A Permissioned Blockchain-based System for Collaborative Drug Discovery

Christoffer Olsson and Mohsen Toorani
Department of Electrical and Information Technology, Lund University, Sweden

Keywords: Decentralized Systems, Blockchain, Hyperledger Fabric, Security, Chaincode, Caliper, Cheminformatics.

Abstract: Research and development of novel molecular compounds in the pharmaceutical industry can be highly costly. Lack of confidentiality can prevent a product from being patented or commercialized. As an effect, cross-organizational collaboration is virtually non-existent. In this paper, we introduce a blockchain-based solution to the collaborative drug discovery problem so that participants can maintain full ownership of the asset and upload partial information about molecules without revealing the molecule itself. A prototype is also implemented using the blockchain technology Hyperledger Fabric and analyzed from security and performance perspectives. The prototype provides a set of functionalities that makes sure that ownership is maintained, integrity is protected, and critical information remains confidential. From a performance perspective, it provides a good throughput and latency in the order of milliseconds. However, further improvements could be done to the scalability of the system.

1 INTRODUCTION

The pharmaceutical industry relies heavily on both *patents* and *trade secrets* to generate value. Intellectual property (IP) management deeply affects the everyday activities of businesses, as it affects their ability to protect and derive value from research (Atkinson and Jones, 2009). A miss in the search of the prior art can mean that a product is either un-patentable or that some other company already owns a patent that will block the commercialization of the developed product. The median for developing a new molecule is estimated to be around $985 million (Wouters et al., 2020). This imposes a substantial risk of confidentiality breaches of ongoing molecule research for the private sector and academia alike. Actors risk that their research gets unlawfully disclosed if they collaborate across organizations. Furthermore, pharmaceutical companies can not rely on trade secrets alone as the world is highly connected. The probability that competitors reverse-engineer, or independently discover a product is also non-negligible (Saha and Bhattacharya, 2011). Patents are needed to protect IP. Technical solutions to share information on ongoing molecule research across organizations in a painless and fair way has been lacking so far (Andrews et al., 2015). We refer to this problem as the *collaborative drug discovery problem* (CDDP).

A chemical identifier is a string that denotes a chemical substance and can be used to create digital representations of molecules (Heller et al., 2015). They are a common way to represent molecules and are ubiquitous in the pharmaceutical industry. *InChi* (Heller et al., 2015), *InChiKey* (Heller et al., 2015), and *SMILES* (Weininger, 1988) are common chemical identifiers. *Inchi* is a linear string representation of a molecule that describes its structure. An *InChiKey* is a character string of length 27 constructed by hashing an InChi string using the *SHA-256* hashing algorithm. InChiKeys can be used as proof-of-knowledge of a molecule structure without revealing the structure. *SMILES* is another linear representation of molecules and is somewhat easier for humans to interpret than InChi. Using a program like *Open Babel* (O'Boyle et al., 2011), one can easily convert between SMILES and InChi, as well as derive an InChiKey from an InChi. To prove knowledge of a molecule at a certain time, SMILES and InChi could be used interchangeably together with other security mechanisms. If the confidentiality of the molecule has to be protected, InChiKeys would be used. This way, the structure of the molecule remains private, but the owner can use the hash value to prove their knowledge of the molecule structure at a later time, e.g. when they wish to patent their discovery.

Olsson, C. and Toorani, M.
A Permissioned Blockchain-based System for Collaborative Drug Discovery.
DOI: 10.5220/0010236901210132
In *Proceedings of the 7th International Conference on Information Systems Security and Privacy (ICISSP 2021)*, pages 121-132
ISBN: 978-989-758-491-6

Since the introduction of the Bitcoin (Nakamoto et al., 2008), blockchain technology has been in the limelight, as it provides a platform where actors who do not trust each other can collaborate. The blockchain enables secure and fully-distributed log management for peer-to-peer networks. The main principle behind the technology is to include transaction logs into a chain of blocks where each block contains a secure one-way hash of the previous block. Bitcoin is a decentralized cryptocurrency and builds a peer-to-peer network where users can submit transactions: A new block is only allowed to be added to the chain if it has gone through a *mining* process which establishes consensus between the peers in the network. Blockchain-based solutions help to build trust, transparency, and traceability. There exist two types of blockchain systems: permissionless and permissioned. In public or permissionless blockchains such as Bitcoin and Ethereum (Buterin et al., 2013), anyone can join the network, while in a permissioned blockchain such as Hyperledger Fabric (Androulaki et al., 2018) only invited members can join and participate. Due to better accountability and auditability, a permissioned blockchain-based system can provide a better solution to the CDDP.

In this paper, we propose a decentralized solution to the CDDP based using the permissioned blockchain framework *Hyperledger Fabric*, where molecules are tokenized using the chemical identifiers and stored on the blockchain. A prototype has been implemented that can be used as a stepping stone to fully realize a system where a multitude of pharmaceutical actors can share resources while providing strong reassurance that the correct actors receive and retain their intellectual propriety. This is a novel solution to the problem as previous work is either centralized or based on public blockchains, which exposes them to extra risks and limitations. The studied problem is integral for the development of new molecules, and any improvement could lead to substantial monetary benefits for any actor involved in the discovery of molecules. However, there are few solutions proposed so far: Astra Zeneca, a major pharmaceutical company, proposed a platform for collaborative drug discovery (Andrews et al., 2015). Since the solution is centralized, any participant has to trust Astra Zeneca not to misuse the database, which is a major drawback for the system. The *collaborative drug discovery* (CDD) platform (Ekins and Bunin, 2013) is another centralized platform for molecule sharing. *Molecule* (Molecule GmbH, 2020) is yet another company that is building collaborative drug discovery using Ethereum. Ethereum is a public blockchain without enough measures for accountability as anyone can participate in the network pseudo-anonymously. Furthermore, their solution is exposed to systemic Ethereum-specific risks. Moreover, it requires users to learn how to handle cryptocurrencies, which makes it harder for both small and large actors to adopt their platform.

Contributions. Our contributions can be summarized as:

- *A New Decentralized Solution to CDDP based on a Permissioned Blockchain*: Previous work on molecule tracking are either centralized or based on public blockchains (Andrews et al., 2015) (Ekins and Bunin, 2013) (Molecule GmbH, 2020). Furthermore, a systematic review and modeling of the problem are accomplished which is absent in previous work.
- *A High-level Security Analysis of the Proposed Prototype*: There exists no overarching discussion on the security properties of Fabric. The paper that introduces Hyperledger Fabric (Androulaki et al., 2018) briefly mentions it, and official documentation (Hyperledger, 2020) of Fabric is still marked as work in progress. Concurrently, Kusters et al. (Küsters et al., 2020) discussed accountability of Fabric.
- *A Performance Evaluation of the Proposed Prototype*: Previous work on performance evaluation of Fabric (Thakkar et al., 2018), (Nguyen et al., 2019), (Nasir et al., 2018), (Hao et al., 2018), (Sukhwani, 2018) focus on either a fixed network topology or a smaller amount of entities. They do not investigate how Fabric scales when one adds a larger amount of entities to the network.

The rest of this paper is organized as follows. Hyperledger Fabric is briefly introduced in Section 2. The proposed solution is introduced in Section 3. Security analysis of the proposed solution and its performance analysis is provided in Section 4 and Section 5, respectively.

2 HYPERLEDGER FABRIC

Hyperledger Fabric (Androulaki et al., 2018) is an open-sourced permissioned blockchain framework with smart contract capabilities. A consortium of organizations join together to form a Fabric network. Each entity in the network is uniquely identified by an identifier, is associated with an organization, and has a role. Moreover, entities can have admin privileges. A Fabric network consists of four entities: *Peers*, *Orderers*, *Clients*, and *Certificate Authorities (CAs)*. Peers

execute smart contracts and maintain the ledgers. Orderers order executed transactions into blocks. Clients are any computer program that interacts with the Fabric network. Organizations can establish one or more channels amongst themselves. Channels can be used to partition data across a Fabric network. Each channel contains one ledger that is maintained and secured by the orderers and peers that are part of that channel. The blocks are chained together using *SHA-256* as the hashing algorithm. Smart contracts are called *chaincode* in Fabric. One or more chaincodes can be installed on each channel. The access control is highly configurable in Fabric and can be defined at the network level, channel level, chaincode level, and inside of chaincode. One can define access control based on roles, organization affiliation, and individual certificates. Any CA can be used to generate certificates, given that they follow some specifications defined by Fabric. By default, Fabric uses X.509 certificates. Certificates can be revoked at a network, channel, or node level. The state inside of the chaincode is modeled as a *versioned* key-value store. The reason the store is versioned is to prevent double-spend attacks. If a first transaction reads a key for a version, any following write to the same key under the same version is marked as invalid in the transaction lifecycle described below.

Figure 1 illustrates a Fabric network with one channel (CH1), and seven entities that belong to two organizations: two clients (C1, C2), two peers (P1, P2), two CAs (CA1, CA2), and one Orderer (O1). The number associated with each entity indicates which organization it belongs to. Each peer has a copy of the chaincode (CC) installed on the channel. The channel is configured with a special transaction called *channel configuration transaction* (CCT) that defines access control for entities interacting with the channel. Admins (A1, A2) create and sign CCTs. *Network creation* (NC) policy defines which entities and under what circumstances they can create channels, and also defines entities that can update the NC policy itself. The NC policy is created and signed by admins.

Fabric is similar to other smart contract platforms, like Ethereum, with some key exceptions. Due to its permissioned nature, there is no mining or crypto-economics in Fabric. The key assumption is that since all messages are signed under a well-known identity, adversaries could face consequences outside of the network, or have their certificate revoked if they misbehave. Fabric uses a novel transaction execution model called *Execute-Order-Validate* as depicted in Figure 2:

1. Clients send transaction proposals to peers that have endorsing capabilities on some channel. Which peers that can endorse is defined by configuration.

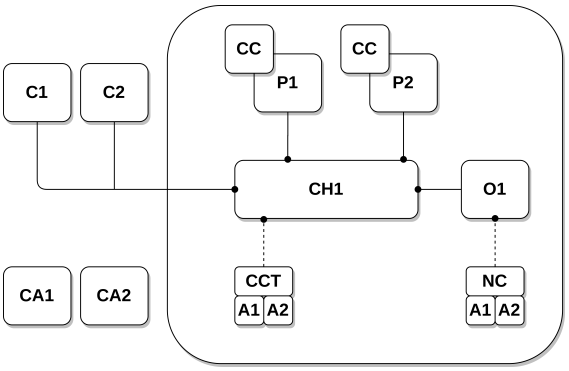


Figure 1: A Fabric network with one channel and two organizations.

2. The peers execute the transaction proposal, but no state is updated.
3. The peers send the execution results to the client.
4. Once the client has collected enough endorsed transactions it sends the endorsed transactions to the orderers. Enough endorsements is also defined by configuration.
5. The orderers order all endorsed transactions they receive into blocks.
6. The orderers deliver the block to all peers on that channel.
7. The peers validate each transaction in every block and update the ledger accordingly.

Any transaction that would create inconsistencies in the state database is marked as invalid. At each step, further action is halted if an identity is lacking the correct level of authority. The main benefit of using this transaction model is that peers can execute several transactions in parallel as double-spend issues are caught in the validation step. This should provide higher throughput of transactions compared to e.g. Ethereum where the transactions have to be executed before they are ordered. Moreover, Fabric has query capabilities. A query is a chaincode invocation that is stopped after the client receives execution results from peers. Queries provide higher throughput and lower latency than full transaction invocations. Currently, Raft (Ongaro and Ousterhout, 2014) is the recommended consensus protocol for the orderers. There exists no production-ready Byzantine fault-tolerant protocol for Fabric yet. As Fabric has no crypto-economic component, we should not expect to see a proof-of-work or proof-of-stake style consensus protocol for the framework.

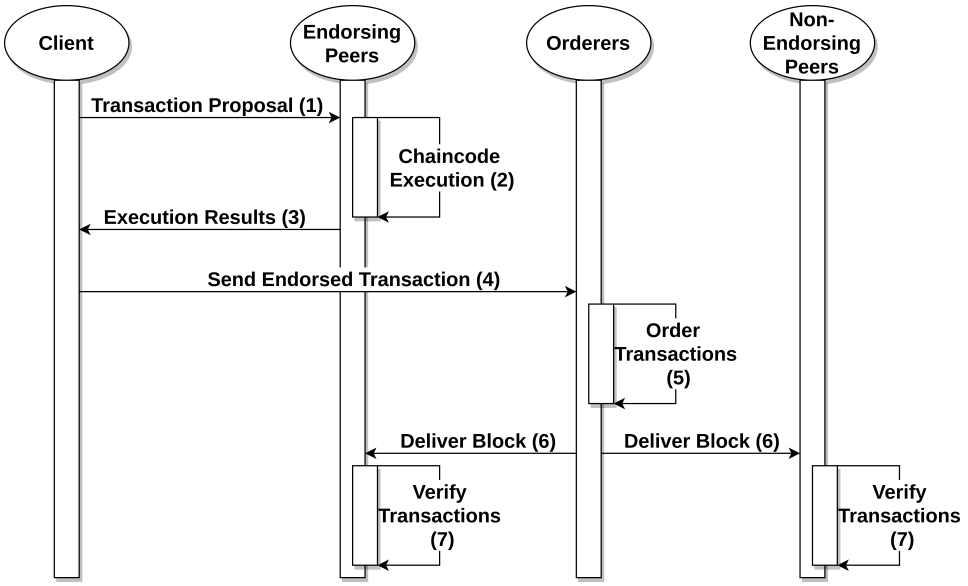


Figure 2: Execute-Order-Validate Transaction Lifecycle.

3 PROPOSED SOLUTION

Our proposed solution to the CDDP is a platform that allows partial or full reveals of tokenized molecules. Actors should be able to reveal the properties of their molecules without making them prior art. This way other actors can search for assets with certain properties. They can then acquire rights to the asset. Using this asset, new assets can be developed and uploaded to the platform. If the newly developed asset is not part of the prior art, it can potentially be patented. The ownership of the token can be transferred using the chaincode. Figure 3 illustrates this cycle: *Organization 1* tokenizes a molecule they want to share. *Organization 2* acquires the token and uses it to develop a novel molecule that will also be tokenized. Organization 1 uses the Fabric network to prove that they were the original uploaders of the first molecule and further contributions to the molecule will also be stored in the system.

A prototype was implemented to analyze the performance and conduct experimental evaluations. The prototype consists of a Fabric network with one of each entity, chaincode implemented in Golang (Hyperledger Foundation, 2020a), and a basic interface that users can use to to interact with the network.

3.1 Data Format

Tokenized molecules consist of the following fields:

1. *Identifier* - A unique identifier for an asset.

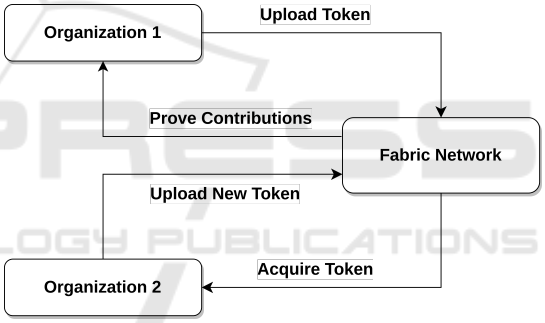


Figure 3: An overview of the proposed solution.

2. *Name* - A text string that represents a natural name for the asset.
3. *Timestamp* - A timestamp of when the asset was uploaded or updated.
4. *Asset Owner* - The owner of the molecule in the real world.
5. *Token Owner* - The Fabric user that controls the token in the ledger.
6. *Structure* - Any of *SMILES*, *InChi*, or *InChiKey* that represent the molecule.
7. *Keywords* - A list of keywords with potentially associated values. For example, *painkiller*, or *boiling point* equals 50°C. Can be used to search for molecules with certain properties.
8. *License Information* - License Information that tells how the molecule can be used.

3.2 Functionalities

The following operations are defined for the platform. *Search Molecule* and *List History* are only invoked as queries, i.e. they only use part of the transaction lifecycle.

1. *Upload Molecule* - Tokenize a molecule and save it in the blockchain.
2. *Search Molecule* - Search for molecules that match some query, e.g. structure.
3. *Update Molecule* - Update information of a molecule. The owner of a token can change any of the fields, except for the structure of the molecule. Due to the nature of Fabric previous states of the molecules are saved in the ledger and can be retained.
4. *List History* - List complete history of a molecule. This is used to prove previous ownership and contributions.
5. *Transfer Ownership* - Move ownership of a token from one user to another.

4 SECURITY ANALYSIS

Fabric is quite new, and much work is required on its security analysis. There is only one recent work on formal analysis of Fabric, mostly considering it from the accountability perspective (Küsters et al., 2020). In this section, we perform a heuristic security analysis of the proposed solution and implemented prototype, discuss different security features provided, and how common attacks are prevented. The chaincode security is a different angle that will be tested using one of the available tools as will be discussed in Section 4.2. We consider tokenized molecules that are uploaded to the network as the information that should be protected. When analyzing the security of the prototype, we assume an adversary in the Dolev-Yao model. We assume perfect cryptography, i.e. deployed cryptographic primitives are secure. The system is also assumed to use the latest version of the *Transport Layer Security* (TLS). Not anyone can join the network due to the use of a permissioned blockchain. Moreover, since there are accountability provisions, users would consider the consequences of their actions.

4.1 Security of the Scheme

The proposed solution provides the following security goals:

- *Confidentiality* - Only invited members can participate in the network. This provides confidentiality against non-members of the network. Furthermore, confidentiality can be provided inside the network by using access control mechanisms. Furthermore, the confidentiality of molecules at the channel level can be provided by uploading an InChiKey instead of the molecule structure.
- *Integrity* - Integrity is protected using a blockchain in combination with signed messages. Peers can verify the state of their ledgers by retrieving blockchain hashes from trusted peers. Using this they can reconstruct the state independently. Peers validate each transaction in the final step of the transaction lifecycle to make sure that the state is updated correctly, and that transaction proposals have not been tampered with.
- *Availability* - Availability is provided by having several entities that provide the same services. The prototype only uses one of each entity. In a production scenario, it would be easy to add more entities.
- *Accountability* - Since all transactions in Fabric are signed one can hold the appropriate actor accountable (Küsters et al., 2020). Transactions that lack authorization will be blocked and logged by orderers and peers.
- *Auditability* - The blockchain provides auditability of authorized transactions. All transactions are stored in the blockchain, even invalid ones. The network entities provide logging that can be analyzed to further enhance auditability.
- *Authenticity* - Since all messages are signed by a certified entity, the prototype provides the authenticity of origin. Moreover, since TLS is used, authentication is mutual.
- *Non-repudiation* - Any transaction in the network contains a digital signature. Non-repudiation is provided by the digital signature since the public keys are authentic due to the deployed CA.
  Another aspect is that an adversary could upload garbage data with the same InChiKey as another molecule. This cannot help him to gain any advantage in any future dispute because when the legitimate owner and the adversary are to claim a patent for the molecule, they have to reveal their molecules to a patent office. It will become known that the adversary has uploaded garbage data just to claim an InChiKey. Due to accountability countermeasures in the permissioned system, the attacker would face consequences and have his certificate revoked.

- *Privacy* - The prototype's permissioned nature denies non-members from learning about identities inside of the network. An organization can enroll users under any name. However, they can not hide the fact that a member is part of their organization. Data privacy is also protected by the fact that the network is permissioned. This can be further enhanced by access control mechanisms. Lastly, the privacy of molecule data can be provided by uploading an InChiKey instead of its structure.

Now we discuss how common attacks to the proposed solution are thwarted:

Impersonation Attacks. All messages and transactions are signed and will be verified by the recipients through corresponding digital certificates and public keys. This provides origin and entity authentication. TLS is also used which provides mutual authentication. An impersonation attack is not feasible unless the private key of an entity is compromised.

Byzantine Peers. Peers communicate with each other using gossip (Hyperledger, 2020), i.e. they send messages to each other in a peer-to-peer manner. A compromised peer has the potential to attack the system and threatens the security consideration in the following way:

1. *Integrity* - Since transactions are signed, any alteration will make them invalid, and thus Byzantine peers cannot threaten the integrity of messages that are passing through them. However, a Byzantine peer could output and sign any arbitrary message at its will. For example, a Byzantine peer could send an arbitrary execution result to a client in step 3 of the transaction lifecycle. To mitigate this, a client should request to have its transactions evaluated at several peers. The larger the network, the more peers would have to coordinate to trick a client.
2. *Availability* - Since gossip is used, the refusal to propagate information of a Byzantine peer can be mitigated by communicating with other peers.
3. *Confidentiality* - A Byzantine peer could be well-behaved but disclose confidential data. Molecules that need their confidentiality protected can be uploaded as an InChiKey, not revealing their structure.

In the end, one has to ask if Byzantine peers are likely. It is assumed to trust peers from one's own organization. However, if communication happens across organizations one could communicate with a peer from a potentially competing organization.

Byzantine Orderers. As there exists no production-grade Byzantine fault-tolerant protocol for orderers, Byzantine orderers could attack a network. This is potentially a fatal security flaw for future systems deployed in more adversarial environments. However, since all transactions are signed, the probability of an orderer diverging from the consensus protocol varies with how much orderers consider reputation. If an orderer is caught being malicious its operator could face legal actions, and subsequently have its certificate revoked.

Replay Attacks. Freshness is provided by including nonces in transactions. This way each transaction will only be handled once by each entity (Hyperledger, 2020).

Man in the Middle (MITM) Attacks. Since all messages are signed and sent over a secure channel over TLS, MITM attacks are not feasible. Some traffic analysis might be possible, but this would be of limited use to the attacker as she would only see that entities are communicating and not the content of the messages.

Denial-of-Service (DoS) Attacks. A single node could be spammed by an attacker that has access to the network. A healthy network should have a reasonable amount of peers and orderers to minimize this risk. However, a Fabric network is susceptible to denial of service attacks from clients that are part of the network. When running evaluations, a client sending more transactions than the peers could commit to a ledger affected the availability of the network. One should ask if clients are likely to spam the network as their actions are logged by all peers and orderers. However, depending on the application it could be hard to differentiate between attacks and a flood of honest transactions. This can be mitigated by introducing some rate limit at the peers.

Sybil Attacks. In a *sybil attack* one participant of a network generates a large number of identities in order to gain a disproportionate influence over the network (Conti et al., 2018). In the case of Fabric that would be an adversary possessing several certificates that they use. Assuming that the CA for an organization is not compromised, Sybil attacks are not feasible as the malicious actor can not generate their own identities. However, the actors controlling a CA could generate as many identities as they like to. This opens up an attack vector if a chaincode implementation relies on identity-based votes.

Eclipse Attacks. An *eclipse attack* happens when all incoming and outgoing connections of a victim entity are to malicious actors (Conti et al., 2018). As all communications are over TLS, entities will establish communications with known and authenticated entities.

Key Compromise. Once the private key of an entity is compromised, it is trivial that an adversary can use the key to masquerade as the compromised entity. If the user is aware of the key compromise, they can request a certificate revocation and stop any further misuse. Regardless, there are at least two aspects that are important in any system when it comes to the key compromise: (1) An adversary should not be able to impersonate himself as a legitimate user to the compromised entity; (2) An adversary should not be able to change or affect communications prior to the key compromise. It is trivial to show that both features are provided: The first feature is provided due to mutual authentication and use of TLS and digital certificates and signatures from all involved parties that will communicate with the compromised entity, so the adversary cannot do it unless they compromise private keys of both parties. The second feature is guaranteed since all transactions will be added to the blockchain after a successful consensus, which means they cannot be altered at a later time. Confidentiality will also be guaranteed if molecules are uploaded in the InChiKey format.
A compromised CA would allow an adversary to generate certificates for some entities at their will. Certificates can be revoked, but if the compromise is not discovered, an adversary can operate in the system for a while. Any attack against the integrity would be caught due to accountability. Misissued certificates will be revoked as soon as they are discovered, and the issuing CA will be kept accountable. If the attack is severe, the network could roll back to a state before the attack. The adversary could probably operate for a longer time if they only query the system for information. Since we assumed that sensitive molecules are uploaded as InChiKeys, this would not break the confidentiality, as an adversary would not learn anything about the hidden molecule structures. It is critical to protect the CAs and take measures to secure them.

Transactions Malleability. A client proxying messages to the ordering service via peers is subject to a malleability attack. This can happen in step 4 of the transaction lifecycle. In this step, the client sends an endorsed transaction to the orderer. If the client does not have access to a direct connection to the ordering service it must proxy it via a peer. The peer can not fabricate endorsements. However, it can remove endorsements from the transactions, potentially making it invalid (Hyperledger, 2020). The prototype allows direct communication between the orderer and client, which mitigates this attack. In future scenarios, the client can mitigate this by proxying the transaction via a peer from their own organization.

Double-spend Protection. A double-spend is a set of two transactions that are in conflict with one another that both get accepted by the network (Conti et al., 2018). In the case of molecules, this would be akin to someone writing to a molecule ($T1$), followed by another read and write to the same molecule, under the same version ($T2$). Let $put(k_i, v_i)$ be an operation that writes some value v_i to the database under the key k_i, for version i. Version i, is a monotonically increasing number, and $get(k_i)$ retrieves the data that was added using *put* for version i. A double-spend would look as follows:

$$T1 = put(k_1, v_1)$$
$$T2 = get(k_1);\ put(k_1, v_2)$$

This would break the integrity of the network, as one user could transfer ownership of a single molecule to several users. Double-spending is stopped at the validation step in the transaction lifecycle. If a key in the ledger is referenced after it has been updated there will exist a conflict. As a result, the transaction will be marked as invalid by the peers (Androulaki et al., 2018).

InChiKey Attacks. We had an assumption on the infeasibility of cryptographic primitives, but there will be some attack vectors if an attacker could break InChiKeys. If an adversary finds the same InChi that produced the InChiKey, the confidentiality of that molecule would be broken. However, any hash collision would not help the adversary, as he should find the same InChi to prevent the owner of the molecule from claiming a patent.
InChiKey uses SHA-256 (from SHA-2 family), and its security is as strong as SHA-256. Some attacks are feasible on SHA-256 (Dobraunig et al., 2015), but they are not still practical. InChiKeys used the latest version of SHA families at the time of its design. However, the latest version of the secure hash algorithm (SHA) family of standards is currently SHA-3, which provides stronger security guarantees. The deployed hash function used to create InChiKeys can simply be updated to make it stronger. However, the output of InChiKeys will then be different for the same input before and after the change of the SHA algorithm, which means that molecule databases will be required to rehash their molecule entries. This will lead to two

lists based on different versions of the InChiKey and could be used in the transition period to avoid any confusion. Future versions of the proposed platform could upgrade the InChiKey derivation to use hashes from the SHA-3 family.

4.2 Chaincode Security

The security of the deployed chaincode can be analyzed independently from the underlying network. When analyzing the chaincode security we assume the underlying network to be secure. Chaincode is a general-purpose program and could have general software vulnerabilities such as buffer overflows, integer overflows, command injections, etc. (Praitheeshan et al., 2019). Security analysis of chaincode and smart contracts is an ongoing research topic and several tools for their *static analysis* and *formal analysis* have been developed (Praitheeshan et al., 2019), (Rouhani and Deters, 2019), (Beckert et al., 2018). Previous work mostly focuses on Ethereum as it is the second-largest blockchain measured in market capitalization, as well as the oldest smart contract platform. Fabric's chaincode is easier to upgrade than public blockchain smart contracts as one needs to coordinate upgrades with a smaller set of participants. One key vulnerability is identified that is relevant for the prototype (Praitheeshan et al., 2019):

- *Timestamp-dependent Contracts*: A timestamp-dependent contract is a smart contract that uses timestamps in its execution. An entity could change time to manipulate code execution. In our case, it is important to know when a user knew about a certain molecule. Users have an incentive to pass wrong timestamps as input when creating or editing molecules. This is mitigated in the prototype by using a timestamp from the peer. This peer functions as a trusted third party. This service could be expanded to include several peers from several organizations.

Yamashita et al. (Yamashita et al., 2019) discussed potential vulnerabilities and identified some components that should not be part of Golang chaincode:

1. Goroutines: Concurrency is discouraged in chaincode as it is easy to get it wrong.
2. Fields: Chaincode should not rely on fields as they are stored locally.
3. Global Variables: Chaincode should not contain global variables as they are stored locally.
4. Non-deterministic libraries should not be utilized.
5. Map Ranges: Map ranges are non-deterministic and should be avoided.

Two static analysis tool for Golang chaincode was found, namely, ChainSecurity's *Chaincode Scanner* (chainsecurity, 2020) and *reviveCC* (sivachokkapu, 2020). *Chaincode Scanner* can only analyze chaincode files that meet certain requirements, while *reviveCC* can analyze any chaincode file. However, *reviceCC* was outdated and could not be used to analyze our chaincode. The chaincode was analyzed using *Chaincode Scanner* and none of the above-mentioned components were found. No formal verification tool for Golang chaincode was found. Beckert et al. (Beckert et al., 2018) modified *KeY*, a formal verification tool for Java to verify correctness of Java chaincode. There also exists a suite of verification tools for Ethereum: *Oyente*, *ZEUS*, etc. (Praitheeshan et al., 2019).

5 PERFORMANCE EVALUATION

Performance analysis of Fabric is not trivial and could be a research topic in itself (Thakkar et al., 2018), (Nasir et al., 2018), (Sukhwani, 2018). We used *Hyperledger Caliper* (Hyperledger Foundation, 2020b) to perform a performance analysis of the implemented prototype and chaincode. Caliper can connect to existing Fabric networks, run benchmarks, and aggregate the results. The implemented chaincode was tested using varying network topologies.

Latency, throughput, and success rate for the operations *Upload Molecule*, *List History*, and *Transfer Ownership* were analyzed. *Update Molecule* operation was not analyzed since it is similar to *Transfer Ownership*. Moreover, *Search Molecule* operation requires a large set of already existing molecules to be interesting. Specifications of our testing environment can be described as following (Performance and Group, 2018):

1. *Hardware*: We used a PC with 128GB of RAM, and a Ryzen Threadripper 1950X (3.4GHz base, 4GHz boost with 16 cores). Running all tests on a single machine removes any networking aspects to the tests.
2. *Network Model*: Different network topologies were tested. Their sizes varied from one up to 26 organizations. Each organization contained two peers. Having two peers seems to be a reasonable starting point for analysis as each organization has access to one backup peer. Moreover, for each network, the number of orderers varied from one up to 31 orderers.
   The block configuration for the orderers was configured as follows: *AbsoluteMaxBytes* (how large a block can be) was 99MB, *MaxMessageCount*

(maximum number of messages in a block) was 10, *PreferredMaxBytes* (how large blocks should be preferably) was 512KB and the *BatchTimeout* (how long the orderers should wait before packaging transactions into a block) was 2 seconds. All other parameters were default values.

3. *Software Components*: The chaincode was written in Golang. For the experiments, LevelDB was used as the state database in the peers.
4. *Type of Data*: The data used in experiments was mock data of size in the order of kilobytes.
5. *Workload*: Caliper sent transactions at a *fixed rate*. For each operation, transactions were sent at the rates 1 tps, and 5 tps. The duration of each round was 30 seconds. For each operation, each round was run 5 times. After every 5 rounds, Caliper paused its transaction sending for 60 seconds to let the network stabilize.

   In summary, a total of 6 test suites were run: 3 operations (*Upload Molecule*, *List History*, *Transfer Ownership*), 2 arrival rates (1 tps and 5 tps). Each round was repeated 5 times and the final result for each test suite was retrieved by averaging the results from the 5 repeated rounds.

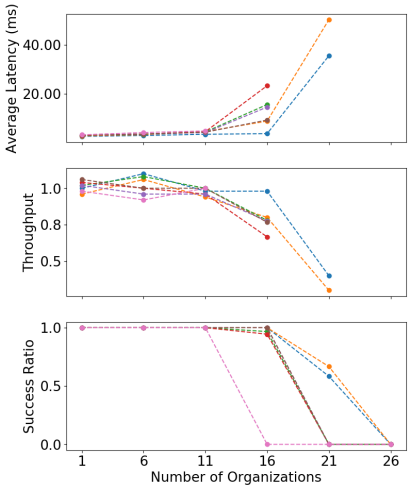
Evaluation results for four different operations at the arrival rates of 1 tps and 5 tps are depicted in Figure 4. For the arrival rate of 1 tps, every network seems to work fine up to 26 orderers and 16 organizations. For 5 tps, a success rate close to 1 could be achieved until 11 organizations and 26 orderers for all operations except *Transfer Ownership*. For *Transfer Ownership* at 5 tps, a success rate of 1 could be achieved until 6 organizations and 26 orderers. *List History* could sustain a success rate of 1 until 21 organizations and 16 orderers. After that, the success rate quickly goes to zero. For 31 orderers, the success rate went to zero after 11 organizations for all transaction rates, but for *List History* it went to 0 at 6 organizations for both transaction rates.

The latency has, in general, a positive trend and the throughput a negative. As expected, query transactions are much faster than full transaction invocations. The highest average latency for the query transaction was around 0.08 ms and the highest average latency for a full transaction was over 40 ms. This is beneficial for the prototype as we expect there to be more reads than writes on the platform.

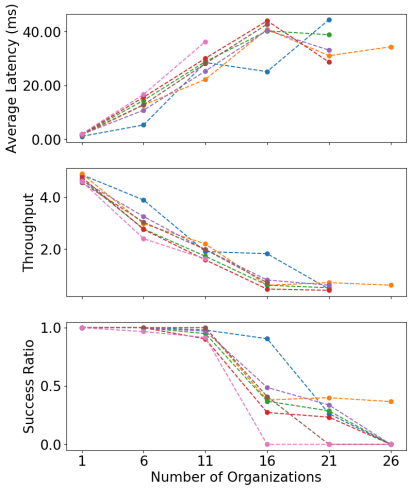
It would be acceptable for end-users of the platform to accept a lower throughput and higher latency when uploading molecules. Most important is that the platform provides strong confidentiality and integrity of the uploaded data. Ideally, the network should be able to support hundreds of organizations, but a network of up to 16 organizations is sufficient to include e.g. a handful of universities. This would provide a good starting point for the network. At this point, it is hard to know exactly what causes the transaction performance to degrade at higher entity numbers. Better error reporting, logging, and documentation from both Fabric and Caliper are needed to understand where bottlenecks exist. After 16 organizations and 26 orderers, we could conclude that the networks are too unstable to derive satisfactory results. Androulaki et al. (Androulaki et al., 2018) managed to run a network with 100 peers at transaction arrival rates that were orders of magnitude higher than what was used in our experiments. However, they did not specify exactly how they configured their networks. This makes it hard to replicate their results. Moreover, other performance evaluations focus on either a fixed network or networks of smaller sizes (Thakkar et al., 2018), (Nasir et al., 2018), and do not analyze how performance changes when networks grow larger. Sukhwani et al. (Sukhwani, 2018) investigated a larger network, but they used a Byzantine fault-tolerant protocol. The following strategies could make the system more scalable:

1. The election timeout for the ordering service could be set too low in the default configuration of Fabric (Hyperledger, 2020). If the ordering service is busy serving blocks, they could miss leader heartbeats and as a result, trigger an election. This way they could end up in a state where they are never able to catch up and as a result, the whole network halts.
2. In order to enable a high transaction throughput, a favorable way to design the chaincode is to use event-sourcing (Betts et al., 2013). Currently, one loads the asset from the ledger and makes a change to it. If there is a high transaction arrival rate, the probability that a particular key version is out of date once it reaches the verification phase increases. Using event sourcing, one only writes the difference in changes to an asset. This way, there will be no read/write-conflicts as all transactions are write-only. Queries are then constructed by replaying the events.
3. Varying block creation parameters could further improve the performance of the prototype (Thakkar et al., 2018).

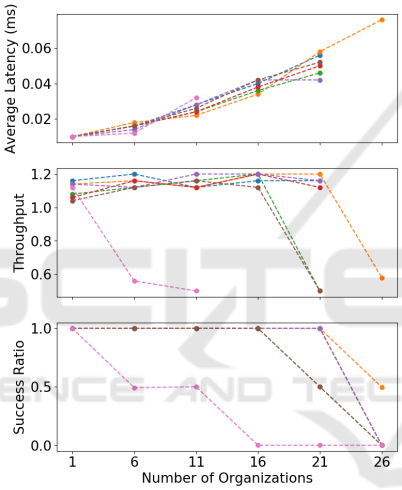
Scaling Fabric to a higher number of participants seems to be non-trivial. One can not simply use any configuration for a higher number of participants and needs to have a deep understanding of Fabric in order to configure the network. Fabric documentation is not satisfactory at the moment but hopefully will be improved.



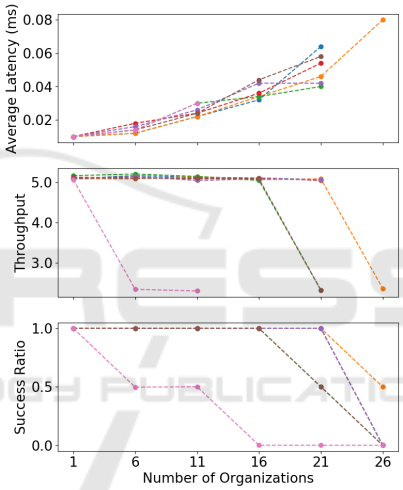
(a) Upload Molecule (1 tps).



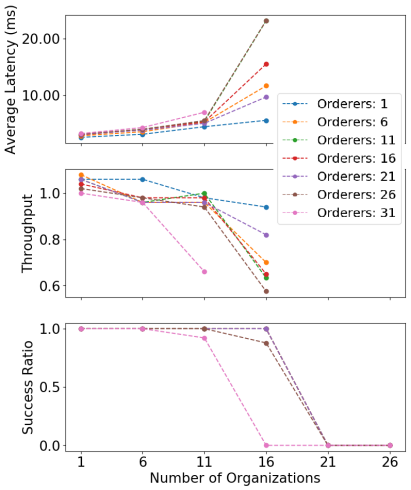
(b) Upload Molecule (5 tps).



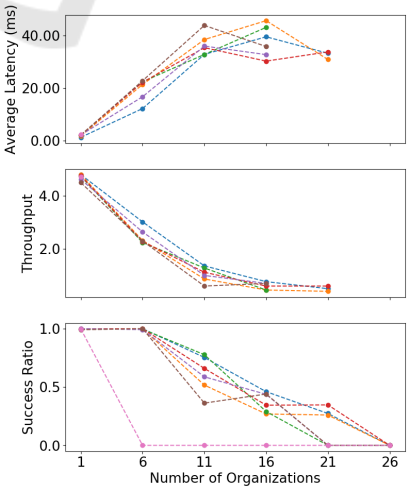
(c) List History (1 tps).



(d) List History (5 tps).



(e) Transfer Ownership (1 tps).



(f) Transfer Ownership (5 tps).

Figure 4: Average latency, throughput, and success ratio for different operations at arrival rates of *1 tps* and *5 tps*. The transactions arrive at a fixed rate for 30 seconds.

6 CONCLUSION

We proposed a solution to the collaborative drug discovery problem (CDDP), based on permissioned blockchain. A prototype was implemented using Hyperledger Fabric. A heuristic security analysis of the proposed solution and static analysis of the deployed chaincode was also accomplished. A performance analysis of the implemented prototype using Caliper shows promising results but extra improvements could be done to the scalability of the system. Larger organizations could use the proposed platform to auction out molecules that they are not interested in bringing to clinical trials or market, smaller organizations could use it to attract funding for a molecule that they have developed, and researchers could use it as proof of their knowledge of properties or structure of a molecule at a certain date.

Future work includes a formal analysis of the proposed solution. Since molecules can be highly-valuable, one needs a strong assurance that the platform works as intended. Moreover, an economic framework that provides incentives for users to participate in the platform would be beneficial. More work on the concept of tokenization is also needed. In particular, how does one make sure that uploaded molecules are authentic? Finally, further research on the Byzantine generals' problem is required. Fabric currently uses Raft as its consensus protocol. However, Raft is not Byzantine fault-tolerant. This imposes restrictions on who can run an orderer, as one has to trust them not to deviate from the consensus protocol. Another question is how to incorporate a Byzantine fault-tolerant protocol into the platform that can support a large number of participants while maintaining acceptable throughput and latency.

ACKNOWLEDGEMENTS

This work was supported by framework grant RIT17-0032 from the Swedish Foundation for Strategic Research and EU H2020 project CloudiFacturing under grant 768892, and was made possible by Vinnova grant 2019-02815. We would like to thank Niclas Nilsson and Paul Stankovski Wagner for fruitful discussions, and anonymous reviewers for a number of useful suggestions for improvement.

REFERENCES

Andrews, D. M., Degorce, S. L., Drake, D. J., Gustafsson, M., Higgins, K. M., and Winter, J. J. (2015). Compound passport service: supporting corporate collection owners in open innovation. *Drug discovery today*, 20(10):1250–1255.

Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., et al. (2018). Hyperledger Fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the 13th EuroSys Conference*, page 30. ACM.

Atkinson, J. D. and Jones, R. (2009). Intellectual property and its role in the pharmaceutical industry. *Future medicinal chemistry*, 1(9):1547–1550.

Beckert, B., Herda, M., Kirsten, M., and Schiffl, J. (2018). Formal specification and verification of Hyperledger Fabric chaincode. In *Proc. Int. Conf. Formal Eng. Methods*, pages 44–48.

Betts, D., Dominguez, J., Melnik, G., Simonazzi, F., and Subramanian, M. (2013). *Exploring CQRS and Event Sourcing: A journey into high scalability, availability, and maintainability with Windows Azure*. Microsoft patterns & practices.

Buterin, V. et al. (2013). Ethereum white paper. *GitHub repository*, pages 22–23.

chainsecurity (2020). Chaincode Scanner. https://chaincode.chainsecurity.com/. Accessed: 2020-09-29.

Conti, M., Kumar, E. S., Lal, C., and Ruj, S. (2018). A survey on security and privacy issues of bitcoin. *IEEE Communications Surveys & Tutorials*, 20(4):3416–3452.

Dobraunig, C., Eichlseder, M., and Mendel, F. (2015). Analysis of SHA-512/224 and SHA-512/256. In *Advances in Cryptology - ASIACRYPT 2015 Auckland, New Zealand, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 612–630. Springer.

Ekins, S. and Bunin, B. A. (2013). The collaborative drug discovery (CDD) database. In *In Silico Models for Drug Discovery*, pages 139–154. Springer.

Hao, Y., Li, Y., Dong, X., Fang, L., and Chen, P. (2018). Performance analysis of consensus algorithm in private blockchain. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 280–285. IEEE.

Heller, S. R., McNaught, A., Pletnev, I., Stein, S., and Tchekhovskoi, D. (2015). Inchi, the iupac international chemical identifier. *Journal of cheminformatics*, 7(1):23.

Hyperledger (2020). Architecture Deep Dive. https://hyperledger-fabric.readthedocs.io/en/release-2.2//. Accessed: 2020-09-29.

Hyperledger Foundation (2020a). Fabric sdk go. https://github.com/hyperledger/fabric-sdk-go. Accessed: 2020-09-29.

Hyperledger Foundation (2020b). Hyperledger caliper. https://www.hyperledger.org/projects/caliperl. Accessed: 2020-09-29.

Küsters, R., Rausch, D., and Simon, M. (2020). Accountability in a permissioned blockchain: Formal analysis of Hyperledger Fabric. *IACR Cryptol. ePrint Arch.*, 2020:386.

Molecule GmbH (2020). Molecule webpage. https://www.molecule.to/. Accessed: 2020-09-21.

Nakamoto, S. et al. (2008). Bitcoin: A peer-to-peer electronic cash system.

Nasir, Q., Qasse, I. A., Talib, M. W. A., and Nassif, A. B. (2018). Performance analysis of Hyperledger Fabric platforms. *Security and Communication Networks*, 2018:3976093:1–3976093:14.

Nguyen, T. S. L., Jourjon, G., Potop-Butucaru, M., and Thai, K. (2019). Impact of network delays on Hyperledger Fabric. *arXiv preprint arXiv:1903.08856*.

O'Boyle, N. M., Banck, M., James, C. A., Morley, C., Vandermeersch, T., and Hutchison, G. R. (2011). Open babel: An open chemical toolbox. *Journal of cheminformatics*, 3(1):33.

Ongaro, D. and Ousterhout, J. (2014). In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference*, pages 305–319.

Performance, H. and Group, S. W. (2018). Hyperledger blockchain performance metrics. Technical report, Hyperledger Foundation.

Praitheeshan, P., Pan, L., Yu, J., Liu, J., and Doss, R. (2019). Security analysis methods on ethereum smart contract vulnerabilities: a survey. *arXiv preprint arXiv:1908.08605*.

Rouhani, S. and Deters, R. (2019). Security, performance, and applications of smart contracts: A systematic survey. *IEEE Access*, 7:50759–50779.

Saha, C. N. and Bhattacharya, S. (2011). Intellectual property rights: An overview and implications in pharmaceutical industry. *Journal of advanced pharmaceutical technology & research*, 2(2):88.

sivachokkapu (2020). ReviveCC. https://github.com/sivachokkapu/revive-cc. Accessed: 2020-09-29.

Sukhwani, H. (2018). Performance modeling & analysis of Hyperledger Fabric (permissioned blockchain network). *Duke University: Duke, UK*.

Thakkar, P., Nathan, S., and Viswanathan, B. (2018). Performance benchmarking and optimizing Hyperledger Fabric blockchain platform. In *26th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2018, Milwaukee, WI, USA, September 25-28, 2018*, pages 264–276. IEEE Computer Society.

Weininger, D. (1988). Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36.

Wouters, O. J., McKee, M., and Luyten, J. (2020). Estimated research and development investment needed to bring a new medicine to market, 2009-2018. *Jama*, 323(9):844–853.

Yamashita, K., Nomura, Y., Zhou, E., Pi, B., and Jun, S. (2019). Potential risks of Hyperledger Fabric smart contracts. In *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*, pages 1–10. IEEE.