POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Informatica
Data Science LM-32

Tesi di Laurea Magistrale

INCENTIVE-COMPATIBLE & PRIVACY-PRESERVING DATA ANALYTICS SYSTEM

Enabled by Blockchain and Multiparty Computation

Relatore:
Prof. Danilo Bazzanella

Candidato:
Andrea Di Nenno
Mat. 234781

Correlatore
Dr. Guglielmo Morgari

Anno Accademico 2018-2019

this is my Robin Hood theory..

- Keith Edward Elam -

Abstract

Blockchain, the technology at the foundation of Bitcoin and other cryptocurrencies, is being praised as a major disruptive innovation with the potential to transform most industries. Yet, considering its base architecture, it lacks some fundamental properties that are preventing a larger scale adoption. Among the others, being the blockchain a public and shared ledger, it doesn't offer privacy of data, precluding its usage to many applications that handle sensitive pieces of information. In this work, the blockchain is integrated with Secure Multi-Party Computation (MPC), a cryptographic tool that allows a number of parties to jointly compute a function over their inputs, keeping them private and guaranteeing that the output (if returned) is correct. Although the two might look apparently distant, they actually address the same set of problems, the ones where mutually mistrusting parties are involved without a trusted third party. Besides they cover different aspects of such problems, thus being considered complementary technologies. Blockchain (by means of Smart Contracts) shapes the MPC system to be incentive compatible, and just as Bitcoin's miners that perform computational work to create valid new blocks in exchange of cryptocurrency, here computing parties run arbitrary functions according to the MPC security properties for the same reason. It also provides a single and incorruptible source of truth of the entire system and, based only upon that, it enforces the contract conditions like an automated trusted third party. The resulting decentralized application (dApp) is implemented over a use-case belonging to the medical domain, in a scenario where patients earn cryptocurrency after having provided their sensitive data under an MPC scheme, while larger groups (i.e. research organizations) obtain aggregate data from arbitrary functions executed by the MPC system. Hopefully, the union between these two technologies could pave the way for a spectrum of new decentralized and incentive compatible applications, for many use-cases never explored before.

Contents

List of Figures

List of Tables

Chapter 1

State of the art

1.1 Blockchain

Blockchain, the technology at the foundation of Bitcoin and other cryptocurrencies, is being praised as a major disruptive innovation with the potential to transform most industries [50]. The motivations behind such enthusiasm are pretty straightforward: they have allowed like never before mutually mistrusting entities to perform financial payments without relying on a central trusted third party while offering a transparent and integrity protected data storage. Since then, blockchain as a technology has gained much attention beyond the purpose of financial transactions: distributed cloud storage, smart property, Internet of Things, supply chain management, healthcare, ownership and royalty distribution, and decentralized autonomous organizations just to name a few.
Even though young, the economic scale this phenomenon has reached cannot be ignored: the total global market capital of blockchain based tokens and cryptocurrencies in late 2018 floats around $130B [15], while having reached $800B in its highest peak so far, occurred in January 2018 [43]. Clearly we are at a crucial point in the evolution of blockchains. The major barrier to their widespread adoption might be represented by the so called *Blockchain Trilemma* [11], which states that blockchain systems can satisfy at most two properties between decentralization, scalability and security. As it will be examined deeper, these properties are fundamentally related to the consensus protocol, the core component of the blockchain, that inevitably has become the real game changer for future developments.

1.1.1 Web3: the new Internet revolution

Over the last ten years internet-based services have been offered in a purely centralized fashion. As the web matured over the years, the internet relied more and more on few companies, making it easy for them to know the information we search for, store our personal data, manage our digital identities and our public and private communications. Google built the fastest search engine, and has been rewarded with control over 74% of all search traffic [32]. Facebook built the most popular social network, and now controls the online identities of 2.2 billion people [26].
In the meantime, a bunch of seemingly unrelated technologies are being developed on the fringes of the tech industry. These range from financial projects (cryptocurrencies), to basic communications technology (end to end encrypted messaging), to mass consumer use-cases (open social networks and p2p markets), to critical internet infrastructure (decentralized DNS) [22]. In common they have the base vision: to create a new, "better" internet. An internet where payments and money are natively digital, where decentralized applications run against centralized ones, and where users have real control over their data and identity.
Web3 is led much more by an ideological motivation than a technological one: it isn't about speed or performance, given that many Web3 applications are, at least today, slower and less convenient than existing products. Instead, it is radically about breaking the schemes that has been shaping the web lately, which is the trade-off between convenience and control. We have become so used to it, that is seems inevitable to be surveilled while using the internet, or to have our personal data controlled by the service providers. How could it be other way? Finally we could: we can have the benefits of the internet without handing the majority of power to a minority of companies.

Digital Money While in the past the internet needed to rely on an offline financial system, with cryptocurrencies in Web3 sending a payment does not require interacting with some offline system, it only requires sending a message over the internet. As an immediate implication, sending or receiving payments will become something that any software could do, and by extension, something that could be done by any person just with an internet connection and a phone.
Costs of transactions (also for micro payments) will likely be lowered, new business models, before impractical, will be unlocked and available to massive new markets (with people that weren't able to enter the traditional financial system); finally new kinds of money will be created and used to trade, such as *non-fungible digital assets* [53]

Decentralized services The core rules that govern decentralized services will be defined in an open-source protocol. Users will interact with that protocol via a client-software of their choice. In other words, there could be a variety of apps, all made by different developers, but all connected to the same social network. These clients may offer different features from one another, but all conform to the same shared protocol, analogous to the way that email clients all use the same standard for sending and receiving emails. This in few words will make a sustainable ecosystems of third party services grow. App developers can build useful products on top of a decentralized protocol without fear that someday their API access will be turned off, because there is no one company who can turn it off. The platform would remain neutral, meaning that a larger network of developers will invest their time and money into building businesses on top of it.

User control of data Web3 is as well laying the groundwork for personal control of online identities, other than more generally of their app data. Similarly to a wallet managing private keys for cryptocurrencies, any kind of blockchain-based data could be stored securely in a personal wallet, including one's ID.
Being the users able to use their own identity, instead of one provided by a third party, the possibility for such identity providers to capture user data inside other applications will be nearly eliminated. Furthermore, in decentralized services ideally there won't be no central company being able to collect user data.
Finally and more importantly, the described control gained by users over their data, will make it easier to earn out of them. In Web3, users will be able to sell their app data, and get paid entirely. The power that turned social media companies into billion dollar businesses will shift into user's hand, that will actually own a piece of the technology they use every day.

1.1.2 Definition

The blockchain is a decentralized, replicated and tamper-proof log: data on the blockchain (public ones) cannot be deleted, and anyone can read data from it and verify its correctness.
It is usually implemented as a linked list of blocks of data, in which the pointer to the previous block is simply the cryptographic hash of it, that serves as integrity code as well. This pattern resolves into a hash chain, where each block implicitly verifies the entire chain before it, and so tampering with already written data gets necessarily detected.

A block can be seen as a set of transactions, that in turn specify some transformation on the state of the blockchain. The state for instance can include information like account balances, reputations, data pertaining to information of the physical world; generally, anything that can currently be represented by a computer. Transactions then represent a *valid* arc between two states. An invalid transaction is for example reducing an account balance without an equal increase of another one.
Blockchains can be categorized in permissioned (private) and permissionless (public) ones. The latter are fully decentralized, meaning that anyone can setup a node and join the network. In permissioned blockchains instead all the node identities are known and are controlled usually by a single authority.
Either way, nodes in the network participate in a collaborative protocol, the consensus protocol, where they agree on the next block to append to the blockchain, and so validating all the state changes in it. The consensus protocol acts essentially as the engine of the blockchain, and so the performances rely upon its efficiency. Indeed, as more extensively explained in the following section, radically different protocols have already been formalized, all of them designed to overcome the Bitcoin's Proof of Work limitations.

1.1.3 Consensus Protocols

Consensus protocols, or *atomic broadcast*, in distributed systems have been introduced because of the need to provide resilience against failures across multiple nodes holding replicas of databases [38]. In this configuration, nodes must receive the same set of messages in the same order. The broadcast is named *atomic* because it either eventually completes correctly at all participants, or all participants abort without side effects. Moreover, that participants recovering from failure, must be able to learn and comply with the agreed order.
Speaking about the properties a consensus protocol should follow in order to consider the blockchain implementing it stable, Bonneau [9] in his thorough and extensive analysis outlines five basic stability properties:

- **Eventual consensus:** at any time, all compliant nodes will agree upon a prefix of what will become the eventual valid blockchain. This won't imply though that the longest chain at any moment is a prefix of the eventual blockchain, given that blocks might as well be discarded in the case of temporary forks;
- **Exponential convergence:** the probability of a fork of depth n should have a complexity of $O(2^{-n})$. This gives high confidence that a

simple "k confirmations" rule will ensure that a transaction is permanently included in a block;

- **Liveness:** new valid transactions will be included in blocks and blocks will be added to the blockchain within a reasonable amount of time;
- **Correctness:** all blocks in the longest chain will only include valid transactions;
- **Fairness:** a miner with a proportion α of the total computational power will mine a proportion $\sim \alpha$ of blocks (assuming they choose valid blocks to mine)

As Bonneau underlines, the blockchain is considered stable if holding correctly such properties, but on the other hand it is still unclear whether they are all necessarily required. For example the Fairness property may be considered irrelevant from a user's perspective, but its absence would lead many miners to cease their participation, threatening the other properties.
Clearly we have that, in order to satisfy the cited conditions, the participants must effectively agree on the order of receipt of the messages. A number of extensions to consensus protocols includes a validation step, that ensures the transactions accepted are necessarily valid. This implies that the validation rules must be deterministic and uniformed across all nodes.
In the following, the two most relevant consensus protocol so far will be described.

Proof of Work

In 2008, Bitcoin [50] was announced and a white paper signed by a pseudonymous author Satoshi Nakamoto was posted to the Cypherpunks mailing list, along with the source code of the original client. Few months later, on January 3rd, 2009, the *genesis block* was mined [1]. The pivotal innovation was represented by its consensus protocol, a Proof of Work, later renamed *Nakamoto Consensus*, that assured to reach a consensus in a permissionless, fully decentralized network.
Yet the singular components forming his consensus model were studied and formalised long before. Indeed the idea of PoW was introduced first in 1993, by Dwork and Naor, as a technique to prevent spam mail: the sender needed to find out the solution to a mathematical puzzle as a prove that some computation was performed [21]. Nakamoto though took the main inspiration

[1] Ironically it contains the string "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks."

from a later work, proposed in 1997 by Back and again for fighting spam, under the name of Hashcash [3]. Here, the computational puzzle is finding a SHA-1 hash of an header containing the email recipient's address and the current date, containing at least 20 bits of leading zeros.
Exploiting the hashing algorithm's preimage resistance [59], the puzzle can be solved only by including random nonces in the header, until the resulting hash meets the formal constraint. Given the relevant amount of computational work required by these guesses, a valid hash is considered to be a PoW. As one can guess, the work must be moderately difficult for the miner to perform, but easy for the network to check.
Nakamoto consensus refined the Hashcash on two features. first, the single SHA-1 hashing is replaced with two subsequent SHA-2 hashes. Secondly, an hash is considered valid when its value is below a certain threshold t. This innovation was crucial, because it made the Hashcash dynamic: just by decreasing t, the number of guesses, and so the computation, increases.
Nodes in the blockchain that generate hashes are called *miners*, and the whole process is referred to as *mining*. In Bitcoin, miners compute hashes of potential blocks of transactions to be appended to the blockchain, and are rewarded with just minted coins if they are the first to produce a valid block. The t value is calibrated automatically by the network, in order to keep the inter-block interval approximately around 10 minutes.

Proof of Stake

One of the main criticism raised against the Bitcoin is that it relies on the PoW that, as we have seen, essentially has no external utility (it is only needed to select the winning miner), thus leading to wasteful computation, other than being prone to centralization (in section 1.1.4, this and other aspects will be treated). Maybe because of the market opportunity many may have seen in producing a much more efficient consensus protocol, a new class of consensus protocols is born.
Proof of Stake as of today stands as the most bright PoW substitute. Instead of voting with their CPU power, as in PoW, here participants vote for new blocks according to some weight, like the amount of currency held. A common pattern between the different implementations of the PoS is to randomly elect a leader among the stakeholders, to append a block to the blockchain. It's in fact based upon user's wealth, or stake, that they get selected.
In order to validate transactions and create blocks, a forger (miner in PoS) first puts his own coins at *stake*: if they validate an invalid transaction, they lose their holdings, as well as their rights to participate as a forger in the

future. Once the forger puts its amount at stake, he is eligible in the forging (mining) process, being in theory now incentivized to validate the right transactions.
The election may be public, that is the leader is known to all the participants, or private, where each player check whether he's the chosen one or not, and after having all the other players able to verify it through public information. This latter approach is resilient to DoS attacks, since players get to know the leader only when he appended the block, at which point it is too late to DoS or generally corrupt him.
Different big players in the industry are prone to adopt this kind of consensus scheme. The Ethereum Foundation has been considering switching from PoW to PoS since 2015, with the so called Casper protocol [12]. The Algorand blockchain, instead, has been natively designed on a PoS defined by them "super fast and partition resilient" [47]. However, as the reader can acknowledge in the following section, many concerns still are alive around the PoS, because few but potentially detrimental attacks have been already theorized.

Attacks and Mitigation As pointed out already in 2014 by Vitalik Buterin [10] three new attacks have emerged against the new consensus scheme. The first one is called *Nothing at stake attack* [27], and is a consequence of the fact that in the PoS it's computationally cheap to extend a chain. Thus miners are incentivized at extending all possible chains so to maximize the probability to be in the winning block and so rewarded. If this happened, an attacker would be able to send a transaction in exchange for some digital good, receive the good, then start a fork of the blockchain from one block behind the transaction and send the money to himself instead. Even with 1% of the total stake the attacker's fork would win because everyone else would be mining on both. A penalty mechanism may be good enough to prevent such scenario: a miner producing blocks on different forks is penalized by loosing part of its stake.
Another one is called *Stake grindin attack*, and require a very little amount of currency. The attacker here goes through the history of the blockchain in order to find blocks where his stake would have won. From that block on, he modifies the next block header until his stake wins once again, until his chain is the longest one. This attack requires a little bit of computation, but isn't impractical. The defence mechanism is straightforward: by means of an unbiased source of randomness contributing to the leader election, an attacker wouldn't be able to choose himself.
The third attack is the *Long range attack*, where an attacker can bribe forg-

ers to sell him their private keys. If these keys had considerable value in the past, then the adversary can mine previous blocks and re-write the entire history of the blockchain. This is technically possible because in the PoS based blockchains, as already said, it is computationally irrelevant to mine a block. The defense mechanism here is to consider only blocks with a certain range of prior blocks to be disputable, while the rest are ultimately considered of the main chain. Unfortunately though, as we can read from the Ethereum's Proof of Stake FAQ [28], this protocol would lead to something called *Weak Subjectivity*.

Weak Subjectivity In Proof of Stake consensus, in order to be secure against long range attacks, a *revert limit* - a rule that nodes must simply refuse to change blocks that are prior to some time limit - must be introduced. Though this causes the security model to change, with undefined behaviors when a node comes online either for the first time or after a considerable time span, because necessarily he will have to ask a trusted source what the hash of the valid chain is.
Even though it completely undermines the trustless nature of blockchains, as Vitalik writes, this is a problem only in the very limited case where a majority of previous stakeholders from some point in time collude to attack the network and create an alternate chain; most of the time we expect there will only be one canonical chain to choose from.

Strengths If implemented correctly, the proof of stake presents three main advantages over the PoW:

1. It doesn't waste any significant amount of electricity. Considering that as of November 2018 only 38 countries worldwide consume more energy than the Bitcoin's mining process [5], this is a significant advantage;

2. It arguably provides an higher level of security, since to conduct a 51% attack, here one must own the 51% of the entire supply of the currency.

3. It possibly allows for much faster blockchains (i.e. NXT has one block every few seconds).

Alternatives Bonneau [9] describe informal consensus protocols based on PoS that have been proposed in the crypto community. As common denominator, these systems require miners to hold or prove the ownership of coins.

- **Proof of deposit** where miners lock a certain amount of coins, which they cannot spend for the duration of their mining. This approach is used in Tendermint [39], where a miner's voting power is proportional to the amount of coins he has locked;

- **Proof of burn** like in Slimcoin [55] miner's voting power is proportional to the amount of coins they have destroyed, by sending them to a verifiably unspendable address.

- **Proof of coin age** where miner's voting power is deduced from the amount of coin weighted by their *age*, that is the time since they were last moved. Peercoin uses this approach [58].

1.1.4 Bitcoin

Ideology *A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution.* This is how the Bitcoin, through its whitepaper [50], was introduced in 2008 by Satoshi Nakamoto.
The main criticism pointed out against the current economical model was that alone it wouldn't be able to survive in a world with no trust. That couldn't be more true. The world is indeed characterized by a lack of trust, and that's why it relies exclusively on financial institutions serving as trusted third parties to process electronic payments.
But everything comes at a price: the security offered by the trusted third party resolved into higher transaction costs, impossibility to do small casual transactions, and finally the possibility for a transaction to be reverted. With this possibility always threatening sellers, they are forced to be wary of their customers, requiring from them lot more information that the ones needed for the selling. When brought to exaggeration, this process also known as *KYC* has led customer to claim back their privacy, like a dog chasing his own tail.
At the end of the introduction, he put the crystal clear solution on the table: *we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions.*

Transactions The state of the world in Bitcoin is represented by a series of messages called transactions. More precisely, transactions are the only thing that exists in Bitcoin. All further higher-level concepts, such as user, account, balances or identities exist as long as they can be inferred from the

transaction list, and as such they are not built-in features. A transaction is a transfer of Bitcoin value that is broadcast to the network and collected into blocks. A transaction is made of an array of inputs and another one of outputs, and the ID is derived from its hash fingerprint.

Basic example The input in transaction reported in Figure 1.1 imports 50 BTC from output 0 in the previous transaction (f5d8...). Then the output sends 50 BTC (50.000.000 Satoshi) to a Bitcoin address (here in hexadecimal 4043...). When the recipient wants to spend this money, he will reference output 0 of this transaction in an input of his own transaction [7].

```
Input:
Previous tx: f5d8ee39a430901c91a5917b9f2dc19d6d1a0e9cea205b009ca73dd04470b9a6
Index: 0
scriptSig: 304502206e21798a42fae0e854281abd38bacd1aeed3ee3738d9e1446618c4571d10
90db022100e2ac980643b0b82c0e88ffdfec6b64e3e6ba35e7ba5fdd7d5d6cc8d25c6b241501

Output:
Value: 5000000000
scriptPubKey: OP_DUP OP_HASH160 404371705fa9bd789a2fcd52d2c580b65d35549d
OP_EQUALVERIFY OP_CHECKSIG
```

Figure 1.1: A simple Bitcoin transaction

Input An input is a reference to an output from the previous transaction in the chain. Since many inputs can be grouped in a single transaction, all of them are added up, and the total (less any transaction fee) is the amount of coin available to use by the outputs of this transaction. *Previous tx* is a hash of a previous transaction. *Index* is the specific output in the referenced transaction. *ScriptSig* is the first half of a script, that will be explained in a specific paragraph here below.

Output An output contains instructions for sending bitcoin. *Value* is the number of Satoshi (1 BTC = 100,000,000 Satoshi) that this output will be worth when claimed in another transaction. *ScriptPubKey* is the second half of a script (explained in the next paragraph). Nothing denies to create multiple outputs, as long as their combined value doesn't exceed the input's one.
Given this nature, that is an output from one transaction can only ever be referenced once by an input of a subsequent transaction, the entire combined input value needs to be sent in an output if you don't want to lose it. If the input is worth 50 BTC but you only want to send 25 BTC, Bitcoin will create two outputs worth 25 BTC: one to the destination, and one back to

you (known as "change", though you send it to yourself). Any input not redeemed in an output is considered a transaction fee; whoever mines the block can claim it by inserting it into the coinbase transaction of that block.

Script The script contains two components, one in the input and one in the output, that is a signature and a public key. The public key is used to verify the redeemer's signature, which is in turn an ECDSA signature over a hash of a simplified version of the transaction. This is needed to prove that the transaction was created by the real owner of the address in question. Various flags define how the transaction is simplified and can be used to create different types of payment.

Consensus Mechanism Bitcoin uses a Proof of Work consensus mechanism, specifically renamed Nakamoto Consensus. For more details about it please refer to section 1.1.3

Block confirmation The intrinsic nature of the Bitcoin's consensus mechanism entails that before being sure that a transaction is permanently included in the blockchain, one is supposed to wait for some amount of time, that is blocks appended afterwards. This is because, during a fork for example, users may see their transaction appended in one chain and not in the other. Anyway, considering the majority of miners following the protocol, users can infer that a transaction is going to increasingly likely end up on the longest chain as more confirmations are sent over. Practically, in Bitcoin a shared though not proven assumption is to wait for 6 confirmations before accepting the transaction.

Mining rewards and fees The size of the block reward is determined by a fixed schedule. Initially, each block created 50BTC, but it is scheduled to halve roughly every four years until roughly 2140 at which point no new bitcoins will be created. This implies that, without any other incentive scheme, the system would gradually get abandoned with time. Because of this, miners do not only profit from block rewards: they are also allowed to claim the difference in value between all input and all output transactions in the block they mine.
Bitcoin's design makes it easy and efficient for the spender to specify how much fee to pay, whereas it would be harder and less efficient for the recipient to do so. The spender is then almost always solely responsible for paying all necessary Bitcoin transaction fees, and it's not an irrelevant parameter to set: the so called *fee-rate* is perhaps the most important factor affecting how

fast a transaction gets confirmed. It is in fact pretty straightforward that a miner would be more attracted to mine transactions which make him earn more.

1.1.5 Ethereum

Motivations Satoshi Nakamoto's development of Bitcoin in 2009 has often been hailed as a radical development in money and currency. That was though just a spark that made an entire new world of researches and projects explodes, because, apart from the Bitcoin use-case, the underlying blockchain technology as a tools for distributed consensus was rapidly gaining attention. Without any doubt Ethereum represents the second milestone in the blockchain's still short history. The futuristic vision that Vitalik Buterin, Ethereum's mastermind, had in mind, was about a blockchain with a built-in, Turing-complete programming language. With such programming language, anyone could create arbitrary code (defined as Smart Contract, section 1.1.5) and decentralized applications, implementing arbitrary rules for ownership, transaction formats and state transition functions, simply by writing up the logic in a few lines of code.

Architecture The Ethereum blockchain is quite similar to the Bitcoin's one, with few yet significant differences. As long as mining is concerned, Ethereum like Bitcoin uses a Proof Of Work approach, but actually the developing team is planning to migrate towards *Casper protocol* [12], which is going to implement a Proof of Stake consensus protocol.
On the other hand, Ethereum differs from Bitcoin mostly for the block structure. Indeed Ethereum blocks contain a copy of both the transaction list and the most recent state, whereas in Bitcoin blocks contain only a copy of the transaction list. This means that in Bitcoin, the entire blockchain history is stored in each block, while in Ethereum "only" the final state. Although this latter approach might seem inefficient, in fact, can be calculated to provide 5-20 times savings in space with respect to Bitcoin [24]. This because the state itself is represented in a tree structure, such that between two adjacent blocks only a small part of it needs to be changes. Therefore data can be stored once and referenced twice using pointers, like hashes of sub trees. The most important components forming the Ethereum's blockchain are the EVM (Ethereum Virtual Machine), Miner, Transactions, Consensus algorithm, Accounts, Smart contracts, Ether and Gas, and they will be expanded in the following sections.

Ethereum Virtual Machine

The Ethereum Virtual Machine or EVM is the runtime environment for Smart Contracts (section 1.1.5) in Ethereum. It is not only sandboxed but actually completely isolated, which means that code running inside the EVM has no access to network, filesystem or other processes. Smart contracts even have limited access to other smart contracts.
Usually the EVM is hosted by a miner node, as the execution for a smart contract transaction occurs when the mining node includes the transaction in a block it generates. Once the block is generated, it's distributed to the network, and any node receiving it then proceeds to run through the list of transactions, ensuring each transaction is valid (as well as running associated code).
When a new node connects to the network, it downloads every block in history and re-validates every transaction in each. The repeated validation is possible because smart contract transactions are deterministic. They may depend on factors such as the block number itself, current storage values for a contract, or the result of another smart contract's computation, but that information is constant and can be recomputed consistently by stepping through transactions from the start of the chain.

Storage, Memory and Stack

The EVM has three areas where it can store data, namely storage, memory and the stack. Each account has a data area called storage, which is persistent between function calls and transactions. Storage is a key-value store that maps 256-bit words to 256-bit words. A contract can neither read nor write to any storage apart from its own, and it is comparatively costly to read, and even more to modify storage.
The second data area is called memory, of which a contract obtains a freshly cleared instance for each message call. Memory is linear and can be addressed at byte level, but reads are limited to a width of 256 bits, while writes can be either 8 bits or 256 bits wide. Memory is more costly the larger it grows (it scales quadratically).
The EVM is not a register machine but a stack machine, so all computations are performed on a data area called the stack. It has a maximum size of 1024 elements and contains words of 256 bits. Of course it is possible to move stack elements to storage or memory in order to get deeper access to the stack, but it is not possible to just access arbitrary elements deeper in the stack without first removing the top of the stack.

Accounts

Accounts are main building block in the Ethereum ecosystem. It is the interaction between accounts that is stored as transaction in the ledger. An Ethereum account contains four fields:

- The **nonce**, a counter to make sure each transaction is processed only once.
- The account's **ether balance**
- The account's **contract code**, if present
- The account's **storage**, which is by default empty

In Ethereum there are two types of accounts: **externally owned accounts**, and **contract accounts**. The difference between the two is that the first comes out with no codes, so can be only used to create and sign transactions. Contract accounts are instead controlled by their code, such that it gets executed every time they receive a message, and through that it can read or write to its internal storage, other than regularly sending messages, transactions, or creating contracts in turn.
Contracts can be seen, citing the Ethereum whitepaper [24], as *autonomous agents, that live inside of the Ethereum execution environment, always executing a specific piece of code when poked by a message or transaction, and having direct control over their own ether balance and their own key/value store to keep track of persistent variables.*
The address of an external account is determined from the public key while the address of a contract is determined at the time the contract is created (it is derived from the creator address and the number of transactions sent from that address, the so-called nonce). Regardless of whether or not the account stores code, the two types are treated equally by the EVM.

Gas

Each operation (from a smart contract call to a simple transaction) is charged with a certain amount of gas, that is a unit that measures the amount of computational effort that it will take to execute such operation.
Each possible operation executable on the EVM is indeed associated with a specific and fixed Gas unit, as specified in the Ethereum Yellow Paper [76] (a few examples are also referenced in Section 2.3.1). This unity measure is used to calculate the amount of fees that the transaction creator has to pay (in Ether) up front when sending the transaction.

Since the Gas itself has not an intrinsic monetary value, the sender specifies a gasPrice, so that he will pay eventually gasPrice * Gas. The higher the gasPrice value he specifies, the sooner miners will mine his transaction. Other than that, the sender might as well specify the upper Gas limit, that is the maximum computational effort he is intended to pay for, but with a risk: if the Gas is used entirely but the transaction would have needed some more to be executed, an out-of-gas exception is triggered, which reverts all modifications made to the state in the current call frame, ending up with the sender losing what had payed. Conversely, if some Gas is left after the execution, it is refunded to the creator again as Gas (left) * gasPrice.
The rationale behind such solution is brilliant. First, it forms the incentivization scheme behind Ethereum. It is called Gas on purpose, as it is literally the Ethereum's fuel. Miners are incentivized to act honestly because they are paid proportionally to the computational effort they offer to the network. This couldn't be other way: being indeed a general purpose blockchain, the mining prize couldn't be specified beforehand and fixed as in Bitcoin, where instead the mining effort is deterministic.
Secondly, it constitutes a pretty strong security measure against denial of service attacks, because to conduct one, an attacker would have to literally pay for it.

Transaction

Ethereum, taken as a whole, can be viewed as a transaction-based state machine. It starts with a *genesis* state and incrementally execute transactions to shape it into a specific final state. The state can include such information as account balances, reputations, data pertaining to information of the physical world; anything that can be represented by a computer. Practically, the state is composed by *accounts*, objects having a 20-byte address, and a state transition is direct transfer of value and information between accounts. Furthermore, a transaction is always cryptographically signed by the sender (creator).
This makes straightforward to guard access to specific modifications of the database. In the example of the electronic currency, a simple check ensures that only the person holding the keys to the account can transfer money from it. There are three types of transaction that can be executed in Ethereum:

1. **Transfer of Ether from one account to another:** the accounts here can be both externally owned accounts or contract accounts, and the transfer may happen in all possible directions among this set.

2. **Deployment of Smart contract:** performed by an externally owned

account by means of a transaction in EVM.

3. **Invoking a function within a contract:** executing a function in a contract that changes state is considered as transactions in Ethereum. If executing a function does not change state (i.e. it just reads the blockchain), it does not require a transaction.

A transaction comes out with several different parameters set by the sender, the most important are:

- **from:** indicates the account that is originating the transaction, signing it, and sending some Gas or Ether along. Can be both externally owned or a contract account.
- **to:** account property refers to an account that is receiving Ether or data (when the receiver is a smart contract) from the transaction. In case of transaction related to deployment of contract, this field is empty.
- **value:** refers to the amount of ether that is transferred from one account to another.
- **input:** refers to the compiled contract bytecode and is used during contract deployment in EVM. Possibly, it is also used when invoking a smart contract function calls to supply its parameters.
- **gas:** refers to amount of gas supplied by sender for the miner node executing the transaction.
- **gasPrice:** refers to the price per gas the sender was willing to pay in Wei (a sub unit of Ether) to the miner node. The higher the gasPrice, the sooner the transaction is likely to be mined in the blockchain.

Smart contract

A smart contract is a contract implemented, deployed and executed within Ethereum environment. A traditional contract is a legal document that binds two or more parties who agrees to execute a transaction immediately or in future. There is a subtle difference though between the two: while a traditional contract technically may as well not be complied, because it provides the consequences of such possibility, a smart contract enforces automatically and inevitably the contract logic when the specified conditions are met.
As it will be explained in the Smart Contract's security section, this feature implies that when bad programmed, Smart Contracts could lead to potentially dangerous side-effects. Smart contracts can store data in their storage

memory, in order to record information, facts, associations, balances and any other information needed to implement logic for real world contracts.
From a programming point of view, the language to write them is called Solidity, and is an object-oriented, high-level language. It is statically typed, supports inheritance, libraries and complex user-defined types among other features. A smart contract can call another smart contract function, and as well as with objects, one can create an instance of the contract and invoke functions to view and update its state.

Deployment steps The first step is compilation of contract. The compilation is done using solidity compiler [68]. The compiler generates two files:

- **Bytecode** is what represents the contract and is deployed in Ethereum ecosystem. It is used to actually deploy the contract over the blockchain. To do so, a new transaction is created passing in the contract bytecode and appropriate quantity of gas for execution of transaction. After the transaction is mined, the contract is available at an address determined by Ethereum.

- **Abi** which stands for Application Binary Interface, is an interface consisting of all external and public function declarations along with their parameters and return types. A new instance of a deployed contract is created using the ABI definition and its address, and that allows any caller to invoke any contract function.

1.1.6 Security considerations

Blockchains are without any doubt a powerful technology. They allow for a large number of interactions to be codified and carried out in a way that greatly increases availability (as there are no single point of failures), removes business and political risks associated with the process being managed by a central entity, and reduces the need for trust. They create a platform on which applications from different companies and even of different types can run together, allowing for extremely efficient and seamless interaction, and leave an audit trail that anyone can check to make sure that everything is being processed correctly.
On the other hand, if a large scale adoption in the business still hasn't arrived after 10 years, it must lack necessarily in something. Two major issues are crystal clear around the blockchain, and are the ones related to scalability and to privacy. The two are going to be discussed in the next two sections.

Blockchain trilemma

Generally speaking about scalability for blockchains, throughout the community it is well-known the term *Blockchain Trilemma* [11], coined by Vitalik Buterin, which refers to the fact that with blockchain there is always a choice to be made that involves the trade-off between decentralization, security and scalability. Creating a blockchain platform that optimizes all three factors seems to be a really challenging task.

Decentralization Decentralization refers to the fact that the blockchain doesn't rely on a central point of control, but on a network of nodes and on a consensus protocol. It is important to highlight that decentralization is not a binary attribute, but always comes at some degree. A common misconception is in fact to think that all blockchains are decentralized to the same degree.
Of course, a centralized organization is controlled by a small group of individuals, i.e. the management, who typically owns the majority of ownership in the company, and as such detains the role of decision-maker. Decentralized networks are controlled in a certain degree by the users. They have the power to use their stakes to vote for some decision. Obviously, this doesn't prevent the fact that users have different influences with their vote: in proof-of-stake protocols, an individual's vote is as influential as the number of tokens they put at stake, while in proof-of-work the influence is proportional to the computational power of the user.
Another important element of decentralization is that the majority of value is accumulated inside the community. Lots of crypto projects are owned entirely by their users, rather than the founders. Precisely because there is no management, nobody is taking a cut of the value before all shareholders are compensated. In the music industry, for example, Apple with iTunes takes a 30% cut of sales fees, with the remaining 70% going to content creators. On a blockchain, content creators would likely get more than 90% of total sales, with a small portion going to nodes involved in running the network. A great point for decentralization then is for the greater security it typically offers, simply because it avoids having a single central point of failure.
Nevertheless, there are downsides to take care of when choosing a decentralised system. The most technical one is, as well known, a relevant slowdown in performances, because of the intrinsic nature of the currently used consensus protocols. Furthermore, the absence of a central moderator may lead disputes astray in a community, ending up with bad outcomes. In a social media use case, a plain blockchain would allow the publication of any kind of material, from fake news to hate speeches and so on. Lastly a decen-

tralised system is nearly impossible to shut down, and even if this is acclaimed as one of the major benefit, it cannot be considered an absolute advantage.

Security Implicitly, the security in a blockchain is directly proportional to its degree of decentralization. In most of the attacks discovered so far against a blockchain, the effort needed is relative to the number of nodes in the blockchain. For example in the *>50% Attack*, an entity (or set of them) should manage to own more than 50% of the total tokens (or computing power), to effectively own the network, or in the less effort demanding *Sybil Attack*, where an entity needs to forge as much identities as possible in the network to gain gradually more control over it.
Clearly, security doesn't imply downsides directly, but rather trade-offs. Security of a blockchain comes at a price of reduced throughput and increased network latency, and these factors constitutes a deterrent for many potential users, used to instantaneous transactions on centralized services. If we look to offer the best user experience possible, this is a strong concern.

Scalability Obtaining infinite scalability, defined as the upper limit on how large a network is able to grow, is a non-trivial problem for a blockchain. Consensus protocols on existing public blockchains are the primary factor that influences scalability, yet they have a critical requirement for validation of transactions: every participating node on the network has to validate each transaction sequentially, and then store transactions on its copy of the ledger. Logically as the number of transactions on the network increases (with an higher adoption), so the number of nodes does. Because of that, the data for each transaction has to travel a lot more before being validated and stored by all the nodes on the network.
Therefore, blockchain scalability seems to inevitably reduce as the network size increases. But still, it's all about trade-offs. The costs to achieving infinite scalability is to give up complete decentralization. Quickly growing networks will require a faster consensus mechanism, in order to validate more transactions while delivering the same speed to individual users. In Proof of Work, this would mean to ease the hash puzzles, but then forks would occur more frequently and the blockchain would loose its consistency.
As far as some numbers are concerned, relatively elder blockchains, such as Bitcoin or Ethereum, are able to process between 3-20 transactions per second, a figure several orders of magnitude away from the amount of processing power needed to run mainstream payment systems or financial markets. Some of the most recent and performing ones are instead increasingly obtaining much higher throughput.

Probably, as of January 2019, Tron [72] is becoming the most performing one, as it's reaching consistently a volume of 2000 transactions per second, other than being one of the largest one, after having incorporated into his ecosystem BitTorrent, the world's largest and most famous decentralized file sharing protocol. Another great candidate to become the *Ethereum killer* is Zilliqa [80], which comes out with a natively Proof of Stake consensus mechanism, and claims to be currently supporting nearly 3000 transactions per second, a number that is supposed to increase after the launch of the main net, planned to occur at the end of January 2019.

Solutions

Although it may look like a relatively emerging issue, many solution to the blockchain trilemma have already been studied and analysed. Such solutions walk along two different paths.
On one hand, there exists many possible *easy solutions*, such as increasing the block size, or as splitting applications among many different chains, giving up scaling a single chain. Both solutions surely would increase scalability, but not for free: the first approach might cut many light nodes out, increasing the centralization risk; the second one instead would reduce security by the same factor as the one gained in scalability, as the resources needed to take control over a sub-chain would be inferior.
On the other hand, harder to implement solutions involve either formalizing a more efficient consensus protocol for a brand new blockchain, or implementing a new layer on top of the existing blockchain architecture. Fortunately, there have been attempts in both ways, as presented below.

Algorand Algorand [1], is the name of a blockchain project that claims to be able to solve the trilemma. Algorand defines itself as *first-of-its-kind, permissionless, pure proof-of-stake protocol supporting the scale, open participation, and transaction finality required to build systems for billions of users*. It has been founded by the Italian cryptography pioneer and Turing award winner, Silvio Micali [67]. The name Algorand comes from the fact that the approach uses algorithmic randomness to select, based on the ledger constructed so far, a set of verifiers who are in charge of constructing the next block of valid transactions. Such selections are ensured to be immune from manipulations and unpredictable until the last minute, and at the same time universally clear.
The key Algorand's features are:

1. **Minimal computation required:** independently from the number

of users in the system, each of fifteen hundred users must perform at most a few seconds of computation.

2. **Higher Performances:** since in Algorand, the only limiting factor in block generation is the network speed. It has been experimentally tested that choosing a 2MB block size, it manages to commit around 327MB of transactions per hour (vs Bitcoin's 6MB per hour). Increasing the block size to 10MB, it even guarantees better performances, committing 750MB of transactions per hour, more than a hundred times higher than Bitcoin.[46].

3. **Negligible forking probability:** with this probability being less than one in a trillion. Thus users could rely on the payments contained in a new block as soon as the block appears.

4. **True decentralization:** for the fact that there are no exogenous entities (as the miners in Bitcoin), who can control which transactions are recognized.

Sharding Sharding is a concept that has existed in distributed systems for a long time, where it is used to improve scalability, performance, and I/O bandwidth [65]. It refers to the process of splitting a blockchain into parallel sub-chains called shards, where each shard processes a small portion of all transactions in conjunction with other shards and appending to the main chain only headers of such *shard blocks*, by means of another relatively small validation step.
Ethereum [64] and Zilliqa [81] are the most active crypto projects that are trying to solve the trilemma with sharding.
New challenges or threat would inevitably emerge in this scenario. First of all, as already said before, splitting the chain into sub-chains has the side effect that an attacker would need less resources to take control of it. The attack is called *single shard takeover attack*, and once succeeded the entire blockchain would be in danger, as the attacker could manage to control the shard blocks as they are appended to the main chain.
More serious issues concerns the cross-shards communication: of course the sharding solution would absolutely need a mechanism to reliably make data available all over the chains, otherwise how could node detect invalid blocks being appended? Another scenario is the one where the effect of a transaction in a shard depends upon events that earlier took place in other shards. How do we add cross-shard communication?

Lightning Network The Lightning Network [40] is a decentralized system for instant, high-volume micro-payments, that has been built on top of the Bitcoin blockchain, and claims to solve two major Bitcoin's limitations.
The first one is the impossibility to perform micro-payments, given both that the fees would be higher than the payment itself, and that the network would likely ignore such payments. The other one is the speed of the network, since it could take an hour to be sure that an appended block wouldn't be removed.
The solution here is one of the first implementations of a multi-party Smart Contract (programmable money) using bitcoin's built-in scripting language. Basically, funds are placed into a two-party, multi-signature "channel" bitcoin address. This channel is represented as an entry on the bitcoin public ledger. In order to spend funds from the channel, both parties must agree on the new balance, sign it, as it will be reflected into the bitcoin's balance sheet. Furthermore, it isn't required cooperation in order to exit a channel: both parties have the option to unilaterally close the channel, ending their relationship.
The network of multi-signature channels that it is generated as such, is then used to send payment from among users of this network, without the need to create direct channels every time. The benefits of such a solution are evident: other than giving the possibility to perform micro-payments (down to 0.00000001 bitcoin) without any risk of loss, it nearly eliminates fees (a user claimed to have paid 5 cents for 42 transactions [41]), it allows instant payments (since they don't need block confirmations).
A very similar approach has been formalized as well on top of the Ethereum Network, and it's called Raiden Network [60].

Privacy issues

Bitcoin was originally thought as a pseudonymous cryptocurrency that maintained privacy, as long as real-world identities couldn't be linked to Bitcoin addresses. However, due to the public nature of its blockchain, it quickly became possible to identify individuals based on usage patterns of certain addresses and transactions. Furthermore, nodes leak their IP addresses when broadcasting transactions. Nonetheless, unlike with scalability, the solutions for privacy are in some cases easier to implement but they are also much less *complete*.
It is probably impossible right now to create a technology which allows users to do absolutely everything that they can do right now on a blockchain, but with privacy. Instead, developers will cope with partial solutions, designed to bring privacy to specific classes of applications. In fact absolutely perfect black-box obfuscation is known to be mathematically impossible [4], as there

is always some information that can be retrieved out of a program. As the need of privacy in blockchain is fundamental for its scale adoption, many researches have been conducted to find a solution, in different aspects of it.
In contrast to Bitcoin's public transactions, many have developed solutions to support private transactions. ZCash [78] for example is built by a strong team of academic cryptographers using zk-SNARKs , or more specifically *zero knowledge-succinct non-interactive adaptive argument of knowledge*. It's a particular type of zero-knowledge proof system [63] that enables one to succinctly (efficiently enough) and non-interactively (because the prover publishes the proof in advance, and anyone can verify it) set a proof of some computation.
Even though they represent a valid solution, they come at a pretty high price: other than being resource intensive (enough to cut light nodes out), zk-SNARKs need for each type of problem to solve an upfront communication step called the setup phase, that other being computationally expensive and time consuming, needs to be *trusted*, because otherwise it would allow fake proofs to be undetected. Because of this restriction, zk-SNARKs aren't a good fit to run arbitrary Turing-complete smart contracts, as each new contract would require a new setup phase.
So ZCash only implemented a private currency, built on one particular SNARK circuit, the ZCash transaction verifier, with its own trusted setup. This trusted setup involved a group of cryptographers and well-known community members coming together in a complex setup ceremony [79]. Trusting the security of ZCash means trusting those participants won't ever collude.

Another famous project implementing privacy of transactions is Monero [49], which instead of zero knowledge proofs uses ring signatures. A ring signature makes use of one's account keys and a number of public keys (also known as outputs) pulled from the blockchain using a triangular distribution method. One of the security properties of a ring signature is that it should be computationally infeasible to determine which of the group members' key was used to produce the signature, and so, ring signatures ensure that transaction outputs are untraceable.
Apart from creating totally privacy preserving blockchains, lots of work has been concentrated on creating a privacy layer on top of the basic blockchain one, towards two directions: privacy of data and privacy of transaction. As far as privacy of data is concerned, Keep [36] is creating off-chain containers for private data, which would allow smart contracts to manage and use private data without exposing the data to the public blockchain.

Another exciting and worth of being cited project is AZTEC [75] (Anony-

mous Zero-knowledge Transactions with Efficient Communication). AZTEC is a protocol comprehensive of a set of zero-knowledge proofs that is able to provide confidentiality of transactions (where the value being transferred is not visible to others), specifically designed to be compatible with blockchains supporting Turing-complete computations. Right now, AZTEC has been implemented for auditing on the Ethereum blockchain. Through a smart contract it is now possible to attach AZTEC zero-knowledge proofs to existing digital assets (i.e. ERC20 token standard). Gas costs for transactions that verify AZTEC proofs are approximately 840'000 gas (less than half a dollar approximately, but it floats according to the gasPrice) for a simple transaction with two inputs and two outputs, though it is possible to batch multiple proofs together to save money.
AZTEC maps real quantities to a so-called *note*, which is an encrypted representation of the quantity. Users send and receive notes through *join-split* transactions, which means an input is destroyed to create one or multiple output.
For example, imagine Alice has two AZTEC notes worth 100 tokens combined. If she wants to send Bob 20 tokens, Alice would create one or more notes owned by Bob, whose values sum to 20. She would then create one or more notes owned by her, the sum of which is 80 tokens. Alice would construct an AZTEC zero-knowledge proof to prove that the balancing relationship between input and outputs hold. After this is done, AZTEC's smart contract would validate the proof, destroy Alice's input notes, and then create the output notes (one or more equal to 20 tokens for Bob, and one or more equal to 80 tokens for Alice).
In order to complicate things to onlookers, users have the possibility to create multiple notes with varying values, even null ones, to obscure the output value. To simplify the process of creating proofs, AZTEC has been working on an application programming interface (API) and other developer tools, that will handle much of the work required to construct such proofs. With the API, users will only need to input a few bits of information, such as the values of their tokens and their public keys, rather than having to perform specific forms of elliptic curve arithmetic.
AZTEC appears to be a quite important game-changer: for the first time it's possible to create confidential digital assets on Ethereum, obtaining the immutability and decentralization benefits of public blockchains without sacrificing privacy. Furthermore, being the AZTEC proofs cheap to construct, the team is confident about giving the possibility of issuing confidential transactions directly from hardware wallets, thus never exposing sensitive private keys. This is not all yet: the team is planning to accomplish even more:

- **Confidential decentralized exchange** where it would be possible to trade AZTEC assets in complete confidentiality (both quantities and prices of orders private).
- **Anonymous identity sharing schemes** that would allow to prove to be part of a group, without revealing one's identity. This would represent another great milestone for the project, being this an essential component for many KYC processes. It is said that all AZTEC tokens would implement this functionality implicitly.

Smart Contract Security

Smart contracts are the application in which blockchain will likely have the most disruptive short-term impact. Millions of dollars are already entrusted to their code and, and according to Etherscan [23], as of January 2019, more than 1 million contracts have been deployed just on Ethereum.
The problem is that, for a series of reasons, coding smart contracts correctly appears to be extremely challenging from a security point of view. A scientific research paper [51], release in early 2018, concluded that right now *Ethereum contracts are full of holes*. They have implemented a new systematic characterization of a class of trace vulnerabilities, coming out after multiple invocations of a contract over its lifetime, by means of an open-source tool, called MAIAN. [42].
The analysis was conducted against nearly one million contracts, simulating a life-cycle for 10 seconds per contract, and found that more than 2 thousands of them were vulnerable, for a total 34000 flags raised. A subset of 3759 contracts was further analyzed, with real exploits reproduced, and the 89% of them yields vulnerabilities. Such threatening figures are explained by several different factors.
First off, smart contracts, unlike traditional software, can't be upgraded or patched once deployed. Secondly, they are written in a new ecosystem of languages and environments, and as such still they lack of standardised best practices. Moreover they are quite difficult to test extensively, since they are able to interact with other smart contracts, other than being invoked repeatedly by many users. On top of these, the fact that contracts sometimes bring with them millions of dollars is bringing probably serious and skilled attackers on the field.
As it will be shown sometimes hacking a contract means draining directly all its balance. It is well known in fact that during the relatively short history of Ethereum, the major disasters have been smart-contract related. The two most expensive ones so far are:

- **Parity freeze:** happened in November 2017 when a user accidentally triggered a bug in the smart contract of cryptocurrency wallet provider Parity, freezing more than a hundred million dollars worth among all users' wallets [56]. Few months before, always the Parity's multi-sig wallet had already been the target of an hack, where a smart contract vulnerability was exploited to steal 150.000 ether from user accounts.

- **The DAO attack:** where DAO stands for Decentralized Autonomous Organization, which is basically a structure with decentralized control, where each rule and decision-making aspect is codified through smart contracts. *The DAO* (the name of a particular DAO) was launched on April 2016, and became so popular that raised more than $100m. In June 2016, The DAO was hacked, and 3.6m Ether (15% of all ether in circulation at the time) was drained from its smart contracts, exploiting a code vulnerability. Since for smart contracts in DAOs *the code is law*, the hacker claimed through a public letter [18] that he didn't steal anything, as he just used a feature coded. This obviously opened up a giant dispute among the community: the only way to recover the funds lost in the hack, and effectively go back in time, was for Ethereum itself to be reset through a hard fork, leading to the creation of Ethereum Classic, which preserved the original version of the blockchain that included the hack.

Back to the aforementioned trace vulnerabilities study, three kinds of families of *contracts with vulnerabilities* have been defined: contracts that either leak funds carelessly to arbitrary users, or lock them indefinitely, or can be killed by anyone.

Prodigal Contracts Belong to the category such contracts that gives away Ether to an arbitrary address, which is not an owner, has never deposited Ether in the contract, and has provided no particularly hard to forge data along with the function call. The tool developed has flagged 438 distinct contracts, which may leak Ether to an arbitrary Ethereum address, with a true positive rate of around 97%.

Greedy Contracts Contracts that remain alive and lock Ether indefinitely, allowing it be released under no conditions. An example of that is linked again to the Parity wallet incident: after the Parity library contracts were killed, the wallet contracts couldn't access the library anymore, becoming greedy. This vulnerability turned out to lock $200M worth of Ether

indefinitely. Greedy contracts may be individuated easily because they accept Ether and either completely lack instructions that send Ether out (i.e. send, call, transfer), or such instructions are not reachable. The tool individuated 1058 contracts accepting ether and not having instructions for realising them, with a 100% true positive rate. In order to test also unreachable function calls that release funds, the analysis had to be manual and statistical, and the extrapolated true positive rate was about 69%.

Suicidal Contracts The Ethereum Virtual Machine provides an op-code (SELFDESTRUCT) after which the smart contract is no longer invocable and called dead. This is a pretty useful feature: often in a contract it is coded a security fallback option of being killed by a specific address (i.e. contract's owner) in emergency situations, for example when under attack. Suicidal contracts are those that can be killed by any arbitrary account. The tool found among the set nearly 1500 possible suicidal contracts (including the Parity wallet library), with a true positive rate of 99%.

Posthumous Contracts After killing a contract, all its code and variables are cleared from the blockchain, in order to prevent any further execution of it. Nonetheless, killed contracts continue to receive transactions, and if Ether is sent along them, it is added to the contract balance, which is locked indefinitely. Killed contract or contracts that do not contain any code, but have non-zero Ether are called posthumous. The tool easily found 853 dead contracts, of which 294 received Ether after being shut down.

1.1.7 Do I need a Blockchain?

After having analyzed from a technical point of view the blockchain technology, it is left to understand where this innovation is really convenient and where instead it's pure hype. Karl Wüst and Arthur Gervais conducted an extensive analysis whether a blockchain is indeed the convenient technical solution for a particular use-case [31]. They also developed a web application out of it [20]. Their conclusion is straightforward: most probably the blockchain is not needed. Let's see what methodology they formalised to claim so.

Database Obviously, being essentially the blockchain a distributed database, the first rule is to check whether or not some data needs to be stored. If not, the blockchain is absolutely not needed.

Writers The second check is about writers, intended as entities with write access in a typical database system or as participant to consensus protocols in a blockchain system. If there's only one writer, a blockchain does not provide additional guarantees and a regular database is better suited, both for efficiency and ease of use. Blockchain provides advantages when the system involves multiple writers. But still, if the writers all mutually trust each other, a database with shared write access is likely the best solution, since the integrity of data would be guaranteed equally.
On the other hand, when there's no trust, using a blockchain could make sense. If the set of writers is not fixed and unknown a priori to the participants, as is the case of bitcoin, a permissionless blockchain could be a suitable solution.

Trusted Third Party Even in a context with multiple writers on a shared database, a solution that is not the blockchain exists: the trusted intermediary. Banks for example control the database and ensures that every transaction is valid and authorized by the customer whose funds it moves.
Customers who don't trust each other trust the bank instead of being impartial. Blockchains remove the need for such trusted intermediaries, so the question here is: do I need to remove the intermediary, if I have one? Good reasons to prefer a blockchain-based database might include lower costs, automatic reconciliation, new regulation or a simple inability to find a suitable intermediary.

Transaction Log So far we could have solved our use-case choosing a distributed database, with transactions coming from many places, propagated in a peer-to-peer fashion and verified by every node independently. Yet what's missing from the scheme is the authoritative final transaction log, which constitutes the blockchain's main power. Why is this log so important?
First, because it enables new nodes to calculate the database content from scratch without the need of trusting another node. Second, because it allows node to be active or inactive whenever they want to, without the risk of finally having divergent copies of it. Third, because it implements natively an automatic conflict resolution mechanism, which is the consensus protocol.

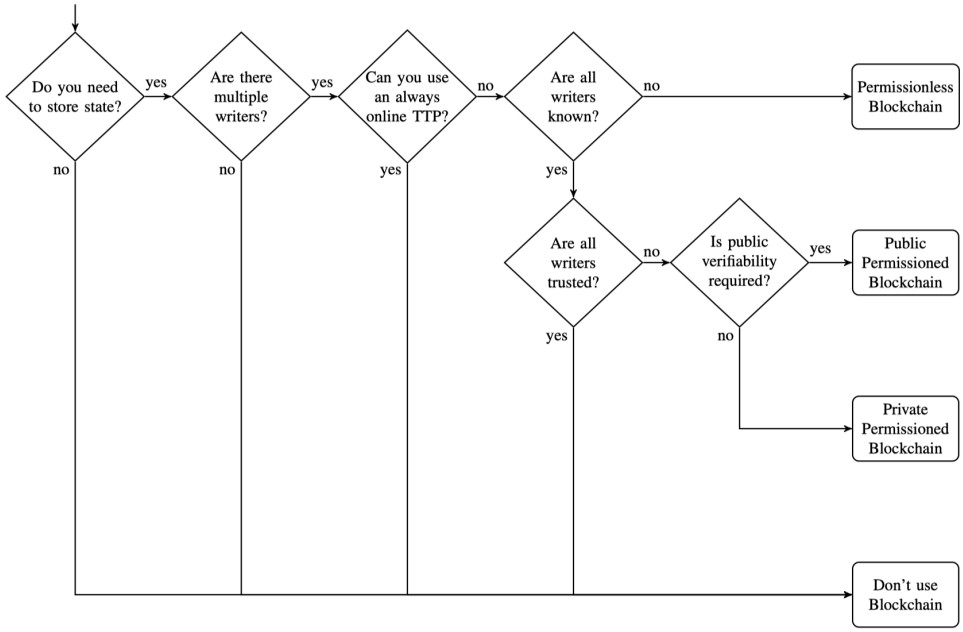

Figure 1.2: A flow chart to determine whether a blockchain is the appropriate technical solution to solve a problem
[31]

1.2 Secure Multiparty Computation

Secure Multiparty Computation (MPC), is a cryptographic tool that allows a number of parties to jointly compute a function over their inputs, keeping them private and guaranteeing that the output is correct (if returned). It was introduced as secure two-party computation (2PC) in 1982, with the notorious Millionaires' Problem, and for any other possible computation in 1986, by Andrew Yao [77].
MPC has been studied for almost thirty years, but just until recently only in the context of academic works, as the protocol requires a fair amount of computational effort and a intensive network communication. More or less in the last decade, MPC has increasingly become more and more feasible, and many real-world projects started to take place [8] [66]. A key part in this transformation from theory to practice is the implementation of techniques that improve performance, while not sacrificing the level of security of the theoretical definitions. It is important to understand that MPC is not a single protocol but rather an expanding class of solutions differing in properties and performances.
MPC is still a lot challenging because other than covering machine crashes or faults, it deals with all those scenarios where one or more parties involved in the computation act maliciously, that is, tries to deviate from the protocol in order to either learn private information or cause the result of the computation to be erroneous. Thus, the definition and implementation of a secure multiparty computation solution involves not only computer scientists but cryptographers as well.

1.2.1 Security Definitions

As mentioned above, the model to consider in this field is one where an adversarial entity controls some subset of the parties and wishes to attack the protocol execution. Such parties are called *corrupted*, as they follow the adversary's commands.

Requirements In order to formally assert that a MPC protocol is secure, a precise definition of security is needed, and this is related to the requirements a general purpose MPC protocol must meet. Although it is hard to formally define a set of requirements that covers all possible applications, there are few yet fundamental ones to be hold for any secure protocol:

- **Privacy:** No party should be able to learn anything more than the function's output. This means that the other parties' inputs must

remain secret, and the only information about them is what can be derived from the output. For example, in an auction it is revealed only the highest bid, thus the only information about all other bids is that they were lower than the winning one.

- **Correctness:** Each party should be guaranteed that the output it receives is the correct one.
- **Independence of Inputs:** No party should be able to choose their inputs as a function of other parties' inputs.

The above informal definition of security misses out a very important factor to consider, which is the power of the adversary attacking a protocol execution. As far as this is concerned, it is important to analyze his corruption strategy and his behavior.

Corruption Strategy The corruption strategy involves the question of when or how parties get controlled by the adversary. There are two main models:

- **Static corruption model:** where the adversary takes over a fixed set of parties before the computation takes place. Throughout the computation, honest and corrupted parties remain so.
- **Adaptive corruption model:** where the adversary is capable of corrupting parties during the computation, arbitrarily upon its view of the execution. Once corrupted, parties remain so until the end.

Adversarial Behavior Another important factor that must be taken care of is related to the actions that corrupted parties are allowed to undertake:

- **Passive adversary:** here even corrupted parties follow the protocol. The advantage of the attacker comes from the fact that he is able to read the internal state of the corrupted parties, including all messages received from honest ones. This may help him to learn information that should remain private. They are sometimes called also *honest-but-curious* or *semi-honest*. Secure computation under this model can be carried out very efficiently, but provides weak security guarantees, not sufficient in many settings.
- **Active adversary:** here corrupted parties can freely deviate from the protocol, according to the adversary's instructions. This adversary may be also referred to as *malicious*. Providing security against this

adversarial model means for a protocol to be highly secure, given that honest parties are guaranteed to be protected independently from the corrupted parties' behavior.

- **Covert adversary:** this rather new model has been formally defined by Auman and Lindell [2]. They describe covert adversaries as in between the semi-honest and malicious model: they are considered active adversaries, because they may deviate from the protocol specification, but they do not wish to be *caught* cheating. This definition might indeed guarantee a fair enough level of security for many business, financial, political and diplomatic use cases, where for example parties cannot afford the loss of reputation and the negative effects with being caught cheating.

1.2.2 Use cases

As already said, for many decades multiparty computation was perceived to be of purely theoretical interest, but lately use-cases have been emerged, proving the feasibility and the consequent opening of many potential unexplored markets. Generally speaking, these use-cases can be grouped basically into two major sets. The first provides database style applications, that allows to extract statistical information, the other concerns the control and processing of cryptographic data, such as keys.

Jana: Private Data as a Service Data as a Service (DaaS) is a well known and scalable model where many parties are able to access a shared data resource. However, due to the poor privacy level such systems offer, more privacy regulations (i.e. GDPR) have emerged, restricting the field of the type of data that can be collected, other than stipulating how long it can be kept and which parties may use it. In order to such services to survive, cryptographic protections needs to be applied on data in transit, at rest and under computation. Such set of technologies can be called Private Data as a Service (PDaaS).
One of the most promising is the Jana project [35]. Jana is a project still under development, and is led by David Archer from Galois [19], who has reunited many researchers from different universities, such as Nigel Smart from KU Leuven, and many others. In Jana, data is encrypted at all times (unless explicitly asked to be stored as plaintext): in transit from input parties to Jana and backwards, data is protected with ephemeral public key generation and encryption; at rest in relational database is protected through public key encryption, deterministic symmetric encryption.

Finally, the Jana ecosystem supports multiparty computations of most query operators, using the SPDZ protocol [16]. Furthermore, the Jana team has planned very impressive long term goals, such as the implementation of machine learning algorithms running in Jana under a multiparty computation scheme.

Sharemind Sharemind is a solution developed over the last ten years in Estonia to deal with the data-sharing problem emerged with the aforementioned privacy regulations, offering similarly to Jana Private Data as a Service. What makes Sharemind particularly innovative is its architecture: it is implemented as a distributed computing system, with each party involved running a different part of the ecosystem.
Input parties (who owns the private data) and *Result parties* (who asks for computations) run client applications that connect to the Sharemind Client API, responsible for automatically applying the needed cryptographic algorithms on the inputs and outputs. *Computing parties* run the Sharemind Application Server software, a distributed system that receives queries from client applications and runs the asked program in the Sharemind's Secure Computing Virtual Machine with inputs received from the client application, after having authenticated him.
Sharemind has provided also a two-level programming language, SecreC, an high-level procedural language for writing privacy-preserving algorithms without knowing the underlying cryptographic primitives applied to the data.

KPI analysis The very first application which utilises Sharemind on real-world data was made by the Estonian Association of Information Technology and Telecommunications (ITL), and concerned the analysis of Key Performance Indicators (KPIs) of their sector [8]. That was the first time where the actual computation took place over the internet, with computing nodes located in different areas.
The problem to solve came from the fact that the official economic report published by the ministry had a yearly occurrence, and by the time it was published, the data was already half a year old. Thus, ITL members uploaded their financial metrics to the computing parties (in this case three), under a multiparty computation scheme. After having collected the data, the multiparty computation system sorted them by each defined KPI, independently from the other.
Through a web application then, the same ITL members were able to see the sorted metrics of all the industry, without knowing the company each of them belonged to. The multiparty computation program, written in Se-

creC, collected as input data coming from 17 companies, and took nearly two minutes to perform the sorting.

Unbound: Virtual-HSM Another potentially great application of the multiparty computation is related to the concept of distributed cryptography. The rationale is straightforward: spreading the operations of a cryptosystem among a group of parties (processes or servers), in a fault-tolerant way.
An example of that is given by the Israeli company Unbound [73], founded by Nigel Smart, which implemented a Virtual Hardware Software Module, built upon the concept of secret sharing. Traditional HSMs are physical computing devices that are able to perform cryptoprocessing, in order to manage and secure digital keys. They can be exploited in any application which makes use of digital keys, as they are able to perform key generation and storage, encryption and digital signatures, among the others, all within a highly secured environment.
The Unbound Tech vHSM product is an MPC engine which is able to carry out any cryptographic operation found in a standard cryptographic API, but in a distributed manner. The main use is probably related to the secure storage of critical keys (as for in the banking sector). With the distributed solution, the private key is split between the vHSM MPC engines, and every time a it is needed, the server calls out to the MPC engine to perform the distributed signing operation.
The integration of such solutions in standard cryptographic APIs, such as PKCS #11, within standard web server software appears to be a triviality. Besides, being the integration in software, it would be easy as well for cloud environments, easier than when using standard HSMs. Perhaps an even more interesting scenario is to perform the above application on a mobile device, having the cryptographic key split between the device and a server. This would lighten the security needed on the mobile device, since a loss of it wouldn't imply the security of the secret, as long as the other piece of it, stored on the server, is safe. Moreover, the key is never transmitted in its entirety, removing the need to secure the mobile application against side-channel attacks.

Chapter 2

Incentive Compatible and Privacy Preserving Data Analytics System

In this chapter the actual thesis work will be presented, following a top-down approach. Section 2.1 is dedicated to the motivations behind this work, along with a set of open problems that this idea might be able to solve. Because of that, the use-case chosen for the implementation of a proof of concept is presented in this section as well.
After such considerations, the high-level overview of the implemented application will be presented in section 2.2, starting from a description of the system in its entirety, in order to understand how the technologies are seamlessly integrated together, to an explanation of the actors involved and of the protocols formalized.
The technical implementation will be extensively described in section 2.3. The pieces composing the system will be presented one by one in more detail: first the blockchain and the smart contract design, then the multiparty computations framework and programs, lastly the decentralized application implemented from the union of the two, including a section with some snapshots of it.
The last two sections (2.5, 2.6) are dedicated to the analysis of the performances of the overall system and to its security model respectively.

2.1 Motivations

2.1.1 Incentive Compatibility

Starting from the mid of the last century, given the rise of two fundamentally opposite economic models such as capitalism and socialism, economists found themselves in the need of formulating new theories and philosophies that were able to include those new factors. Such theories took the name of Mechanism Design [44].
Some of the early contributions to these theories can be due to well-known economists such as Mises, von Hayek, Barone and others, that were the first to debate over topics such as the feasibility of a centralized socialist economy, or how to collect decentralized information and allocate resources. Very likely, the modern growth and formalization of these theories came from the inclusion of incentive issues. Leonid Hurwicz, in his *Mechanism Design Theory* (1960), had been the first coining the term *incentive compatibility*. Incentive compatibility revealed to be a total innovation, and in 2007 was one of the key ideas that costed him, along with Eric Maskin and Roger Myerson, the Nobel Prize in Economic Sciences [34].
Eric Maskin, in his lecture about the topic [45], used a simple example to explain clearly what the concept was all about: the one having a parent that wants to find a mechanism to divide a cake between two children, in a way that makes the two equally happy. The solution is trivial: the mechanism could be that one child cuts the cake, while the other has the first choice of the piece he wants. Assuming that the children acts rationally, this mechanism is claimed to be incentive-compatible, because it allows both children to achieve their goal, that is to have at least half of the cake.
It is undeniable that our lives are governed by situations where collective decisions have to be made while taking into account also individual preferences, other than that individual participants may act for their own gain, rather than for the general well-being. Mechanism design can ultimately be referenced as the effort to produce institutions that align social goals to individual needs. Bitcoin is a great example of an incentive-compatible network: it has intentionally been designed in a way that the mining process is difficult and inefficient, thus being costly and counter-productive for malicious actors. On the other hand, actors are incentivized economically to secure the network with their computing power, being rewarded with new coins when this happens.

2.1.2 Complementary technologies

Back to computer networks, and more precisely to multi-party systems, incentive compatibility can be applied to obtain configurations where each party loses only by deviating from the protocol. As already said, with the addition of economic assumptions to purely cryptographic ones, security can therefore be shaped by considering that parties might not be just honest or not, but also that are rational and their gain can be quantified, such that with a proper incentive scheme, many of them are discouraged from doing any damage to the system.
As already stressed in Section 1.2, Multiparty Computation (MPC) allows to analyze data while keeping the inputs private at all times. Although it may sound contradictory, completeness theorems have demonstrated already back in 1988 its feasibility for whichever type of function [48]. Because of that, the literature around it is quite vast and demonstrates how promising this technique is. Yet, practical implementations still are few.
Foremost this is due to the fact that, given the performances it offers in the current state of the art, MPC is poorly scalable. Another important factor comes from the security model it addresses, and the consequent assumptions to make. Indeed, under the assumption that the majority of parties are always honest, privacy, correctness and output delivery are ensured; but if this assumption turned out to be wrong, the entire system would immediately be broken. On the other hand, protection against dishonest majority involves the possibility to abort; but then a single corrupted party would be able to deny the service for all honest ones.
It looks evident that for multiparty computation, as long as honesty is concerned, cryptography alone is not enough, simply because it cannot model strategic decisions or human behaviors. The blockchain appears to be the missing piece in this configuration, as it represents the proper technology to implement an incentive compatible model for a multiparty computation scheme, exploiting all its revolutionary features. First, exploiting its integrity property, the blockchain would constitute the single source of truth for the entire MPC system, as a distributed log. Then through smart contracts the entire application logic (including the economic model) could be coded and, reading from the shared log, honest and dishonest parties could be detected. Finally, using cryptocurrencies and the direct payment channel the blockchain provides, smart contracts would be able to inevitably enforce the penalization and the rewarding of parties according to their behavior, without the need of intermediaries.

2.1.3 Use case: Medical Privacy

The fast-paced progress in technology and electronic data processing has led nations to adapt their data protection framework. The increasing digitalisation of all kinds of data, inevitable on one hand, has clearly exposed personal and sometimes very sensitive data to security breaches, hacking and unlawful kinds of processing, trespassing the fundamental individual's privacy right.
In January 2012, the European Commission proposed a major reform for the data protection framework, the so called General Data Protection Regulation (GDPR). It aims principally to re-balance rights in the digital worlds, providing stronger rules on data protection, in order for data owners to detain more control over their data.
As long as medical data are concerned, even more special treatments have been established for their processing. The GDPR has included in this category also *genetic and biometric data*, since upon their process, the unique identification of a natural person is possible. The article 9 thus dictates that the processing of such special categories is generally prohibited [30], yet at the same time (Article 9[2][j])) legitimising it only in the domain of health and healthcare management, according to few criterias. In this case the processing has to guarantee respect for principles of the so-called data minimisation, such as the anonymization of them.

Private medical data are for sale For decades researchers have conducted studies to gain knowledge over health and illness, by regularly collecting information about the same people's medical history over the years. Unfortunately, they have not been the only ones: many companies have specialized in gathering information from hundreds of millions of medical records, coming from hospitals, doctors, prescriptions, insurances and laboratory tests. It is pretty clear why they've done so: this gigantic databases are worth billions of dollars. Many other businesses indeed are willing to pay for such collections in order to guide their investments, for instance in the pharmaceutical industry.
As stated by law, identities of data owner's in these commercial databases are supposed to be kept secret. The law is actually respected by stripping off all those information that can connect to the data owner's identity from their records. Clearly this is not enough anymore: through some not even sophisticated data mining algorithms, data brokers are able to match sparse pieces of information to the same individual (without knowing his name), and by scanning multiple public databases may end up re-identifying the individual himself.

Back in 1997, the today Harvard University Professor Latanya Sweeney, while being a graduate student at MIT was able to identify the Massachusetts governor William Weld in the publicly available hospital records, just by comparing the supposedly anonymous data to voter registration rolls of the governor's hometown. She is now the director of the Data Privacy Lab at Harvard, and in 2013 she has re-identified more than 40% of people who have shared their DNA data for the Personal Genome Project [70]
Citing Adam Tenner [71], the dominant player in the medical-data-trading industry is IMS Health, that recorded $2.6 billion in revenue in 2014. Nowadays, it receives automatically from the computerized records of pharmacies, insurance companies and other, petabytes of data, and claimed to have assembled half a billion dossiers on individual patients across the United States and Australia. According to Marc Berger, in charge of the analysis of anonymized patient data for Pfizer, one of the most powerful pharmaceutical company, they spend $12 million a year to buy health data from a variety of sources.

The need for a solution To draw things even worse, it is important to underline that all of the aforementioned system is very often unknown to patients. Considering the 9-to-12 figures number that such business is worth, it is pretty much a shame that the real data owner don't get anything out of it.
Riding the wave of the new GDPR regulations, a blockchain based multiparty computation solution at the same time would be compliant with the privacy law, other than contributing to a redistribution of wealth that is so unfairly in the hand of few companies. Through the blockchain, data owner could get back control over their data and would have a real incentive on supplying them to institutions for research, as well as to pharmaceutical companies for a more precise strategy. Through the multiparty computation, calculus over medical data would be conducted in a totally obfuscated way, so to avoid the risk to cross-check them to obtain further private information, as it already happens.
The work of this thesis is an experimental implementation of such solution.

2.2 High-level overview

2.2.1 Actors

The entire application logic is coded into a smart contract, and in order to interact with it, users will need to send (and thus digitally sign) and receive

transactions (sometimes including cryptocurrencies as well). To do so, the actors involved are all registered on the blockchain, that means they own a pair of keys of a wallet. Since the blockchain used is Ethereum, it is appropriate to talk about actual accounts.
There exists three kinds of actors in the system:

- **Data Producers:** are the ones that own the actual data (i.e. patients). They share such data in a secret-shared form (Section 2.3.2), and are rewarded in cryptocurrency every time their data are input in a computation.
- **Data Consumers:** are the ones that ask for a specific calculus (and pay for it), that will be performed in a multiparty computation fashion by computing parties. They will never get access to input data, but just to the output of the computation they asked.
- **Computing Parties:** are essentially servers that perform multiparty computations. They have to be registered on the blockchain as well both to use it as log of the computation, and to obtain rewards for their work. Input data is secret-shared among the parties such that in order to reconstruct it, all of them have to collude and exchange their shares. For this reason, each computing party is supposed to be held by a different organisation, possibly with opposing interests.

2.2.2 System

From an high level point of view, the Multiparty Computation system acts as a distributed cloud system, where sensitive data are stored, that is able to ensure privacy of data both at rest and while computing any arbitrary function. Differently from traditional cloud systems, this one must be strictly decentralised, since the only yet fundamental requirement for the security to be held is that parties don't collude between themselves and unify their shares of data. To clarify the concept more, the reader may think of computing parties like of miners in Bitcoin: physical independent servers who do computational work in exchange of rewards.
The blockchain is as well part of this decentralized cloud, because provides a key property to the MPC system, the *incentive-compatibility*. Other than that, the blockchain represents the base layer upon which the client-side is built: is responsible of managing authorisation and authentication of users, as well as of handling disputes directly and automatically by paying (or penalizing) for good (or bad) behavior the actors. It is important to stress that

all payments involved in the system happens on the blockchain via cryptocurrency.
Application users in the system are then Data Producers and Data Consumers who, through a software library that runs on the client, interacts seamlessly with the blockchain and with the computing cloud. On one hand Data Producers are required only to share their sensitive data: this is easily achieved with a single interaction through a web application, while remaining agnostic of the whole underlying process (explained in Section 2.3.2). No further interaction is needed for them, other than for redeeming the rewards they have obtain anytime they want always via web application.
On the other hand, Data Consumers pays to obtain aggregate data coming from arbitrary functions, which are computed by the distributed cloud in Multiparty Computation, again just by interacting with a web application. Their are guaranteed to either obtain the correct output, or being refunded. In fact, in order to take part of the computation, each computing party must put at stake an amount of coins as well. Thus, before the computation takes place, the blockchain holds a security deposit coming from the actors involved, with which will reward or penalize parties according to the computation outcome.

2.2.3 Protocols

The overall application logic as already mentioned is composed of three specific protocols, *Data Sharing*, *Data Consuming* and *Post-Computation*. Since these protocols involve operations happening both on the blockchain and off of it, in the pictures below the former will be referred to with normal lines, while the latter with dashed ones.

Data Sharing The Data Sharing protocol (Figure 2.1), involves a Data Producer requiring the Multiparty Computation input process, which is described even in more detail in Section 2.3.2. As it's shown, the Data Producer authenticates itself with the MPC system using a kind of authentication token, issued upon authorization by blockchain.
Once the authentication has occurred, on an private end-to-end channel (off the blockchain) between the Data Producer and the MPC system, the actual data sharing takes place, following the SPDZ protocol. This requires each server to send his share of a random data $\langle r \rangle$, produced in the so called SPDZ pre-processing phase (Section 2.3.2), to the Data Producer. Although explained in the section, the reader should bare in mind that each server just know a piece of the random value (his share), and not the random itself.
Upon reception of all shares from all servers, the Data Producer reconstructs

the random value by adding all shares, uses it to mask his input data, and sends that to each MPC server. As soon as a server receives the masked data, he confirms to the blockchain the fact, so that when the blockchain receives all the confirmations, marks the Data Producer as payable: from that point on he will be rewarded when his data is used.

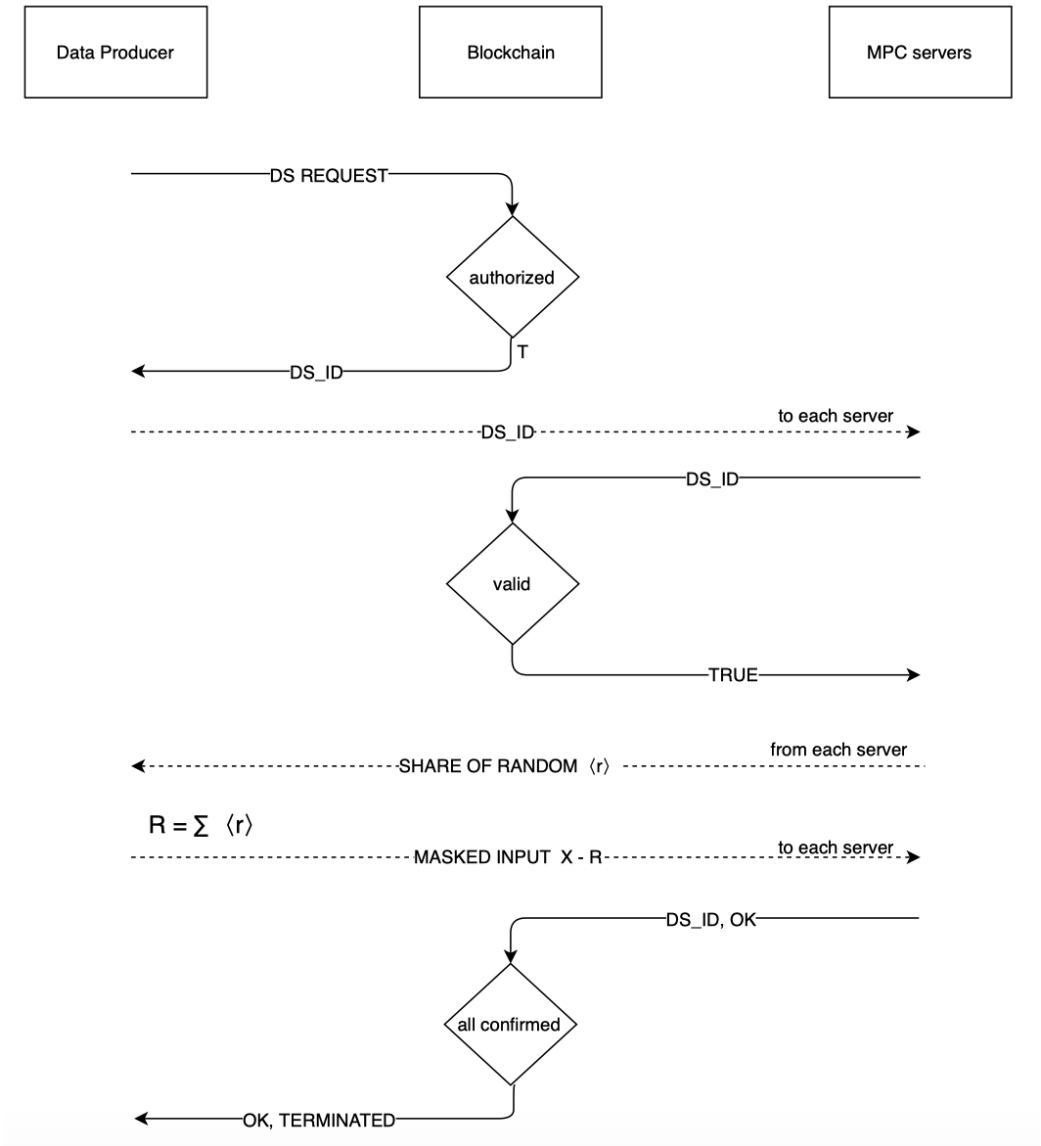

Figure 2.1: Sequence diagram of the Data Sharing protocol

Data Consuming The Data Consuming protocol (Figure 2.2), involves a Data Consumer asking for a specific computation to the MPC system over the Data Producers' data.
Again, the Data Consumer uses the authentication token given him by the blockchain to connect off-chain with the MPC system. Differently from the Data Sharing protocol, the DC has to send along with the request some cryptocurrency directly to the contract address, that serves as security deposit, other than as a simple payment for a service. Each MPC server, in order to take part to the computation just asked, has to do the same; this time the cryptocurrency put at stakes constitutes an incentive for him to be honest in the computation.
The smart contract, after having collected all the stakes by the parties, authorizes the computation to take place off-chain, in a private channel between the MPC system and the Data Producer. Being the MPC protocol in this case used secure against active adversaries, the computation either delivers the correct output to the Data Producer, or aborts. Either way, each MPC server append to the blockchain the output (or an abort code) hash, in order from the smart contract to understand what happened off-chain and proceed with the post-computation phase

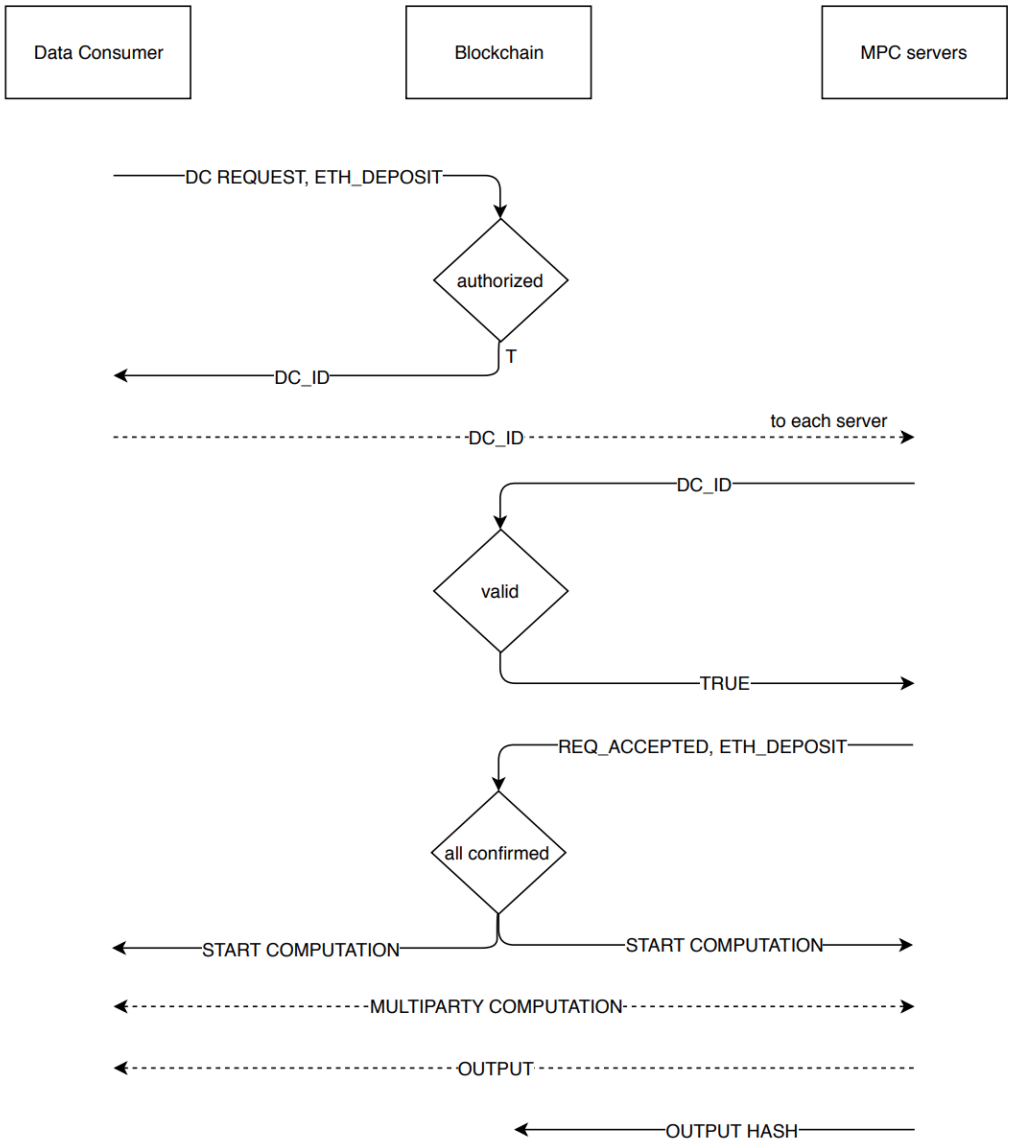

Figure 2.2: Sequence diagram of the Data Consuming protocol

Post-Computation This last phase (Figure 2.3) is enforced by the Smart Contract automatically after each computation, and is part of the incentive-compatibility framework of the entire system. The Smart Contract simply by reading the hashes appended by MPC server is able to understand whether the computation was honest or not. Of course, this is possible given the security definitions that the SPDZ protocol is built upon, that it is secure just by having at least one honest player among the n involved in the computation.

Under this assumption, the honest player will log in the blockchain always the correct information, either the output hash or the abort code. Thus the contract makes this deduction: if all hashes are equal and different from the abort code hash, the computation is necessarily gone well, otherwise it's not. In the latter case, the contract will send the data consumer, who hasn't actually received any service, his caution back, and will instead detain the MPC servers' ones, that are responsible for the misdeed. Clearly this basic solution is pretty rough, because penalises honest parties as well. Ideally the Smart Contract should be able to individually detect each party's behavior, and proceed accordingly. In Section 3.2.1, this issue is going to be discussed more.
Instead, when the output is correctly delivered to the Data Consumer, the Smart Contract will reward Data Producers and MPC servers by splitting the DC deposited amount, other than pay back the latter with the security deposit put at stake beforehand.

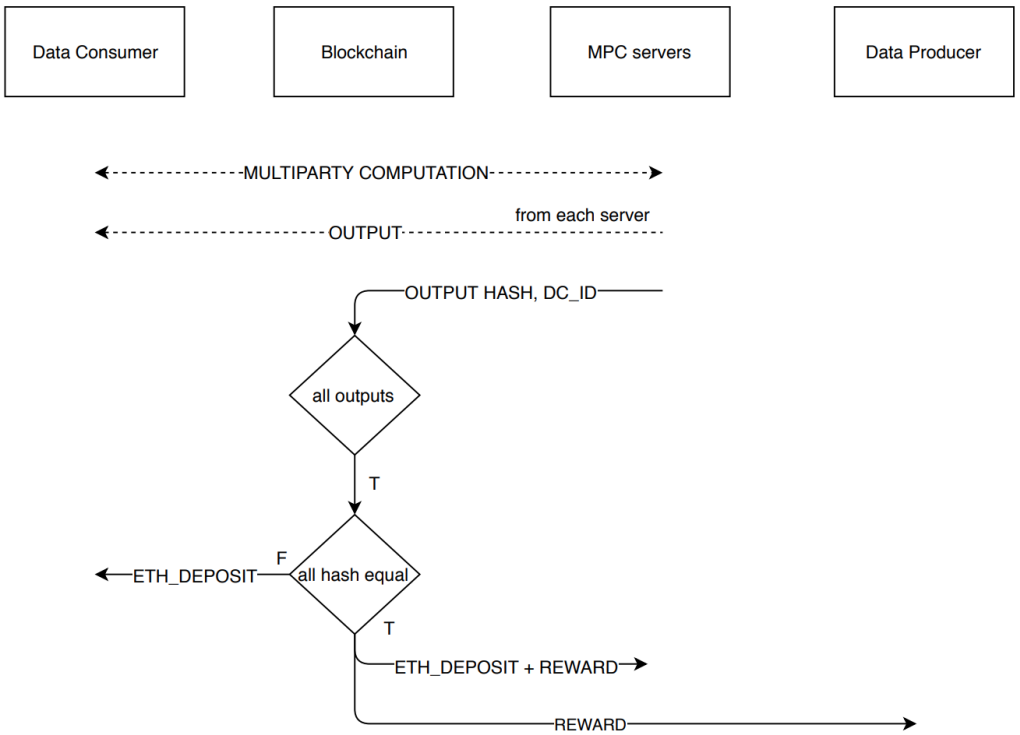

Figure 2.3: Sequence diagram of the Post-Computation Protocol

2.3 Implementation

2.3.1 Blockchain

The Blockchain chosen for the development is Ethereum, clearly because it's been designed to be a general purpose blockchain and so, being its virtual machine Turing-complete (even infinite loops are allowed, as long as one pays for it), one can implement whichever application logic. Besides it provides useful tools and a good documentation for the development of decentralized applications, among which the possibility to run smart contracts in a test environment, either local or not. Since when running on the main blockchain each operation has to be paid by someone, having a test environment it's not such an irrelevant convenience.

Smart Contract

The Smart Contract developed is formalized in Figure 2.4, and comprehends State Variables, Functions and Events.
State variables are variables whose values are permanently stored in contract storage. They can be either *public* or *private* to other contract calls, and the datatypes are pretty much the same as in other programming languages. Just worth to mention are the *address* datatype, which holds a 20 byte value (size of an Ethereum address), and the *mapping(keyType => valueType)* datatype, that can be imagined as hash tables and initialised such that every possible key exists and is mapped to a default zero value. The key is not stored in a mapping, only its keccak256 hash, that is needed to look up the value.
Generally, a function header in Solidity can be made of different keywords among the following:

```
function (param types) {internal|external|private|public} [
    pure|view|payable] [returns (return types)] methodName;
```

where the first set corresponds to the function visibility, more precisely:

- **internal** functions can only be called inside the current contract and the ones deriving from it
- **external** functions cannot be accessed from the contract but only externally, represent the contract interface
- **public** functions can be accessed both internally and externally by other contracts

- **private** functions can be accessed only by this contract

Interestingly in Solidity there is the distinction between functions which modifies the blockchain and functions that just read from it, namely:

- **pure** refers to those functions that do not modify the contract storage (i.e. they don't write the blockchain), nor read any state of it.
- **view** refers instead to functions that still do not write into the blockchain, but make a read out of it. When marked as view or pure, the function call doesn't end up with a transaction that will be broadcast to the network, but simply in a lookup to the current node's blockchain.
- **payable** refers to those functions that are supposed to receive Ether along with the transaction to add it to the contract balance. Considering the Smart Contract in Figure 2.4, the two functions which require the caller to send ether along are marked as payable.

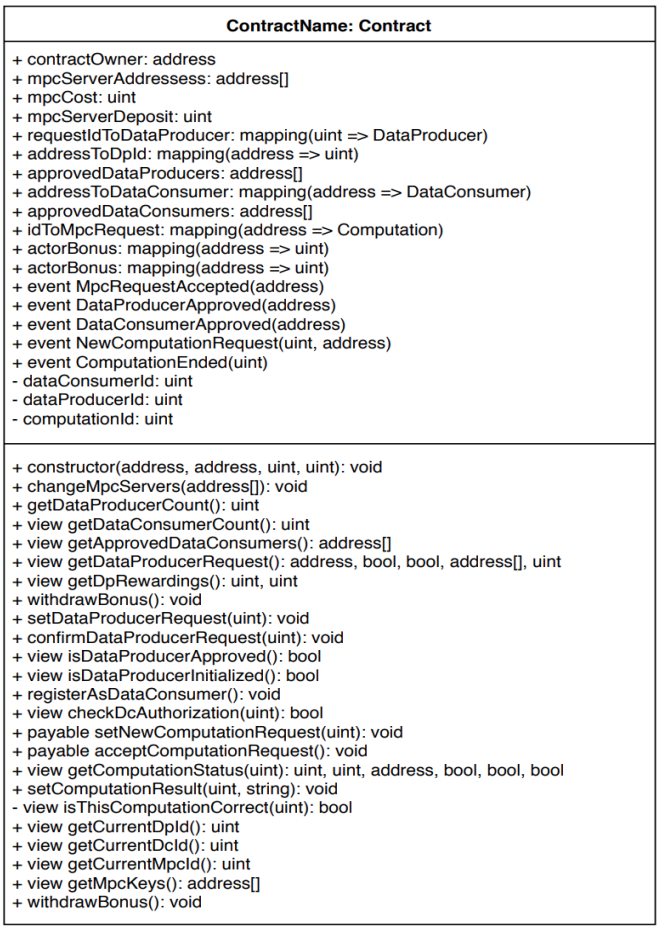

Figure 2.4: Class Diagram of the Solidity Smart Contract

As the reader can see from the class diagram, in Solidity there exist another special type which is called Event. Events are convenient types and are used for logging purposes, which in turn can be used to trigger JavaScript callbacks in those interfaces that are listening for them. In this context for example, MPC servers are listening on the blockchain for the *NewComputationRequest* event, which is emitted inside the function *setNewComputationRequest*: as soon as a new event is emitted, servers' callbacks activate to proceed with the computation.
Also three structs are defined in the contract, as formalized in Figure 2.7, 2.5, 2.6. This way, through mapping variables, each Ethereum address is associated with its specific data structure, containing the needed information.

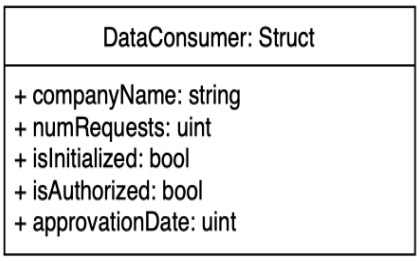

Figure 2.5: Data Consumer Struct

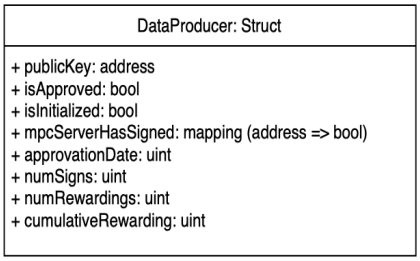

Figure 2.6: Data Producer Struct

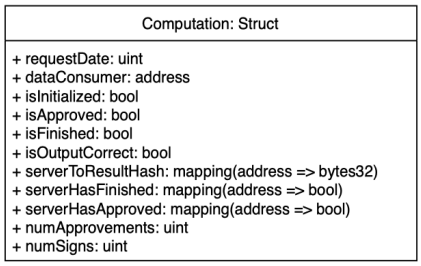

Figure 2.7: Computation Struct

Costs

In Table 2.3.1, it is shown the estimation of the Gas consumption (please refer to Section 1.1.5 for more details) for each function of the Smart Contract. The gas price used is 9.6 Gwei/gas [1] [2], and the average ether price over February 2019 of 123 USD/ether.
The number of Gas, that is the computational cost mining nodes will have to pay by running the code, is tightly dependent on the function instructions, each of which has a fixed Gas amount. A complete documentation of such costs can be found in the Ethereum Yellowpaper [76], while the most common ones are reported in table 2.3.1.
Nonetheless, there exist techniques of formatting the code that, by exploiting the underneath architecture, allow to optimize the gas consumption of a contract. For example a well-known pattern is called *tight-packing*, and exploits the fact that the storage in Ethereum is a key-value store with keys and values of 32 bytes each. When using smaller data types (i.e. uint8 or bytes16), the EVM automatically packs them together to fit a single 32 bytes slot, but following the order of their declaration. By just changing the variable

[1] 1 GWei $= 1 \times 10^{-9}$ ether
[2] `https://ethgasstation.info/`

order, one can then perform multiple reads or write in a single operation, thus reducing the gas consumption.

Function	Gas	Cost (USD)
contractDeployment	5073065	7.50
setDataProducerRequest	67883	0.94
registerAsDataConsumer	115176	1.6
confirmDataProducerRequest	66081	0.92
setNewComputationRequest	85379	1.1
acceptComputationRequest	24696	0.34
setComputationResult	87787	1.22
withdrawBonus	19838	0.028

Table 2.1: Smart Contract functions cost on Ethereum

Operation	Gas	Description
ADD/SUB	3	Arithmetic operation
ADDMOD/MULMOD	8	Arithmetic operation
ADDMOD/MULMOD	8	Arithmetic operation
AND/OR/XOR	3	Bitwise logic operation
LT/GT/SLT/SGT/EQ	3	Comparison operation
POP	2	Stack operation
PUSH/DUP/SWAP	3	Stack operation
MLOAD/MSTORE	3	Memory operation
JUMP	8	Unconditional jump
JUMPI	10	Conditional jump
SLOAD	200	Storage operation
SSTORE	5,000/20,000	Storage operation
BALANCE	400	Get balance of an account
CREATE	32,000	Create a new account using CREATE
CALL	25,000	Create a new account using CALL

Table 2.2: Estimation of operation costs on Ethereum

Deployment

The deployment of the Smart Contract developed in this work has been performed on a test Blockchain, specifically the one provided by Truffle Suite, called Ganache [29]. Ganache offers a local implementation of the Ethereum blockchain with a set of accounts, which is necessary to run tests, other than

to examine blockchain responses, logs and blocks.
The deployment has to be done exactly the same way as if on the Ethereum main net, just by setting the web3 provider into a local one (via Https). For more details regarding the deployment step, please refer to Section 1.1.5.
Web3.js is a collection of libraries that allow to interact with a local or remote Ethereum node. The provider instead is how web3 talks to the blockchain. Providers simply get JSON-RPC requests and return responses, via an HTTP/IPC socket based server.

2.3.2 Multiparty Computation

The Multiparty Computation framework used in this work is called SPDZ and is owed to Ivan Damgård and Nigel Smart, among others. They have designed and implemented a general MPC protocol, which is secure against an active adversary corrupting up to $n - 1$ of the n players [17].
It is an open source project, still under development, but that already offers a fair enough flexibility and independence from the application logic to build upon it. A surely convenient tool is the SPDZ compiler, which turns high-level programs (written in a pseudo-Python language) into the proper bytecode needed by the Multiparty Computation base layer.
As long as the communication between Blockchain and MPC system is concerned, this is achieved through the official Ethereum Javascript API, Web3.js [74].
Application clients instead are able to interact with the SPDZ software through a library [69]. Since it offers several different interfaces, in this project, it is used the one to the SPDZ Proxy web socket APIs, the most reactive one, where the SPDZ Proxy pushes data as soon as it becomes available.

Architecture

The SPDZ architecture is made of three main components:

- **Engines:** they are the parties actually involved in the multiparty computation.
- **Proxies:** they run in front of a single engine, providing a web friendly interface that allows external clients to provide input and receive output. Each proxy interact with clients either via Web Sockets or via HTTPs, and with the engine via a stateful TCP socket connection. Moreover they implement the appropriate encoding/decoding scheme into/from the binary formats expected by SPDZ.

- **Clients:** end user applications that wish to interact with the SPDZ software. They communicate with proxies to send inputs and receive outputs again either via Web Sockets or via HTTPs.

Protocol

The SPDZ protocol consists of a pre-processing phase (or offline phase) that is both independent of the function to be computed and of the inputs, and a much more efficient online phase where the actual computation takes place. The online phase is claimed to be unconditionally secure and has total computational (and communication) complexity linear in n, the number of players. The offline phase instead is based on a somewhat homomorphic cryptosystem, with a computational complexity of $O(n^2/s)$ operations per secure multiplication, where s is a parameter that increases with the security parameter of the cryptosystem. Clearly, the complexity here is dominated by the public-key operations of distributions of the pre-processed data between the computing parties.
In the following sections secret shared values will be signed by inserting such values into $\langle\rangle$ delimiters.

Offline Phase

The purpose of this phase is to produce batches of data that are necessary during the computation. This stage is executed by a *Trusted Party*, that will eventually produce five batches of random data, each of which satisfying some properties:

- **Multiplication Triples:** triples of a, b, c values, such that $c = a \times b$. These triples are used during the computation to multiply secret shared values according to the Beaver technique (Section 2.3.2);

- **Input:** random single values r that are used by a client to secret share his data before sending them over as input of the computation;

- **Square:** pairs of a, b values such that $b = a^2$, useful to perform squaring of shared values more efficiently than general multiplications;

- **Bit:** single bit values $\{0,1\}$ that will be used to compute bitwise boolean operations;

- **Inv:** pairs of a, a' such that $a' = a^{-1}$, used for constant round protocols for integer comparison operations.

After having completed this step, the Trusted Party secret shares the batches between the engines (multiparty computation servers), delivering to each of them five files containing shares of the values divided as described above, through a public-key scheme. So it's clear that these are independent both from the input secret-shared values they are used for and from the circuit to be evaluated. It is intended that the needed size of the batches produced in this phase cannot be predicted a priori without knowing the function to compute. Nonetheless as soon as the batches have been exhausted, this step will be run again to produce some more.

Online Phase

In the Online Phase the actual multiparty computation takes place: parties compute on open values (i.e. values which are not secret shared) and secret values (i.e. values which are secret shared).

Input The data sent as input by a Data Consumer are sent under a secret-sharing scheme, and as such they never run over the network in clear. Let's say that he wish to input x^i; obviously it must be secret shared before being received by proxies, and this is achieved as follows:

1. When a new input request is received, each proxy sends a new $\langle r \rangle$ to the Data Producer.
2. As soon as the Data Producer receives all $\langle r \rangle$ by all proxies, he calculates:
$$r = \sum \langle r \rangle$$
3. Then he broadcasts to all proxies $\epsilon = x^i - r$
4. Now party P_1 sets its share of x^i as $x^i_1 = r_1 + x^i - r$ while P_j for $j > 1$ sets it as $x^i_j = r_j$.

Even though it might seem counter intuitive to send the same data to all the proxies, this data is secret shared. Indeed none of the proxies could retrieve the clear data, having just a share of the value r used to mask the input. So at this point proxies have a secret shared values, that they store in an external database for later use.

Addition Since the input data have been secret shared according to an additive scheme, the following properties holds for additions:

- Having two parties some secret-shared field elements $\langle a \rangle$ and $\langle b \rangle$, each party P_i can compute locally $a_i + b_i$ obtaining so a secret sharing $\langle a+b \rangle$ of the value $a + b$;
- Each party can multiply its share a_i by some public value α, obtaining a share of $\langle \alpha a \rangle$, since $\sum_i \alpha a_i = \alpha \sum_i a_i = \alpha a$

Being the secret sharing linear, no communication costs are required for such operations.

Multiplication The intuition on how to multiply secret shared data comes from Donald Beaver, in 1991 [6], and it's the one used in the SPDZ implementation. Suppose we need to compute $\langle xy \rangle$, it works as follows:

- First off parties need to already have the so called multiplication triples, produced in the offline phase as described above, i.e triples of a, b, c values, such that $c = a \times b$.
- Each party calculates and broadcasts to the others $\alpha = x_i + a_i$ and $\beta = y_i + b_i$
- At this point each party holds a secret shared value of $\langle xy \rangle$, because:

$$\langle xy \rangle = (\alpha - a_i)(\beta - b_i) = \alpha\beta - \beta a_i - \alpha b_i + c_i$$

The multiplication is so composed of three primitive operations of local additions, opening of values and consuming pre-processed data. It's important to say that a multiplication triple cannot be reused because this would reveal information about the secrets we are trying to multiply (since $x - a$ and $y - b$ are made public in the process above).

Implemented functions

The actual functions to be carried out under a multiparty computation scheme have been written in a special Python-like language, provided with the SPDZ ecosystem, and subsequently compiled down to a bytecode that can be interpreted by a virtual machine, which instead is implemented in C++. The library provides all kinds of functions needed when dealing with MPC calculations, such as for creating, storing and loading both clear and secret shared data types, or for retrieving pre-processed data (cited in section 2.3.2).
For the sake of implementing functions that are relevant to the use-case chosen and significant in order to conduct some testing of the performances,

other than the traditional statistical functions (i.e. max, min, mean), it has been implemented the *Chi-squared test* [13].
This kind of test is used widely to determine the degree of correlation between an expected model of frequencies and the observed ones for a certain type of data, according to the formula:

$$X^2 = \sum_{i=1}^{k} \frac{(o_i - e_i)^2}{e_i}$$

where $o_1, o_2, ..., o_k$ are the observed frequencies, $e_1, e_2, ..., e_k$ the expected ones and k is the number of classes in which the models are divided. Clearly, $X^2 = 0$ indicates that the observed frequencies correspond exactly to the expected one, while the higher X^2 the higher the discrepancy with the model. Once the chi-square value is calculated, the observer has to fix the number of *degrees of freedom*, which corresponds to $k - 1$, and then find the corresponding *p-value* from the chi-square distribution table [14]. By means of the p-value found, the correspondence between the observed and the reference model is probabilistically evaluated, similarly to other statistical test based on the concept of null hypothesis.
The data hypothetically taken into account in this work are related to the HDL-cholesterol, which in literature is known to follow a Gaussian Distribution, with mean value $\mu = 57$ (mg/100ml) and standard deviation $\sigma = 10$ (mg/100ml) [33].
The reference model instead has been created as follows. First the distribution has been arbitrarily divided into ten intervals, eight of which belonging to the $\mu \pm 2\sigma$ area, and the other two covering the tails. Each interval extreme x has then been translated in the corresponding Z score in the standard normal distribution [3], according to the formula:

$$Z = \frac{x - \mu}{\sigma}$$

Then by checking the standard normal distribution table [54], also its frequency is obtained.
As long as the observed frequencies are concerned, each Data Producer's input has been used to increment the number of occurrences of the interval it belongs to. Of course, such inputs have been generated through a script, randomly but according to the HBL-cholesterol distribution, so to obtain pertinent data.

[3]The normal distribution is a special Gaussian distribution with $\mu = 0$ and $\sigma = 1$

2.3.3 Decentralized Application

Including the blockchain into the traditional application stack brings up a set of new challenges around the design of the architecture and the layout of the decentralized application, or dApp.
Generally speaking a dApp can be designed in two different ways. The simplest one is serverless, where the entire application flow happens between the client and the blockchain, since the latter can substitute a traditional server as long as permanent storage, business logic, and client coordination are concerned. On the other side though, off-chain services cannot interact directly with on-chain code, so if the application needs to be integrated with third-party services a server is also required to complete the architecture.
In this particular use-case, although the entire application logic is managed by the smart contract, the integration with the MPC system requires a server to be added to the stack. The most delicate work, since both client, server and MPC system interacts with the blockchain, is to coordinate them seamlessly to achieve consistency, other than a user-friendly interface. In Figure 2.8, the architecture of the implemented dApp is reported.

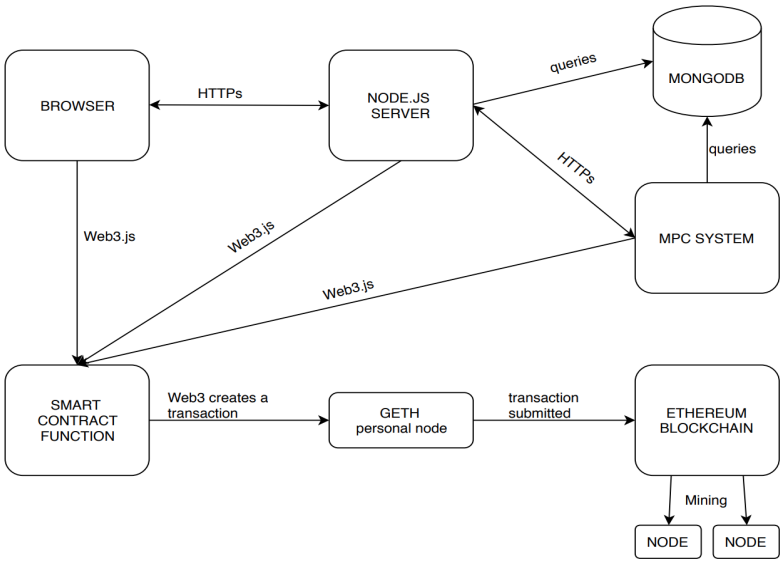

Figure 2.8: Decentralized Application Architecture

Communication with the blockchain

The coordination through the blockchain of clients (DataProducers and DataConsumers through browser) and servers (Node server and MPC system), is obtained by means of the so called Solidity Events. These (along with their

parameters) are emitted by explicitly coding this instruction inside a Smart Contract function, so that when the function is run, nodes which subscribed to that specific event intercept that, and often a callback is triggered. It is also possible to filter the subscription according to the event parameters: usually clients filter their subscription only to events related to themselves, while servers listen to all of them. Different events have been implemented in this work, for a complete list please refer to Section 2.3.1.
Figure 2.9 reports an example of what just said. First, through the browser, a DataConsumer append to the blockchain a dataConsuming request, which contains the instruction for emitting a specific event. This is intercepted by MPC servers, that needs to accept the request (and send their security deposit) in order to take part of it. When the needed number of servers have accepted, another event is emitted and intercepted by the DataConsumer in the same way, telling him that the computation (which will occur off-chain between the Node server and the MPC system) is allowed to take place.
The drawback of such a system reflects the blockchain forking probabilities. Due to chain splitting or forking issues, reacting right after a transaction containing an event is mined might eventually lead to an invalidation of such transaction. A good practice is to wait for a fair enough amount of blocks to be appended before acting off-chain, and in the meantime update the user-interface to let him know of what's going on underneath.

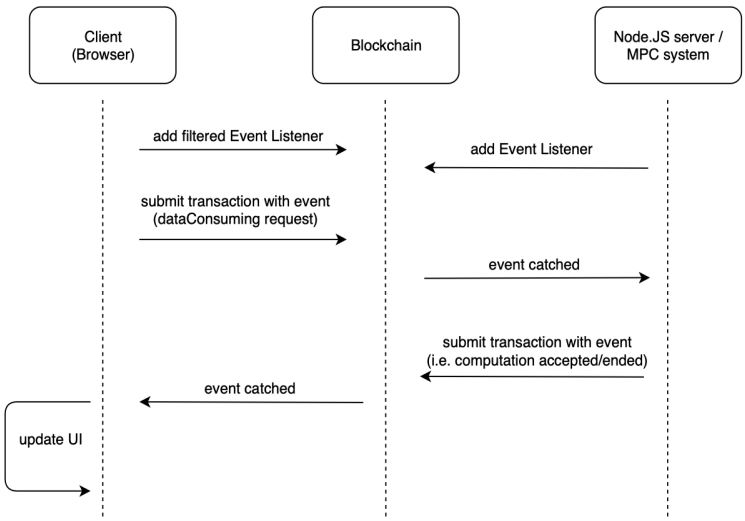

Figure 2.9: Event-based communication in the Data Consuming phase

Front-end

The front-end has been built by using the modern React.js and Redux.js Javascript frameworks [61] [62]. React has been designed by the Facebook team to create interactive user interfaces, by means of class-based components. It applies to web pages the concept of application state, in order to make components render their views according to state changes, without the need to refresh the page. Considering the high amount of state changes in the application, coming both from the blockchain and from the MPC system, a traditional server-side rendering would have lead to continuous page refreshing and a poor user-friendly interface. Besides, being actually objects, components come up with integrated life-cycle hooks: pre-defined methods that are triggered following the component life-cycle. For example, inside the *componentDidMount()* method, which is triggered once right after the component is inserted in the DOM, all the events listeners and fetching operations from the blockchain may be performed.
The state in React applications comes at two levels: component-level state and application-state. If the former is a kind of component local state, bound for example to on-screen actions (buttons, links) the user perform and that cannot be seen by other components, the latter is used to synchronize components between them other and with server upon a single-source state in more complex applications.
In this work the application state is needed to manage the concept of authentication and session of users: when a user logs in, the back-end server will update the state by associating to a unique ID (corresponding to the client's cookie) the authentication status and any other information needed, and components check that authentication status and render the view accordingly.
Redux is the framework that implements the concept of application state, designed to operate in React/Node Javascript applications.

Back-end

The back-end server is implemented by means of Node.js and Express.js [52] [25]. Node is an asynchronous event driven JavaScript runtime, designed to build scalable network applications. Express instead is a flexible and lightweight framework for Node.js web-based applications, that provides an easy way to create APIs.
In this context, the back-end is responsible of managing the authentication flow (through the integration with Passport.js middleware [57]), that hap-

pens off-chain with the credentials saved in a NoSql database (MongoDB), other than the communication with the MPC system.
The used model for communication with the front-end is REST (REpresentational State Transfer): both in case of a data sharing and a data consuming, after having obtained the authorization by the blockchain, the client initiates through the browser an HTTPs post request to the back-end, which will manage the interaction with the MPC system (spawning a new process), and finally sending back the client the responses always via HTTPs.

2.4 Application Snapshots

This section is dedicated to the presentation of some application screenshots of the most relevant states. For convenience, it has been split into those related to the blockchain explorer interface, to the Data Consumer dashboard and to the Data Producer's one.
For the sake of simplicity, in this context the multiparty computation system is composed only by two parties. Besides, a bug has been discovered in the local blockchain software Ganache [29], that results in an hash collision for those transactions made separately by the MPC parties: probably this is due to the fact that the public key of the sender is not included as well as other data (i.e. timestamp, function header, receiver, value) in the hash calculation, and since transactions by the MPC are identical except for the sender, the hashes are the same. Given that the transaction details are retrieved from its hash, in the table this produces the same transaction retrieved twice.

2.4.1 Blockchain Explorer

A blockchain explorer has been implemented inside the application, in order to decode hexadecimal data coming from blockchain's blocks into a more user-friendly format. This component keeps track of the approved Data Producers and Consumers (Figure 2.10), of the transaction history (Figure 2.11), and of the contract and MPC parties' balances. Clearly this last information should remain private, but for this purpose it is instead useful to understand the flow of cryptocurrency during a computation.

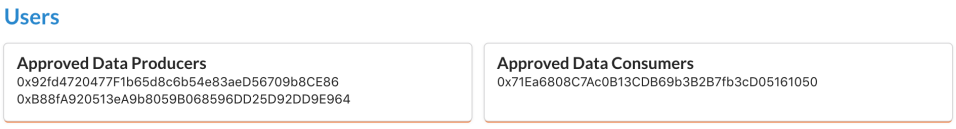

Figure 2.10: Approved Data Producers and Data Consumers Public Keys

Transactions history

Trx hash	Block	From	To	Value	Gas	Method
0x0058c469...	20	0xc3001e08...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	0 Eth	179886	setComputationResult
0x0058c469...	20	0xc3001e08...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	0 Eth	179886	setComputationResult
0x2fa8759c...	22	0xc3001e08...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	0 Eth	19838	withdrawBonus
0x2fa8759c...	22	0xc3001e08...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	0 Eth	19838	withdrawBonus
0xbe432921...	23	0x71Ea6808...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	1 Eth	85379	setNewComputationRequest
0x6b3bb3c6...	25	0xc3001e08...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	1 Eth	57057	acceptComputationRequest
0x6b3bb3c6...	25	0xc3001e08...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	1 Eth	57057	acceptComputationRequest
0xeddadb07...	26	0x71Ea6808...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	0 Eth	92932	refundComputation
0x326ef439...	28	0x5bf328D2...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	0 Eth	19838	withdrawBonus
0x326ef439...	28	0x5bf328D2...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	0 Eth	19838	withdrawBonus

Figure 2.11: Transactions history

As we can see from the Picture 2.11, for each transaction it is reported its hash, the block number where it is included, the sender and receiver public keys, the amount in Eth sent along with the transaction, the Gas consumed and the Smart Contract method called. The receiver here is always the Smart Contract address, because users interact only with it, while the value field is greater than zero only for those function calls that are related to a multi-party computation, where a security deposit is required to be sent along by the caller.
As long as the Smart Contract method in the last column, this is achieved by reading the *input* field of the transaction: the first 4 bytes of it are called the *function signature*, and are nothing more than the first 4 bytes of the Keccak-256 hash of the function header (including its parameter types). So, during the compilation of the Smart Contract, the hash table that maps each function signature with its name is created, and then used to obtain from a transaction the Smart Contract method called.

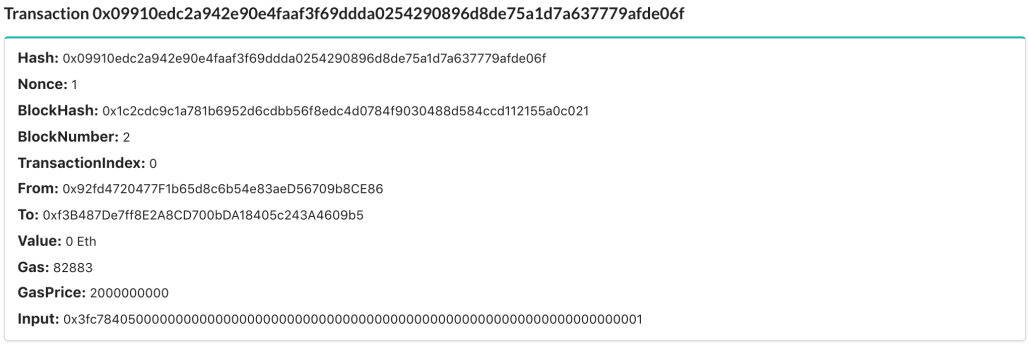

Figure 2.12: A transaction in detail

By clicking on the transaction hash, another component shows the transaction fields (Figure 2.12). As we can see, the first 4 bytes of the input field constitutes the aforementioned function signatures. The reader might as well check here [4] that:

```
3fc78405=keccak256(setDataProducerRequest(uint256)).substr(0,8)
```

2.4.2 Data Producer

Figure 2.13 shows the Data Producer's dashboard, clearly right after he has submitted his data and has been approved by mpc parties (Figure 2.14). From the dashboard, the user sees the list of computations that used his data among the others, and he is able to know the public key of the Data Consumer, the date and the computation outcome.

[4] `https://emn178.github.io/online-tools/keccak_256.html`

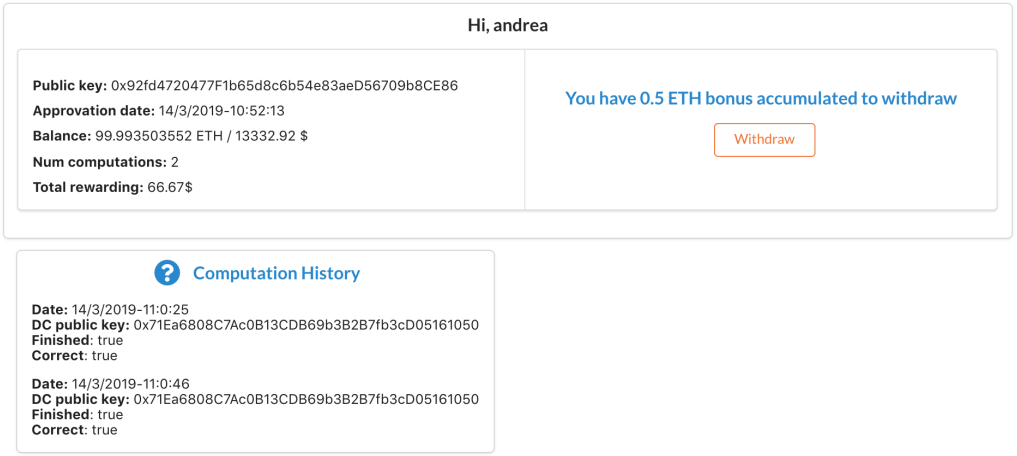

Figure 2.13: Data Producer dashboard

Besides, the top-right box invites the user to withdraw from the contract the rewarding accumulated with the computations. The button will trigger the *withdrawBonus* Smart Contract function, that will transfer the Ether value, which inside the contract storage is associated to the user's public key, from the contract's wallet to the user one.
Figure 2.14 reports instead the sequence of transactions involved in the Data Sharing phase: the user sets, through a simple form in his dashboard, a request containing a Solidity Event, that will activate MPC servers for the off-chain data sharing, and upon completion they will singularly confirm that particular request. As soon as the contract receives all the confirmations (in this case two), the user is approved.

Transactions history

Trx hash	Block	From	To	Value	Gas	Method
0x09910edc...	2	0x92fd4720...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	0 Eth	82883	setDataProducerRequest
0x6474ee2e...	4	0xc3001e08...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	0 Eth	143894	confirmDataProducerRequest
0x6474ee2e...	4	0xc3001e08...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	0 Eth	143894	confirmDataProducerRequest

Figure 2.14: Data Sharing transactions

2.4.3 Data Consumer

Figure 2.15 shows the Data Consumer's dashboard. From the bottom-right panel, the DC is able to ask the MPC system for a specific computation and get back the result on the same panel, just by clicking the orange button. On

the bottom-left side instead, the computations he asked for are listed out, again with their final outcome. For example, it is reported that the last one went badly, with an output that is not correct. This can be due to many things, from an abort of the MPC system due to malicious actors to a simple network error. Either way, when the computation is not correct, the smart contract reserves the Data Consumer the caution he deposited. He is then prompted to withdraw it just as for the Data Producer by clicking on the button.

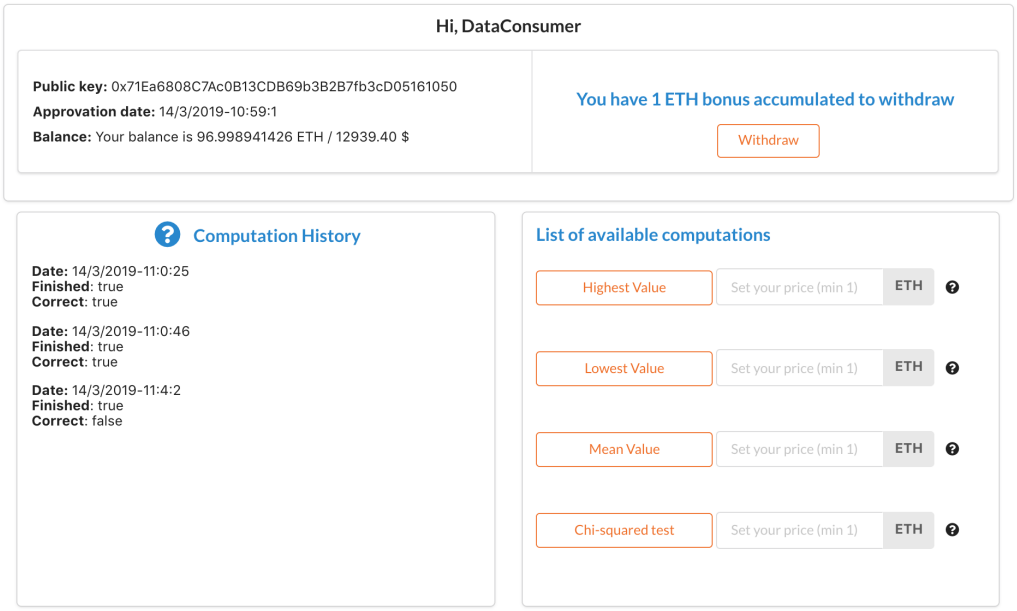

Figure 2.15: Data Consumer dashboard

Finally, in Figures 2.16 and 2.17 are shown the sequences of transactions occurring in the context of a Multiparty Computation both in the case of negative and positive outcome. As one can see, both start with the Data Consumer setting a new computation request and sending his deposit of 1 Eth along. Right after, the MPC servers that intend to take part to the computation accept the request, as well sending 1 Eth along. Clearly at this point the Smart Contract's wallet detains 3 Eth. As soon as the required number of computing parties have done so, an Event is triggered, and the computation takes place off-chain in a secure channel between Data Consumer and MPC system.
Then, in the first of the two figures we may see a call for *refundComputation* from the Data Consumer, right after an error in the communication with the MPC system occurred. This call fires an event which is caught by computing

parties, that will get their deposit back right after. Evidently this is a naivë approach: given the high latency between the call and the effective event to be triggered on the blockchain, a Data Consumer would be able to receive the result off-chain and at the same time set an interruption call, with his deposit received back; being this work just a proof of concept, this scenario has been omitted in the implementation.

Transactions history

Trx hash	Block	From	To	Value	Gas	Method
0xbe432921...	23	0x71Ea6808...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	1 Eth	85379	setNewComputationRequest
0x6b3bb3c6...	25	0xc3001e08...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	1 Eth	57057	acceptComputationRequest
0x6b3bb3c6...	25	0xc3001e08...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	1 Eth	57057	acceptComputationRequest
0xeddadb07...	26	0x71Ea6808...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	0 Eth	92932	refundComputation
0x326ef439...	28	0x5bf328D2...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	0 Eth	19838	withdrawBonus
0x326ef439...	28	0x5bf328D2...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	0 Eth	19838	withdrawBonus

Figure 2.16: Data Consuming transactions (negative outcome)

In Figure below instead it is shown the flow of transactions when the computation happens correctly. Both MPC computing parties set on the blockchain separately the Keccak-256 hash of the result they obtained, and right after the Smart Contract receives the last result, it does the integrity check, and if positive it will automatically share the security deposit among Data Producers and computing parties, and will finally trigger the appropriate Event. The last two transactions refer to the MPC parties that withdraw their rewarding after having been triggered by the Event.

Transactions history

Trx hash	Block	From	To	Value	Gas	Method
0xa00f5ed0...	16	0x71Ea6808...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	1 Eth	85379	setNewComputationRequest
0x2dc488a9...	18	0xc3001e08...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	1 Eth	57057	acceptComputationRequest
0x2dc488a9...	18	0xc3001e08...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	1 Eth	57057	acceptComputationRequest
0x0058c469...	20	0xc3001e08...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	0 Eth	179886	setComputationResult
0x0058c469...	20	0xc3001e08...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	0 Eth	179886	setComputationResult
0x2fa8759c...	22	0xc3001e08...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	0 Eth	19838	withdrawBonus
0x2fa8759c...	22	0xc3001e08...	0xf3B487De7ff8E2A8CD700bDA18405c243A4609b5	0 Eth	19838	withdrawBonus

Figure 2.17: Data Consuming transactions (positive outcome)

2.5 Performances

Some tests have been conducted against the MPC system, in order to understand how it scales according both to the data input size and to the number of computing parties. Blockchain operations have not been considered in this, as they don't happen during a computation but only before the beginning and at the end of it, once for each computing party. The timing of such operations varies along with the block time of the chosen blockchain, that is the time for the network to produce and broadcast a new block including the transaction: Ethereum has an average block time of 17 seconds.
Two bash scripts have been written for this purpose. The first one simulates a data sharing phase for the generation of the secret-shared input data (Section 2.3.2), according to the passed number of inputs and the number of computing parties. The second one simulates multiple data consuming phases, where the computed program is the chi square test. The program is run repeatedly varying the number of inputs, from a minimum of 250 to a maximum of 1000 (128 bits for each input) with an increase of 50 for any iteration, and the number of computing parties, from 2 to 5. For each iteration a specific log file is printed with the time of the computation, the total megabytes of data sent and the total rounds of communication between parties, along with the amount of pre-processing data used (Section 2.3.2).
For convenience, during the tests all computing parties were located on the same machine, a 2015 MacBook Pro with a 2,7 GHz Intel Core i5 Processor and a 8 GB 1867 MHz DDR3. This influences the total time for a computation, as the latency of each communication message in this setting is null, and so the numbers related to that have to be read bearing this is mind. Moreover, the total rounds of communication and the amount of megabytes sent among parties have been analyzed.

2.5.1 Communication Model

As already stressed in Section 2.3.2, a round of communication between parties during the computation is required for each multiplication (other than in the beginning and in the end of the computation), where parties randomize their secret shared values by means of a *Beaver multiplication triples*, produced in the offline phase. This step instead is not required for additions, that can be computed locally. From the SPDZ technical document [37], we read that the communication between parties of their value a_i for a multiplication has been designed as follows :

1. The players pick an arbitrary *nominated player*, Player P_1 for example;

2. Players P_i for $i = 2, ..., n$ send a_i to P_1;

3. Player P_1 computes $a = a_1 + ... + a_n$ and sends a to all players.

Clearly, for those parties that are not the nominated one, this operation has a complexity which increases linearly with k, the number of inputs, that will in turn influence the number of multiplications. Differently, the nominated player sees his computational load increasing also linearly along with the number of parties n (excluding himself, with $n \leq 5$) and the inputs k (with $k \leq 1000$).
The asymmetric solution adopted prevents computation to be distributed and carried out in parallel, and doesn't speed up operations, because parties have to wait for all the others to having sent their value to the nominated one before proceeding with the computation. Nevertheless, it can be very convenient in those settings where the nominated player either has more computational power than the others, or is connected through higher speed channels with the others, thus optimizing the available resources.

2.5.2 Results

Given the above considerations, in Figure 2.18 it is reported the trend of the total amount of megabytes sent by the nominated player during a chi square computation, according to a variable number of inputs and of computing parties. Such numbers confirm what's written in the technical documentation [37], namely that the system scales linearly to the number of inputs and to the number of players.
As we can see, in a setting of 1000 inputs and 5 players, the nominated player has to send less than 140Mb of data, which is considerably doable in few seconds in a fiber optic network.

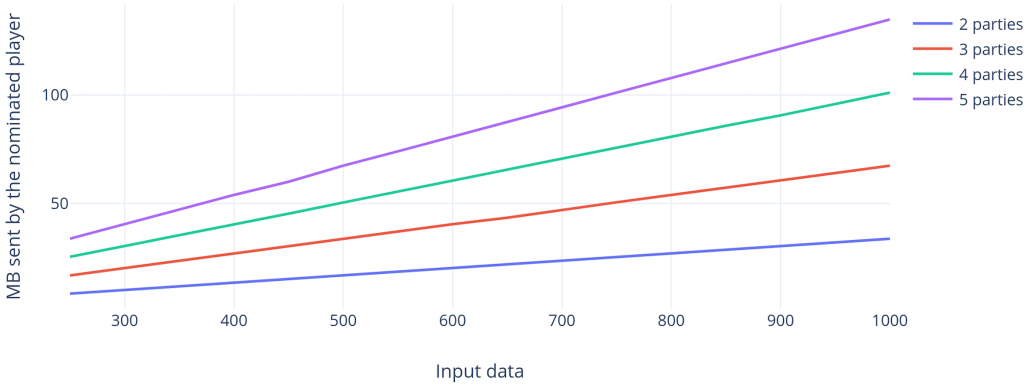

Figure 2.18: The total megabytes sent by the nominated player

Figure 2.19 shows instead the amount of time from the start to the end of a computation. As we can see, results indicate that the computation is carried out in the order of seconds. Again, the reader should keep in mind that these numbers are related to a best-case scenario, since they are obtained in a configuration where the latency of the communication between parties is null. Nevertheless, considering from the graph above the rather low amount of data to be exchanged, these figures provide confidence that the system could conduct a computation in a reasonable amount of time with tens or even hundreds of thousands input data.

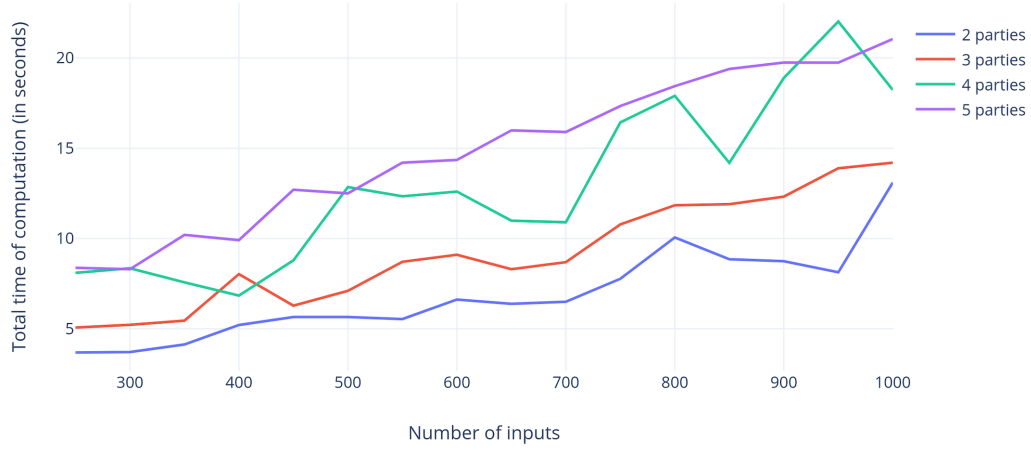

Figure 2.19: The time in seconds of a chi square computation

2.6 Security Model

In this model the attacker is someone who manages to "corrupt" at least one of the k parties involved in the computation, in order to control or alter the messages sent to the other parties, trying to obtain extra information, and more importantly trying to convince the other parties about the correctness of an output that's actually wrong.

2.6.1 Assumptions

In this section the assumptions taken during the construction of the system are listed out below.

- **Honest party [ASS-1]:** this assumption states that at least one party involved in the MPC computation is honest, that is he acts following the protocol. This assumption doesn't require that the honest party would be the same at each multiparty execution;
- **Blockchain properties [ASS-2]:** this assumption states that the Blockchain security properties are always valid;
- **Input data [ASS-3]:** this assumption states that the input data supplied to the system and used in the computations are truthful and homogeneous (standardized) .
- **MPC communication [ASS-4]:** this assumption states that the parties involved in the multiparty computation exchange their messages in a secure network (i.e. via HTTPS).
- **Registered parties [ASS-5]:** this assumption states that all the actors in the system are regularly registered to the Blockchain, that is they have a wallet with some funds that never below a given threshold. This because each new execution of the protocol is allowed only if parties involved put a security deposit at stake, which will be later used to penalize dishonest ones and reward honest ones.

2.6.2 Requirements

In this section the requirements, both functional and security, the system must have, are listed out. Each of them will be extensively analyzed in Section 2.6.3.

Functional Requirements

These requirements concern the functions the system should guarantee, precisely:

- **FUNREQ-1:** a data producer must be able to provide his data to the system;
- **FUNREQ-2:** a data producer must be rewarded every time his data is used in a computation;
- **FUNREQ-3:** a data producer must be able to set and update the permissions over his data. Such permissions may specify for example for which data consumer make data available for analysis;
- **FUNREQ-4:** a data consumer must be able to require a computation;
- **FUNREQ-5:** a data consumer must be certain that if he pays for a computation, then the output he obtains will be the correct one. This implies that the system must be able to abort when some party misbehaved;
- **FUNREQ-6:** a computation must not be started without having left a security deposit;
- **FUNREQ-7:** a computation must not be started without having done a request through the Blockchain;

Security Requirements

These requirements concern the security properties that the proposed system should guarantee, precisely:

- **Correctness of computation [SECREQ-1]:** a data producer must obtain correct output from a computation, i.e. given some input the computation is executed according to the protocol.
- **Privacy of sensitive data [SECREQ-2]:** sensitive data must be stored maintaining the privacy towards all actors in the system.
- **Non repudiation [SECREQ-3]:** an actor in the system must not be able to deny of having taken some action.
- **Authentication [SECREQ-4]:** all actors involved in the system must be known in advance and all actions they could take must be authenticated.

- **Independence of input [SECREQ-5]:** parties must not be able to choose their inputs as a function of other parties' inputs.
- **Fairness [SECREQ-6]:** if one party learns the output, then all parties learn the output.
- **Covert security [SECREQ-7]:** the system guarantees covert security with a good enough probability of detecting dishonest parties.
- **Passive security [SECREQ-8]:** the system guarantees that an honest but curious attacker must not leak out sensitive information.

2.6.3 Solution

- **SECREQ-1:** this requirement is guaranteed through the MAC checking phase of the MPC protocol (SPDZ) that is used in this context, given the assumption [ASS-1]. Of course the output is bound to the input given to the computation, so we can say that the computation is really correct given the assumption [ASS-3]. Either way, the system is made up so to abort a computation in case of a dishonest part, avoiding so to output a wrong value.
- **SECREQ-2:** this requirement is guaranteed through the mathematical properties of the Additive Secret Sharing scheme upon which sensitive data are stored in the system, given the assumption [ASS-1]. Only by having all parties corrupted an attacker could be able to reconstruct those data.
- **SECREQ-3:** this requirement is guaranteed given the assumption [ASS-2] and [ASS-5]. In fact all actions taken in the system pass through the Blockchain, which digitally sign all of them.
- **SECREQ-4:** this requirement is guaranteed given the assumption [ASS-2] and [ASS-5]. Non registered actors wouldn't be able to perform any action, other than simply read the Blockchain (that is public), as authentication and authorization are performed by the Blockchain in use.
- **SECREQ-5/SECREQ-6:** these requirements are guaranteed by the MPC protocol used in this solution (SPDZ), with the assumption [ASS-1].

- **SECREQ-7:** in this scenario the attacker is an adversary who deviates from the protocol only with a low probability of not getting caught. The SPDZ protocol offers covert security again through the MAC checking phase, given the assumption [ASS-1]. The probability an attacker could get away with cheating is the same he would have to guess the global MAC key, that is $\frac{1}{\mathbb{F}_p^k}$

- **SECREQ-8:** in this scenario the attacker does not deviate from the protocol, but tries to extract information that he wouldn't know otherwise. The outcome of such an attack would be an information leakage. Given the domain of the application, related to sensitive and personal data, this attack may constitute a real threat. The MPC solution adopted (SPDZ) is not itself secure against a passive attacker. Nonetheless, security against a passive adversary is guaranteed by the mathematical properties of the secret sharing on which the sensitive data are stored, given the assumption [ASS-1].

Chapter 3

Conclusions

This thesis work represents an experimental dive into uncharted waters, in order to hopefully acknowledge the reader that the world wide web is probably going through another huge transformation. The blockchain technology, in all its implementations, from the first secure, peer-to-peer electronic payment system [50], to the visionary peer-to-peer general purpose worldwide computer [24], is leading an increasingly larger community towards an entirely new set of applications. Some might see the correlation with the Internet in 1994, when the technology emerged, but actually everyone was clueless about where it was heading.
Apart from the technological tools it relies upon, that without any doubt are not definitive and require great effort in terms of research from the most brilliant computer scientists and cryptographers, the blockchain's unprecedented innovation consists in the inclusion of economic theory principles inside computer protocols. The term *cryptoeconomics* have been coined to describe the union between cryptography and economics, in order to build decentralized marketplaces and applications, for an entirely new and even unimaginable set of services.
This work tries to formalize one of such services, by combining the model of incentives enabled by the Blockchain with the privacy preserving distributed computing of the Multiparty Computation. It is demonstrated how each actor involved in the system could be incentivized to act according to the protocol, assuming him to be rational and so looking first at maximizing his profit.
Nonetheless, this is just the beginning: many tough challenges need to be undertaken in order to overcome the few yet significant technical limitations that both technologies still presents, around which the skepticism of many is rightly grounded. As usual, it will be the push of innovation and science to drive this revolution towards a more democratic and righteous version of the

internet.

3.1 Related Works

The projects that aim at integrating Blockchain and Multiparty Computation are few and still under development. Nonetheless, two of them claim to provide a working solution, and will be reported briefly here below. Both share the vision of building a platform for data analysis that puts control and monetization back in the hands of data owners.

3.1.1 Decentralized Computation Platform with Guaranteed Privacy

Enigma [1] was born thanks to the groundbreaking research of Guy Zyskind, that in 2015 was the first to theorize the union between Blockchain and MPC, at MIT. Enigma claims to use an optimized version of MPC, guaranteed by a verifiable secret-sharing scheme and a distributed hashtable for the storage of secret-shared data. Enigma consists of a privacy layer (Enigma Network) on top of the blockchain: they coined the term *secret contracts* for those smart contracts that are located in the Enigma Network, that is responsible of running the code while keeping input data hidden to underlying blockchain nodes. On top of this they provided an application layer, to allow programmers to create decentralized applications that exploits the Enigma Network.
The first application they provide is called Catalyst [2], and it allows to analyze the performances of a particular trading strategy (of crypto-assets) against historical data, and the live-trading of cryptocurrencies already in four major exchanges.

3.1.2 Secure infrastructure for data exchange

Insights Network [3] and Partisia [4] are working together to construct a decentralized platform that will allow to conduct marketing researches, other than allowing users to securely store their data, by the combination of Blockchain and MPC. Although still in early stages, the platform has included in the

[1] `https://enigma.co/`
[2] `https://enigma.co/catalyst/`
[3] `https://insights.network/`
[4] `https://partisia.com/`

ecosystem the iOs app Gambeal [5], an application that rewards restaurants' owners for submitting surveys related to their activities. The migration towards a Blockchain-MPC solution will allow for instant payments through cryptocurrencies, rather than via PayPal, other than ensuring an higher level of privacy for users by means of a MPC-based authentication, rather than the traditional one occurring via Facebook.

3.2 Future developments

3.2.1 Dishonest Parties Identification

As at its current implementation, the Smart Contract is not able to individually identify among computing parties honest and dishonest ones. Even though purely in theory, the actual dispute resolution between parties in case of bad outcome is needed to happen off-chain.
Without any doubt, in a real world use-case this wouldn't be much convenient: as one of the base requirement for the entire system, which guarantees both the input privacy and the output correctness, is that MPC servers don't collude between each other, it would be appropriate to have computing parties belonging to different organisations and companies, in order to maintain the highest decentralization degree as possible. It wouldn't be acceptable then, to have honest players penalized.
Given such considerations, it would be necessary for the smart contract to receive from players some kind of proof that the computation had been made correctly. Since the proof ideally should be verifiable at any given time in the future, zkSNARKs (seciton 1.1.6) are a good candidate of zero-knowledge proofs that could be applied here.

3.2.2 Reputation System

Once a more precise dishonest parties identification is integrated into the Smart Contract, it will be possible to implement, always through smart contracts, a reputation system of the computing parties, such that after every computation a score indicating the honesty of the party is updated accordingly. Such a system would increase the security against covert adversaries 1.2.1, those adversaries that would be active but are worried about the consequences (for instance the loss of reputation) of being caught cheating.
Moreover, recalling the base security assumption that at least one computing party must be honest in order to guarantee the security of the system, the

[5] `https://gambeal.netlify.com/`

reputation score could be used to influence the choice of the set of computing parties at each computation, in order to minimize as much as possible the probability to have the entire set malicious.

3.3 Acknowledgements

I would like to thank my mentor Guglielmo *Coach* Morgari for having significantly helped me all along this thesis period with great expertise, kindness and wisdom. I consider myself very lucky for the opportunity I had to work by his side and to learn from his considerable experience. Thanks also to all the other people of Telsy SpA, for the stimulating and pleasant environment where I spent several months.
Many thanks to my professor Danilo Bazzanella, for having always been present when needed and for having connected me to Telsy.
To my family, for the unimaginable sacrifices made to support my studies for six years with love and trust, there aren't words to express my gratitude.
To my true friends, for having believed in me when I didn't, without your affection this journey would have been impossible.

Bibliography

[1] Algorand website. https://www.algorand.com/.

[2] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. 2009. https://eprint.iacr.org/2007/060.pdf.

[3] Adam Back. A partial hash collision based postage scheme. 1997. http://www.hashcash.org/papers/announce.txt.

[4] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the impossibility of obfuscating programs. 2010. https://www.iacr.org/archive/crypto2001/21390001.pdf.

[5] Countries that consume more or less electricity than bitcoin mining in late 2018. https://powercompare.co.uk/bitcoin-mining-electricity-map/.

[6] Donald Beaver. Efficient multiparty protocols using circuit randomization. *CRYPTO 1991: Advances in Cryptology*, 1991.

[7] Bitcoin wiki. https://en.bitcoin.it/wiki/Transaction.

[8] Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis. 2011.

[9] Joseph Bonneau. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. 2015. Published in: 2015 IEEE Symposium on Security and Privacy.

[10] Vitalik Buterin. On stake. 2014. https://blog.ethereum.org/2014/07/05/stake/.

[11] Vitalik Buterin. On sharding blockchains. *Ethereum Wiki*, 2018. https://github.com/ethereum/wiki/wiki/Sharding-FAQs#this-sounds-like-theres-some-kind-of-scalability-trilemma-at-play-what-is-this-trilemma-and-can-we-break-through-it.

[12] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. 2017.

[13] Chi-squared test - wiki. https://en.wikipedia.org/wiki/Chi-squared_test.

[14] Chi-square distribution tables. http://www00.unibg.it/dati/corsi/40025/74822-tavola_chi2.pdf.

[15] Coinmarketcap website. https://coinmarketcap.com/it/.

[16] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel Smart. Practical covertly secure mpc for dishonest majority - or: breaking the spdz limits. 2013.

[17] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. 2011. https://eprint.iacr.org/2011/535.pdf.

[18] The dao attack - hacker letter. https://pastebin.com/CcGUBgDG.

[19] David archer - galois inc. https://galois.com/team/david-archer/.

[20] Do you need a blockchain? - web application. http://doyouneedablockchain.com/.

[21] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. *Crypto '92*.

[22] Decentralized dns. https://ens.domains/.

[23] Etherscan token tracker. https://etherscan.io/tokens.

[24] Ethereum whitepaper. https://github.com/ethereum/wiki/wiki/White-Paper#ethereum.

[25] Expressjs. https://expressjs.com/it/.

[26] Facebook reports third quarter 2018 results. https://investor.fb.com/investor-news/press-release-details/2018/Facebook-Reports-Third-Quarter-2018-Results/default.aspx.

[27] Ethereum Foundation. Ethereum problems. "https://github.com/ethereum/wiki/wiki/Problems.

[28] Ethereum Foundation. Proof of stakes faq. https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs.

[29] Truffle suite - ganache. https://truffleframework.com/ganache.

[30] Gdpr - processing of special categories of personal data. http://www.privacy-regulation.eu/en/article-9-processing-of-special-categories-of-personal-data-GDPR.htm.

[31] Arthur Gervais and Karl Wüst. Do you need a blockchain? 2017.

[32] Search engine market share. https://netmarketshare.com/search-engine-market-share.aspx.

[33] Hdl-cholesterol distribution. https://www.unich.it/med/papers/Metodologia\%20Medico-Scientifica\%20di\%20base/statistica/04-distribuzione\%20normale_nuovo.pdf.

[34] Optimizing social institutions - nobel prize. https://www.nobelprize.org/prizes/economic-sciences/2007/speedread/.

[35] Jana: Private data as a service. 2018. http://www.bu.edu/hic/files/2018/06/2018-06-05-Anand.Swarte.pdf.

[36] Keep - privacy layer for public blockchains. https://backend.keep.network/whitepaper.

[37] Marcel Keller, Peter Scholl, and Nigel Smart. An architecture for practical actively secure mpc with dishonest majority. 2013. https://eprint.iacr.org/2013/143.pdf.

[38] Ajay Kshemkalyani and Mukesh Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press, 2008.

[39] Jae Kwon. Tendermint: Consensus without mining. 2014. https://tendermint.com/static/docs/tendermint.pdf.

[40] Lightning network. https://lightning.network/.

[41] Lighting network transaction costs. https://www.reddit.com/r/Bitcoin/comments/8q7fj3/so_far_ive_made_42_transactions_on_lightning/.

[42] Maian: automatic tool for finding trace vulnerabilities in ethereum smart contracts. https://github.com/MAIAN-tool/MAIAN.

[43] Andrew Marshall. Combined crypto market capitalization races past $800 bln. *Cointelegraph*.

[44] Mechanism design. https://en.wikipedia.org/wiki/Mechanism_design.

[45] Eric maskin - an introduction to mechanism design - warwick economics summit 2014. https://www.youtube.com/watch?v=XSVoeETsEcU.

[46] Silvio Micali and Jing Chen. Algorand. 2016. https://arxiv.org/abs/1607.01341.

[47] Silvio Micali, Jing Chen, Sergey Gorbunov, and Georgios Vlachos. Algorand agreement super fast and partition resilient byzantine agreement. 2018.

[48] Avi Wigdemon Michael Ben-Or, Shafi Goldwasser. Completeness theorems for non-cryptographic fault-tolerant distributed computation. 1988. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.116.2968&rep=rep1&type=pdf.

[49] Monero. https://www.getmonero.org/.

[50] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008. https://bitcoin.org/bitcoin.pdf.

[51] Ivica Nikolic, Aashish Kolluri, Ilya Sergey, Prateek Saxena, and Aquinas Hobor. Finding the greedy, prodigal, and suicidal contracts at scale. 2018.

[52] Nodejs. https://nodejs.org/en/about/.

[53] Ethereum erc-721. http://erc721.org/.

[54] Normal distribution table. https://www.math.arizona.edu/~rsims/ma464/standardnormaltable.pdf.

[55] P4Titan. Slimcoin: A peer-to-peer crypto-currency with proof-of-burn. 2014. http://www.slimcoin.club/whitepaper.pdf.

[56] Ethereum's parity hack. https://www.coindesk.com/ethereum-client-bug-freezes-user-funds-fallout-remains-uncertain.

[57] Passport.js. http://www.passportjs.org/.

[58] Peercoin. https://peercoin.net/.

[59] Preimage attacks. "https://en.wikipedia.org/wiki/Preimage_attack".

[60] Raiden network. https://raiden.network/.

[61] Reactjs. https://reactjs.org/.

[62] Reduxjs. https://redux.js.org/.

[63] Charles Rackoff Shafi Goldwasser, Silvio Micali. The knowledge complexity of interactive proof-systems. 1985. https://groups.csail.mit.edu/cis/pubs/shafi/1985-stoc.pdf.

[64] On sharding blockchains. https://github.com/ethereum/wiki/wiki/Sharding-FAQs.

[65] Sharding - wikipedia. https://en.wikipedia.org/wiki/Shard_(database_architecture).

[66] Sharemind multi-party computation. https://sharemind.cyber.ee/sharemind-mpc/.

[67] Silvio micali. https://it.wikipedia.org/wiki/Silvio_Micali.

[68] Solidity compiler. https://solidity.readthedocs.io/en/v0.5.2/installing-solidity.html.

[69] Spdz client library. https://github.com/bristolcrypto/spdz-client-lib.

[70] Adam Tanner. Harvard professor re-identifies anonymous volunteers in dna study. *Scientific American*, 2013. https://www.forbes.com/sites/adamtanner/2013/04/25/harvard-professor-re-identifies-anonymous-volunteers-in-dna-study/.

[71] Adam Tenner. How data brokers make money off your medical records. 2016. https://www.scientificamerican.com/article/how-data-brokers-make-money-off-your-medical-records/.

[72] Tron-decentralise the web. https://tron.network/.

[73] Unbound tech. https://www.unboundtech.com/.

[74] Web3 - ethereum javascript api. https://github.com/ethereum/web3.js/.

[75] Zachary J. Williamson. The aztec protocol. 2018. https://github.com/AztecProtocol/AZTEC/blob/master/AZTEC.pdf.

[76] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. 2014. https://github.com/ethereum/yellowpaper.

[77] Andrew Chi-Chih Yao. Protocols for secure computations. 1982.

[78] Zcash. https://z.cash/.

[79] Zcash setup ceremony. https://www.wnycstudios.org/story/ceremony.

[80] Zilliqa website. https://zilliqa.com/.

[81] Zilliqa technical whitepaper. https://docs.zilliqa.com/whitepaper.pdf.