Università degli Studi di Milano Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di laurea magistrale in Informatica

A CONSORTIUM BLOCKCHAIN FOR TRACEABILITY IN THE FASHION INDUSTRY SUPPLY CHAIN

Relatore: Prof. Maurino Andrea
Correlatore: Prof. Leporati Alberto Ottavio

Relazione della prova finale di:
Alessandro Provenza
Matricola 769330

Anno Accademico 2018-2019

TABLE OF CONTENTS

LIST OF FIGURES

This thesis does not include an Appendix. If you'd like to review the networks' configurations or the Solidity source code of all the demos developed and discussed in Chapter 5, please refer to the public repository at this address:

https://gitlab.com/FoxHounder/consortium-luxury-blockchain

1 Introduction

Supply chain management is an important piece of enterprise resource planning, defined as the oversight of funds, raw materials, components and finished products as they move from suppliers to the final consumers through manufacturers, wholesalers, carriers and retailers. A good supply chain management will keep product quality consistent and will also prevent either understocking or overstocking of inventory; it will not, however, offer any solid countermeasure to counterfeiting, a problem across all industries that has been damaging both the customers and the economy to a larger extent for years. Companies all around the world have indeed struggled with this phenomenon for a long time, and the industries most affected are the ones from which the most significant rewards can be reaped, such as Fashion and Luxury, where falsification of goods is a common and known reality.

While companies are still trying their best to curb the problem in their supply chain through tighter human controls, better tracking and new technologies, a fairly recent solution has proven to be more effective than others: **Blockchain**, or **Distributed Ledger Technology** (**DLT**) more generically (the terms will be used interchangeably throughout this thesis). With blockchain technology, transactions can be documented in a permanent, decentralized record and monitored securely and transparently. This can greatly reduce time delays and human mistakes. It can also be used to monitor costs, labour, waste and emissions at every point in the supply chain while verifying the authenticity or status of products by tracking them from their origin. That is, with transparency being such a pressing issue, DLT is being used to not only increase intellectual property protection for designers and companies, but also to rapidly trace the entire journey of the finished product as it travels through the distribution chain, right from the initial raw material stage up until it reaches the consumer, thereby giving rise to greater clarity about its origins for both brands and consumers alike. Thanks to big IT tech companies such as IBM, some Proof-of-Concepts (PoC) or real DL networks for supply chain management have already been deployed. However, as we'll see later, more often than not these solutions are based on a private network, meaning that the governance of the entire system is handled by one and only one company. Truth be told, hype and marketing rather than actual business needs were the driving factors in the conception of these systems, in what's now called "*the first generation of blockchain*". Luckily, in current generation, a lot of attention is being paid to the actual use cases and the business implications of a shared ledger, in order to offer real, tangible value to both the end customer and the participants as well: a **Consortium Blockchain**. In this permissioned environment, competitors share the understanding of the ruleset, such as how things work, how transactions should be validated and under which conditions. A shared governance also means that customers have more trust in the system, because the decisional power is, at the very least, shared among few parties, and illegal operations are much less likely to happen and left unreported. Also, since companies often share the same chain of suppliers and distributors, a collective commitment to the chain and its main features, like a score for trust or performance, will help to prevent bad business decisions that can lead to bad quality products, or worse, counterfeits.

Since its conception, DLT has seen a constant rise in interest by both IT companies and brands from all industries. While still not as mature or accessible as other technologies, we can now choose between a vast range of platforms, tools and consensus algorithms, even in known cloud environments such as Azure and AWS. Analysts' predictions all agree in the disruptive power of Blockchain and its potential to revolutionise a broad range of industries, not only financial services, but also government, healthcare, education, energy, entertainment and consumer markets.

The goal of this work is to propose the design for a DL architecture, as well as to implement some key functionalities by basic demos. The wishes are to make counterfeiting a way more complex task to accomplish, possibly unfeasible. While this can be applied to a broad range of applicative domains, we've chosen the Textile industry as the business case, being one of the favourite targets for this kind of illicit operations. The following *Chapter 2* will provide more insights in this regard, along with some very interesting data that will help us understand the economics at play and some social aspects of the Fashion industry.

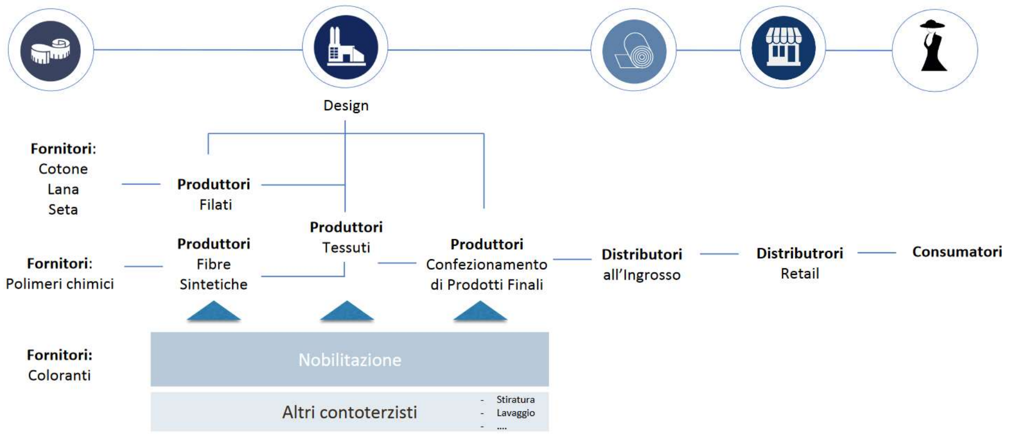


Figure 1 - Basic representation of textile supply chain (MISE and IBM)

The basics of blockchain are covered in *Chapter 3*, where we'll overview the academic and industry interest in it, some use cases and a more detailed explanation of the technology, so that we can better understand if, how and why it can help us to solve our problem. We will then discuss some of the most successful blockchain implementations as of today, as well as some interesting PoCs still in development. We'll see that DLT can be applied to a very wide range of scenarios; however, its design principles are often stretched to a significant degree in order to fit a client's needs. In *Chapter 4* we'll finally describe the requirements of our problem and we'll introduce, mostly by comparison with other successful projects and researches, some key points that can better drive our choices in the design of the architectural solution. These choices will then be revealed and explained thoughtfully and we'll also reason about the most appropriate platform for us and the main features we need. *Chapter 5* will provide details about some core elements of the solution and it will overview the fundamentals of its functionalities. The actual code of the demos will be hosted to a public repository in GitLab, available for review. Finally, conclusions will be drawn in *Chapter 6.*

2 Business Case Study

A quick look at the market reveals that the textile industry itself is healthy. This data from the Italian market covering the past two years not only confirm that Italy's Fashion industry is still one of the strongest in the world, but it also reminds us of the value of the **Made in Italy** brand and economic damage of counterfeits. Italy is indeed home of 20% of world production and 27% of world exports of final goods (over 50% in Europe, for a total of €3.8 billion in 2017). It is, however, victim of counterfeit products that amount to a loss of business for over €250 million, with the Fashion industry being the primary target (38% of the total, 66% counting accessories) with over €100 million of economic loss that also damaged the workforce leading to around 30 thousand potential jobs lost.

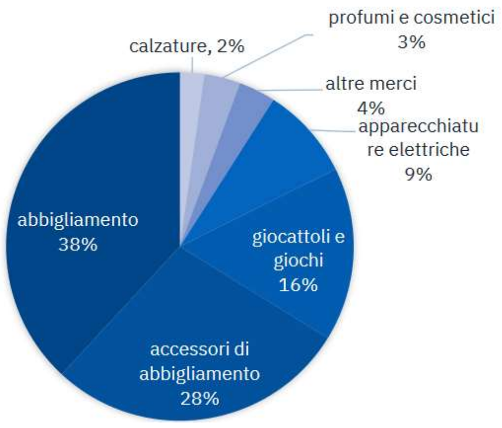


Figure 2 - Counterfeits percentage by area in Italy
(in descending order: fashion goods, clothing accessories, toys, electronic devices, others, cosmetics, footwears)

Unfortunately, studies from the OECD (Organisation for Economic Co-operation and Development) reported that the trend is not slowing down, rather, there has been a substantial increase in counterfeiting trends [1]. Between 2008 and 2013, counterfeiting increased from 1.9% of the total volume of international trade to 2.5%, a figure that represents more than 460 billion dollars' worth of transactions. As for governments, counterfeiting means lost tax revenue, higher unemployment and expenses associated with compliance with anti-counterfeiting legislation.

The very nature of supply chain is part of the issue: a supply chain is essentially a network of interconnected supply companies. As part of a general trend towards specialisation, which is in turn the result of the increasing complexity of products as well as permanent cost and price pressure, many companies have decided to outsource production processes. In many cases, manufacturing is outsourced to countries with lower production costs (developing countries for the most part), where controls are harder to apply and be standardized. *Figure 3* is a simplified example of a company producing sports goods, highlighting the many different layers of the supply chain. In order to manufacture a T-shirt, for example, the cotton crop first needs to be planted and picked, and then the raw cotton sorted, washed woven and dyed. Only then does it move on to the labour-intensive stage of garment manufacture, before finally being delivered to the wholesaler or retailer. It is not uncommon for some producers of shoes and apparel to have way more than a dozen suppliers scattered across several countries. Today, all these steps are weakly connected with exchange of information, mostly on distributed databases that back up different applications from different vendors, making interoperability a challenge. Even with a good system overall, there are still plenty of blind spots that allow fakes to be created and sold. Most companies realize that there's a need for a more robust solution, heavily based on unique identifiers, reliability, precision, privacy, real time data and security; however, most companies involved have little to no experience with blockchain. The most quoted reasons are complexity and lack of standards or regulations, that is, the business domain and the relationships seem hard to arrange. The alternative for most brands is to maintain customer relationship management (CRM) technologies to record interactions with their customers. However, the customer records are often unreliable or even non-existent. The current owner of a product is often unreachable by the brand as products can change hands after purchase. This can lead to mistargeted communications, hence to a waste of resources.

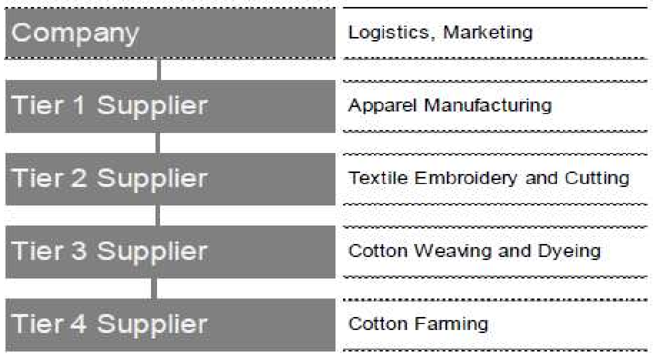


Figure 3 - Schematic illustration of the value chain of a textile company (Puma)

The implementation of a DL-powered system would bring some relevant benefits for the textile industry, such as the guaranteed traceability along the supply chain, meaning:

- Certification of origin and composition of the product. Identification of the composition of the item, the place where the fabric was produced, and which chemicals were used by participants in the supply chain for the benefit of consumers.
- Transparency along the supply chain and ownership. A link between goods and their digital identities: the passage of a product from one actor to another is recorded in the blockchain where each transaction is visible to all those with the right permissions.

Secondly, it would help to protect the brand itself, such as the Made in Italy, by:

- Combination of DL and RFID technology. This would make counterfeits generation much more complicated, because even if one applied a legal RFID on a fake item, there would be no history of its raw materials on the ledger, as they were never registered to begin with.
- Protection of intellectual property. Authenticity can be easily verified by customers, while designers and brands' owners can trust the ledger when it comes to their own work.

But even more importantly, perhaps, it's about **transparency** for the customer and **sustainability** for the industry and all workers across the globe. Typically, information made public about a company's sustainability, if it's made public at all, is self-reported by that very company. Even if its environmental or social performance is independently substantiated, it's through a narrow certification body [33]. Blockchain has the potential to make a change, because it can authenticate a richer, more accurate global ledger of a company's actual social and environmental performance, providing society with a more realistic assessment of its impact.

In the two markets posting the highest export growth of raw materials in recent years, Bangladesh and Cambodia [2], the legal minimum wage is only a fraction of what people need to survive, creating problems associated with poverty. These include malnutrition, poor access to healthcare, lack of social security, bad living conditions, restricted access to education and limited participation in the country's cultural and political life. While, admittedly, this is out of scope from an IT perspective, I believe that these observations are totally relevant and important in their wider, social impact.

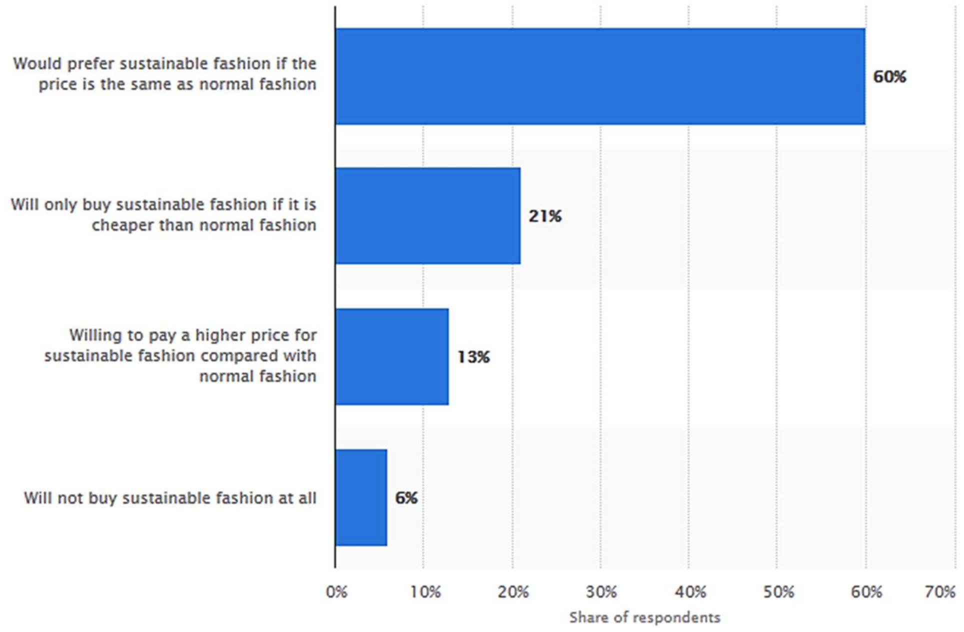


Figure 4 - Willingness to purchase sustainable fashion (Statista)

In this regard, in 2019, 200 fashion brands and retailers took part in the *Fashion Transparency Index* [3] study which showed that sportwear and outdoor brands were leading the way on the transparency front. The five brands who scored the highest on their supply chain transparency are Adidas, Reebok and Patagonia, each scoring 64%, Esprit with 62%, and fast-fashion clothing chain H&M scoring 61%. However, as of 2019, only 10 brands are disclosing information regarding where and who are they getting their supply of raw materials like viscose, wool, etc. Ideally, if we could record data for several KPIs, such as resource management (e.g. emissions), financial management (e.g. R&D), employee management (e.g. work environment and benefits) and more, as it's done annually by research institutes (e.g. Corporate Knights Global 100 [4]), we would be able to picture up-to-date visualizations not only to reward the "most green" brands, but also to help those brands drive their business decisions. The relevance is also backed up by statistics such as *Figure 4*, based on 5000 respondents worldwide, that shows the willingness of consumers to buy sustainable fashion worldwide. In 2018, 60 percent of respondents stated that they would buy sustainable fashion if it were the same price as normal fashion. The same study lists a good number of reasons and real case studies as to why more transparency and sustainability would be a great step forward for the industry. More data from the *Lanieri Fashion Tech Insights 2018* [5], the annual report about new tech trends in the fashion world, reveals that for 6 out of 10 millennials, luxury shopping will be impacted by new technologies, blockchain included, which is more welcomed than technologies such as facial recognition due to privacy concerns. For example, in Italy 1 out of 5 highly favours data protection and the certainty that data do not get shared to third parties without consent.

3 Blockchain Today

The concept of *chain of blocks* was first described in 1991 by Stuart Haber and W. Scott Stornetta [6], whose goal was to design a system where data timestamps could not be tampered with. The technology known as Blockchain, however, was first detailed in Satoshi Nakamoto's whitepaper [7] and popularized by its most successful implementation yet, the Bitcoin. Today, the most notable form of this technology is still the cryptocurrency in all its versions, such as Bitcoin, Ethereum, Ripple, EOS, Litecoin, Cardano and Monero, just to name a few, and Facebook's Libra more recently. That being said, modern Blockchain research is moving ahead fast, trying to come out of the cryptocurrency shadow.

Indeed, as the technology grasped the attention of governments, large corporates and international IT companies over the years, the investments in research outside of cryptocurrencies paid off with a good number of early implementations, especially for the supply chain management. Currently, the interest and commitment are very high, with a worldwide spending projected to hit $12.4 billion by 2022 according to *International Data Corp* [8] and $2.9 billion spent by governments in 2019 alone, a stunning 89% increase over 2018. In PwC's 2018 blockchain survey [9] of 600 executives from 15 territories, 84% say their organisations are actively involved with blockchain. *Gartner's Hype Cycle* predictions in 2018 (Figure 5) also explain the very fast rate of developments in recent months.

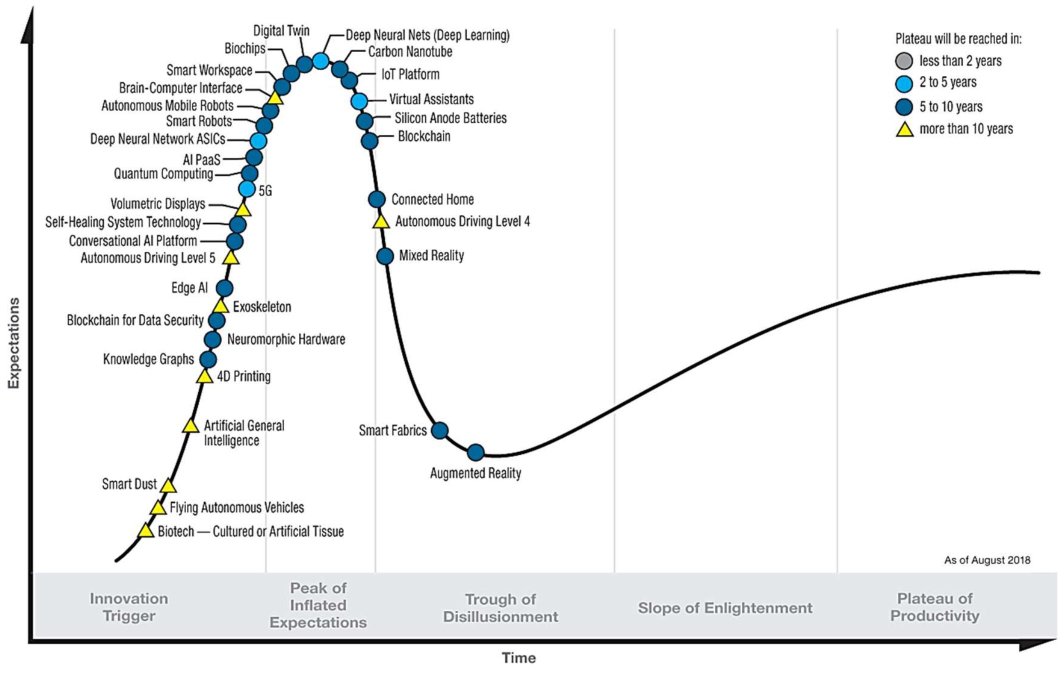


Figure 5 - Gartner Technology Hype Cycle 2018

But Blockchain technology comes in all shapes and sizes. Figure 7 shows *Gartner's Hype Cycle* specific for Blockchain as of July 2019, that highlights some of the most discussed business use cases and academic interest for this technology. The *Gartner 2019 CIO Agenda Survey* reports that CIOs believe that the business impact of Blockchain will be transformational across their industries within 5 to 10 years and around 60% of them expects to adopt the technology in 3 to 5 years, once the governance policies are well defined, which seems to be the greatest obstacle so far [24]. In that regard, it's reported that for Blockchain capabilities to reach the *Plateau of Productivity*, considerable work needs to be completed in nontechnology-related activities such as standards, regulatory frameworks and organization structures.

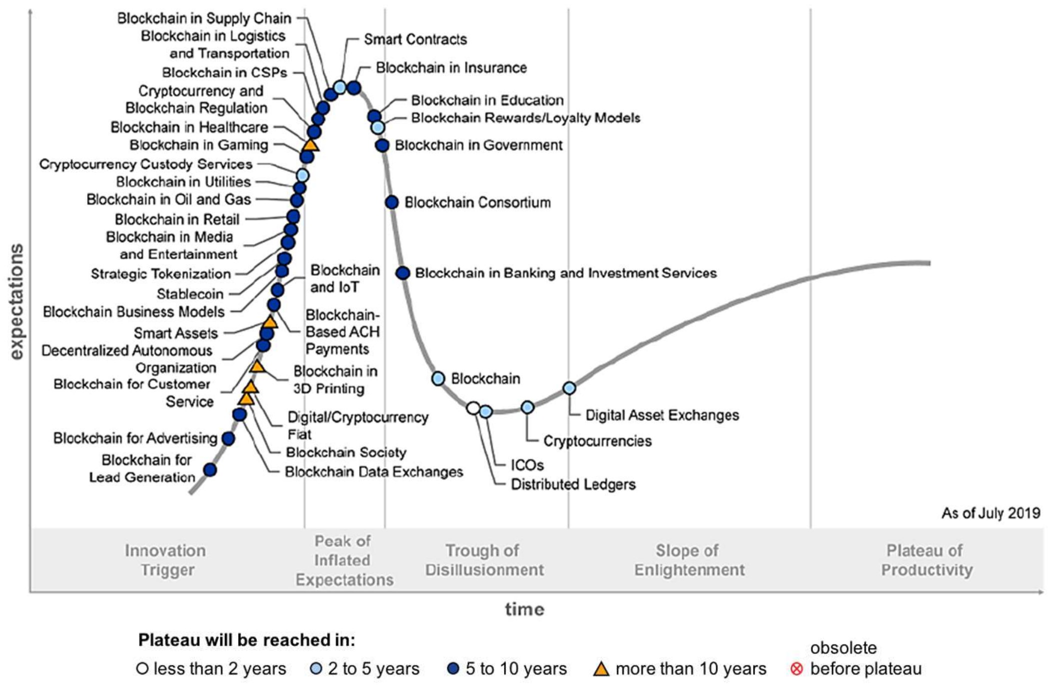


Figure 6 - Gartner Blockchain Business Hype Cycle 2019

On the front of research, Deloitte is committed to make an impact [10]. In its annual *Tech Trends 2019* report, Deloitte reported Blockchain as one of the 4 core areas that will lead the digital revolution for the near future.

"Large enterprises and consortia are deploying enterprise-grade blockchain solutions, avoiding complexities in traversing multiple disparate databases. With technical hurdles and policy limitations being resolved, we will likely see breakthroughs in gateways, integration layers, and common standards in the next few years. Concerns around scalability and cost-performance of transaction processing are being addressed as proof-of-stake becomes a viable alternative to proof-of-work consensus, and enterprise tools have emerged to manage and maintain high-performance blockchain stacks. When further breakthroughs occur, expect blockchain to become even more ubiquitous. The door will be open for cross-organizational business process reengineering, an arena that encompasses transformation and possibilities across industries, functions, and geographies."
(Deloitte Tech Trends 2019, Beyond the digital frontier)

Blockchain is growing in different directions, forcing new business models to come along and IT companies to adapt. The list includes digital identity, digital content storage (such as Public Administration's documents or a governments' services), customs declaration and notary, digital assets and intellectual properties in the media industry, insurance and banking, digital voting, healthcare for medical history and energy trading [25].

Also, as Blockchain made a name for itself in the finance world, governments had to act quickly to regulate its use and to promote more research. Italy, which so far has been slow in the adoption of Blockchain, is one example. Its government has made a first step with *DL Simplification Decree 2019*, which acknowledges and gives definitions to common DLT terms such as *Smart Contracts*, also detailing how they will be evaluated. The Decree Law was a cooperation between the Ministry of Economic Department (MISE) and Fulvio Sarzana, *National Strategy Blockchain* member. Moreover, MISE has been working intensively these years on more projects, one of which involves traceability in the textile industry in order to protect and certify the Made in Italy. This first pilot project, presented on 13th March 2019, is a collaboration with IBM and it involved a first feasibility study and design thinking sessions, together with some Italian Fashion Houses, in order to highlights the advantages of traceability of items in a supply chain, the defence against counterfeits and the added value of a more sustainable industry. The results were presented in November 2019 and, in February 2020, it has been announced that a Ministerial Decree is under work to boost companies that uses Blockchain. This also represents Italy's contribution to a European initiative launched by UNECE (United Nations Economic Commission for Europe) called *Transparency and Traceability for Sustainable Textile and Leather Value Chains*. More details and new updates are freely available at MISE official page for Blockchain[1].

Finally, one thing that stands out is the nature of today's deployed Blockchain. More research unveils that most of the solutions are private blockchain, that is, blockchain with very restrictive access such as an Intranet. It comes with no surprise, then, that even in *Forbes's Blockchain 50* [11], a list of the most influential companies in terms of Blockchain research and development, most projects are private blockchains. Early adopters will be disappointed to hear that most profit comes from a centralized use of Blockchain with known intermediaries. As we'll see later, a private blockchain is a questionable implementation of DLT design principles, considering the overall potential of a distributed system that features a strong layer of security. The reasons for this span from hype to lack of trust, both in people and in the technology, the need for profit and more frequently a false urgency to sell a Blockchain in scenarios that can probably, perhaps even surely, be solved by other means. An interesting case is the MISE's pilot project with IBM mentioned above, being a government-driven blockchain; since the goal is to protect the Made in Italy, the Italian government is the one making the rules in and out of the ledger. We'll explore in more depth these considerations in later chapters.

3.1 S.W.O.T. Analysis

The SWOT analysis is a method of analysis developed by Albert S. Humphrey which is a useful technique for identifying the Strengths, Weaknesses, Opportunities and Threats of any given technology or product. Strengths and weaknesses are often internal to the technology itself, while opportunities and threats are generally related to external factors. Many issues like unwanted centralization, slow transaction verification times and low throughput aren't easy to solve, and they can put a client's will to invest to a halt. Let's review the SWOT points for Blockchain.

The main **strength** of Blockchain is its distributed trust enabled by distributed power, through an immutable, shared and public ledger. Strengths also include the operational efficiency of the system: it facilitates easier sharing of verified information, without the needs for documents to be passed along, not to mention that it eliminates any central authority with full access to the data (although this is one of the most disregarded aspect of today's DLs). On top of that, it features a very robust layer of security granted by encryption, certificates and tamper-proof data storage. Altogether, this has the potential to move us to the *Internet of Value* from the *Internet of Information* we have today.

[1] https://www.mise.gov.it/index.php/it/blockchain

The **opportunities** come in good numbers. In addition to the actual business cases that will be explored later, a DL provides a great platform for analytical research. More importantly, it gives back control to the user when it comes to sensitive data: all permissions protocols are stored on the Blockchain and they are all publicly known beforehand. With the world becoming more and more digital in all aspects of life, people will be more willing to accept the concept of Blockchain in a relatively short time, opening to an interesting public scenario, with the certainty of data being as secured as possible.

However, there are also **threats** to be considered. Looking at the future, a potential and frequently discussed threat is quantum computing: the "*very robust layer of security granted by encryption, certificates and tamper-proof data storage*", as said before, relies on cryptographic hash functions. The most common are known as SHA, *Secure Hash Algorithms* (Bitcoin protocol for example uses SHA-256), and they're widely used outside of blockchains, while others were designed for cryptocurrencies, such as X11. They are all very robust standard against today's computing power and brute force attacks, since it takes on the order of 2^{128} basic operations to get a private key (even less if other hash functions are used), but it's a known fact that a relatively large quantum computer could decrypt that data in a considerable shorter amount of time, around the order of only 128^3 basic quantum operations to get the result [12], which is not immediate but it's still very practical. This is a real problem as the possibility of mining attacks and hacks is not low: they're already common today against Bitcoin and Ethereum. Also, the goal to displace the central banks may not be warmly welcomed by governments, as demonstrated by the recent announcements of European Union and its desire to have "more control" over cryptocurrencies, sometimes for security reasons [13]. Finally, it's important to consider that DLT is currently a very fast changing environment surrounded by hype, which make easier for design errors to occur.

This brings us to one of the **weaknesses** of DL: business rules change frequently but a blockchain does not. This is a fundamental aspect to keep in mind during the design phase, as blockchain is mostly not a modular system. For example, an old encryption module cannot be easily replaced. Same goes for data structures: what if a business rule change and there's the need to export data to a new blockchain with the correct data models? It can be done, but it's very far from being a no-brain operation. Any changes, however small or big (e.g. introduction of GDPR regulations) will make a heavy impact on the maintenance and evolution of the existing ledger. Secondly, scalability issues: while this is a problem that's being tackled right now with promising results, a DL is in itself unable to manage a situation of overload (too many transactions at the same time). This is mainly a problem of public DLs in areas which require throughput, such as finance, while areas such as supply chain management are more easily manageable. Another high-risk factor, one that's affecting Bitcoin today, is unwanted centralization, that is, mining pools and large mining farms created by few people with lots of resources. This is more a problem with consensus algorithms than DLT itself, but it's one that undermines the very core idea and reason to be of the system. Other algorithms, such as Proof-of-stake used by Ethereum, often lead to the same problem, as fundamentally it's due to the reliance of miners who are essentially and solely motivated by an incentive scheme, which is also contingent both on diminishing supply of bitcoin and market price of it, as well as transaction fees. The enormous energy consumption required for the mining process for some blockchains also raise continuous concerns on environmental costs.

3.2 Technology Deep Dive

What makes DLT so attractive in its trustworthiness lies in the design of its architecture. Fundamentally, it's a smart use of known technologies and components that interact with each other. It's important to note, however, that not all DL platforms work the same. Some of the functionalities that we'll mention here do not apply to all of them, but we can get a basic definition from the Hyperledger blog that highlights the main concepts common to all:

- A p2p network, with different governance models.
- A consensus protocol, which is how parties agree about the state of the system.
- Smart contracts (or chaincode), computer programs deployed to the chain that automatically execute under predefined conditions.

"A blockchain is a ***peer-to-peer distributed ledger*** *forged by* ***consensus,***
combined with a system for ***smart contracts*** *and other assistive technologies"*
(hyperledger.org)

The ledger is updated with blocks. A block is a set of transactions bundled together, taken from the transaction pool. Each transaction has a fee associated, and miners choose which transaction to use to create the block also based on these fees. The race to win the chance to add a block, and get the transaction fees, is part of the consensus protocol (e.g. proof-of-work). We can say, then, that a generic DL block consists of the following four pieces of metadata [21]:

- The reference to the previous block.
- The proof of work, also known as a nonce.
- The timestamp.
- The Merkle tree root for the transactions included in this block.

Each block is timestamped and has a reference to the previous block. Combined with cryptographic hashes, this timestamped chain of blocks provides an immutable record of all transactions in the network, from the very first block (called the genesis block). Any change to a block's content (e.g. the amount of an asset in a transaction) will inevitably change its hash, therefore breaking the link between block N and block N+1. This is easily verifiable thanks to the data structure employed, the **Merkle Tree**, that enables the efficient reading and storing of transactions data in large datasets.

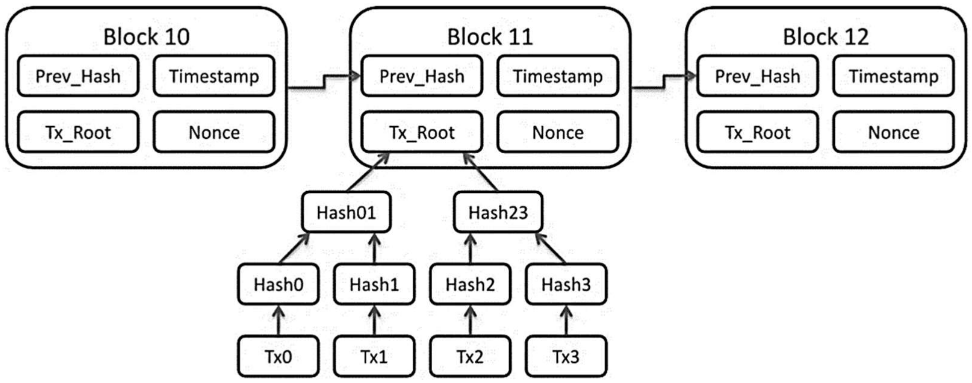


Figure 7 - Merkle Tree data structure in a block (edX)

It suffices to present only a small number of nodes in a Merkle tree to give a proof of the validity of a branch; at the same time, any attempt to change any part of the Merkle tree will eventually lead to an inconsistency somewhere up the chain, as the hash propagates upward (see Figure 7). The hashing algorithm also plays a key role in the performance, therefore the SHA-256 is the standard choice. Finally, the nonce is the string to find (in a proof-of-work consensus) that, concatenated with the block's hash, will output another hash with the desired difficulty.

As said above, a block consists of transactions. When it comes to spreading new transactions and blocks, being a p2p network, something called *gossip protocol* is generally used. It works as follow: a node performs a transaction, signs it and sends it to the neighbour nodes. When these nodes receive the transaction, they will check it in a process called *validation*. If the validation passes, they send it to their peers. Eventually, all connected nodes, including miners, will receive the transaction and will race to find the nonce required to win the challenge in the consensus protocol. Again, being a p2p network where latency is a factor to consider, miners will likely receive different transactions at different times, so we could say that each miner is actually building its own block. One key distinction to make, in this process, is between validation and consensus. A validator performs validation by verifying that transactions are legal. This is done, for example, by checking the wallet of the parties involved or the signature that gets sent along with the actual data. However, consensus involves determining the ordering of events in the blockchain and coming to agreement on that order. Essentially, consensus involves agreeing on the ordering of validated transactions. The validation, then, is an important step that precedes the consensus, meaning that we may very well have something that is valid that the network does not agree upon.

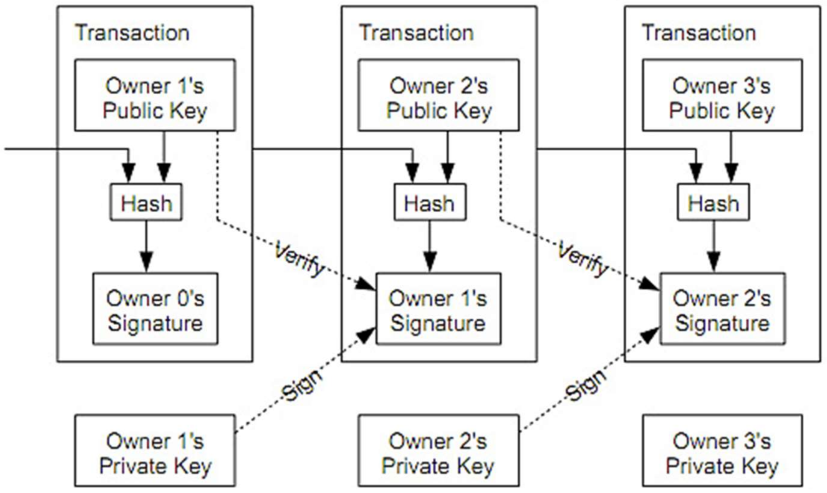


Figure 8 - Digital signatures and validation in transactions

3.2.1 Governance Models

Distributed Ledgers have seen an evolution in how their governance work. In the first generation, the common models were public, mostly for cryptocurrencies, and private, used as an internal system for the organizations to manage their data. As time passed, the main players such as IBM realized that the enterprise world would have greatly benefited from a shared governance, hence conceptualizing the permissioned network and releasing the first versions of permissioned platforms (e.g. Hyperledger Fabric). The choice between governance models should always be driven by the application at hand, and the supply chain management is an ideal use case for permissioned blockchains. Nonetheless, let's overview each one of these models to better understand pro and cons of each one and why a permissioned model will likely be our choice.

A **public ledger** is permissionless, that is, the read and write access to the ledger is available to anyone that's willing to set up a node, which makes it impossible for all peers to know each other. This means that the level of trust in the network is extremely low, leading to tighter, more frequent checks per transactions that have consequences on performance. The obvious positive return is that a transaction added to a block is unlikely to be invalid as the network is more secure and fully decentralized. The cost of the infrastructure is relatively low, except for the enormous energy expenses, in part due to the consensus protocol and the mining activities to which all peers can participate. As the architecture requires, peers need an incentive to set up a node: in exchange for their computational power, public ledgers use proof-of algorithms such as proof-of-work (Bitcoin) or proof-of-stake (Ethereum) that often offer a monetary prize for the winning mining block. The fundamental idea is that the more peers, the more secure the chain gets, as it would be unfeasible for anyone to take over 51% of the network. However, it must be noted that the degree to which someone can have an influence in the consensus process is proportional to the quantity of economic resources that they can spare. In the proof-of-work protocol, this leads to enormous mining farms. In the proof-of-stake protocol, richer peers enjoy more power by design.

The opposite model of a public ledger is a **private ledger**. Permissioned, the read and write access is given upon invitation and identity verification. Peers all know each other or, as it often stands, they belong to the same company. Since there's a fundamental trust between peers that leads to less checks per transactions, a private ledger is generally a lot faster thanks to different consensus algorithms and a lower number of peers compared to public ledgers. Costs per transactions are lower as well as energy costs, as the custom designed proof-of algorithms such as proof-of-authority do not require mining. There's no need for an economic incentive to set up a node, as it's in the interest of the peers themselves to keep the system online and secure. That said, it's the weakest choice in terms of compliance with the DLT design principles as it could be seen as a glorified central database, although robust against external attacks as every node keeps a copy of it.

The last model we're going to cover is the **consortium ledger**. This third category of blockchains, conceptualized and suited for most enterprise needs, is composed of permissioned blockchains which allow a mixed bag between the public and private blockchains with lots of customization options. The available options include allowing anyone to join the permissioned network after suitable verification of their identity, and allocation of select and designated permissions to perform only certain activities on the network. Ethereum co-founder Vitalik Buterin has defined the differences between private and consortium ledgers, saying:

"So far there has been little emphasis on the distinction between consortium blockchains and fully private blockchains, although it is important: the former provides a hybrid between the 'low-trust' provided by public blockchains and the 'single highly-trusted entity' model of private blockchains, whereas the latter can be more accurately described as a traditional centralized system with a degree of cryptographic auditability attached."
(Ethereum Blog, On Public and Private Blockchains, August 6th, 2015) [14]

In short, with a consortium blockchain model, rather than allowing anyone with an internet connection to participate in the verification of transaction processes or vesting full control in only one company, some selected nodes are predetermined. Accordingly, the consortium platform model offers many of the advantages associated with a private blockchain, including enhanced efficiency and transaction privacy, but it doesn't consolidate the power within a single company.

3.2.2 Consensus Protocols

The consensus protocol is a key element of design in a DL system. It refers to the process of achieving agreement among the network participants as to the correct state of data on the system. Consensus leads to all nodes sharing the exact same data. A consensus algorithm, hence, does two things: it ensures that the data on the ledger is the same for all the nodes in the network, and, in turn, prevents malicious actors from manipulating the data. The consensus algorithm varies with different blockchain implementations. While the Bitcoin blockchain uses proof-of-work as the consensus algorithm, other blockchains and distributed ledgers are deploying a variety of algorithms, like the proof-of-stake, proof-of-burn, proof-of-capacity, proof-of-elapsed time and many others, depending on their unique requirements. It's worth to quickly overview the most common available options, so that we can start to understand why some are more attractive than others for our permissioned scenario.

Most people have heard about **proof-of-work** (PoW), the consensus protocol used in Bitcoin. At its core, the algorithm proceeds as follow: transactions are bundled together in form of blocks, and miners verify that these transactions are legitimate and choose the ones they want to create a block with, often based on the transaction fees they would get. Miners then race to solve a mathematical problem, consisting in finding a valid hash with the given seed and the pre-set difficulty (a precise number of zeros in front of the hash), which is defined by software and is readjusted periodically based on the hash rate (the combined computing power of all miners) so that the block generation rate remains somewhat constant. The PoW consensus protocol is probably the safest when it comes to network security, as it was designed for a P2P network where peers don't know nor trust each other, at all. Such feature is paid with lower performance and an enormous use of computing power, which PoW is often and rightfully criticized for. Also, security is based on the assumption that nobody can take control of 51% of the network, as it would be economically unfeasible; unfortunately, Bitcoin and the big mining farms proved the fragility of this assumption. The most common alternative to proof-of-work is **proof-of-stake** (PoS). Here, block producers are selected based on how many coins they have. This selection can be made randomly or by choosing those who get the most votes, but the more coins you have the higher the probability that you'll be selected as the next producer. The process of staking can be delegated to other nodes, either because the user can't vote regularly or to increase the voting power of two users; in this case, profits are then shared. The amount of coins required changes similarly to the difficulty in PoW, and there's a lot more to it that we'll not cover; fundamentally, though, the idea is that those with the most coins in circulation have the most to lose. The PoS consensus protocol is cost efficient in terms of energy and hardware and it gets more secure as more people use the network. However, it's really no different from PoW when it comes to cons: whereas PoW trade power with energy, PoS does the same with coins, and who's rich get richer faster. Moving to a different idea, chipmaker Intel has come up with its own alternative consensus protocol called **proof-of-elapsed time** (PoET). This system works similarly to proof-of-work, in the sense that the fastest participant wins, but it consumes far less electricity. Simply explained, instead of having participants solve a cryptographic puzzle, the algorithm uses a trusted execution environment (TEE) to create a *real* random number. Each participant waits for its own random amount of time and the winner is the first participant to finish waiting. PoET is a consensus mechanism that is often used on the permissioned blockchain networks to decide the mining rights or the block winners on the network. Based on the principle of a fair lottery system where every single node is equally likely to be a winner, the PoET mechanism is based on spreading the chances of a winning fairly across the largest possible number of network participants. The downside of the protocol is its dependency on TEEs enabled hardware.

All these protocols are suited for public, permissionless networks. However, when it comes to private or permissioned blockchains with a relatively small number of interested parties, our requirements change. Restricting membership greatly reduces the need for elaborate algorithms to prevent fraud on permissioned blockchains; instead, these applications must ensure that complex workflows and transactions are implemented correctly. The protocols that follow are faster, more scalable, consume minimal computational resources and offer a higher transaction throughput by design, but only work in more centralized networks. The **Practical Byzantine Fault Tolerant** (PBFT) [23] is one of these consensus protocols. Each node in the network maintains its own internal state and when it receives a message, it runs a computation and prepares a decision about the new message received. The individual decision of each node is sent to the leader of the nodes, which confirms the trust on the new message based on the decisions from all the nodes. It incorporates several important optimizations that improve the response time with respect to previous algorithms by more than an order of magnitude. Even more apt to a permissioned environment is the **proof-of-authority** (PoA) protocol. At the price of full decentralization, PoA networks validate transactions using pre-approved accounts known as validators. The right to be a validator is earned, so there's an incentive for them to retain that position, and by attaching a reputation to their identity, they're also incentivized to act correctly. With this idea, **proof-of-reputation** (PoR) was designed as an upgraded, stronger, and more secure form of proof-of-authority. This consensus model depends on the reputation of the participants to keep the network secure. A participant (a block signer) must have a reputation important enough that they would face significant financial and brand consequences if they were to attempt to cheat the system. Finally, a very interesting proposal from *TrustChain* is called **proof-of-trust** (PoT) which introduces the idea of a *TrustToken*, something that can only be earned by being a good player in the system and which defines a validator's level (Junior, Middle or Senior).

As previously stated, we're looking for something that works best in a permissioned environment, protecting participants from unfair rights assignment. The choice between the ones mentioned here will be dependent on what the platform of our choice can support; we'll continue this analysis in Chapter 4, but we can already anticipate that it will be a protocol that takes into account the identity of the validator, as we'll want to ensure accountability.

3.2.3 Smart Contracts and Tokens

Before businesses can transact with each other, they must define a common set of contracts covering common terms, data, rules, concept definitions, and processes. Taken together, these contracts lay out the business model that govern all of the interactions between transacting parties. Using a blockchain network, we can turn these contracts into executable programs known in the industry as **smart contracts**. It's called a *contract* because the code can control valuable things like cryptocurrencies or other digital assets. They're written in high-level languages such as Solidity or Go and they are deployed on the platform using a special contract creation transaction, which gives the smart contract an address. Smart contracts only run if they are called by a transaction, that is, even though a contract can call another contract, the first contract in such a chain of execution must always be called by a transaction. Along with the code itself, smart contracts keep a state (the overall value of all their data) and they are defined by two key characteristics:

- They are **Immutable**. Once deployed, the code of a smart contract cannot change. Unlike with traditional software, the only way to modify a smart contract is to deploy a new instance.

- They are **deterministic**. The outcome of the execution of a smart contract is the same for everyone who runs it, given the context of the transaction that initiated its execution and the state of the Ethereum blockchain at the time of execution.

Moreover, transactions are **atomic**, regardless of how many contracts they call or what those contracts do when called. Just like a regular database transaction, a DL transaction executes in its entirety, with any changes in the global state recorded only if the entire execution terminates successfully; in case of error, a rollback is performed.

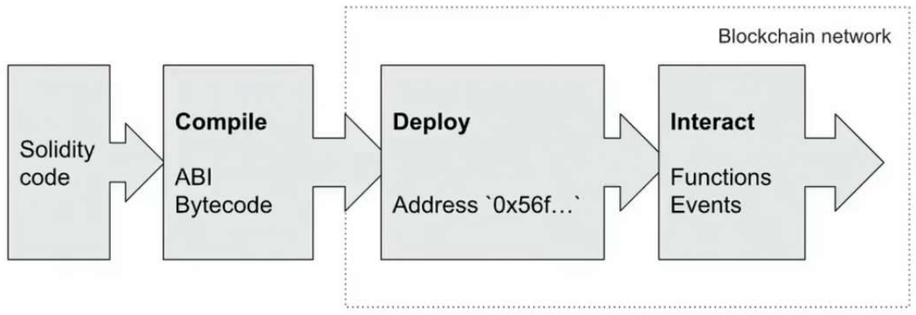


Figure 9 - Development steps of a smart contract

With smart contracts we can also create and manage assets, or tokens. A **token** is a digital asset that can be exchanged between two parties without intermediaries. The most obvious use of tokens is as digital private currencies. However, tokens can be programmed to serve many different functions: they can be resources, tangible assets (e.g. a car), an identity or more. Standards allow for uniformity between tokens and for wider adoption and use of tokens. There are two main groupings of tokens, fungible and non-fungible. Quickly explained, fungible tokens mean that one token is worth any other token, like money, where $1 is always equal to any other $1, while non-fungible tokens all have a unique ID. In more depth:

- Fungible tokens are interchangeable. We can exchange 1 BTC for another 1 BTC and it makes no difference to the owners. Non-fungible tokens hold more sensitive, not-exchangeable information (e.g. a birth certificate cannot be exchanged for another).
- Fungible tokens are uniformed, as there’s no structural difference between a BTC and another. That’s not necessarily true for non-fungible tokens.
- Fungible tokens can be split into more tokens, the sum of which makes the original value. Non-fungible tokens are already an elementary unit that cannot be split.
- Fungible tokens are defined with the ERC-20 standard, whereas non-fungible tokens are often defined with the ERC-721 standard.

3.3 Distributed Databases

When talking about blockchain and distributed ledger, what we’re actually talking about is the data layer of the architecture. On top of that lives the application layer, like a Java EE or Node.js application, and the client. However, calling blockchain just a database would be selling it short, as we’ve seen there are several unique characteristics that make it stand out for certain types of applications. That being said, the very same principles and technical considerations apply: to put a database on a single server is undesirable, as we would want redundancy, load distribution and backup, to name a few. The three main architectures to consider when setting up a database cluster are centralized, decentralized, and distributed. Blockchain is a *distributed* database.

In terms of points of failure and maintenance, centralized systems are the easiest to maintain, as there is only a single point of failure. Decentralized systems have more of them but still finite, whereas distributed systems are the most difficult to maintain. However, when it comes to fault tolerance, centralized systems can be highly unstable, having a single leader. If this leader goes down for any reason, the system will go down as well. A failure in a leader for a decentralized system will lead to many decentralized systems. Distributed systems, on the other hand, are very stable and a single failure will not do much damage. A distributed system is also better for scalability, as it's designed (on paper) for an infinite number of nodes. Centralized system does not allow for high scalability while decentralized systems support a moderate degree of scalability, before performance gets compromised. Another thing to consider is how easy it is to set up a network and this is a clear advantage for centralized systems, that can be created quickly with one framework applied everywhere. For decentralized and distributed systems, instead, there's a whole pack of details to think about, such as resource sharing and communications. The complexity, though, is well justified by the possibility of evolutions and diversity in technology. Since, as mentioned, centralized systems follow a single framework, they don't support diversity and their evolution is slower. But for decentralized and distributed systems, once the basic infrastructure is in place, to make things bigger and richer is fairly straightforward.

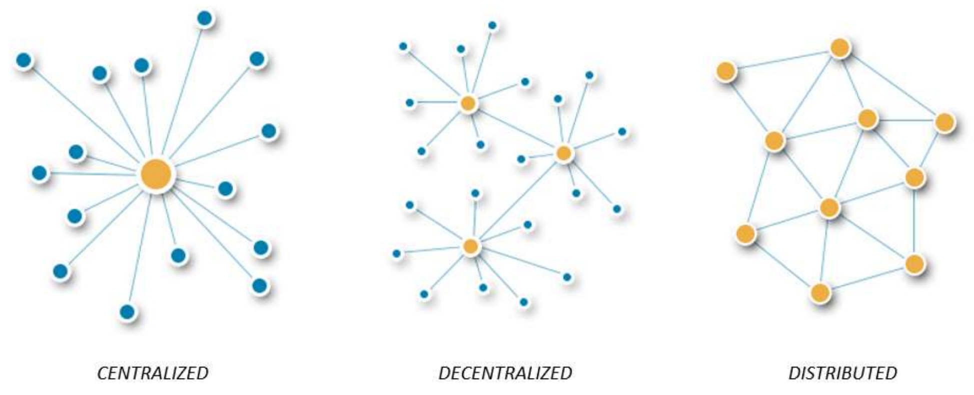


Figure 10 - Centralized, decentralized and distributed systems

Now that we know why Blockchain is designed as a distributed system, it's time to consider if it's indeed the right choice for a given scenario. Some Blockchain critics often question the use of a DL, indicating a distributed database as a viable and cheaper solution. These doubts are legitimate and it's worth explaining the differences between the two and what makes them ideal for different use cases. We'll soon realize that it's more about "who" rather than "what" or "why".

Distributed ledgers, or decentralised databases, are systems that *enable parties who don't fully trust each other to form and maintain consensus about the existence, status and evolution of a set of shared facts.* The heart of this sentence is "*parties who don't fully trust each other*". In a distributed database, we often have multiple nodes that cooperate to maintain a consistent view for their users. The nodes may cooperate to maintain partitions of the overall dataset or they may cooperate to maintain consistent replicas, but the principle is the same: a group of computers, invariably under the control of a single organisation, cooperate to maintain their state. These nodes trust each other, and the trust boundary is between the distributed database system as a whole and its users. Each node in the system trusts the data that it receives from its peers and nodes are trusted to look after the data they have received from their peers. In a distributed database, nodes cooperate to maintain a consistent view that they present to the outside world; they cooperate to maintain rigorous access control and they validate information they receive from the outside world. So, it's no surprise that distributed databases are invariably operated by a single entity: the nodes of the system assume the other nodes are just as diligent as them: they freely share information with each other and take information from each other on trust. Clearly, if an organization has a business problem where the team is happy to rely on a central operator to maintain the records, as it could be sometimes in finance, then a distributed database will do just fine because the central operator will run it for them. But if there's the need to maintain one's own records, in synchrony with peers, this architecture simply won't do. And there are huge numbers of situations where we need to maintain accurate, shared records with our counterparts. Indeed, a vast amount of the cost and inefficiency in today's financial markets stems from the fact that it has been so difficult to achieve this so far. At the end, it all comes down to one key element: in a distributed ledger, a node cannot assume the data it receives from a peer is valid. The peer is probably operated by a completely different entity and even if the other peers know who that entity is, it's still extremely prudent to verify the information. Moreover, if a node sends data to another node, it must assume that such a node might share the information with everyone with no care for privacy. The design principles of Blockchain solve all these issues.

IBM offers a very basic infographic to help decide whether Blockchain is the right choice for a given business scenario, as it's often the case that Blockchain is used for just one of its strengths (security, validation, availability of data). A solution architect must consider that a lot of these scenarios can be solved with older, yet very practical and cheaper solutions, such as central database, backup services and more. We'll dig deeper into these considerations later, when designing the solution.

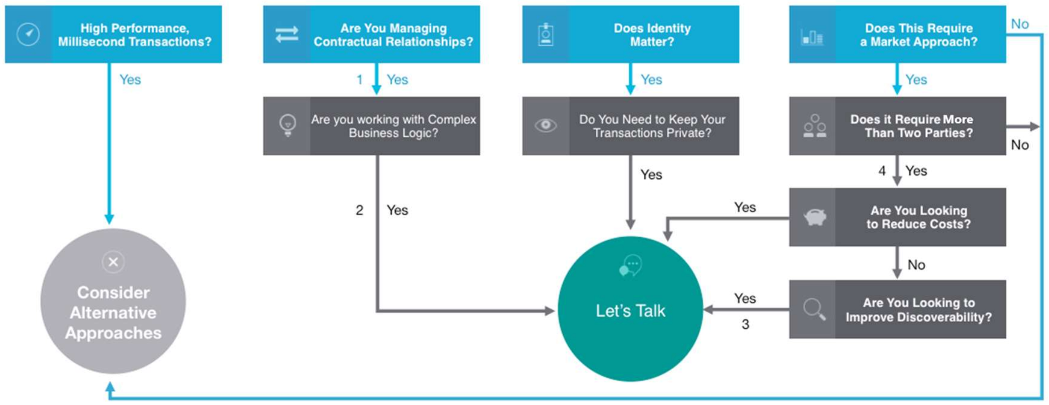

High Performance, Millisecond Transactions?
Yes
Are You Managing Contractual Relationships?
1 Yes
Are you working with Complex Business Logic?
2 Yes
Does Identity Matter?
Yes
Do You Need to Keep Your Transactions Private?
Yes
Does This Require a Market Approach?
No
Yes
Does it Require More Than Two Parties?
No
4 Yes
Are You Looking to Reduce Costs?
Yes
No
Are You Looking to Improve Discoverability?
Yes
3
Let's Talk
Consider Alternative Approaches

Figure 11 - How to decide when to use a Blockchain (IBM)

3.4 State of the Art and PoCs

Blockchain can be used in many different areas of business, but the most common one so far has been the supply chain. What follows is a brief list of PoCs and existing projects featuring DLT from a variety of industries and use cases.

One of the very first and largest Blockchain project in the retail and food industry was led by Walmart, in USA, with the cooperation of IBM. They're using Blockchain to track some products back to their roots. They can scan any product and get information about which farm produced it, or which company transformed it. This application for supply chain tracking is extremely useful and can bring many different use cases, from security factors and loss reduction in case of an infection to more informed users. The problem it can bring associated is the extra cost to put IoT sensor all around the chain to gather data during the whole process (this is a common trade-off, as we'll see). Clearly enough, the most detailed control one wants to have, the highest are the costs. Moving to Europe, Carrefour is using blockchain in order to track where products come from, as consumers increasingly look to ensure that products meet standards regarding ethics and general safety, and would enable shoppers in France to trace where certain food products are sourced. It currently uses the blockchain to trace the production of free-range chicken in the Auvergne region in central France, but it will be extended to trace honey, eggs, cheese, milk, oranges, tomatoes, salmon and hamburgers by end-2025. Other examples include Nestlé, that in July 2019 announced a program that will use blockchain technology to track the origin of milk supplied by New Zealand breeders and Middle Eastern deposits. Finally, another interesting use scenario is brought by the United Nations, that with the *World Food Program* [15] is currently helping refugees to purchase food with only an eye scan, coordinating everything through Blockchain. With the Office for the Coordination of Humanitarian Affairs they're improving donor financing, securing and monitoring supply chains, and data protection, using also Blockchain securing the whole chain of information.

In the automotive industry, Volkswagen Financial Services and Renault led PoCs in 2017 testing vehicle telematics tracking. In this use case, a vehicle's mileage data, engine usage history, repair and maintenance history, and other data can be captured on the Blockchain so that manufacturers, dealers, buyers, insurance companies, and other players know a vehicle's history and activity with accuracy. It is a good use case, as an estimated one-third of used car sales in Germany have manipulated odometers [16].

In the Shipping industry, DHL is already working with Accenture establishing a Blockchain-based track-and-trace serialization system in six areas worldwide. They have a working system populated by more than 7 billion unique pharmaceutical serial numbers and handling more than 1,500 transactions per second. As soon more governments, institutions and pharmaceutical companies join in, the system will offer a better way of cutting costs, elevating security and trust, eliminating error-prone data movements and enabling real-time supply chain transparency.

3.4.1 Consortium Models

The DL systems just mentioned are built with private governance models. As noted in the introduction, this type of architecture is, as of today, the most implemented by far. However, consortium models are being explored; these implementations either have the whole ownership of the network shared among parties or, at the very least, they let the participants use the same network as long as they agree to its rules.

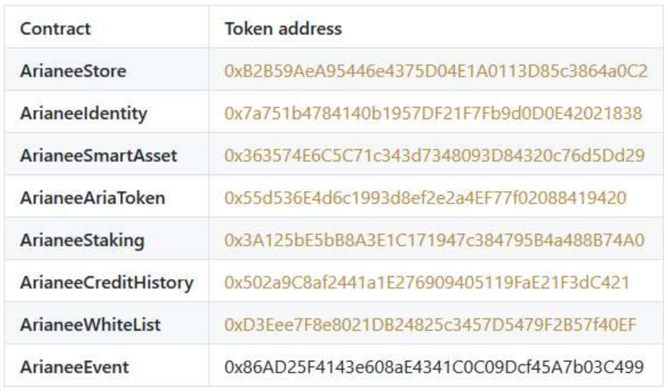
The program TradeLens, developed by IBM for Maersk, the largest container shipping company on the planet, is an interesting case falling in the second category. The project involved not just Maersk but a series of third parties, such as the shipper Dutch customs and the U.S. Department of Homeland Security, with all of them tracking containers remotely. The tech's reliance on cryptographic signatures makes it harder for anyone to mislay goods or tamper with labels while the cargo is on the move. However, it's hard enough to get enterprises that compete to work together as a team, but it's especially tricky when one of those rivals owns the team. The platform was designed such that Maersk's rivals would act as *trust anchors* and run full blockchain nodes on the network. The problem was that Maersk's rival shipping carriers were concerned about joining the platform on a less than equal footing, that is, competitors did not trust Maersk in guaranteeing them the same rights. In all fairness, building big blockchain networks is not an easy, straightforward task, especially from a business perspective, and IBM is exploring different approaches to the problem.

"If you start off small and centralized, the challenge will be getting the next big trust anchors on board. On the other hand, with the decentralized/consortium approach, you can have multiple competitors and their lawyers all asking questions and it's going to take some time. You've got to pick your poison"
(Jerry Cuomo, Big Blue's vice president for blockchain technologies)

Three very promising Consortium Blockchain projects in development today are *AURA*, *Virgo* and *Arianee*. AURA is a PoC announced in March 2019 from luxury brand conglomerate LVMH, owner of the iconic Louis Vuitton label. It's supposed to go live in Summer 2019 for LV and Dior. As it develops, LVMH plans to open the platform up to its remaining luxury brands, and eventually even to its competitors. AURA is based on a permissioned version of the JPMorgan-developed Quorum, an Ethereum blockchain focusing on data privacy. LVMH hired a full-time team to work on the project for a year now, alongside Microsoft Azure and Ethereum design studio ConsenSys.

"To begin, AURA will provide proof-of authenticity of luxury items and trace their origins from raw materials to point of sale and beyond to used-goods markets. The next phase of the platform will explore protection of creative intellectual property, exclusive offers for each brands' customers, as well as anti-ad fraud."
(Ethereum design studio ConsenSys)

On occasion of the Fashion Global Summit in Milan held in October 21st, 2019, Virgo has been announced as the collaboration of four companies: Temera, PwC, Luxochain and Var Group. In addition to the blockchain, Virgo also includes other technologies, such as RFID (Radio Frequency Identification), UHF (Ultra High Frequencies) and NFC (Near Field Communication). Virgo features a smartphone application where the consumer can check the luxury product in advance, prior to the purchase. Finally, Arianee is an ambitious, independent organization whose mission is to build the first perpetual, anonymous and trusted record of all the world's luxury asset, enabling a revolutionary link between owners and brands, and to establish a global standard for digital certification of valuable objects (e.g. jewellery) with the adoption of the Arianee protocol. The idea was born back in 2016 and the main network is online since the end of 2019 (it runs on the POA Network Core mainnet, id=99), with future milestones already planned. Arianee also recognizes and uses different technologies to enable anti-counterfeit controls, such as serial numbers, security labels, RFID and NFC. All of them merge into one Smart-Asset, defined as a digital unfalsifiable certificate representing a unique asset, connecting items, owners and brands [17].



Contract	Token address
ArianeeStore	0xB2B59AeA95446e4375D04E1A0113D85c3864a0C2
ArianeeIdentity	0x7a751b4784140b1957DF21F7Fb9d0D0E42021838
ArianeeSmartAsset	0x363574E6C5C71c343d7348093D84320c76d5Dd29
ArianeeAriaToken	0x55d536E4d6c1993d8ef2e2a4EF77f02088419420
ArianeeStaking	0x3A125bE5bB8A3E1C171947c384795B4a488B74A0
ArianeeCreditHistory	0x502a9C8af2441a1E276909405119FaE21F3dC421
ArianeeWhiteList	0xD3Eee7F8e8021DB24825c3457D5479F2B57f40EF
ArianeeEvent	0x86AD25F4143e608aE4341C0C09Dcf45A7b03C499

Figure 12 - Arianee's mainnet contracts' addresses

4 A Shared Ledger for Supply Chain Management

As we've seen, the investments in DLT for supply chain management are getting more common and that's because the risks for global supply chains are continuously fed by numerous factors, such as globalization and global connectivity, which are increasing the complexity of global supply chains and their costs. We learnt that an ideal supply chain, in order to be reliable, efficient, trusted and resilient, requires end-to-end visibility for monitoring assets, operational efficiency and a degree of flexibility to adapt to issues or critical demands. On top of that, the actors may not trust each other but they need to trust the data in the system, which is also supposed to be managed by privacy policies. Today's supply chain stakeholders are global and multi-layered. They form an intrinsic layer of suppliers, producers, end customers and logistics, between which data is transferred. This data can represent features of goods, traceability information, compliance of regulations and more, and it can be written and reviewed by humans, forcing the stakeholders to trust one another in their willingness to share true information. The ability of the collected data to effectively and accurately represent real-life objects or events is limited by the systems put on the supply chains to monitor them and track the supply. Accurately tracking an entire supply chain from raw material extraction to a factory-made product ready to be sold to a retail customer might quickly become discouraging.

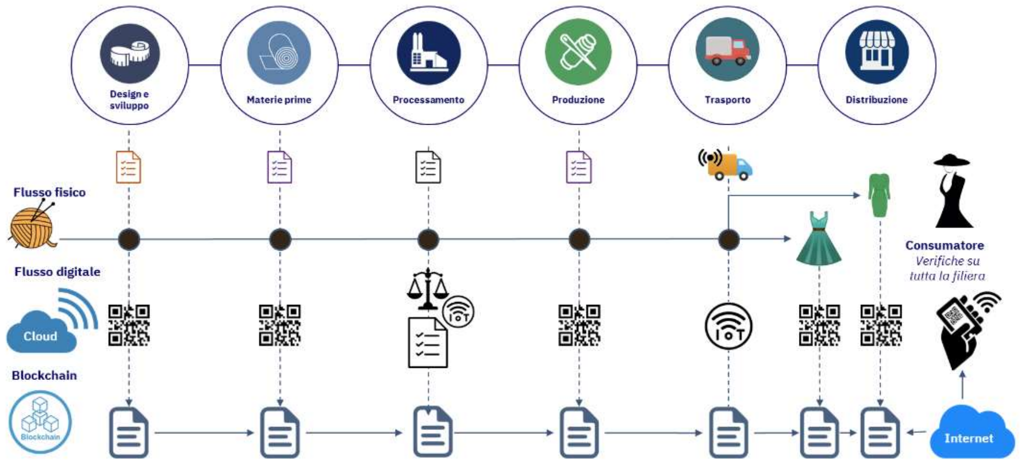


Figure 13 - High level representation of a DLT-powered supply chain (MISE and IBM)

Specifically, for the Fashion Industry, the main actors involved are:

- Brand owners (e.g. LVMH, Nike, etc). They design, create and sell high quality goods.
- Carriers. They transport goods or materials to destination.
- Brokers. They're the interface between brand owners and suppliers and work out the deals to buy raw or half-processed materials.
- Suppliers. They provide the raw or half-processed materials. It's common to have more layers of suppliers, but brokers work out the deals with the one at the highest level of the chain. It's a known fact that, in the lowest levels of the suppliers' chain, tracking is often impossible due to illegal agencies that employ underpaid workers.
- Producers. They receive and work the materials to create the final goods. The brand houses know in advance the expected output of a given quantity of raw materials, with a small percentage of errors, and the production line is expected to respect those numbers.

- Warehouse. They store the final luxury goods ready for selling. They communicate directly with carriers to set up the transportation to official retailers.
- Retailers. They sell the goods to the customers.
- Customers. They buy the goods. They can also sell their own goods to other customers as second-hand goods.

The information sharing between these different stakeholders is a major challenge. As it often stands, each has its own information system, and this affects the traceability of information along the chain [27]. A DLT can solve part of these difficulties but its power can be enhanced using IoT technologies. By its intrinsic properties, an IoT-enabled blockchain can address these visibility and traceability challenges, solving the trust issues and providing even more detailed information by autonomously collecting data from real-world objects and environmental inputs such as speed, acceleration, temperature and humidity. Using analytics on the gathered data through these intelligent devices gives systems the possibility to aggregate data, analyse trends and perform predictive monitoring. For instance, an RFID chip can be attached to a parcel to identify it. When it goes out of a warehouse, a sensor positioned at the gate detects the parcel and sends a signal over the network that the parcel has left the building. This kind of automation is useful to simplify management and data ingestion, but it can also be used for more specific situations. The following picture shows the use case for ice cream transportation, during which temperature has exceeded the limits allowed and previously defined in a smart contract.

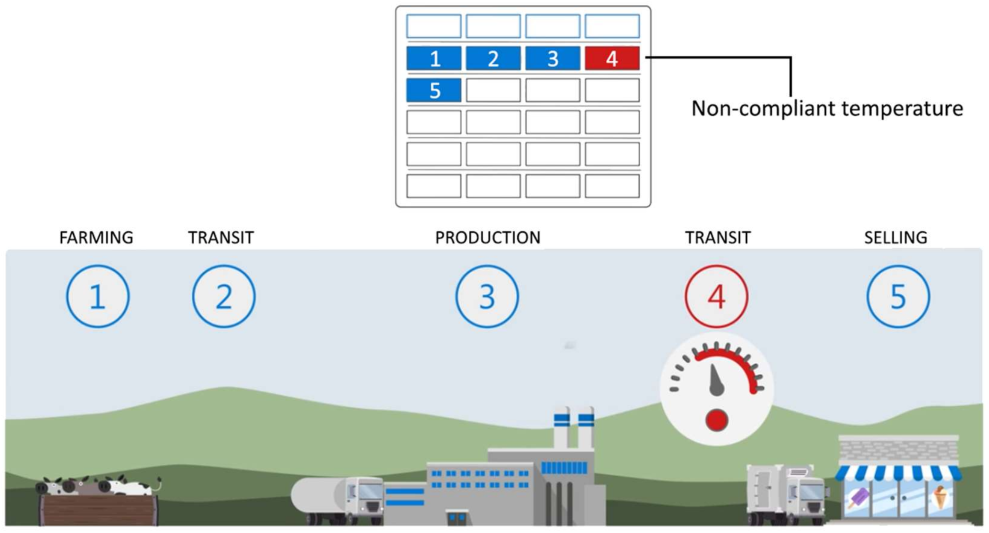


Figure 14 - DLT and IoT-powered supply chain use case (Microsoft Azure)

This brief overview and the facts learned from previous chapters, lead us to make some further considerations. The first point is about costs. The design of the architecture alone is a significant investment. The hardware itself is a high expense and skills in blockchain development are still uncommon, making developers and architects expensive to hire. Once deployed, the energy costs cannot be ignored even if one does not rely on the most power-hungry consensus protocols, such as PoW. One or more *Decentralized Application (DApps)* must be developed for the end users to interact easily with the blockchain. If, as it's likely to be, the architecture is hosted in a cloud environment, such as Azure, then one must consider the yearly licensing and services costs for an Azure subscription and the data ingest per quantity of byte. Also, the assembly lines must be adapted to integrate with the new ecosystem. This includes the cost of IoT devices and RFID management. The second, most delicate point is governance and how to bring competitors on board. The TradeLens experience taught us that one of the guiding principles should be that the ownership of the platform of the consortium must not be restricted to the founding parties. The governance model must ensure that additional partners will be able to join with equal rights and obligations. Also consider that it's not uncommon for brand owners to share suppliers and carriers' services, and obviously nobody wants to leak information about specific deals between parties.

The analysis of our case and the recent development in consortium models has convinced us that this might be the right track to follow. In general, a consortium is formed by a group of business entities of common interests and, in this specific case, common suppliers, carriers and raw materials as well. Every participant in the consortium has the incentive to run and maintain this blockchain as far as they benefit from it; say, for example, data about the trustworthiness of suppliers. From a user perspective, it's a bit different. The knowledge that a consortium blockchain is being used, rather than a private one, bears little to no importance unless people can get insurance claims faster or gain more transparency on the whole process. But comparatively speaking, users should feel higher trust on a consortium blockchain because it's not kept and maintained by one single party, but by a group of enterprises who have an incentive to maintain its operation. It is believed that a company can modify data for whatever reason, but a consortium is less likely to do so.

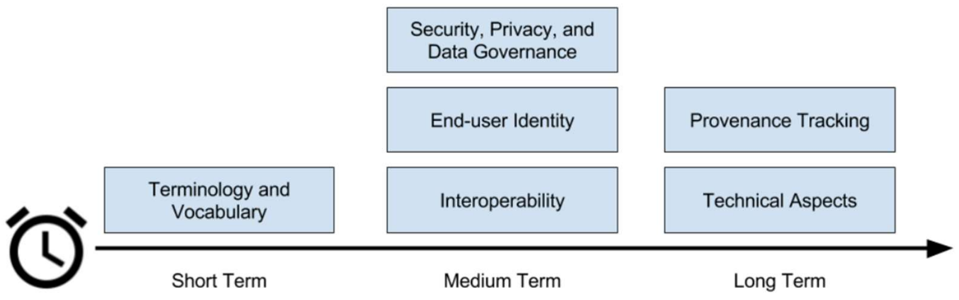


Figure 15 - The process of defining a DLT-powered system (edX)

4.1 AIDC Technology

Physical certificates have been introduced as a way to authenticate objects without the need for a detailed expert assessment, but they were relatively easy to manipulate. As the years passed, new modern equivalents have been introduced in the form of *Automatic Identification and Data Capture* (AIDC) capable devices, such as RFID and NFC, both contemporary tools intended to make a product more difficult to copy. On top of that, brands provide little or no information that enables a consumer to understand how to distinguish a genuine product from a copy, as each brand has its own way to create certificates, which it doesn't communicate publicly. For the customer, the only way to confirm the authenticity of a product often is to request an assessment from an expert who will compare the details, materials, and craftsmanship of the object to confirm or deny its authenticity. This process is expensive and is only as reliable as the brand's network of specialists and their skills. Therefore, AIDC and blockchain together can greatly help the end customer and, in some extents, the brand itself in reducing this phenomenon.

Along with NFC, the most common AIDC in the market by far is RFID, which stands for *Radio Frequency Identification*. It's a cheap technology based on air propagation of electromagnetic waves, consisting of an integrated circuit and an antenna that allow for automatic identification of relatively distant objects. They also feature a protective material that holds the pieces together and shields them from various environmental conditions, making them ideal for transportation. It can be seen as the natural evolution of a barcode, with less limitations:

- Objects with barcodes cannot be scanned in group: each barcode must be read separately. The operation is time consuming and it may require a human to do the job, which makes it undesirable for our purpose. This is not true for RFID tagged objects, that can be passed together in a pallet under a big RFID reader.
- There's a finite number of objects that can be identified with a barcode. This leads to identification of the kind of object rather than the object itself (think about the barcode in a supermarket, where the same barcode identifies all goods of the same type). RFID tags, on the other hand, can feature an extendible UUID (universally unique identifier) to track the single object, which in our case could be the single roll of fabric imported from country C in date D.
- A barcode can be easily falsified, and it must be readable from the outside. This is not optimal for DL security standard. RFID instead can be hidden inside the object itself, such as the sole of a shoe or the frame of a pair of glasses. This is perfect for our goals, since the object would need to be ripped apart in order to swap the tag inside and that is often enough to discourage counterfeits attempts.

A RFID tag defined as *active* lives with its own battery, a small, custom battery with long longevity. They can transmit data up to 100 meters, but the battery life will eventually end. The kind of RFID tag that best fit our requirements is a *passive* tag, due to its ability to operate through the energy emitted by the reader's antenna, ensuring longevity of the chip. The distance by which these tags can be read is limited by their frequency: low, high and ultra-high. In our case, a Ultra-High Frequencies (433 MHz - 2,4 GHz) RFID, often suggested for pallets and containers, can reach up to 10m distance of transmission, making it the best choice for us even from a cost perspective. The other alternatives, such as the mentioned NFC technology, or even security labels, are more expensive or impractical due to the expected high volume of goods and frequency by which controls are performed.

4.2 Data Store Technology

We've given for granted the use of DLT, but as explained in earlier chapters the choice could be different. As a matter of fact, it could be any data store technology such as a distributed database. To make an informed decision about the best possible technological data store for us, it's important to consider all the existing options. To do that, let's first recap some of the key requirements:

- We have more assets we want to track.
- We care about the evolution/lifecycle of the assets.
- The lifecycle of the assets is governed by well-defined rules and processes.
- We need to store states in time about assets.
- There are multiple writers on the data store. The problem of privacy and transaction details visibility needs to be addressed.
- Writers are known and they know each other. However, while they do trust each other to a certain extent, they're still competitors and their actions must be verified.
- There's no need for extremely high performance, millisecond transactions. We're not expecting a very high throughput of data.

- Identity matters.
- We cannot rely on a central operator to maintain the records, but we can expect third party agencies for random controls at the lowest levels of the chain.
- It should not be hard to be part of the consortium. We want it to grow, steadily.
- Information must be stored forever and cannot be lost or modified directly.

Thanks to similar supply chain management problems solved with it, we realized early in the process that a blockchain implementation was ideally suited to our needs as well. However, it's worth to dive deeper about some of the technological and functional elements that helped us to make this decision.

First is about **disintermediation**. Blockchain does not need to share its data with an intermediary for validation, as transactions are inherently validated and authorized by the network and the chosen proof-of. The consensus mechanism also ensures that nodes stay in sync, hence creating one and only one version of the truth. The second point is **robustness**, because DL architecture is also a highly fault-tolerant system. As each transaction has to be processed by every node, and since nodes are connected in a dense peer-to-peer network, a great number of communication links would have to fail in a short time period for the system to go offline, which is unlikely to happen. It's also easy for a node to catch up with the ledger on missed transaction thanks to the high redundancy. One other point to consider is **performance**. Distributed ledgers are slower than databases because transactions have to be signed and verified, nodes have to reach consensus and new blocks have to be transmitted over the network. This is a classic case of trade-off and, reading the requirements, our scenario heavily favours security and consistency over performance. Therefore, this redundant processing is well justified. Finally, the importance of **immutability**. We are designing this system to protect brands from counterfeits and to offer the end customer the absolute certainty of the information. A new record written in the ledger cannot be undone or modified. This core feature of blockchain allows for a provider of data to prove that data hasn't been altered, ever. At the end it comes down to the concept of trust, both in the data and in the infrastructure as well. A distributed database would make the development easier and it would even be more cost efficient; it would not, however, offer the level of trust we require.

Which type of DL best fit our scenario is another question to answer. The commonly known models are three: public, private and consortium blockchains. Considering our requirements, we're not going to use any trusted third party for validation. Privacy policies, as well as access rules, are a must. About trust, peers know each other but do not fully trust each other; at the same time, we want to identify who writes what on the ledger, as there will be more writers. This is enough to draw conclusions, as these considerations leave us with one obvious choice: the consortium model. Let's also consider that the consortium running the blockchain can easily, if desired, change the rules of the blockchain. As businesses grow and change, this is an important feature. From a security perspective, the validators are known, so any risk of a 51% attack does not apply. Transactions are cheaper, since they only need to be verified by a few nodes that can be trusted to have very high processing power, and do not need to be verified by the whole network. For the same reason, thanks to apt consensus protocol, finality is reached faster. Drawing conclusion, our choice is a consortium blockchain as it offers many of the advantages of a private blockchain, yet it still operates under the management of a group, the consortium, rather than a single entity, making it the best fit for our scenario.

4.2.1 Choosing a Distributed Ledger Platform

There's a good number of options available in terms of Blockchain platforms but only a few managed to distinguish themselves from the rest: Ethereum, Quorum, Hyperledger and Corda, all open source solutions. Despite sharing some core features, their vision about Blockchain vary significantly as well as their scope.

Ethereum is perhaps the most known framework thanks to its associated cryptocurrency, the Ether (ETH), and the number of DApps developed on it are a testament to its popularity. It's also been the first DL platform to introduce the concept of *Smart Contract*. Ethereum is a well-documented platform, supporting more programming languages for different purposes (Go and Java for node interaction via client, Solidity for smart contracts development) and it has been tested with numerous PoCs. However, being born in the early days of blockchain, the original code of Ethereum only allowed for a public, permissionless platform, hence not suitable for enterprise purposes. This is the main reason why the *Enterprise Ethereum Alliance* (EEA) was born: to customize the Ethereum blockchain network to apply to industries, developing open, blockchain specifications that drive harmonization and interoperability for businesses and consumers worldwide. The platforms all stand on three key pillars:

- Signature validation, so that all the users on the platform are only those that are legitimate.
- Freely available source code. This encourages experiments and diverse perspectives within the ecosystem.
- Integration with Ethereum network, meaning that any improvements that happen on Ethereum will take place in the platform. Furthermore, it will ensure that the architecture of the platform is not left behind given the fast-moving nature of the blockchain ecosystem.

One of the EEA project based on Ethereum is J.P. Morgan's **Quorum**, a private blockchain network that focuses on enterprise solutions. Notably, the platform has a clear bias toward the financial industry, hence the focus on high transaction speeds and security, but developers can create different kinds of applications. Quorum is not as commonly used as Ethereum but it's getting traction with time, and it doesn't require a built-in cryptocurrency because consensus is not reached via mining. A block's commit to the chain after verification relies on the Raft consensus method, with the leader that can be rotated with each transaction for added resiliency; the protocol itself is called *QuorumChain* and consensus is reached by simple majority voting. Both private and public states are required to keep consistency between public and private contract transactions, therefore enabling a correct verification process. Quorum provides a good start documentation and it integrates with development environments and SDK such as Truffle.

The main competitor of Quorum is **Corda**, another platform built with finance at its core, designed and developed by R3 consortium to record and automate legal agreements between identifiable parties. Corda only seeks to achieve consensus among parties of an agreement on the state of that specific agreement as it evolves, as opposed to seeking agreement on the state of a globally distributed ledger. This makes it vastly different from other DL platforms, with the advantage of greater scalability and less privacy concerns. In Corda, the finality of a transaction is ensured by a notary node that, by performing the function of the miners, validate the transactions.

Finally, **Hyperledger** is an umbrella for open source DL platforms, established under the Linux Foundation. As of today, it includes over 10 platforms and tools such as:

- Hyperledger Burrow, for deploying and executing smart contracts in a permissioned environment.

- Hyperledger Iroha, which emphasizes mobile application development with libraries for Android and iOS.
- Hyperledger Indy, for identity management.
- Hyperledger Sawtooth, ideal for building scalable, broad networks of hundreds or thousands of nodes thanks to the PoET (Proof-of Elapsed Time) consensus protocol, which provides a degree of scalability that's unmatched in other blockchain platforms.

However, for our use case the most interesting are Hyperledger Fabric and Hyperledger Besu, a very recent entry in the Hyperledger umbrella (announced in August 29th, 2019 [30]). Hyperledger Fabric is meant to be a foundation for developing blockchain applications with a modular architecture. Of all the Hyperledger blockchain projects, Fabric is the most mature and has the most active community of developers. Smart contracts are deployed through chaincode in Golang, and privacy in Fabric's distributed ledger is protected by private channels. Hyperledger Besu is kind of unique, being the first open source Ethereum client hosted in Hyperledger. Previously known as Pantheon, Besu includes several consensus algorithms such as PoW, PoA, and IBFT, and has comprehensive permissioning schemes designed specifically for uses in a consortium environment.

As Corda and Quorum are both consciously designed as DL for the Banking and Financial Industry, the focus is on financial services transactions. The good thing is that the architectural designs are simple when compared to Fabric or Besu; unfortunately, though, Corda relies on a notary node for the addition of new blocks, which makes it a less suitable choice for us, while Quorum feels like it targets a more expert audience. Admittedly, Hyperledger Fabric is probably the best choice left, thanks to its maturity, a richer documentation and its overall modularity that would allow us to design what we aim for (and a lot more). The language and Fabric's concepts are fairly easy to grasp, and Hyperledger Composer makes it for a very nice environment for beginners. Unfortunately, with the release of Hyperledger Fabric v1.4 and the upcoming v.2.0, Hyperledger Composer is officially unsupported since Summer 2019.

Our choice, though, is Hyperledger Besu. It's the most recent entry in the Hyperledger umbrella and the documentation is still far from being comprehensive, but its collection of consensus algorithms and the potential interoperability that comes from the fruition of the Ethereum network with Hyperledger makes it far too interesting for us to ignore. Also, it was designed from the ground up to be a permissioned environment and It shares a lot with a classic Ethereum fork, meaning that we can take a lot of inputs from what's perhaps the most active community of DL developers in the world, especially regarding smart contracts and the Solidity language.

4.2.1.1 Hyperledger Besu Architecture

Hyperledger Besu is a Java-based open-source **Ethereum client** created under the Apache 2.0 license. It can be run both on the Ethereum public network or on private permissioned networks and test networks such as Rinkeby, Gorli, and Ropsten. Being an Ethereum client, it can be used as a node that verifies a blockchain and its smart contracts and everything else related to a blockchain. In other words, it is a software that implements the Ethereum protocol. An Ethereum client contains:

- Storage for persisting data related to transaction execution.
- APIs for application developers for interacting with the blockchain.
- An execution environment used for processing transactions in the Ethereum blockchain.
- Peer-to-peer networking to communicate with other Ethereum nodes to synchronize state.

Besu supports Ethereum functionalities such as smart contracts and DApps development, deployment, and operational use cases. The tools that enable these activities are Remix, Truffle, and web3j; we'll use some of them for our development as well. The Ethereum client implements standard **JSON-RPC APIs** that simplify integration with ecosystem tooling. Besu includes a command-line interface and HTTP and Web socket-based APIs for running, monitoring, and maintaining nodes on an Ethereum blockchain. Besu does not (yet) support key management, so one must manage private keys via third-party tools. It supports account permissioning and local configuration-based nodes. Private transactions are also made available in the client through zero-knowledge methods.

Hyperledger Besu is an *order-execute architecture*, in which the consensus protocol first validates and orders transactions, then propagates them to all peer nodes to execute the transactions sequentially. This means that all nodes in the network will execute all transactions, as taxing as that sounds. This is a profoundly different approach to other solutions in the Hyperledger family, such as Fabric, that has a *execute-order-validate architecture* [26] where transactions are executed first, in any order, by a subset of nodes, and then they're sent to the ordering service for actual ordering. The transactions in this order are then executed by all peers (and validated at the same time).

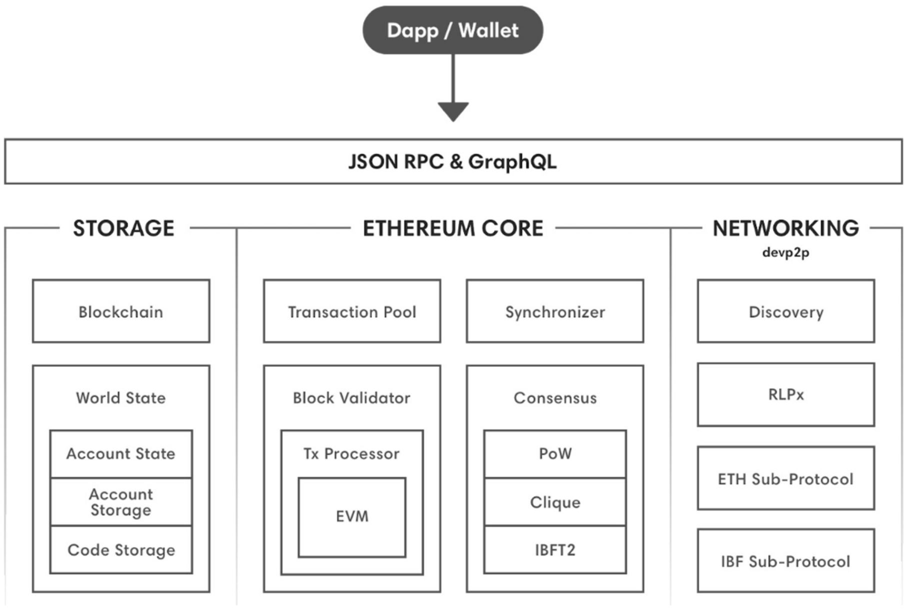


Figure 16 - Hyperledger Besu architecture (official docs)

The consensus protocols available are both PoW (Ethash) and PoA (Clique, IBFT 2.0 and Quorum IBFT 1.0). In terms of privacy, Besu offers flexible privacy groups and policies. This ability to keep transactions private between the involved participants is managed by a *Private Transaction Manager* such as Orion. Orion generates and maintains private/public keypairs, stores privacy group details, self manages and discovers all nodes in the network and provides API for communications. Each Besu node that sends or receives private transactions requires an associated Orion node. A private transaction is first passed from the Besu node to its associated Orion node, which encrypts and distributes the transaction to all and only the Orion nodes participating in the transaction. These nodes must be online at the moment of submission, otherwise the transaction will fail, and the process must be repeated. Signature is performed by standard public/private keypairs. Note that the mapping of Besu node addresses to Orion node public keys is off-chain, meaning that the public keys must be known beforehand from the sender Orion nodes and communicated by other means between participants. This ensures compliancy with EEA standard for privacy, but Besu also supports privacy groups for extended privacy capabilities. They're identified in Orion by a group ID and work in transactions with the attribute `privacyGroupId` as the recipient. It's also very important to remember that a separation between *public and private states* exists. That is, Besu nodes maintain the public world state for the blockchain and a private state for each privacy group. The private states contain data that is not shared in the globally replicated world state. What this means, practically, is that a public contract cannot access a private contract, while a contract deployed via a privacy group:

- Can read and write to a contract in the same privacy group.
- Can read from the public state, including public contracts.
- Cannot access contracts from a different privacy group.

Permissioning is also possible, allowing only specified nodes and accounts to participate. Permissioning can be configured *locally*, to restrict the use of a specific node owned by participants, or *onchain*, where rules apply to the entire network. The latter is definitely the best choice for production environment: onchain permissioning uses smart contracts to store and maintain the node, account and admin whitelists, so that every node can read from a single, updated source. The smart contracts for permissioning are provided by Besu's developers themselves and can be deployed as ingress contracts with the genesis file, defining the desired address.

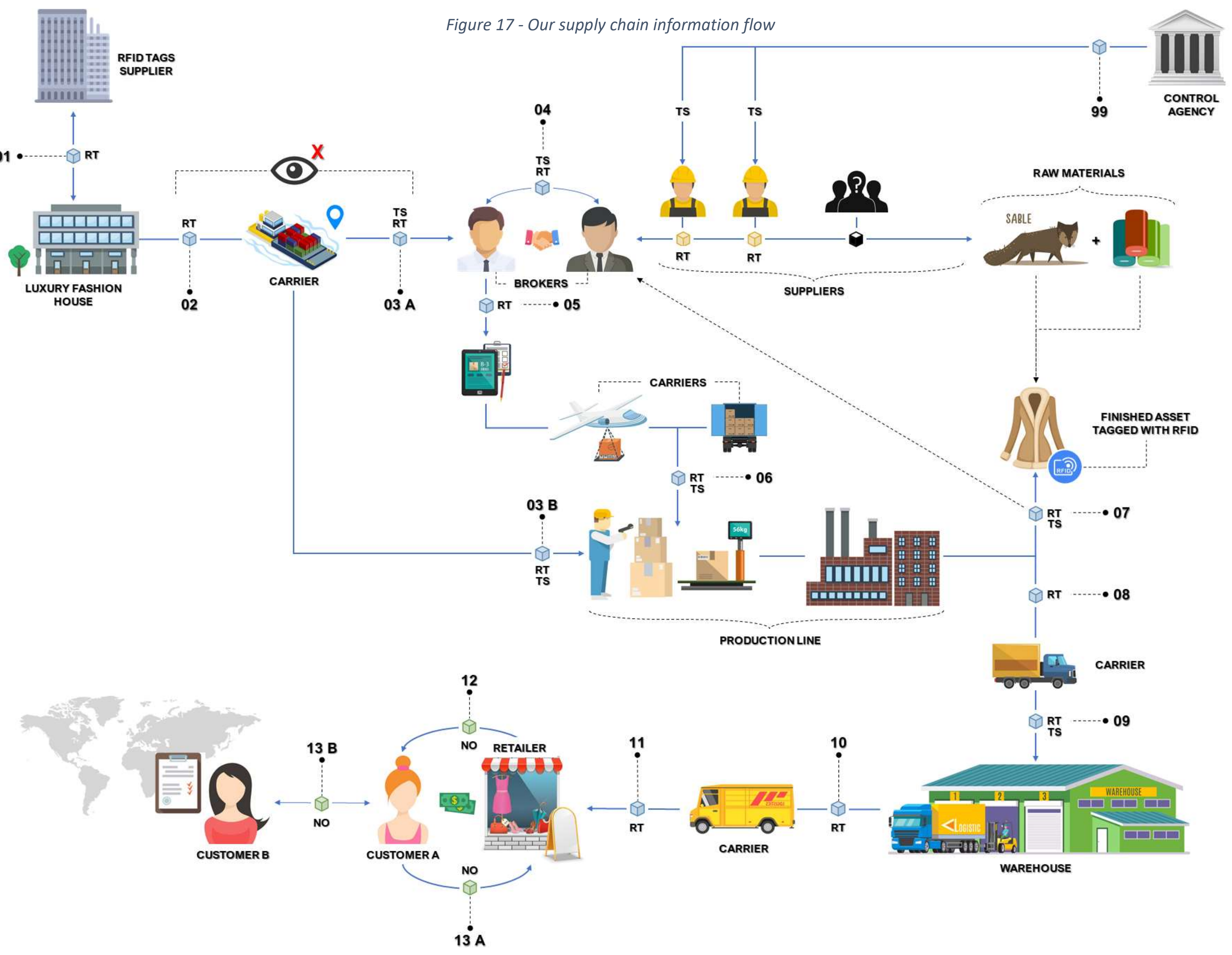
4.3 Information Flow and Architectural Components

Before getting into the details of a possible implementation with Hyperledger, let's picture what the information flow and overall architecture might look like, listing players, transactions, assets involved and defining when the blockchain and the Trust Score are updated. As an example, let's consider a generic luxury house producing a coat with sable fur, a very precious and peculiar material from Russia; such clothing item can cost up to $10000, making it a perfect case where the provenance and authenticity of the materials are very important for the buying customer.

LEGEND

- 🟦 a generic blockchain transaction TX
- ■ an operation or deal that's potentially untraceable
- 🟩 a blockchain transaction TX that change the ownership of a defined ASSET from one OWNER to another
- 🟨 optional blockchain transactions TX (e.g. 3rd tier SUPPLIER deals, there might be none)
- RT: a transaction TX that involves moving or verifying a RFID TAG
- TS: a transaction TX that has consequences on the TRUST SCORE of one or more PARTICIPANT
- NO: a transaction TX that changes the OWNER of the main ASSET to an end customer

Figure 17 - Our supply chain information flow



Let's describe every step in more details.

(T01) The cycle starts with a Luxury Fashion House registered as a PARTICIPANT (PAR) A ordering a set of RFID TAGS (RFT) from its supplier. This supplier may or may not be registered on the Distributed Ledger as a PAR: if it is, the set of RFT is exchanged like any other ASSET (AST), otherwise the Fashion House will register the set by itself. We can trust PAR A in this because there would be no advantage for them in declaring an RFT that does not exist or vice versa.

(T02, T03) Once registered, the RFT are delivered by a known Carrier (PAR C) to both Brokers[2] (PAR B) and Production Lines (PAR P). The transportation phases are perhaps the most delicate of this journey: although carriers can use GPS technology to keep PAR A updated on position and ETA (estimated time of arrival), there's still a lot of things that could go wrong and that may damage the shipping. For this reason, there are always two transactions to confirm the delivery, one on each end:

- T02 represents PAR A giving PAR C a set of RFT for delivery, let's say for example a set of RFT with codes going from CB001 to CB500 to PAR B and another set of RFT with codes going from CP700 to CP900 to PAR P.
- T03A and T03B represent the receivers confirming that they got exactly the set of RFT that PAR A claimed to have sent. This can be done easily by scanning the pallet of RFT in the cargo with technology that's already on the market, known as *RFID Warehouse Tracking*.

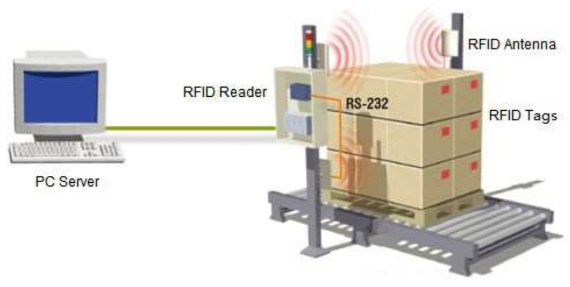


Figure 18 - RFID warehouse tracking technology (leadercolor.com)

In our case, the PC would run the application interfaced with the DL to check the registry. Note that the RFT that are not registered in T01 will not be used, as they're officially non-existent, and the RFT that were registered by PAR A but were not found by the receivers are invalidated, to avoid future use of them for forgeries.

This process of confirmation is repeated every time that RFT are moved from one place to another, even if carriers are owned by PAR A itself. While this is repetitive and expensive for PAR A to implement, it's a necessary evil for this system to work properly and to reduce its flaws. Also, the pool of checks can be extended with the adoption of IoT sensors, as we'll see in the next steps where that kind of metadata can come in handy. Finally, upon confirmation of a good delivery, the TRUST SCORE of PAR C can be updated.

(T04) At this point, it's time to make a deal to get the raw materials from Suppliers (PAR S). These deals involve more layers (more PAR) but for this example they have been abstracted for simplicity:

[2] We identify as "brokers" the whole group of people engaged in the activity of buying

hence let's say that a broker working for PAR A buys from a broker working for PAR S a quantity Q1 of AST A (sable fur) and a quantity Q2 of AST B (dye).

More realistically, PAR B sends PAR S a subset of RFT so that they can tag the raw materials closer to the source. This can be done up to only some layers of the supply chain, though, since the Textile Industry is known to make use of workers with a very limited amount of resources and often under no regulation whatsoever [2]. After that point, the latest traceable participant is held responsible for everything that's under his governance. The process of assignment of RFT from raw materials to more refined materials is no different than the one that will be explained at T07, at the production line.

Note that the quantity of finished items that can be produced by Q1 of AST A and Q2 of AST B is known with a margin of error by PAR A. This information can be registered on the DL, for further verification on the production line's performance and to avoid huge spread of authentic fakes without RFT; that is, real Luxury items sold in the black market for lower prices.

(T05) The material is then prepared in pallets for distribution, following the production lines' needs. To each pallet is assigned an RFT, ideally in a way that any tampering attempt would break the chip. This ensures that the box stays closed until it arrives to the production line. On top of that, more information can be linked to the pallet: weight, for example, but also data from IoT sensors to register information about temperature, humidity or any element that might damage the raw materials or might suggest a tampering attempt.

Finally, the expedition is prepared to the production lines using a known carrier. To continue with our example, let's say that PAR B send to PAR P an amount of Q1 = 20 of AST A and Q2 = 20 of AST B, with codes going from CB101 to CB120 and CB301 to CB320 respectively.

(T06) Upon arrival, the cargo is checked for integrity and to make sure that the RFT sent by PAR B arrived as intended. The same considerations of T03 apply here, but this time workers can also check all the metadata and IoT data available to make better decisions and update PAR C TRUST SCORE accordingly.

(T07) At this point the final product AST F is made. The production line is composed of more layers, but the simplified process stands as follow: a pallet of AST A and AST B is prepared for work and their UUID are stored for later reference (for example, CB101 and CB301). When the item (the fur coat in this example) is completed, a new RFT is assigned to it, for example CP701; this will be the UUID of reference for AST F throughout his lifecycle. The UUID of AST A and AST B are stored as *history of provenance* for this new asset.

Hypothetically, the TRUST SCORE for the broker who provided the raw materials could now be updated by another PAR, given the right prerequisites. One could argue that his TRUST SCORE should be updated sooner, as he should not be held responsible for the steps T05 to T07. However, raw materials can deteriorate easily if the quality is not as good as claimed, for example due to humidity.

(T08) to **(T11)** are just another iteration of the procedures described previously.

(T12, T13) The sable fur coat is finally sold to a customer, who will be able to read the history of this item and prove its authenticity thanks to the UUID in the RFT. This is a special kind of transaction because for the first time the item changes ownership from one business PARTICIPANT to a paying customer. This transaction will be identified in the DL by another UUID, for example a combination of the item's UUID, the store's UUID, the timestamp of the purchase and possibly even some information about the buyer, although due to privacy reason this could be hard to get. In other word, we would like to create a *digital, timestamped and verified certificate* of ownership of the asset, by a combination of data that's easily verifiable by everyone, anytime and anywhere. This would allow to track the ownership of the item from this point forward. However, it's important to note that human intervention is always required for this to work as intended: to track the ownership of the item even in the second-hand market, transactions still must be made. This includes physical stores as well as online stores. Like a proof-of ownership certificate of high valuable jewellery, there's no way to guarantee the ownership if the deal is not recorded in the system.

(T99) Parallel to this system, there's another known, 3rd-party trusted PARTICIPANT that acts as control agency. Its goal is to periodically and randomly select a supplier to perform quality checks and to confirm the truthfulness of the declarations about the raw materials. False claims will turn into sanctions and more. This control agency will then update the TRUST SCORE accordingly.

When it comes to the separation between public and private data, we need to consider that our desired end goal, which is also the reason why we picked Besu over Fabric and took the path of a consortium ledger, is to share non-sensitive data in an effort for collaboration, transparency, sustainability and an overall growth of the market. Clearly, then, RFID tags information will be public, because we need to be able to associate any RFID UUID to its current owner, the item and the producer, anytime. We also want the Trust Score to be public; if desired, participants can keep a private Trust Score for each relationship as well, for business related metrics. Metadata about items should be public, while the information related to the exchange of raw materials must be hidden from competitors, therefore they will be kept in the private state of each brands. Finally, some side details about the remaining components of the architecture. As far as front-end is concerned, we've a good number of options and none of them will make any difference in terms of functionalities. It really is a choice of preference and targeted platforms (mobile, browser, etc), so Angular or React will work the same. More important is the backend: Besu supports Java as well as Node.js, with the latter being the preferred choice according to their papers. In the demo, we'll use Node.js as it seems to be, indeed, simpler and better documented. The database, as we'll see, can be RocksDB by default or a more known option such as Oracle. Each and every node will also have a second node, the Orion node, that will manage the privacy transactions and, on top of that, they will need a third-party software to manage their own wallet, such as Metamask. The architecture will include both a client and a server component, with the former being used to view products' history and the latter to process operations as expected by the blockchain platform. A realistic scenario might be the following:

- End user connects to a cloud service provider via a mobile or web app. End users can be real operators (e.g. operators in a warehouse) or customers. They will have different grants to the ledger.
- Through edge services, the request reaches a security gateway, where credentials are checked to grant the correct permissions.
- The request is then forwarded to the Ethereum client to process.
- Integration Services can be used to integrate data with external systems as well. Hypothetically, a cloud environment such as Azure that already supports Ethereum clients, can manage some of these interactions via Azure Active directory, Azure B2C, with Azure Data Factory and Azure Data Bricks for data reporting.

4.4 Pain Points and Benefits

The proposed flow of information presents weak links that can lead to counterfeits. Thanks to the DL-powered architecture discussed so far, some of these links get solved automatically, while some still require human intervention. Let’s review some plausible scenarios.

A RFT gets lost or isn’t found to be associated with any PAR. A lost RFT can represent a risk in the early steps of the chain, as an ill-intentioned participant could use it to tag some low-quality, unwanted raw materials. This is why step T03A is important, as it’s expected that the broker will move forward on the chain all and only the verified RFT received by PAR. Anything else that’s not verified, from now on, is considered invalid or expired and cannot be used to tag materials or items. A similar process should be put in place for RFTs that seem not to be associated with any PAR.

PAR A claims false ownership of RFT. This is easy to verify at any point in the chain, as the ownership of a RFT is the main information exchanged on the ledger with every transaction. It’s just not possible to claim ownership of another PAR’s RFT, as every RFT lifecycle is registered from its very creation down to the final customer.

The details about an exclusive offer between PAR S and PAR A get leaked to another PAR. This is a legitimate privacy concern, considering that all PARs share the same ledger. However, we’ll prevent this from happening using privacy groups and the Orion nodes to create private channels between the interested parties alone.

The number of expected products to be made with the amount of raw materials differs significantly from the expected production output. This is a very interesting case. Brand owners know in advance, with a small margin of errors, how many goods can be produced given a quantity X of raw materials. Lower volume of production can happen for various technical reasons, but we’ll assume it’s due to counterfeits (the so called real-fake products). This scenario should be solved thanks to the continuous check of RFID tags, in this case from T09 onward. As this specific real-fake good never reached the warehouse, or the retailer, its UUID will be invalidated automatically by a smart contract after a predefined period of time. The amount of goods made out of a given set of RFID tags, each one associated with a known quantity of raw materials, can be then monitored by querying the ledger periodically in a data analysis process.

IoT metadata suggests a problem with the shipping. When this happens, a smart contract can automatically perform a predefined action.

Broker has a trust score that’s too low for PAR A standards. If PAR B finds out that its usual supplier has received enough sanctions for its Trust Score to be lower than the standard, his attempts to buy the raw materials will fail thanks to the restrictions imposed in a smart contract. That is, the transaction will be attempted but it will not pass the validation process.

Raw materials are not what the broker/suppliers claimed to be. This point refers to T07. For argument’s sake, let’s suppose that workers in the production line realize that the raw materials are not what they’re supposed to be. Say, for example, that the dye consistency is off and can’t be applied correctly. Just as PAR C gets penalized for disservice in transit, suppliers should be held accountable for this kind of shortcomings, more so if they claimed a different quality. This can lead to the broker’s Trust Score receiving a hit.

There are still many cases, however, where a DL-powered system can’t help us, unless we take targeted actions. For example, if a transaction in the second-hand market is not registered via DApps, the ownership of the asset will not be updated. A possible solution could be to grant a retailer the powers to update the ownership of an asset that’s already been sold, given the appropriate documentation and proof of purchase. Even better, if the basic contact information of the previous owner were given at the time of purchase, then a retailer could contact the owner for further assessment.

5 Modelling with Hyperledger Besu

In this chapter we're going to detail the choices we made for our test deployment in Hyperledger Besu, how we managed its features, such as gas, and how we programmed the business logic using Solidity and the smart contracts.

For this purpose, we set up a Virtual Machine with Ubuntu 18.04.3 LTS. After installing the basic necessities and prerequisites, we downloaded the binary of Hyperledger Besu and we set up the environment variables for the shell command by editing the `.bashrc` file in the `home` directory, so that we could use the shell script `besu.sh` and others from anywhere. Note that the binary has been updated often, from version 1.2.3 (it was still called Pantheon back then) to version 1.3.6, and no compatibility problems were encountered. We also used some tools to make the development experience smoother. Here's a quick overview of what it's needed or desired to develop the demos:

- Curl is used for all the JSON RPC calls we'll make to interact with our nodes. If one prefers a GUI, there are tools such as Postman. We will also need the web3js library to interact with the Ethereum blockchain and the smart contracts.
- Docker, the de-facto standard for containerization, and Portainer, a very useful docker image that can be used to manage the same Docker engine it is running on or remote ones.
- Remix Ethereum or Ethereum Studio, both online development environments for Smart Contract development and testing. Remix supports Solidity as well as Vyper. We also used Visual Studio Code with the Solidity language extension and the Solidity compiler.
- Metamask is a lightweight, browser extension to manage Ethereum wallets. It connects to all the main Ethereum networks and can also be used to connect to a custom RPC endpoint.
- Alethio offers a whole analytics platform for Ethereum networks, but we'll only use the Block Explorer application, which offers a minimal interface to check our transactions, and Ethstats[3], a browser-based network monitoring application.
- Another known and very appreciated tool in the Etheruem community is the Truffle Suite, that offers an easy-to-use environment for smart contracts development, deploy and network management.

What follows is a step-by-step process that will lead us to deploy some demos for presentation and testing purposes. Starting from a basic public network where we'll move Ether from one account to the other, we'll build privacy-enabled networks, making choices about settings and functionalities along the way based on requirements, performance and ease of use. The Solidity code and nodes configurations are hosted in GitLab.

5.1 Assumptions

Before we dive into the development, there are some assumptions we need to make that will help us to abstract the problem to a more easily solvable scenario. The first assumption we'll make is about AIDC: we'll assume that their identifiers (UUID) are exactly that, *universally* unique identifiers. Let's assume that we'll have a 256-bit code in this pattern, stored in hexadecimal format:

```
pattern: AAAABBBBB-CCCCDDDDD-YYYYMMDDHHMM
example: RFTDA0002-LVMHA0012-201911201245
base16 : 5246544441303030322d4c564d4841303031322d323031393131323031323435
```

[3] The Ethereum mainnet has its own Ethstats at https://ethstats.io/

It reads as follows:

- AAAABBBBB is the code of the AIDC supplier.
- CCCCDDDDD is the code of the brand owner who bought that set of UUID.
- YYYYMMDDHHMM is the timestamp (year, month, day, hour and minute) at which the UUID was generated.

This string pattern should guarantee the uniqueness across AIDC suppliers and brand owners, even if the same AIDC supplier deals with more than a single brand, which is a certainty.

The second important assumption we'll make is about the suppliers' chain. As mentioned earlier, there are suppliers of different tiers and what the brands' brokers buy is often a pallet of half-processed materials, rather than raw, unprocessed materials. Realistically, a supplier would apply an AIDC to the lowest possible tier of the chain (the lower tiers are hard to track due to unregulated labour) and track its journey from there, back to the highest level where the brand's broker makes the exchange. This means that half-processed materials would already have a long history in the blockchain when they arrive to the production line, composed of one or more UUIDs. However, we'll assume that there's at most one, traceable level of depth in the suppliers' chain.

Third and last assumption is about customer's privacy. It might be necessary to store some basic customer information in the transaction, such as the identification card number and a contact. This is solely to demonstrate the potential use of the system when it comes to targeted marketing.

5.2 A Basic Network

Let's start with the basics. The first thing to do is to prepare the directory structure for our peers. Each node in this network has a dedicated main directory, storing keys and few others information that we will generate, and a data subdirectory that will hold the blockchain data, both public and private states. As anticipated, Hyperledger Besu outsources keys management to third-party tools such as EthSigner or Metamask. This is done by generating the key via a specific command:

```
besu \
--data-path=<NODE-DATA-DIR> \
public-key export --to=<NODE-DATA-DIR>/<NAME-OF-KEY-FILE>
```

Details about this and other Besu commands are provided via the `help` argument as usual, including the formats of the expected data in the argument.

As we are not connecting to an existent ledger, but to a custom one, we also need to define the so-called blockchain *genesis file*, which is responsible for generating the first block and for laying out the configuration details for each successive block created, such as the consensus protocol, the difficulty, the contract size limit and a lot more. It also allows for a node to identify the other nodes in that same network. The documentation provides a pre-compiled skeleton for this genesis file, that only needs to be customized for our needs. So, if we generate a 4-nodes network, we copy-paste the default genesis file with Clique as the consensus protocol, and we run the above command for Node-00, ultimately this will be the directory structure that we should be looking at:

```
├── D00-private-network
│   ├── genesis.json
│   ├── Node-00
│   │   └── data
│   │       ├── key
│   │       └── N00-Address
│   │
│     ├── Node-01
│     │   └── data
│     ├── Node-02
│     │   └── data
│     └── Node-03
│         └── data
```

Node discovery in the network requires one or mode nodes to be labelled as *bootnode*. One of these nodes will also be the initial signer for the genesis block. We'll use only Node-00 as bootnode, hence it will also be the initial signer. We define this by appending the node address to the genesis file, in the `extradata` field. The genesis block can now be created by starting Node-00, with this command:

```
besu \
--data-path=<NODE-DATA-DIR> \
--genesis-file=<PATH-TO-GENESIS-FILE> \
--network-id <NETWORK-IDENTIFIER> \
--rpc-http-enabled \
--rpc-http-api=ETH,NET,CLIQUE \
--host-whitelist="*" \
--rpc-http-cors-origins="all"
```

With `rpc-http-enabled` we're enabling the *JSON-RPC API* because that's how we're going to interact with our nodes. Once started, the command line log will provide basic information about the node's status, even though there are no transactions to be stored yet. Among all the information, the most important at this point are the JSON RPC HTTP Service Endpoint (127.0.0.1:8545) and the *enode URL*, which is the hexadecimal node ID used in Ethereum for node identification and peer discovery. It's stored as `enode://<id>@<host:port>`, where

- `<id>` is the node public key excluding the initial 0x.
- `<host:port>` is the host and port the bootnode is listening on for P2P peer discovery. Specified by the `--p2p-host` and `--p2p-port` options.

In our case, the enode for Node-00 is

```
enode://6d3ecbcded1ccb97c4e9b569929a7df92fe3dec4b8296f93f5db1b82c4e9088913d903ac5c13
4587fd47adfa32803eac108c242d63f2a97490e6a59cfeb93a76@127.0.0.1:30303
```

and its address in this custom network, for future reference, is

```
0x3f5aa82175339f070e6f04a659b2be1f7b509911
```

The enode is always displayed when starting a Besu node but it can also be obtained using the `net_enode` JSON-RPC API method, like the following (note that we're using Node-00's RPC endpoint):

```
curl -X POST \
-d '{"jsonrpc":"2.0","method":"net_enode","params":[],"id":1}' http://127.0.0.1:8545
```

We can now start the remaining nodes, that will act as regular nodes. We will start them so that they will connect to the same network as the bootnode and will be looking for peers to sync with. To do so, this time we will have to specify the bootnode by adding the `bootnode` option, where we'll put Node-00's address, and we'll also set the same network id as before. We will also give these nodes another RPC HTTP port to expose and another p2p port to listen to, in order to prevent conflicts with Node-00. For example:

```
besu \
--data-path==<NODE-DATA-DIR> \
--genesis-file=<PATH-TO-GENESIS-FILE> \
--bootnodes=<Node-00 enode URL> \
--network-id <NETWORK-IDENTIFIER> \
--p2p-port=30304 \
--rpc-http-enabled \
--rpc-http-api=ETH,NET,CLIQUE \
--host-whitelist="*" \
--rpc-http-cors-origins="all" \
--rpc-http-port=8546
```

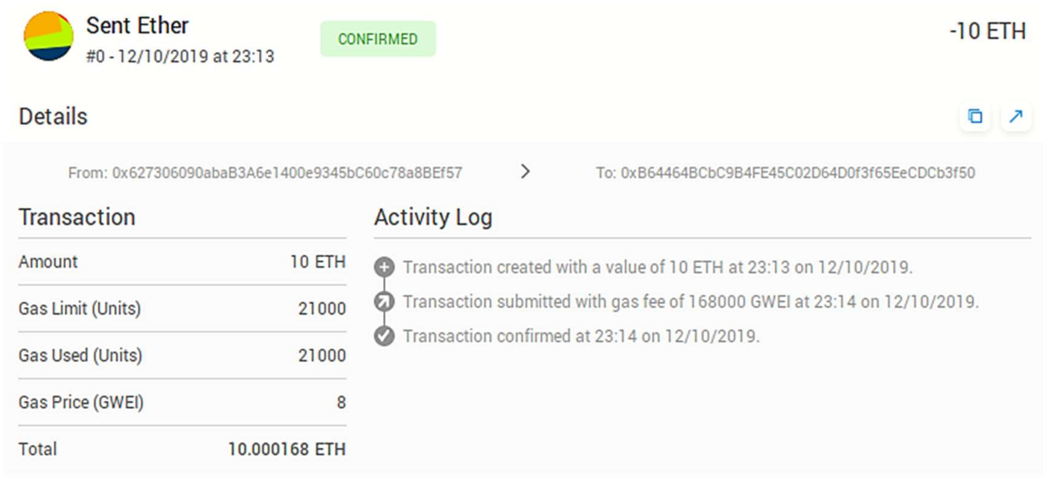
Once started, we may notice that Node-00's log stop reporting the following message:

```
FullSyncTargetManager | No sync target, wait for peers.
```

That's because a peer was found, which is the Node-01 that started syncing with Node-00 as soon as it entered the network:

```
SyncTargetManager | Found common ancestor with peer 0x6d3ecbcded1ccb9... at block 0
```

Once all four nodes are online, we can make dummy transactions using *Metamask*. We set up a connection to our network by passing our JSON-RPC endpoint to Metamask. We then import our Node-00's key to manage its wallet (which of course is empty at this point), as well as a couple of fake accounts commonly used for test purposes and pre-allocated in our genesis file with a very high amount of Ether[4]. Finally, when we try to send 10 Ether from one account to another, we're faced with a choice about *Gas*. This concept is key in the Ethereum networks and will be explained later in this chapter; as of now, let's just say that it's a fee for a transaction.



Sent Ether
#0 - 12/10/2019 at 23:13
CONFIRMED
-10 ETH

Details

From: 0x627306090abaB3A6e1400e9345bC60c78a8BEf57 > To: 0xB64464BCbC9B4FE45C02D64D0f3f65EeCDCb3f50

Transaction

Amount	10 ETH
Gas Limit (Units)	21000
Gas Used (Units)	21000
Gas Price (GWEI)	8
Total	10.000168 ETH

Activity Log

- Transaction created with a value of 10 ETH at 23:13 on 12/10/2019.
- Transaction submitted with gas fee of 168000 GWEI at 23:14 on 12/10/2019.
- Transaction confirmed at 23:14 on 12/10/2019.

Figure 19 - Transaction in Besu via Metamask

After few seconds, the transaction gets mined and added to a block. This is confirmed by the recipient's wallet but also by Node-00's log, with the following message:

```
BlockMiner | Produced and imported block #320 / 1 tx / 0 om / 21,000 (0.2%) gas /
(0x781ab3ad7ba93c8af445361a0460c46edcbae88bfb6795b61c7924dbc4f613f5) in 0.024s
```

While the other three nodes, being peers, import the proposed block:

```
BlockPropagationManager | Imported #320 / 1 tx / 0 om / 21,000 (0.2%) gas /
(0x781ab3ad7ba93c8af445361a0460c46edcbae88bfb6795b61c7924dbc4f613f5) in 0.039s
```

The JSON-RPC API is rich of options and Besu, along its exclusive functions, includes common Ethereum's RPC. We can, for example, propose a new signer, remove a signer or get a wallet's

[4] Note that this amount is available only in our localhost network. If we change network, we'll notice that these same accounts have a different balance. For the mainnet of Ethereum, that balance is zero.

balance. As an example, let's check the address of the node who signed the transaction we just made, issuing the following request to the usual endpoint:

```
curl -X POST --data '{
 "jsonrpc":"2.0",
 "method":"clique_getSignersAtHash",
 "params":["0x781ab3ad7ba93c8af445361a0460c46edcbae88bfb6795b61c7924dbc4f613f5"],
 "id":1}'
http://127.0.0.1:8545
```

To be fair, we know we only have one signer, which is Node-00, so the response is obvious:

```
{
  "jsonrpc" : "2.0",
  "id" : 1,
  "result" : [ "0x3f5aa82175339f070e6f04a659b2be1f7b509911" ]
}
```

Before we wrap this up, let's also try the deployment of a smart contract called `SimpleStorage`. Using *Remix Ethereum IDE*, there are two ways we can do this: via bytecode and JavaScript, or directly from the IDE. If we take the first route, using the embedded JavaScript VM environment, we compile the contract and we get the *ABI (Abstract Binary Interface)* directly from the IDE, which also provides the skeleton of the web3 calls necessary for the deploy.

```
contract SimpleStorage {
  uint public storedData;
  event StorageChanged(uint x);

  function set(uint256 x) public {
    storedData = x;
    emit StorageChanged(storedData);
  }
}
```

When we deploy this contract using `deploy.js`, we get the contract's address. We can use this address and the ABI as inputs for our `getStorage.js` and `setStorage.js` scripts. Here the values are placeholders and running these scripts simply changes the value from 0 to 5. The transaction receipt we get when running `setStorage.js` should show, among other things, the data field with value 5.

```
data: '0x0000000000000000000000000000000000000000000000000000000000000005'
```

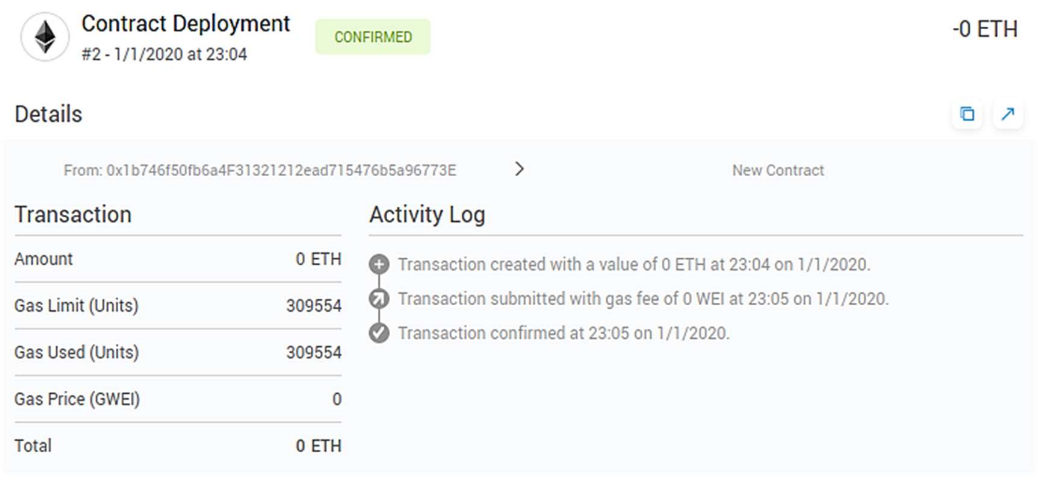
Alternatively, we can connect our node to the Remix Ethereum IDE[5]; this way, we'll be able to combine Remix and Metamask very easily. For example, we can deploy this `HelloWorld` contract using any account in the same network from Metamask.

```
contract HelloWorld {
  string public message;

  constructor(string memory initMessage) public {
    message = initMessage;
  }

  function update(string memory newMessage) public {
    message = newMessage;
  }
}
```

[5] In order to connect to a custom RPC endpoint, we have to select "*Injected Web3*" as the Environment option and enable "*Personal Mode*" in the settings panel

Just as before, we now have a contract address to call, that changes with every new update to the contract's code (recall that smart contract cannot be edited, they can only be re-deployed). Metamask also reports that the transaction, correctly identified as *Contract Deployment* this time, has been mined successfully. We can then use Remix's interface to easily `Update` and `Get` the message in the contract, using any other account from the same network; that is, this being a public contract with no restrictions whatsoever, anyone can read and write the value `message` stored in it.

Between the two methods, the key difference is the privacy features. Solidity, as a programming language, does provide some important keywords to reduce the scope of what an account can and cannot do. For example, we can limit the use of a `Set` function exclusively to the owner of the contract (the account who deployed the contract), which is something that will come in handy soon in our development. Unfortunately, Besu has a richer set of privacy features that cannot be replicated in Solidity, such as the privacy groups, that can only be programmed via the web3 client library.


Contract Deployment
#2 - 1/1/2020 at 23:04
CONFIRMED
-0 ETH

Details

From: 0x1b746f50fb6a4F31321212ead715476b5a96773E > New Contract

Transaction

Amount	0 ETH
Gas Limit (Units)	309554
Gas Used (Units)	309554
Gas Price (GWEI)	0
Total	0 ETH

Activity Log

Transaction created with a value of 0 ETH at 23:04 on 1/1/2020.
Transaction submitted with gas fee of 0 WEI at 23:05 on 1/1/2020.
Transaction confirmed at 23:05 on 1/1/2020.

Figure 20 - Contract deployment via Remix and Metamask

That being said, Remix remains an excellent GUI to query the contracts in a local network. Rather than compiling and deploying the Solidity code from the web IDE, Remix allows to specify the address at which the contract has been deployed, thus making possible, thanks to the integration with Metamask, to choose an account to use to set and read the values from it. We'll provide an example of this integration in later developments.

What we've done here is to set up a basic private network with the default settings, make a few transactions with the Metamask interface and deploy a smart contract. A private network like this provides a configurable test environment, and by configuring a low difficulty and enabling mining, blocks are created quickly. However, we can already point out a few things we'll need to address:

- Clique might or might not be the best consensus protocol for us.
- The requirement for Gas is questionable for our case.
- It's a private network, meaning that nobody from outside can access the transaction data. However, there's no privacy *inside* the network; that is, all transactions can be read by everyone with access to the network.

Let's see what it can be done for each one of these points, with a deeper dive into Hyperledger Besu configuration options.

5.3 Proof-of-Authority

The consensus protocol in a consortium environment is a very important choice. Remember that, with consensus, we're basically deciding how the power to validate transactions gets assigned in a decentralized system. This protocol should ensure equal opportunity for all participants to sign a block and for all transactions to be processed, sooner or later, regardless of them being accepted or rejected. We cannot afford for a transaction to sit in the transaction pool for long periods of time, for example because the low transaction fees make it unattractive, nor we can let any participant to gain more control over any other based on the amount of resources they have, as that would make the ledger more centralized than we'd like and governed by big conglomerates such as LVMH.

Given these inputs, the ideal choice for us would clearly be a proof-of-trust or proof-of-reputation consensus protocol, as used in VeChain or GoChain, but they're not supported in Hyperledger Besu. It's the genesis file that specifies the consensus protocol for a chain in the config property and the choices are Ethash, Clique and IBFT 2. **Ethash** is a PoW consensus protocol used to carry out mining activities on Ethereum Mainnet. This kind of protocol is unfit for a consortium environment, as we've seen previously in Chapter 3, so we'll just rule it out. On the other hand, Clique and IBFT 2.0 are PoA consensus protocols, used when there's a minimum level of trust between participants. They allow for faster block production times and have a much greater throughput of transactions than the Ethash protocol or any other PoW protocol for that matter. **Clique** is used by the Rinkeby testnet and can be used for private networks. In Clique networks, transactions and blocks are validated by approved accounts, known as signers. Signers take turns to create the next block and they propose and vote to add or remove signers. The attributes `blockperiodseconds` and `epochlength` in the configuration file allow us to specify the block time in seconds and the number of blocks after which to reset all votes. **IBFT 2.0** can be used for private networks as well, where transactions and blocks are validated by approved accounts, known as validators. Validators take turns to create the next block, and a majority (greater than 66%) of validators must sign the block before it can be inserted into the chain. Existing validators propose and vote to add or remove validators. A majority vote (greater than 50%) is required to add or remove a validator. IBFT 2.0 shares the same `blockperiodseconds` and `epochlength` attributes as Clique and it adds a `requesttimeoutseconds` attribute as a timeout for each consensus round before a round change. Being so similar, we must decide on the few but key differentiating factors. IBFT 2.0 requires 4 validators to be Byzantine fault tolerant while Clique can operate with a single validator, although that offers no redundancy in case of node failure. IBFT 2.0 has immediate finality, meaning that all valid blocks are included in the chain and no forks can occur; Clique does not have that feature and forks must be dealt with. In terms of liveness, Clique is more fault tolerant as it keeps working with half the validators offline while IBFT 2.0 networks require at least 2/3 of validators to be fully operational (immediate finality cannot be guaranteed, otherwise). These characteristics make Clique the faster choice for reaching consensus compared to IBFT 2.0, that becomes slower with each new validator; however, with the increasing number of signers, the probability of a fork increases for Clique.

Our demos include few validators and there's no risk of fork whatsoever, so both are equally good. In a real scenario, with this being a multi-party, decentralized use case, we stand to the opinion that consistency (transaction finality) is to be preferred to performance, so we would recommend IBFT 2.0 which is an improvement over the original IBFT [18].

5.4 Zero Gas Network

In Ethereum based platforms like Besu, transactions use computational resources, hence they have a hidden associated cost. The cost unit is called gas and it's commonly defined as the unit that measures the amount of computational effort that it will take to execute certain operations. This computational effort is estimated by looking at the instructions used to execute a transaction in the Ethereum Virtual Machine (smart contracts are pre-compiled into bytecode); each and every line of code in a smart contract requires a certain amount of gas to be executed, hence it costs Ether. More specifically, each instruction belongs to a group with a different weight in terms of gas (details are available in the Ethereum Yellow Paper [19]), making some operations more expensive than others; basically, it's an estimated count of how much work will be performed at execution time. With one of the API of web3, *web3.eth.estimateGas*, we can get a rough idea about the gas we would need to execute a transaction. The gas price, instead, is the price per gas unit and it's defined by the sender of a transaction. The average gas price is typically on the order of about 20 Gwei (or 0.00000002 ETH) but can increase during times of high network traffic as there are more transactions competing to be included in the next block. Along with the gas price, the sender must also specify a gas limit, defined as the maximum amount of gas that he's willing to pay for that transaction. If that gas limit is too low, it will run out during processing and the transaction will be reverted (remember, transactions are atomic). If it's too high, it might exceed the block gas limit or take too much gas space, making it less economically attractive to miners who could earn more by mining more of the smaller transactions. That is because the transaction cost, ultimately, is estimated as gas used * gas price and it is paid in Ether by the account that submits the transaction to the miner that includes the transaction in a block. This works not only to avoid spammers and limit resource use, but also and most importantly as an incentive for people to run mining nodes.

In a private network, however, such incentive might not be needed. In our case, for example, the validators are run by the same participants who take benefits from the system being online. Adding gas to each transaction would be impractical and unnecessary, given that our miners are not decentralized, untrusted participants, but rather nodes within a closed network that do not need an incentive to finalize a transaction. Therefore, we decided to configure our network as the so-called *zero gas network*, that is, a network where the gas price is always set to zero. By doing so, even the block size and the contract sizes become less important (the former will never be reached, anyway), so we can set them to their maximum value. In order to do so, we modify the genesis file to allow for maximum block gas limit (in gas) and maximum contract size limit (in byte), by setting the following parameters:

```
"gasLimit": "0x1fffffffffffff"
"contractSizeLimit": 2147483647
```

Finally, we can start Besu specifying that we allow for gas price to be zero, adding the following parameters at startup:

```
--min-gas-price=0
```

5.5 Permissioning and Private Transactions

Now that we've settled for a Zero Gas Network with Clique as our consensus protocol, we need to define permissioning and keep the transactions private between the interested parties alone. In Hyperledger Besu, this is done with Orion, a private transaction manager that uses its own set of API methods to generate keypairs and manage privacy group details. It also features the possibility to use a standard RDBMS, such as Oracle or PostgreSQL, to store the payloads of the private transaction. If not specified, by default it will use RocksDB, a high-performance embedded database for key-value data.

Taking our previous demo, we'll start by defining permissioning to support node and account whitelisting, hence restricting communication to only the nodes in the whitelist. We'll keep the same core architecture as the first example, so Node-00 will be our only bootnode. Permissioning rules can be set at the chain level or at the node level; however, keep in mind that in both cases the rules defined should apply to the entire network, meaning that a whitelisted account must be whitelisted for everyone. We'll define local permissioning by creating a `permissions_config.toml` file that must be copied to each node's data directory. In the `accounts-whitelist` parameter we'll put a couple of fake test accounts, while in the `nodes-whitelist` we'll add the enode URLs of our nodes, one by one by hand or with the `perm_addNodesToWhitelist` JSON-RPC API method. We also use the `admin_addPeer` JSON-RPC API method to add Node-00 as a peer for Node-01 and Node-02, but not Node-03 which will remain unlinked from the others. With this setup, using Metamask again, we can demonstrate that only transactions from whitelisted accounts can be performed, and that only whitelisted nodes can be peers. That is, attempting a transaction (Tx) with an account not in the whitelist will fail almost immediately; more precisely:

- A Tx from an account in the whitelist to a node in the network is accepted and processed as usual. This is also true for recipients that do not have an active node in the network.
- A Tx from an account not in the whitelist to anyone else in the network is rejected immediately.
- Nodes not in the whitelist cannot be peers. Starting Node-03 with Node-00 as the `bootnodes` option will result in Node-00 ignoring the request, because Node-00's whitelist is missing Node-03's enode URL. Trying to add it manually via API will also result in an error. The only way to make two nodes peers is to have them in their whitelist, reciprocally.
- A Tx from an account in the whitelist of node A, but not in node B's, will force node B to reject the transaction and disconnect from node A as peer. Node A will accept and process the transaction as usual. That is why, as mentioned before, permission must be agreed at the network level.

```
Invalid transaction: Sender 0xca202e3ab4792e8826d7c2ad6e4e75e0d5136937 is not on the
Account Whitelist
Failed to import announced block 197
(0xc111a85237920abc77d281b2885504fcb83222f4313918dceceba608ee322afd)
FullSyncTargetManager | No sync target, wait for peers.
```

This is exactly why *onchain* permissioning is a better choice for enterprise ledgers, as these lists are defined once and for everyone by smart contracts, the first version of which can be deployed together with the genesis file.

Now that we've got the permissioning rules laid out, the second step is to make transactions private. We create the Orion subdirectory for each node, as we did earlier with the data subdirectory, and we store the keys for that node that we'll generate using this command:

```
orion -g Node-00/Orion/Node00Key
```

The prompt will ask for a password of the keypairs. We need an Orion configuration file for each Orion instance, where we'll specify the usual information as well as the keys' path (`othernode` is like `bootnode`, necessary for all non-bootnodes alone):

```
nodeurl = "http://127.0.0.1:8081/"
```

```
nodeport = 8081
clienturl = "http://127.0.0.1:8889/"
clientport = 8889
publickeys = ["Node01Key.pub"]
privatekeys = ["Node01Key.key"]
passwords = "node01password"
tls = "off"
othernodes = ["http://127.0.0.1:8080/"]
```

Once we've done this operation for all the nodes, we should be looking at something like this:

```
├── D01-private-network-withOrion
│   ├── genesis.json
│   ├── Node-00
│   │   ├── data
│   │   │   ├── key
│   │   │   └── N00-Address
│   │   └── Orion
│   │       ├── Node00Key.key
│   │       ├── Node00Key.pub
│   │       ├── orion.conf
│   │       └── password
│   ├── Node-01
│   │   ├── data
│   │   │   └── key
│   │   └── Orion
│   │       ├── Node01Key.key
│   │       ├── Node01Key.pub
│   │       ├── orion.conf
│   │       └── password
│   ├── Node-02
.   .
.   .
```

We can now start our Orion nodes, with this simple command

```
orion <orion-configuration-file>
```

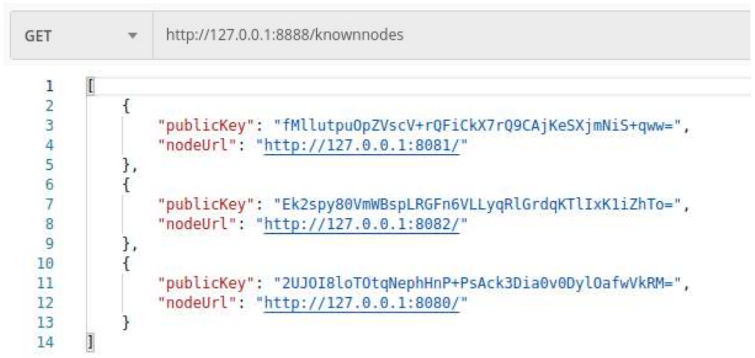
If we look at the logs, we'll see that the Orion nodes are finding each other. We can check "who sees who" by making this call

```
curl -X GET http://127.0.0.1:8888/knownnodes
```

which returns the nodes URL and their public keys of their own Orion nodes. It might be useful to mention that all the API calls we've done so far can be imported and run in more user-friendly applications such as Postman.

With this information, we start our permissioned network, this time specifying that we're also running Orion nodes and that our gas price can be zero. For example, for Node-01:

```
besu \
--data-path=Node-01/data \
--genesis-file=genesis.json \
--rpc-http-enabled \
--rpc-http-port=8546 \
--rpc-http-api=ETH,NET,CLIQUE,PERM,EEA,PRIV \
--host-whitelist="*" \
--rpc-http-cors-origins="all"
--privacy-enabled \
--privacy-url=http://127.0.0.1:8889 \
--privacy-public-key-file=Node-01/Orion/Node01Key.pub \
--bootnodes=<Node-00 enode URL> \
--p2p-port=30304 \
--min-gas-price=0 \
--permissions-nodes-config-file-enabled \
--permissions-accounts-config-file-enabled=false
```



```
http://127.0.0.1:8888/knownnodes
```

```
[
    {
        "publicKey": "fMllutpuOpZVscV+rQFiCkX7rQ9CAjKeSXjmNiS+qww=",
        "nodeUrl": "http://127.0.0.1:8081/"
    },
    {
        "publicKey": "Ek2spy80VmWBspLRGFn6VLLyqRlGrdqKTlIxK1iZhTo=",
        "nodeUrl": "http://127.0.0.1:8082/"
    },
    {
        "publicKey": "2UJOI8loTOtqNephHnP+PsAck3Dia0v0DylOafwVkRM=",
        "nodeUrl": "http://127.0.0.1:8080/"
    }
]
```

Figure 21 - Orion API call in Postman

Since we're starting to have way too many parameters, it might be a good idea to start our nodes with a TOML configuration file, like this:

```
besu --config-file=<Node-0X config.toml>
```

Note that, since account permissioning cannot be used with random key signing [22] (which is how Besu is configured to work by default), we had to set to false the `permissions-accounts-config-file-enabled` parameter. In order to have both functionalities, we would need to set a specific signing key to mark private transactions, using the `privacy-marker-transaction-signing-key-file` parameter in the node's TOML configuration file and adding it to the accounts whitelist. Failing to do so will lead to a permissioning error when trying to deploy the smart contract.

With this setup, we can now try to use Metamask again to send ETH, but the documentation itself suggests that, for private transactions, the web3.js-eea client library is the best choice. As a first example, let's try a `HelloWorld` contract, private between Node-00 and Node-01, using one of the scripts in the web3js-eea library. The solidity code of this contract simply stores a value and broadcasts it to the network. We update the file `keys.js` to store our nodes' RPC URLs and the private keys of both Besu and Orion, then we run

```
node deployContract.js
```

to deploy the contract to our network. This operation will assign the contract an *address*, to which we can send calls and transactions. Here's the output:

```
Transaction Hash  0x6e15c84eebc5de82d3c6120962cbda5997176998edef80827b0e3bb8b8e46fc6
Waiting for transaction to be mined ...
Private Transaction Receipt
{ contractAddress: '0xe63a49e124e89a19d51938bd4ab8b9b6b3f480c3',
  from: '0x3f5aa82175339f070e6f04a659b2be1f7b509911',
  output: '0x6080604052600436106100575760ffffffffffff...',
   commitmentHash:
    '0x5f6b8a698d4638ef14f395c84d3dfde172ed3ca5f9d384af4ae6008a0c7672fc',
  transactionHash:
    '0xc8d3fb5f729794999628a644b16fc56463b4b6b343aa39112ceb3cf7456b07a8',
  privateFrom: '2UJOI8loTOtqNephHnP+PsAck3Dia0v0DylOafwVkRM=',
  privateFor: [ 'fMllutpuOpZVscV+rQFiCkX7rQ9CAjKeSXjmNiS+qww=' ],
  status: '0x1',
  logs: []
}
```

By checking the logs, we'll notice that this is treated like any other transaction. Now, with the smart contract's address, we can launch `storeValueFromNode1.js` and `storeValueFromNode2.js` that respectively store value 1000 from Node-00 and value 42 from Node-01 in the contract. Here's the output of the first one:

```
Transaction Hash: 0x4a63a83cb4f4995b2f809f1a77b334087fe857d06472c2138f54f490252e591a
Waiting for transaction to be mined ...
Event Emited:
0x0000000000000000000000003f5aa82175339f070e6f04a659b2be1f7b509911000000000000000000
00000000000000000000000000000000000000000003e8
Waiting for transaction to be mined ...
Get Value from http://127.0.0.1:8545:
0x00000000000000000000000000000000000000000000000000000000000003e8
Waiting for transaction to be mined ...
Get Value from http://127.0.0.1:8546:
0x00000000000000000000000000000000000000000000000000000000000003e8
Waiting for transaction to be mined ...
Get Value from http://127.0.0.1:8547: 0x
```

In more detail, this is what's happening step by step:

- Tx #1: the value in the smart contract is updated to 1000 and broadcasted to the network
- Tx #2: we perform a `GET` operation from Node-00 and it returns 1000 (hex 3e8)
- Tx #3: we perform a `GET` operation from Node-01 and it returns 1000 (hex 3e8)
- Tx #4: we perform a `GET` operation from Node-02 and it returns nothing

Since Node-02 is unable to see the payloads of these transactions, the private configuration is working properly. We can check Node-02's logs to see that it's importing the blocks, although it reports it doesn't have full access to their content (enclave is another name for the Orion node).

```
ERROR | PrivacyPrecompiledContract | Enclave probably does not have private
transaction with key VEZa+1G69azW4Hp6YtCQR5HWIfiSiOFhd8XlMirieHI=.
```

Another interesting way of making privacy work is through *Privacy Groups*. A private group is a named list of Besu addresses and their own Orion public keys that takes the place of the `privateFor` parameter in the creation of a contract, where normally we would put the recipient's Orion's public key. A private group is created and distributed very much like a private transaction, therefore making it impossible for the not-included parties to see the details of the transaction (in this case, the nodes in the list); the Orion node will, in fact, ignore transactions addressed to private groups it doesn't belong to. The behaviour is identical to the example we just made: if we specified a private group composed of Node-00 and Node-01, and recompiled the previous contract with this quick change after having created a private group

```
privateFor: [orion.node2.publicKey] → privacyGroupId
```

we would get the same output as before, except that for Node-02 the JSON-RPC response would be the following

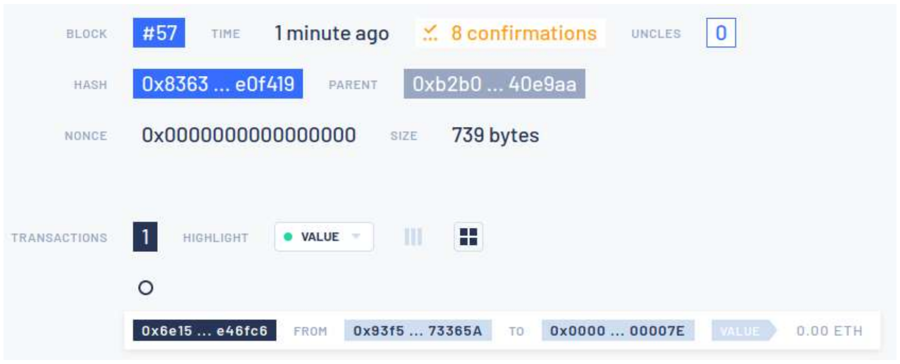
```
{"jsonrpc":"2.0","id":1,"error":{"code":-50200,"message":"PrivacyGroupNotFound"}}
```

which is correct, since Node-02 was never part of this group.

We can explore the transaction details more easily by running an instance of the *Ethereum Lite Explorer*, starting a docker container from the official image with the following command, specifying a port and our node's JSON-RPC URL as usual:

```
docker run --rm -p 9191:80 -e APP_NODE_URL=http://localhost:8545 alethio/ethereum-
lite-explorer
```

Opening the browser at localhost:9191 will reveal the web-application homepage. Here we can either search for the transaction or query the database by writing down the transaction hash:



BLOCK #57 TIME 1 minute ago 8 confirmations UNCLES 0

HASH 0x8363 ... e0f419 PARENT 0xb2b0 ... 40e9aa

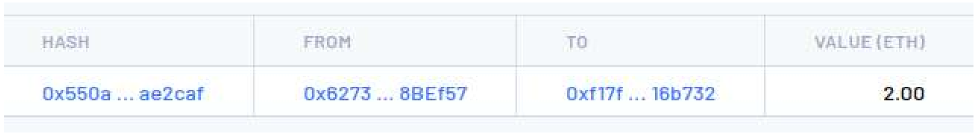
NONCE 0x0000000000000000 SIZE 739 bytes

TRANSACTIONS 1 HIGHLIGHT VALUE

0x6e15 ... e46fc6 FROM 0x93f5 ... 73365A TO 0x0000 ... 00007E VALUE 0.00 ETH

Figure 22 - Transaction details in Alethio Block Explorer Lite (1/3)

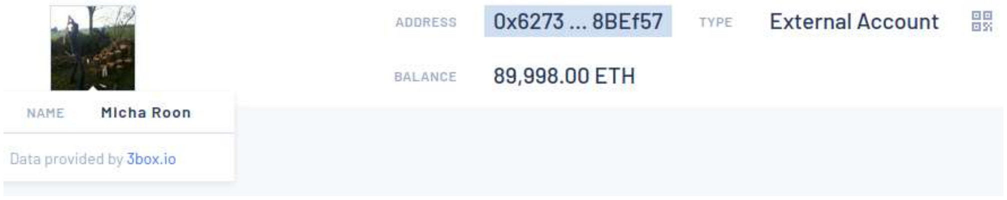
If we moved simple ETH, we would look at something like the following:



HASH	FROM	TO	VALUE (ETH)
0x550a ... ae2caf	0x6273 ... 8BEf57	0xf17f ... 16b732	2.00

Figure 23 - Transaction details in Alethio Block Explorer Lite (2/3)

Clicking the hash will take us to the transaction details, while clicking the address of one of the parties involved will take us to the account details. This being a test account for Ethereum, we can even get a name for the account's owner:



ADDRESS 0x6273 ... 8BEf57 TYPE External Account

BALANCE 89,998.00 ETH

NAME Micha Roon

Data provided by 3box.io

Figure 24 - Transaction details in Alethio Block Explorer Lite (3/3)

This privacy feature, in both its forms (detailed or via private groups), comes in handy for many reasons, but we could say that the two most practical and realistic applications are the aggregation of one's brand entire supply chain's list of nodes under the same privacy group, as well as many others private groups for commercial reasons, if desired, as this would keep the sales sensitive data hidden and secured from competitors of both the brands and the suppliers.

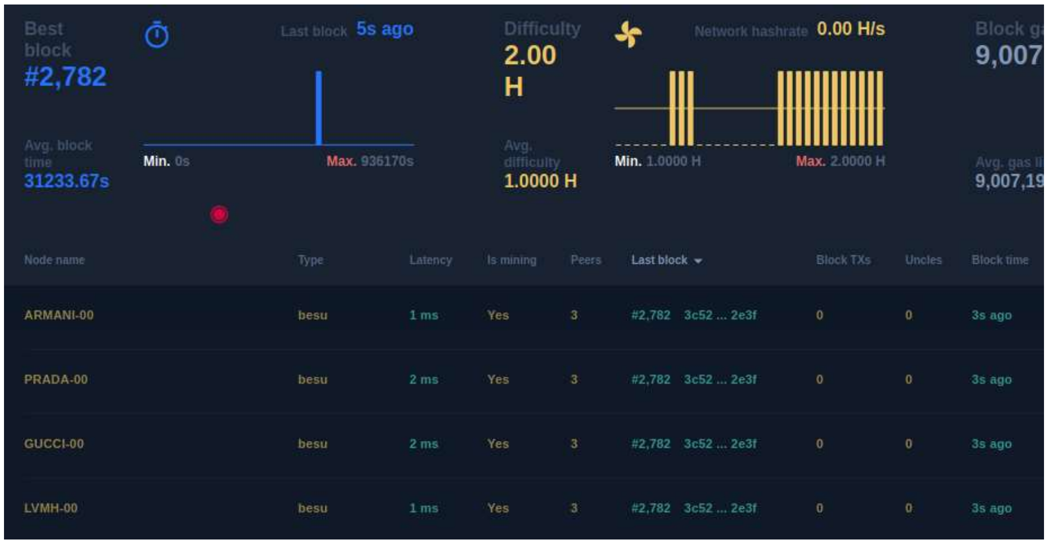
5.6 Real Case Deployment

Now that we have a better practical understanding of the Besu platform, we can finally give a context to our simulation, building upon the permissioned network of the previous demos. We create a custom network, by configuring yet another genesis file with `network-id=2020`. If we wanted to, we could connect to a known network by either giving the right `network-id` (e.g. `id=1` is the Ethereum public mainnet) or by specifying its name with the parameter `network`; keep in mind, however, that these networks have specific consensus protocols that cannot be changed (e.g. Rinkeby uses PoA, Ethereum mainnet uses PoW). The genesis file we deployed has a slower output of blocks compared to our previous demos, one every 30 seconds; during development, we suggest keeping this at a faster pace. The parameters `gasLimit` and `contractSizeLimit` are both specified to enable a zero-gas network, while the `extraData` field stores the addresses of all four brand owners as bootnodes and validators. By doing so, the network will never actually start producing blocks until 3 of these 4 nodes are up and running; also, the block production frequency is evenly distributed between these nodes. A valid alternative is to specify a set of static nodes by listing the enode URLs in a `static-nodes.json` file stored in the data directory of each node and disabling automatic discovery. The nodes whitelist has been disabled by setting the parameter `permissions-nodes-config-file-enabled=false`; please note, however, that in a production environment it must be set to True, although it's highly suggested to set up a separated server to centralize the permission management, as previously explained, so that every node is always up to date with the list being either pushed by the server or read by polling the server. The default quantity of blocks needed for transaction confirmation is 32, but we can reduce it with a simple instruction:

```
web3.transactionConfirmationBlocks = 10;
```

Also, please know that a couple of scripts has been prepared to run the docker containers necessary for Alethio Block Explorer and Alethio Ethstats Network Monitor, both useful tools to quickly check the ledger and the network:

```
ext-tools/alethio-start.sh
ext-tools/alethio-stop.sh
```


Best block #2,782
Avg. block time 31233.67s
Last block 5s ago
Min. 0s Max. 936170s
Difficulty 2.00 H
Avg. difficulty 1.0000 H
Network hashrate 0.00 H/s
Min. 1.0000 H Max. 2.0000 H
Block g 9,007
Avg. gas li 9,007,19

Node name	Type	Latency	Is mining	Peers	Last block	Block TXs	Uncles	Block time
ARMANI-00	besu	1 ms	Yes	3	#2,782 3c52 ... 2e3f	0	0	3s ago
PRADA-00	besu	2 ms	Yes	3	#2,782 3c52 ... 2e3f	0	0	3s ago
GUCCI-00	besu	2 ms	Yes	3	#2,782 3c52 ... 2e3f	0	0	3s ago
LVMH-00	besu	1 ms	Yes	3	#2,782 3c52 ... 2e3f	0	0	3s ago

Figure 25 - Alethio Ethstats Lite for Network Monitoring

5.6.1 Domain

Our network is composed by these participants. The naming convention should be intuitive: nodes with suffix -XX are general purpose, while nodes with a numerical suffix, such as -00, belong to the same supply chain of the brand owner with the same suffix:

- 4 brand owners (BRO-XX)
- 1 production line (PRL-XX)
- 1 supplier (SUP-XX)
- 1 broker (BRK-XX)
- 1 RFID tags distributor (RFD-XX)
- 1 control agency (CAG-XX)
- 1 official reseller (RES-XX)

Each one of the brand owners has one or more peer nodes representing the entities of its own supply chain; that is, a brand owner might have two production lines, one broker and one official reseller, counting four total peers. Bootnodes are only used to discover other peers and have no consequences on transaction visibility, but we used it also as a way to keep things organized: node PRL-00, for example, being the Production Line of Brand Owner 00, only needs the enode of BRO-00 to stay connected to the network.

For the same reason, it might be a good idea to keep BRO-00 and PRL-00 in the same Privacy Group, so that private transactions (e.g. deals with suppliers) stay private for that company. We can do that by calling the `createPrivacyGroup` JSON-RPC API, specifying the Orion public keys of each node taking part into the group, a unique name and a description:

```
curl -X POST \ --data '{
"jsonrpc":"2.0",
"method": "priv_createPrivacyGroup",
"params":[{
  "addresses":[
    "2UJOI8loTOtqNephHnP+PsAck3Dia0v0DylOafwVkRM=",
    "/0W702sDhd2o95RQiYTswgixNlsCx58bvEWNJ5dfajU="],
  "name":"Group SupplyChain-00",
  "description":"Private channel for BRO-00 and all its supplychain nodes"}],
"id":900}'
http://127.0.0.1:8545
```

As usual, the logs will give us some more details: the request for a privacy group creation is sent by the Orion node of BRO-00 to PRL-00's, that responds with a 200 ACK. The privacy group is then assigned a unique identifier and a digest, that will be used in the private transactions. These common CRUD operations can be performed on privacy groups via a node script as well.

We do the same for all other participants and the result is the following. Note that only some participants have a private component: resellers, RFID suppliers and the control agency, for example, are not expected to make private transactions, therefore they do not have an Orion node and their toml configuration files reflects this too.

```
├── BRK-XX
│   ├── data
│   │   ├── database
│   │   │   └── private
│   └── Orion
├── BRO-00
│   ├── data
│   │   ├── database
│   │   │   └── private
│   └── Orion
├── BRO-01
│   ├── data
│   │   └── database
│   │       └── private
│   └── Orion
├── BRO-02
│   ├── data
│   │   └── database
│   │       └── private
│   └── Orion
├── BRO-03
│   ├── data
│   │   └── database
│   │       └── private
│   └── Orion
├── CAG-XX
│   └── data
│   │   ├── database
├── PRL-00
│   ├── data
│   │   └── database
│   │       └── private
│   └── Orion
├── SUP-01
│   ├── data
│   │   ├── database
│   │   │   └── private
│   └── Orion
├── RES-00
│   ├── data
└── RFD-XX
    └── data
```

5.6.2 Business Logic

The supply chain system we decided to model, at its core, is a collection of many pieces of code that, once put together, create a more complex and complete distributed software, with all the privacy features discussed so far. Indeed, if we look back at Figure 17, every block in the figure is a smart contract call or a smart contract transaction. There's more than one possible path that we can take to code these functionalities: a brand owner could deploy its own business logic or, this being a consortium, a group-effort might be preferred to decide how to manage specific use cases, trying to find a common ground for common necessities.

The next steps will provide a possible solution, implemented in Solidity and tested mostly via Remix. The test cases will be also used as an excuse to demonstrate some characteristics of the Solidity language; luckily, we opted for a zero-gas network, or we would need to pay great attention to instructions' costs and gas management overall. Note that the examples that follow solve one problem at a time and they play in a restricted scope. A production deployment would require for a more cohesive, structured, precise code and contracts management; moreover, the code that will be posted is a partial extract from the GitLab repository.

5.6.2.1 Generating and assigning RFID tags

The first case we're going to cover is the RFT generation and assignment. As stated back in Chapter 4.3, there are many ways this can work but the most plausible are:

- Brand A buys a set of RFTs from a supplier registered in the network. This supplier writes the information in the ledger.
- Brand A buys a set of RFTs from a supplier external to the network and declares the ownership of the codes by itself.

Programmatically, the choice comes down to who owns the contract, who writes the RFID tag data in the ledger, who can update its values and who can read them. We decided early on that everybody could read this data, as that's the only way we can perform checks later in the supply chain. The contract `RfidTagsCollection.sol` tries to offer a solution with just one supplier, based on the assumptions that:

- Every RFT supplier deploys its own contract to manage its codes. This is because each supplier may generate RFTs with different properties, and they may want to manage the whole cycle differently. Also, this would allow for privacy between suppliers.
- Only the supplier can generate a RFT and store it in the ledger.
- Brand A can updates the values of its own RFTs alone.

The basic structure for a RFID tag (RFT) is defined as follow:

```
struct RFT {
  string uuid;
  uint exchanges;
  address brand;
  address[] owners;
  address currentOwner;
  bool isValid;
} mapping(string => RFT) private rfts;
```

The mapping is a data key-value structure in Solidity that resembles a hash table. The binding with the `uuid` allows for indexing and faster search (see Figure 26). The variable is private as we provide a way to get the important information via a `GET` function. In a real scenario, we would add an entire set of RFTs at once from a CSV, rather than one code at a time by hand. The `uuid`, in the format specified in chapter 5.1, will be given as input from DApps.

What makes possible the assumptions listed above is the implementation of Solidity's modifiers, which requires that certain criteria must be met before performing an action. We implemented two modifiers, one to check if the account who calls the contract is the contract's owner (the tags supplier), one to check if the account is trying to perform an action on someone else's tag. By letting modifiers manage these cases, we can easily change our rules by deploying an updated contract: if we decided to let a brand owner add its own RFTs, we could edit the program as follows:

```
function create(string memory _uuid, address _brand)
public notOnlyContractOwner(_brand)
{...}

modifier notOnlyContractOwner(address _brand) {
  require(
    msg.sender == contractOwner || msg.sender == _brand,
    "Only the contract's owner and the brand owner can perform this action."
  ); _;
}
```

These modifiers prove to be useful in many cases. For example, if we wanted to add a sensitive information on the RFT set (e.g. sell price), we would add a private variable `price` and a `get` function that can be called exclusively by the owner of those RFTs, although it's preferable to have this managed in a separated contract, where Besu' extended privacy features can be implemented. Solidity also provides programmers a set of keywords to "protect" a contract from an incorrect use of its functions (view, pure, constant, etc.), as we'll see in a moment.

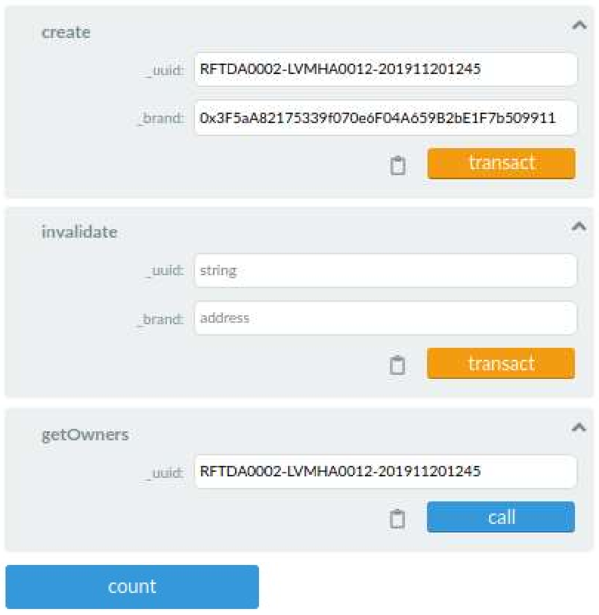
Being a public contract, we can test our code with Remix before deploying it into our network. We deploy the contract with account `0xcA202e3aB4792E8826d7C2AD6e4E75e0D5136937` in our custom network, hence representing our RFT supplier. We generate the RFID tag `RFTDA0002-LVMHA0012-201911201245` and we store it for brand `0x3F5aA82175339f070e6F04A659B2bE1F7b509911`. If we try to mark as invalid this RFT, calling the function `markInvalid` with another account, we get the following message as expected.

```
transact to RfidTagsCollection.markInvalid errored: VM error: revert. revert The
transaction has been reverted to the initial state. Reason provided by the contract:
"A brand owner can mark as invalid only its own UUIDs."
```

The same principle applies if we tried to create a new RFT from an account that does not match with the owner of the contract:

```
transact to RfidTagsCollection.create errored: VM error: revert. revert The
transaction has been reverted to the initial state. Reason provided by the contract:
"Only the owner of this contract (RFID supplier) can generate new UUIDs."
```

This solves one of the potential issues mentioned earlier that can arise in our information flow, that is, when a RFT seems lost in the supply chain. If we assume the tag never reached the expected destination due to illegal activity, invalidating the code will also invalidate the real-fake good that will be associated with it.


create
_uuid: RFTDA0002-LVMHA0012-201911201245
_brand: 0x3F5aA82175339f070e6F04A659B2bE1F7b509911

invalidate
_uuid: string
_brand: address

getOwners
_uuid: RFTDA0002-LVMHA0012-201911201245

Figure 26 - RFID tags smart contract interface in Remix

The provided function `getOwners()` covers the requirement to query the contract for a full list of owners of the requested RFT. It's a public function, as we want this information to be publicly available, and it's also a `view` function, meaning that it cannot change the state of the contract (only read operations are possible), and that includes the option of emitting events. In Solidity, an *Event* is like a broadcasted message for JavaScript DApps that are listening for those events and they cause the arguments to be stored in the transaction's log, a special data structure in the blockchain. By using events, then, we can promptly communicate to the connected applications that a transaction occurred, and by simple parsing of the JSON response given as input, data can be shown or used.

A good practice to have events for significant operations that change the public state, although it is not mandatory. If we wanted to emit events from the function `getOwners()`, we would need to apply a few changes, and to declare such event. The following could be a possible implementation:

```
function getOwners (string memory _uuid)
  public {
    for(uint i=0; i<rfts[_uuid].owners.length;i++)
      emit LogOwnersHistory(rfts[_uuid].uuid, rfts[_uuid].owners[i]);
  }

event LogOwnersHistory(string uuid, address owner);
```

The creation of the RFT already involves two participants, hence after its creation we can call the function `getOwners()` and see that the RFT had indeed two owners: the participant who generated the tag (RFD-XX) and the brand to whom it was assigned (BRO-00).

```
[
  {
    "from": "0x67028408e18CDC11BfdB3E57d728717b99Cdd8f1",
    "topic": "0x976030417fec254b3380121b400ca6ebb772246f7d86d83d5f1ccf1caf457b53",
    "event": "LogOwnersHistory",
    "args": {
      "0": "RFTDA0002-LVMHA0012-201911201245",
      "1": "0xcA202e3aB4792E8826d7C2AD6e4E75e0D5136937",
      "uuid": "RFTDA0002-LVMHA0012-201911201245",
      "owner": "0xcA202e3aB4792E8826d7C2AD6e4E75e0D5136937",
      "length": 2
    }
  },
  {
    "from": "0x67028408e18CDC11BfdB3E57d728717b99Cdd8f1",
    "topic": "0x976030417fec254b3380121b400ca6ebb772246f7d86d83d5f1ccf1caf457b53",
    "event": "LogOwnersHistory",
    "args": {
      "0": "RFTDA0002-LVMHA0012-201911201245",
      "1": "0x3F5aA82175339f070e6F04A659B2bE1F7b509911",
      "uuid": "RFTDA0002-LVMHA0012-201911201245",
      "owner": "0x3F5aA82175339f070e6F04A659B2bE1F7b509911",
      "length": 2
    }
  }
]
```

As a last note, since we're going to use the RFT object quite often, it's better to move the definition of the `struct` to a library that can later be imported and referenced from other contracts. We'll call this library `SharedObjectsLibrary`.

5.6.2.2 Updating and checking a participant's Trust Score

One of the key elements of the solution is a feature called *Trust Score*. Participants can evaluate who to do business with depending on this value, as it proves a participant's "green" performance and the sustainability of its own business. We mentioned earlier that a Trust Score can be designed to be unique for each participant or unique for each business relationship between participants, in which case it would assume another meaning. In our basic design, we'll only consider the most valuable feedback given by the control agency who randomly checks suppliers to validate their claims and their business against important metrics of transparency and sustainability. If we simplify by assuming that CAG-XX is the only owner of this activity, we can write a smart contract named `TrustScoresCollection.sol` that's fairly simple. We define a Trust Score as an object with an history of `score` values and the address of its owner, although nothing would stop us from adding more information such as a textual report or, even better, the base64 of the documents generated as a result of the inspection. It's worth saying that usually, however, the Ethereum community suggests using distributed file systems such as IPFS[6] for storing documents.

```
struct TrustScore {
  address participant;
  uint currentScore;
  uint[] allScores;
  string latestReport;
  uint scoresCount;
} mapping(address => TrustScore) private trustscores;

function getCurrentScore(address _par)
public view returns (uint) {
  return trustscores[_par].currentScore;
}
```

This object is yet again associated with a `mapping` data structure, rather than an array, for performance reasons. The important thing to consider in this public contract is that the Trust Scores are also public, therefore with the aid of a standard get function we can check the score for a given participant's id, calling the function from another contract such as `TrustScoreExample.sol`. From the same contract, we can call a function to update the score; however, in this case we would need to declare the function `update` as `external`, making it callable exclusively from other contracts.

```
import "./TrustScoresCollection.sol";

contract TrustScoreExample {
  function get(address _ts, uint _id)
  public view returns (uint) {
    return UpdateTrustScore(_ts).getTS(_id);
  }

  function set(address _contract, address _par, uint _score, string memory _report)
  public {
    TrustScoresCollection(_contract).update(_par, _report, _score);
  }
}
```

The choice of the function's modifier (public, private, internal, external) is also extremely important. They set the visibility scope of the functions they're applied to in respect to the network, allowing or restricting particular use cases such as the `delegatecall`, a low-level function of the EVM that works as follows: when Account A calls Contract C from yet another Contract B, `delegatecall` will still see

[6] https://ipfs.io/

Account A as the message's sender, whereas a standard call would see Contract C's address as the message sender [29].

In conclusion, this setup allows for a brand owner to check the Trust Score of its business partner, both at the moment of a new transaction or even periodically, to update an internal list of trusted suppliers. In a process of data analysis, this kind of metric can be ingested and studied to drive business decisions; indeed, given a context, a simple Trust Score becomes significant for the stakeholders in their monthly or quarterly meeting to update the company's business relationships.

5.6.2.3 Exchanging assets

Looking back at Figure 17, the most frequent type of transaction is the exchange of a set of RFT from one participant to another. As described in Chapter 4.3, this exchange is managed by two transactions: one performed by the sender, who declares the set of RFT in transit and its destination, and one from the receiver, who checks he received all and only what he was supposed to receive. This contract will have to interact with both previous ones, so it's necessary to get the deployment address in order to reference the correct instance. The developers of Arianee, one of the solutions discussed in Chapter 3.4.1, had the great idea to store in a contract the list of addresses of the other main contracts, so that their management becomes easier for everyone else.

We'll start by creating instances of the other contracts and the Exchange struct. To each of these variables, we'll assign the corresponding contract's address at the moment of the contract's deploy.

```
RFTagsCollection rftContract;
TrustScoresCollection tsContract;

constructor(RFTagsCollection _rftAddress, TrustScoresCollection _tsAddress)
public {
  rftContract = _rftAddress;
  tsContract = _tsAddress;
}

struct Exchange {
  uint transferId;
  address from;
  address to;
  string[] set;
  string status;
} mapping(uint => Exchange) public exchanges;
```

The exchange is a simple `struct` with an identifier, addresses of both senders and recipients, and an array of RFT codes. The transfer operation checks, for each RFT in the given set, that the current owner is the same node sending the request for transfer; all the RFTs for which this condition is false will be not be added to the list. It also checks, for demonstration purposes, that the recipient has a Trust Score higher than 70, refusing the transaction otherwise; clearly this threshold is a placeholder and should be given as input from the contract's caller, as each brand may decide to have different minimum acceptable value for the Trust Score of its own business partners, on a case-by-case basis. Values in the RFT object get updated accordingly, such as the `currentOwner` after a successful exchange. On the other side, a `closeTransfer()` does the opposite: it checks, for a given set of RFT and a `transferId`, that the content was all declared from the correct owner and that the pallet had been, indeed, sent.

```
function beginTransfer(address _to, string[] memory _set)
public returns (uint transferId) {
  exCount++;
  Exchange memory _exchanges;
  if(tsContract.getCurrentScore(_to) > 70) {
    for(uint i=0;i<_set.length;i++)
      if(msg.sender==rftContract.getCurrentOwner(_set[i])) {
        _exchanges.transferId = exCount;
        _exchanges.from = msg.sender;
        _exchanges.to = _to;
        _exchanges.set = _set;
        _exchanges.status = "SENT";
        exchanges[exCount] = _exchanges;
        rftContract.updateOwner(_set[0], _to);
      }
    }
    return exCount;
  }
```

The RFTs that are not supposed to be in the package can be communicated via an event, so that a DApp (for each brand owner) can listen and take actions, such as calling the function `invalidate()` we've discussed before. The following example only checks for lost RFTs, but checks can be extended to owners, IoT data and more.

```
function closeTransfer(uint _transferId, string[] memory _set)
public {
  bool exists=false;
  for(uint i=0; i<exchanges[_transferId].set.length; i++) {
    exists=false;
    for(uint j=0; j<_set.length; j++)
      if( keccak256(abi.encodePacked(_set[j]))
          == keccak256(abi.encodePacked(exchanges[_transferId].set[i]))) {
        exists = true;
      }
     if(!exists)
     emit WarningRFT(
       msg.sender, exchanges[_transferId].from, _transferId,
       exchanges[_transferId].set[i], "The RFT seems lost"
      );
  }
}

event WarningRFT(address receiver, address sender, uint transferId, string uuid,
string message);
```

We simulated the expedition of two RFTs, codes `RFTDA0002-LVMHA0012-201911201245` and `RFTDA0451-LVMHA0015-202012201111`, but only one reached the destination. Here's an event log of `WarningRFT`:

```
{
  "from": "0xC14E1dd8a8BFceD9C7360C766E13E5fd4d6C06df",
  "topic": "0x9a67d99146db6c429e29990e36a801c60752b9cd50fde992b725f9def72c6cdb",
  "event": "WarningRFT",
  "args": {
    "receiver": "0xcA202e3aB4792E8826d7C2AD6e4E75e0D5136937",
    "sender": "0xFB9DBB4CC9818930b2D3DEa68C60615f56e054D0",
     "transferId": "1",
    "uuid": "RFTDA0451-LVMHA0015-202012201111",
     "message": "The RFT seems lost",
    "length": 5
  }
}
```

5.6.2.4 Making private deals while announcing publicly the ownership of RFTs

Perhaps one of the trickiest transactions to deal with is the one that needs data to be both public and private at the same time. This is where Solidity's modifiers and privacy keywords can't help us, as those apply to the last layer of the information flow, whereas we need strict governance control starting from the data layer. An example would be the exchange between a broker and a supplier: let's suppose that BRK-XX buys 5 units of asset A at $2500 and that this pallet is assigned a tag. We want to hide the value of the exchange, but we also want to publicly take ownership of both the RFID tag and the pallet of asset A.

Ideally, we would accomplish this by deploying two smart contracts, one called by the other. That is, the first one, that will be private between the two main nodes or private to the Privacy Group, records the exchange, but it also calls the second, public contract to link the RFID tag with the asset A. Unfortunately, we weren't able to reproduce this behaviour in Solidity alone. There seems to be no way to directly connect a public transaction with a private contract or otherwise cross that boundary, which is intuitive as that would make impossible for other nodes to verify the public transactions without the data that comes with the private bytecode. A private smart contract may read a function of a public smart contract, but not the other way around, as clearly indicated in Besu official documentation:

The Private Transaction Processor executes the transaction. The Private Transaction Processor can read and write to the private world state and read from the public world state.
(Hyperledger Besu Documentation, Processing private transactions) [20]

This is also consistent with most of the others Ethereum clients that keep a public and private state for these situations. If we assume, then, that a private contract may not update the state of a public smart contract either, a viable (and generally used, as a matter of fact) workaround would be to let this process be handled by the application backend, that will call the two contracts sequentially, passing the correct inputs twice; of course, the sensitive data would not be passed to the public contract.

6 Conclusions and Future Developments

The impact that blockchain can deliver on the fashion industry is huge, and that's true for both the supply chain management area as well as retail, but considerable investment is required for that to materialize, in order to construct the systems to support it. The real-time access to streamlined product information that blockchain technology provides will enable for new, exciting possibilities for both brands, retailers and customers. Indeed, this specific analysis has highlighted how blockchain technology appears to be particularly suitable for supply chain management solutions and brand protection, such as the Made in Italy, as it solves historical issues which typically arise in traceability processes currently in use. We identified some scenarios which could be addressed with a series of actions in order to favour the progressive growth and participation of interested players, focusing on a more solid governance that should guarantee the model's long-term sustainability. As a side, yet important note, we discovered that metrics such as sustainability, transparency and a general attention to social and environmental elements are getting more and more scrutinized by the public, and blockchain technology can help brands to address these requests. The study carried out is then worthy of further exploration, by means and the help of a continuous dialogue with the interested parties, detailing the requirements for the engineering of a system that ensures full project feasibility and viability. As developers and novices of Blockchain, instead, along with the benefits and excitement of looking into cutting-edge technologies, we found some clear challenges. For instance, building a demo architecture in an open source development environment that's evolving so rapidly, and sometimes faster than its documentation, can lead to moments of frustration and a developer will find himself to repeat the learning process continuously. The Solidity language, while being fairly easy to understand, is also quite new and the differences between compiler versions can still be significant, leading to time-wasting headaches. Nevertheless, Hyperledger Besu has proved to be a good choice in hindsight, as it can be used with widely appreciated tools, such as Remix and Metamask, that make development easier and smoother, standing the test of time regardless of the number of platforms being developed out there. And that's maybe another confusing aspect of this world right now: the number of platforms, ideas and alternatives one can choose from is growing rapidly, yet there's no real commitment for a standardization process in the enterprise area, partially because this is still a race between big IT corporations, for both innovations and clients' loyalty. That is, distributed ledgers are a significant investment in both time and money, and not something a corporation can step back from very easily, years after deployment. Therefore, arriving first matters a lot, which leads to the debatable implementations discussed at the beginning, often results of a rushed analysis process. Fortunately this trend is shifting, although slowly, and the project led by IBM with MISE to protect the Made in Italy seems to be a good example of this; of course, there was no real software evaluation phase (they proposed the IBM Cloud Blockchain Platform) but we think they identified the right requirements and use cases, thanks to the collaborative effort with the interested brands. This variety of platforms has obvious pros and cons, but in the wider scope of the so-called Internet of Value that we've mentioned, it makes harder to implement any plan for interoperability between ledgers, which is a very hot topic in the Blockchain scientific community.

At the start of this work, we stated that the goal was to build a system that would tackle the widespread problem of counterfeits in the fashion industry, not by making reproduction of fakes impossible but by giving brands and customer the ability to distinguish them as easily as possible. Although the supply chain scenarios were simplified, we managed to find a solution with the combination of few technologies, such as RFID tags, IoT sensors and distributed ledgers, that we believe can be extended to a more complex, feature-rich case, with a reasonable effort in the analysis of the processes. We faced and discussed the options we had in the architectural design, simulating what a PoC proposal would have to go through to present a potential solution. The demos from Chapter 5 worked as a tool for us to study the Besu platform and to confirm that it was indeed a good fit for our requirements; thanks to its privacy features, the general ease-of-use and the compliancy with Ethereum standards, we managed to configure a use case closer to reality, with a good number of nodes online. The business logic shaped in Solidity has proven to be a successful tool to automate common operations and to address few subtle privacy concerns as well, while the open source community provided us useful docker images to deploy a set of monitoring applications. Overall, the system feels like a solid start point for further developments that might lead to deploy an actual network in a test environment under the Ethereum umbrella. For our specific business case study, we have seen that the Trust Score is a good tool that can boost competition among parties (suppliers, carriers, brokers), which helps the brand company economically by getting better deals and services, but it also indirectly helps the customer who will buy something that is certified by a control agency for its quality and provenance. However, an argument could be made as to why suppliers, brokers, carriers or any other actor in this blockchain would want to invest money on something that has no return of investment except for the brand owners themselves; as a matter of fact, it could easily worsen their business by exposing their weaknesses to the competition. We've seen in the introduction chapters that similar attempts had already been tried and they failed for this very reason, not to mention the low level of trust among the competitors. This is probably the hardest part to figure out when modelling a Consortium Blockchain: how to get competitors on board, make them trust the system in a private/permissioned environment and justify the expenses. This last point is particularly significant, and it involves the concept of reward. We previously stated that in this architecture, reward is not necessary as the brand owners do not need an incentive to keep the network secure since their own business depends on it. That statement is still valid, but not entirely true, because brand owners do enjoy a particular kind of reward: traceability and transparency in their processes, both of which can play a significant role in business performance, as we'll explain soon. It would only be fair, then, if something was rewarded to the other participants as well. In a PoW network, nodes are rewarded with coins, which is not a path we can take in our case; however, not all rewards have to be monetary. For example, we could reward with trust, reputation and visibility, at the cost of all of them if any participant decides to play unfairly. While this already applies to brand owners, it does not apply to participants that are not in the selected group of authorities. Practically speaking, a good evolution to this architecture would be to enable the system to shuffle the set of authorities who sign the blocks, taking from a pool of potential new authorities with high Trust Score, so that they can get their own opportunity for visibility. That is, we would need to implement a way to move nodes up the hierarchy, from a standard peer to an authoritative node. This way, they could sign blocks and play an active role in the validation process, rather than passive, not only to the network's participants but to the attention of the customers themselves. Such argument could be the foundation of future research; as of now, this idea is being explored by other platforms such as GoChain [31], that runs on a proof-of-reputation consensus algorithm. They believe that proof-of-reputation is the key to open enterprise chains in the public space, as it works as an additional layer of access control over the standard proof-of-authority, whereas Besu took a different path to challenge this problem. Similar research work is in progress by TrustChain and their proof-of-trust consensus algorithm [32], based on the TrustToken and currently still a PoC.

In conclusion, we need to realize that the transparency we're aiming for might also be an obstacle for its adoption by the very same worldwide fashion brands that can decree the success of the initiative, so we're hoping for a long-term game that hopefully ends at the community's advantage. Everyone knows that our luxurious items are often crafted by underpaid workers, and that raw materials come from third-world economies where working conditions are unacceptable; truth is, we pretend we don't know. The core idea, however utopic it might sound, is to give visibility to this ignored reality, and to gift "greener" brands a strategic advantage over "not-so-green" competitors. Most likely, this will help smaller companies to get a brighter spot on the stage, as statistically speaking they'll feature a higher Trust Score. The scenario we imagine is the following: at one point, a customer will have to decide between two luxurious set of shoes, one of which is certified to be crafted with high quality materials in a sustainable process, while the others is not certified at all. Here's where the customers can play their part in the success of this initiative, and while there's no guarantee that they will buy the first set of shoes, there's also no denying that the transparency index can play an important factor in the decision. Customers, then, would scan the item and see a timestamped list of information: the provenance of the materials, who bought it, where it was shipped, assembled, stored and distributed; plus, they would be able to check the score for each and every one of the players involved in the manufacturing process, including downloadable archives of the control agency's reports throughout the history of those participants in the ledger, as well as the owner of the nodes who signed the blocks to which that verified information belongs to. In other words, the best way to sell this idea is to think the Trust Score not only as a metric of social importance, but as a marketing asset that contributes to add value to the brand. If customers' behaviour turns as expected, the bigger companies will have no choice but to seriously consider investing in the same asset, taking part in the consortium to leverage on its ability to share truth through distributed data.

References

[1] **Estimating the global economic and social impacts of counterfeiting and piracy** by Jeffrey Hardy for World Trademark Review *(Last visit: 11-02-2019) (https://www.worldtrademarkreview.com/anti-counterfeiting/estimating-global-economic-and-social-impacts-counterfeiting-and-piracy)*

[2] **Supply Chains in the Clothing Industry - A House of Cards?! (2014)** by Philipp Mettler, Makiko Ashida for J. Safra Sarasin Sustainable Investment Research *(Last visit: 11-02-2019) (https://www.eticanews.it/wp-content/uploads/2014/09/Report-Bank-J-Safra-Sarasin-Supply-Chains-in-the-Clothing-Industry.pdf)*

[3] **Fashion Transparency Index 2019** *(Last visit: 11-02-2019) (https://issuu.com/fashionrevolution/docs/fashion_transparency_index_2019?e=25766662/69342298)*

[4] **Mapped: Where Are the World's Most Sustainable Companies?** by Iman Gosh *(Last visit: 15-03-2020) (https://www.visualcapitalist.com/most-sustainable-companies/)*

[5] **Lanieri Fashion Tech Insights 2018** *(Last visit: 11-02-2019) (https://www.lanieri.com/blog/wp-content/uploads/2018/07/Lanieri-FashionTech-Insights-2018.pdf)*

[6] **How to Time-Stamp a Digital Document (1991)** by Stuart Haber, W. Scott Stornetta *(Last visit: 11-01-2019) (https://www.anf.es/pdf/Haber_Stornetta.pdf)*

[7] **Bitcoin: A Peer-to-Peer Electronic Cash System** by Satoshi Nakamoto *(Last visit: 10-01-2019) (https://bitcoin.org/bitcoin.pdf)*

[8] **Worldwide Blockchain Spending Forecast** by Michael Shirer, Jessica Goepfert, Stacey Soohoo for International Data Corp *(Last visit: 29-01-2019) (https://www.idc.com/getdoc.jsp?containerId=prUS44898819)*

[9] **Blockchain in Business**, PwC (Last visit: 14-02-2019) *(https://www.pwc.com/gx/en/issues/blockchain/blockchain-in-business.html)*

[10] **Continuous interconnected supply chain, Using Blockchain & Internet-of-Things in supply chain traceability** *(Last visit: 03-03-2019) (https://www2.deloitte.com/content/dam/Deloitte/lu/Documents/technology/lu-blockchain-internet-things-supply-chain-traceability.pdf)*

[11] **Blockchain 50** by Michael del Castillo, Matt Schifrin for Forbes *(Last visit: 10-08-2019) (https://forbes.it/2019/04/17/forbes-blockchain-50-le-aziende-leader-della-nuova-rivoluzione-digitale/)*

[12] **Capricoin Whitepaper** *(Last visit: 04-09-2019) (https://capricoin.org/whitepaper)*

[13] **Cryptocurrencies and blockchain, Legal context and implications for financial crime, money laundering and tax evasion (2018)** by Prof. Dr. Robby Houben, Alexander Snyers for the European Parliament *(Last visit: 08-05-2019) (http://www.europarl.europa.eu/cmsdata/150761/TAX3%20Study%20on%20cryptocurrencies%20and%20blockchain.pdf)*

[14] **On Public and Private Blockchains** by Vitalik Buterin for the Ethereum Blog *(Last visit: 23-03-2020) (https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/)*

[15] **World Food Programme Building Blocks** *(Last visit: 02-04-2019) (https://innovation.wfp.org/project/building-blocks)*

[16] **Blockchain as a Privacy Enabler: An Odometer Fraud Prevention System** by Mathieu Chanson, Elgar Fleisch, Andreas Bogner, Felix Wortmann *(Last visit: 02-04-2019) (https://cocoa.ethz.ch/media/documents/2017/08/None_UbiComp_2017-Privacy_Poster_final_1.pdf)*

[17] **Arianee Whitepaper** *(Last visit: 27-11-2019) (https://static1.squarespace.com/static/5c489ebf7c9327393b1ab84b/t/5c87f0efe5e5f0d5e48f6ac4/1552412941224/Arianee_White_Paper.pdf)*

[18] **Another day, another consensus algorithm. Why IBFT 2.0?** by Gina Rubino for Pegasys.Tech *(Last visit: 30-01-2020) (https://pegasys.tech/another-day-another-consensus-algorithm-why-ibft-2-0/)*

[19] **Enterprise Ethereum Alliance (EEA) Technical Documents** *(Last visit: 05-01-2020) (https://entethalliance.org/technical-documents/)*

[20] **Hyperledger Besu Documentation** *(Last visit: 23-03-2020) (https://besu.hyperledger.org/en/stable/)*

[21] **Bitcoin and Blockchain Technology course slides (University of Milano-Bicocca)** by Ferdinando Maria Ametrano *(Last visit: 02-07-2019) (https://www.ametrano.net/bbt/)*

[22] **Blockchain and Ethereum Training: Hyperledger Besu Essentials** by ConsenSys Academy *(Last visit: 23-01-2020) (https://learn.consensys.net)*

[23] **Blockchain Consensus Encyclopedia** *(Last visit: 19-12-2019) (https://tokens-economy.gitbook.io/consensus/)*

[24] **Blockchain for Business - An Introduction to Hyperledger Technologies** by LinuxFoundationX for EdX Course *(Last visit: 02-10-2019) (http://www.edx.org/course/introduction-to-hyperledger-blockchain-technologie)*

[25] **Blockchain: Understanding Its Uses and Implications** by LinuxFoundationX for EdX Course *(Last visit: 02-10-2019) (https://www.edx.org/course/blockchain-understanding-its-uses-and-implications)*

[26] **Hands-on Blockchain with Hyperledger (2018)** by Nitin Gaur, Luc Desrosiers, Venkatraman Ramakrishna, Petr Novotny, Dr. Salman A. Baset, Anthony O'Dowd

[27] **Enterprise Blockchain Solution Lifecycle** by The Blockchain Training Alliance *(Last visit: 20-07-2019) (https://blockchaintrainingalliance.com/blogs/news/enterprise-blockchain-solution-lifecycle)*

[28] **Mastering Ethereum** by Andreas M. Antonopoulos, Gavin Wood *(Last visit: 25-02-2020) (https://github.com/ethereumbook/ethereumbook)*

[29] **Ethereum white paper** *(Last visit: 05-01-2020) (https://github.com/ethereum/wiki/wiki/White-Paper)*

[30] **Hyperledger Besu proposal** *(Last visit: 21-09-2019) (https://wiki.hyperledger.org/display/HYP/Hyperledger+Besu+Proposal)*

[31] **GoChain, What is Proof of Reputation?** *(Last visit: 23-03-2020) (https://gochain.io/proof-of-reputation/)*

[32] **The TrustChain Initiative** *(Last visit: 23-03-2020) (https://www.trustchainjewelry.com/)*

[33] **How the blockchain could transform sustainability reporting**, by Russ Stoddard for GreenBiz *(Last visit: 23-03-2020) (https://www.greenbiz.com/article/how-blockchain-could-transform-sustainability-reporting)*