Enhancing the privacy and security of Hyperledger Fabric smart contracts using different encryption methods

Rado Stefanov, Prof. Dr. Kaitai Liang
TU Delft

Abstract

Blockchain networks have gained recent popularity among organisations that want to make use of the security aspects that blockchain provides. Fabric is one of the most used distributed network technologies, most commonly applied in scenarios that require confidential data to be stored securely and privately. Use case examples are finance, trading, dispute resolution and healthcare record-keeping. Multiple research has shown that Fabric has vulnerabilities that can allow malicious attackers to obtain access to the data stored in the ledger or the state database. This research presents a symmetric encryption methodology that can be implemented in most of the Fabric smart contracts to protect the stored information in both ledger and state databases. Some drawbacks of the method are increased smart contract execution time, increased storage size, slightly higher code complexity, and limitations when executing CouchDB range queries. In conclusion, although this implementation enhances the security levels of Fabric, other approaches can be used to additionally improve data protection, such as ZKPs and MPC.

1 Introduction

Hyperledger Fabric (Fabric for short) is one of the most used blockchain platforms among corporate organisations and businesses. By definition, Fabric is a distributed ledger technology (DLT) blockchain platform [1]. The blockchain concept has recently gained a high reputation because of cryptocurrencies like Ethereum and Bitcoin. Fabric, however, is a blockchain platform that does not rely on a cryptocurrency. The reason for this is that unlike Bitcoin and Ethereum, which are public and anonymous blockchain networks, Fabric is private and permissioned. It uses the security properties of blockchain to allow identified parties to create their own blockchain networks in order to maintain a shared ledger without the need for cryptocurrency mining, which is well known to utilise high quantities of energy [2]. Blockchain technology allows for data to be stored without a single centralised party responsible for keeping the information. Instead, data is placed in chained immutable blocks, distributed among all the different parties in the network. Every peer in the network has a local copy of the ledger.

Fabric is built with the concept of smart contracts in mind (or *chaincodes* as called in Fabric). Smart contracts are used to encode the business logic of the interaction between the parties of the blockchain network using a programming language. Fabric allows smart contracts to be developed using general-purpose languages like JavaScript and GoLang [3]. *Transactions* generated from smart contracts are stored in the blockchain network. Therefore, the parties will be ensured that once the data is in the ledger, other parties cannot modify it. One major drawback of public blockchain networks is that all of the information is public, and all members of the network are anonymous. For most businesses, this is unacceptable, as their contracts might be confidential, and the identity of the participants must be known. Fabric addresses these issues by allowing companies to create private blockchain networks where all participating organisations need to be identifiable.

The popularity of Fabric is increasing among top-level business companies. Examples of companies that implement Fabric are Oracle, Amazon, SP Global, and others [4]. Common use cases include dispute resolution, trading [5], supply chain finance [6], healthcare record-keeping [7]. In general, use cases often arise in areas where it is a must to keep information private and with limited access to specific members only. Because of this, the security of the platform needs to be perfected as much as possible.

This research paper will focus on the security aspects of smart contracts specific to Fabric. The researched issue in this paper is that data in the ledger is stored in plain text. In the case that an attacker gains access to the information inside the blockchain network, the assets stored will be fully visible to the attacker. This issue allows malicious peers or outside attackers to read private information, which by itself creates privacy issues regarding the data stored in the ledger. In addition, this allows attackers to read keys or other secret data that can be used to actively attack the network and successfully execute malicious transactions. An example is knowing the seed of a random number generator, thus being able to predict what will be the generated random numbers.

Given these issues with the lack of cryptography and privacy, this study aims to answer the following question.

How to enhance the level of protection of data

stored in the ledger of Hyperledger Fabric networks using strong methods of encryption?

The rest of this paper is created to answer the above-stated question. In section 2, a background of how Fabric works is presented. Section 3 shows current research regarding encryption in distributed ledgers and the motivation on why this research is necessary. In section 4, the methodology used to analyse the security and privacy issues is described. A detailed study of the presented methods in this paper is described in section 5. Section 6 will comment on alternative encryption methods involving ZKPs. Section 7 presents a general overview of the responsible research methods used in this paper and the possible problems from implementing the methods presented in this study. Finally, section 8 will conclude the paper and give future recommendations for more research.

2 Background

This section will present an overview of the main architectural components of Fabric that are relevant to the current research. Information on existing built-in privacy tools in Fabric can be found in appendix A.

Hyperledger Fabric main components

Fabric was created with the idea to allow enterprise organisations to make use of the possibilities that blockchain networks give, without the need to spend additional time and resources to teach developers new programming languages or pay fees to validators or maintain a proof-of-work network. The way Fabric works is that it allows organisations to create private blockchain networks. The list of organisations that can participate in the network is strictly defined.

Fabric is composed of the following main components. Each one serves a specific role in the network. Figure 1 shows a schema of a simple Fabric network.

- **Organisations** represent each party participating in the network. There should be at least two organisations in a network for it to be useful. Although organisations are not a physical part of the network, they represent a collection of other Fabric nodes that represent and are controlled by the corresponding organisation.
- **Peers** are the main physical participants of the network. Each peer belongs strictly to one organisation, and each organisation can have multiple peers. The function of the peers is to store a copy of the ledger and execute chaincode functions when an application requests it. Orderers are network peers that ensure deterministic consensus of the transactions. Each peer has a private key stored in the *MSP* [8] that can be used to sign transactions generated by the smart contracts. The respective public certificate is known to all other peers in the network.
- **Orderers** receive transactions from applications, group them into blocks, and distribute the newly created blocks to all connected peers in the network. More on the transaction flow will be presented in the following subsection.

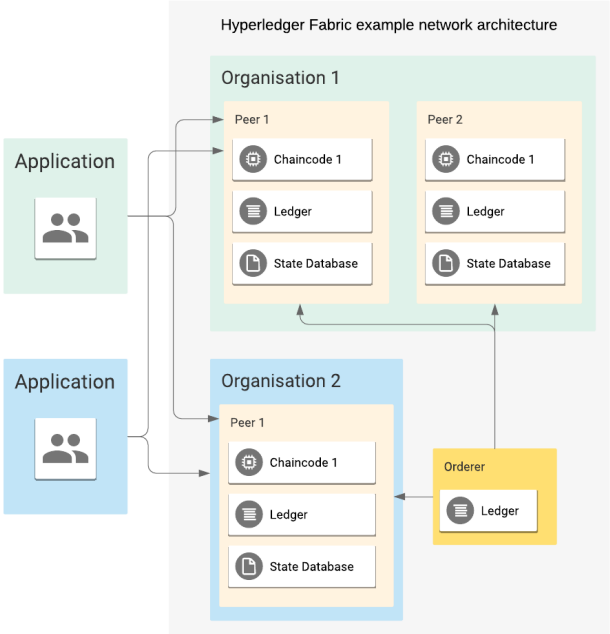


Figure 1: An architecture diagram of a simple Fabric network, consisting of two organisations and one ordering service. One organisation has two peers, and the other organisation has one peer

- **Chaincode** (or smart contract) is the program code that defines the logic of what type of transactions will be stored in the ledger and what are the rules for creating, editing or removing entities in the ledger. Smart contracts can be programmed to fit any business case, as they are written in Turing-complete general-purpose programming languages.
- **Applications** belong to organisations and are the connection between clients and the actual blockchain network. Applications take care of executing transaction requests. They can also make requests to the chaincode to read existing data in the ledger and present it to the application clients. Applications can be implemented as a web interface, mobile application, console application, and many other forms.
- **Ledger and state database** The ledger represents the entire history of all transactions made in a blockchain network. Assets stored in the ledger cannot be modified once committed to the blockchain. Each peer stores a local copy of the ledger, which is the same as for all other peers. Given a ledger, the current state of all assets stored can be derived. For example, if there are N transactions in the ledger, each transaction representing a money transfer between two companies. In that case, a single state can be derived, showing the current amount of money for each person. Peers, together with the ledger, also store a state using a key-value database. Fabric supports LevelDB and CouchDB. Smart contracts can invoke requests on the state database to read necessary information. Peers make sure that the state database

is always up to date.

Architecture of transactions in Fabric

Most of the permissionless blockchain technologies make use of the order-execute transaction architecture. Unlike those, Fabric introduces a new architecture, called execute-order-validate. This method is more scalable and much faster than the other methods used in the other blockchain networks. With this architecture, Fabric allows transactions to be validated in a deterministic way. All of the three steps will be explained below. Figure 2 shows a sequence diagram of the complete transaction process in a network with two organisations and one orderer.

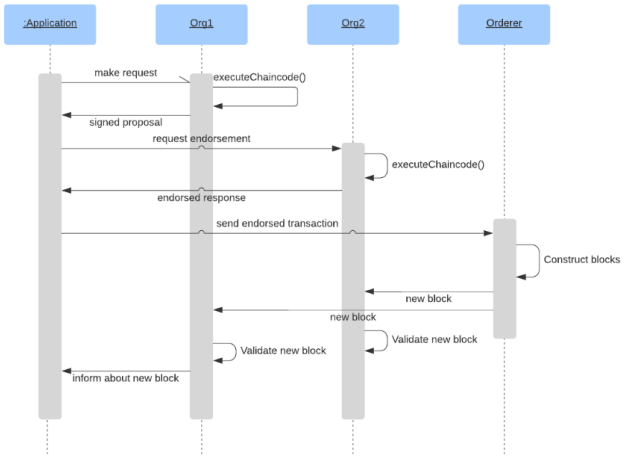


Figure 2: Sequence diagram demonstrating the main steps of a transaction inside Hyperledger Fabric

Execute

When an application wants to write data to the ledger, it needs to get approval from a specific amount of organisations. The rules on who needs to approve a transaction are defined in the *endorsement policy* [9] of the channel. To get the transaction approved, the application sends a proposal to peers from all organisations specified in the endorsement policy. The peers will run the chaincode, and if everything goes well, return a response to the application signed with the private key. For the transaction to be valid, all peers need to return the same result. If the results are different, the transaction is not valid and cannot proceed further. If the transaction is valid, the application sends it to the ordering service, together with all signatures, where the next step of the process begins.

Order

The ordering services have the task to receive transactions from applications, order them in a strict order, group them into blocks, and finally distribute them to all connected peers in the network. Multiple ordering peers can work together using CFT or BFT algorithms like Raft or Kafka [10]. No more details are necessary for this research.

Validate

This is the final step of the transaction. After receiving a new block, peers do not necessarily trust it. Thus, all peers are required to individually validate the received transactions. The peers check if the transaction has been signed by enough peers, as described in the endorsement policy. Only valid transactions become part of the ledger. The others are still kept for audit purposes. All peers execute the validation in the exact same way, and they do not need to invoke the chaincode to ensure validity. This means that all peers have the same copy of the ledger, and no forks can be created in the network.

3 Related work and motivation

According to Putz and Pernul [11], Hyperledger Fabric is one of the most researched distributed ledger frameworks. Their analysis from 2019 shows that Fabric has three times more academic publications than the second most popular distributed ledger network. There are multiple studies performed on the security of Fabric, and some of them showed many present vulnerabilities in the platform. Examples are the potential of executing a DoS attack on the endorsers, as the identity of the endorsers is known to everyone within a channel [12]. The researchers have also shown the possibility of extracting all of the ledger information for all members by compromising a single member. Dabholkar and Saraswat [13] have researched the possible security issues of Hyperledger Fabric. Their paper lists a large number of potential security vulnerabilities involving the implications of a compromised MSP, a malicious Ordering Service and malicious validators. Huang et al. [14] presented a good summary of multiple possible vulnerabilities that can be exploited in Fabric, including the range query risk, program concurrency, read your write, and many others. Yamashita et al. [15] and Brotsis et al. [16] also showed potential vulnerabilities in Fabric chaincodes.

Both channels and private data are available in Fabric out-of-the-box and explained in the documentation. However, they suffer from drawbacks. Research from Benhamouda, Halevi, and Halevi [17] stated some of the disadvantages of using channels and private data. A still open issue with channels is that data is stored in plain text, although part of the organisations would not be able to access it directly. In case one of the peers gets compromised, and an attacker gains access to the ledger, all of the data inside will be readable. This can be catastrophic if sensitive data is stored, which is often the case when smart contracts are used in practice.

Given that the state database is protected by itself, the peers will rely on the protection of CouchDB and LevelDB. Research from Dabholkar and Saraswat [13] stated that the web interface of CouchDB by default has no password protection. Furthermore, the study shows that data can be modified using this vulnerability and other weaknesses present in CouchDB. Therefore, if attackers can change the data, they can also read it, thus see all secret information stored inside. Research from Putz and Pernul [11] supports this claim, stating that because the databases do not offer encryption-at-rest, “anyone with access to the database can read all historical data contained in the ledger. Corda marks the exception: it relies on relational

databases, some of which offer encryption". Private data in Fabric suffers from the same problem. Even though only hashes are stored in the ledger, the original data will not be encrypted unless all peers take care of data encryption themselves.

Multiple research papers state that encrypting data before storing it in the ledger can protect the network [18] [19] . The documentation of Fabric states that data encryption can be used as a form of privacy protection, besides channels and private data [20]. Research performed by Lv et al. [21] finds that "212 chaincode samples in 300 chaincode samples had potential risks". Fabric users can tend to neglect the need to protect their data and write secure and audited smart contracts, as users rely on the fact that Fabric is a private network and that Fabric allows chaincodes to be easily updated if a problem is found after deployment. However, if, due to a vulnerability in the smart contract, private data gets leaked, there is nothing that can be done after that. Even though it would be hard to read the ledger, there are known existing methods for breaching the security of Fabric. In case the chaincode has a weakness that allows exposing the data in the ledger, or in the case where the state database is compromised. Although many sources supporting the idea of encrypting data in Fabric before it is submitted in the ledger, there is a lack of research on how exactly can data be encrypted, what methods to use, and when and where to execute the encryption and decryption. This means that each group of organisations that want to deploy a Fabric network needs to execute their own research on what is the best encryption method.

4 Methodology

At the beginning of the research period, an extensive study on the current state of Fabric security was performed. The study primarily analysed existing research papers regarding blockchain security, smart contract security and Fabric specific security. In addition, the documentation of Fabric [1] was thoroughly reviewed. Although Fabric is a relatively new platform, a number of helpful papers with extensive security research were found. After analysing the papers and concluding that encryption can enhance the security of Fabric networks, the specific research questions were formulated, and the research was now focused specifically on enhancing encryption methods in Fabric.

Furthermore, during the research, smart contracts were created to test and verify several encryption systems. To achieve this, the Test Network [22] documented in the Fabric documentation was deployed. The network consists of two organisations, each one with one peer. In addition, the network includes one ordering service. Because network security is not the main focus of this research, no further improvements were applied to the network to keep it as simple as possible. All parties in the network were installed in a single machine, each deployed in a separate Docker v3.5.1.7 container. The version of Hyperledger Fabric used and tested was v2.3, which was the latest version of Fabric at the time the research was performed.

For developing smart contracts, Fabric allows three programming languages: Java, JavaScript and GoLang. The smart contracts developed in this study use JavaScript. The reason behind this decision is that JavaScript is a more simple and well-known programming language, which will make the code in this paper easier to read. In addition, more libraries related to cryptography are available for JavaScript, which makes it easier to showcase implementations regarding encryption. However, all code shown in this research paper can also be implemented in GoLang and Java. All of the three are general-purpose languages. If a particular library is not available in GoLang or Java, smart contracts can invoke OS-level commands, which can be used to call libraries written in other languages.

Finally, a performance evaluation was executed on the main symmetric encryption method demonstrated in this paper. To evaluate the change of speed in both encryption and decryption operations, a docker container created by a peer was used. The performance evaluation was performed only on the source code, without using real Fabric transactions, because this research was performed in a very limited time frame. The `encrypt` and `decrypt` functions were evaluated by calculating 100 times the average of 1000 operations.

5 Enhance protection from external attackers

This section will present an encryption method for Hyperledger Fabric meant to encrypt the information stored in the ledger and the state database. The data will be encrypted in a way that when external attackers breach the security and gain access to the state database or the ledger, they will not be able to actually read the stored data. The complete implementation can be found in GitHub[1]

Encryption method

The presented encryption method in this section is an extension of the transaction architecture in Hyperledger Fabric. Figure 3 shows the flow of the encryption with all steps. When a new transaction needs to be created, a specific number of peers need to execute a function inside the smart contract and produce the same result. The encryption process adds an additional step after the chaincode has computed the result. Before returning it to the sender, the output data will be encrypted.

Given that the final output of the chaincode needs to be deterministic, meaning that all endorsing peers need to compute the same result, the peers must use the same encryption mechanism and make use of the same encryption key. Because of this, a symmetric encryption approach must be used to ensure determinism. The secret key needs to be shared among all members of the network. Methods on how to store and protect the encryption key will be discussed later in this section.

Implementation

The implementation example extends the smart contract defined in the Hyperledger Fabric test network tutorial [22]. The smart contract stores assets and has functions that can read,

[1] https://github.com/radostefanov/fabric-samples

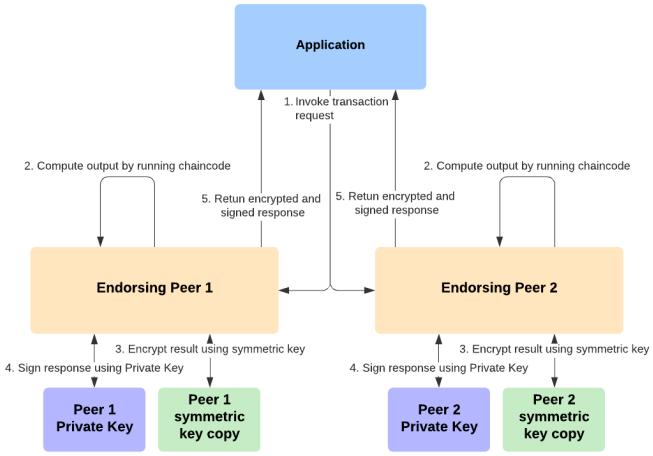


Figure 3: Schema of the required steps needed to perform a transaction. Step 3 is additional to the transaction architecture of Fabric, and it ensures that the data is encrypted, before being sent back to the Application

update and delete the assets. The original implementation stores the information in plain text.

In Listing 1, a version of the `CreateAsset` function is demonstrated. The logic of the code is preserved the same as the original. The only difference is that before invoking `putState`, the code will encrypt the JSON object using the encrypt function. The details of this function will be presented in the next subsection.

```
// CreateAsset issues a new asset to the world state with given
    details.
async CreateAsset(ctx, id, color, size, owner, appraisedValue) {
  const asset = {
    ID: id,
    Color: color,
    Size: size,
    Owner: owner,
    AppraisedValue: appraisedValue,
  };
  const encryptedAsset = encrypt(asset, asset.ID);
  await ctx.stub.putState(id, Buffer.from(JSON.stringify(
    encryptedAsset)));
  return JSON.stringify(encryptedAsset);
}
```

Listing 1: Encryption example inside a Fabric smart contract

In Listing 2, an updated version of the `ReadAsset` method is shown, where the data is decrypted before being sent to the chaincode invoker.

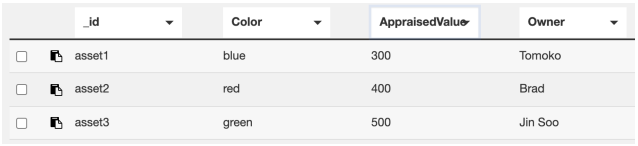
```
// ReadAsset returns the asset stored in the world state with given
    id.
async ReadAsset(ctx, id) {
  const assetJSON = await ctx.stub.getState(id); // get the asset
    from chaincode state
  if (!assetJSON || assetJSON.length === 0) {
    throw new Error(`The asset ${id} does not exist`);
  }

  const asset = JSON.parse(assetJSON.toString());
  const decryptedAsset = decrypt(asset, id);

  return JSON.stringify(decryptedAsset);
}
```

Listing 2: Decryption example inside a Fabric smart contract

Both encryption and decryption are executed via methods defined in the chaincode. Chaincode developers can easily modify existing contracts by including those methods in the smart contract and making sure to encrypt and decrypt data in each of the functions of the ledger. Returning decrypted information must be done carefully. Applications that execute read operations that return decrypted data will be able to read anything. The chaincode can use the decrypt method to read information for validation purposes without returning it to the invoker.

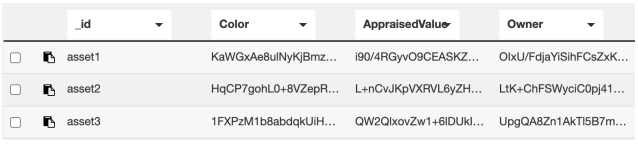
Another way to execute this implementation is to use an Endorsement plugin [23]. With this approach, the logic for encrypting can be extracted in a plugin, and it will take care of encrypting the information before endorsing it. The use of such a plugin can make the code in the chaincode simpler. However, the organisations will need to install and maintain an additional plugin, which might outweigh the benefits of not including the encryption in the smart contract itself. Besides that, all information that is retrieved from the ledger still needs to be decrypted. For these reasons, this implementation example uses only code defined in the smart contract.

Figure 4 shows the state of the CouchDB database both with and without encryption.



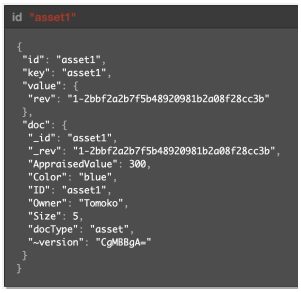
	_id	Color	AppraisedValue	Owner
	asset1	blue	300	Tomoko
	asset2	red	400	Brad
	asset3	green	500	Jin Soo

(a) CouchDB overview of assets without encryption

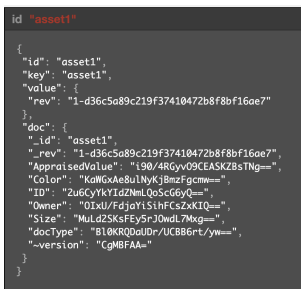


	_id	Color	AppraisedValue	Owner
	asset1	KaWGxAe8ulNyKjBmz...	i90/4RGyvO9CEASKZ...	OlxU/FdjaYiSihFCsZxK...
	asset2	HqCP7gohL0+8VZepR...	L+nCvJKpVXRVL6yZH...	LtK+ChFSWyciC0pj41...
	asset3	1FXPzM1b8abdqkUiH...	QW2QlxovZw1+6IDUkl...	UpgQA8Zn1AkTl5B7m...

(b) CouchDB overview of assets with encryption



(c) CouchDB single asset without encryption



(d) CouchDB single asset with encryption

Figure 4: CouchDB overview of encryption vs no encryption

Encryption and decryption functions

The purpose of the encrypt and decrypt functions is to accept a JSON object and execute respectively encryption or decryption of each field in the JSON object. The received object represents a single asset. Listings 3 and 4 show a suggested example of how to implement the functions. The algorithm

used is AES [24]. Atwal and Kumar [25] and Ebrahim, Khan, and Khalid [26] showed that this is the most secure symmetric encryption algorithm, compared with other popular symmetric encryption algorithms, such as RSA DSS, 3DES, Blowfish, and others. The library providing the algorithm is called `crypto-js` [27]. The library contains different algorithms. This paper will not focus on the specifics of the library.

```
const AES = require("crypto-js/aes");
const SHA256 = require("crypto-js/sha256");
const deterministicEncryption = { mode: CryptoJS.mode.ECB }

function encrypt(asset, id) {
  // Step 1 and 2
  const symmetricKey = getSymmetricKey();
  const secret_key = CryptoJS.enc.Utf8.parse(
    SHA256(symmetricKey + id).toString()
  );
  // Step 3 and 4
  const encryptedResult = {};
  Object.keys(asset).forEach((key) => {
    encryptedResult[key] = AES.encrypt(
      JSON.stringify( {v: asset[key] } ),
      secret_key, deterministicEncryption
    ).toString();
  });
  return encryptedResult;
}
```

Listing 3: Decryption function implementation

The encryption and decryption process consists of the following four steps.

1. Obtain the shared symmetric key that all peers must use to encrypt and decrypt information. This is achieved through the `getSymmetricKey()` function. The implementation details of that function will be discussed in the next subsection.
2. Compute the secret key that will be used for the AES encryption. The key is computed by hashing a combination of a secret private key and the ID of the transaction, using the `SHA256` hashing function. This ensures that each transaction has a different symmetric key and adds an additional layer of protection. In the case that brute force is applied, and one of the transactions is decrypted, the others will require an equal amount of work, as the key used there will be completely different from the keys in all other transactions.
3. Both encrypt and decrypt functions must traverse the passed JSON object.
   For each field, the encryption function creates a new JSON object that acts as a wrapper and contains a single key with a value equal to the raw text value of the corresponding field. After that, the JSON is converted into a string. This is done to preserve the data type of the raw input value. After the JSON is converted into a string, the resulted string is encrypted using AES. The response is assigned to a new object, which will be later returned.
   Similarly, the decryption function uses AES to decrypt the corresponding JSON field. The original raw value is extracted from the wrapping object and inserted into a new object that will represent the decrypted version of the JSON input.
4. The encrypted / decrypted object is returned as a copy. The original input remains intact.

```
const AES = require("crypto-js/aes");
const SHA256 = require("crypto-js/sha256");
const deterministicEncryption = { mode: CryptoJS.mode.ECB }

function decrypt(asset, id) {
  // Step 1 and 2 same as encryption method from Listing 3
  ...
  // Step 3 and 4
  const decryptedResult = {};
  Object.keys(asset).forEach((key) => {
    const decryptedString = AES.decrypt(
      asset[key],
      secret_key, deterministicEncryption
    ).toString(CryptoJS.enc.Utf8);
    decryptedResult[key] = JSON.parse(decryptedString).v
  });
  return decryptedResult;
}
```

Listing 4: Encryption function implementation

Private key management

This symmetric encryption method requires that each organisation participating in the network has access to a common private key. The organisations must agree in advance of what the key will be, and make sure to communicate the key using a protected channel, for example, by using asymmetric encryption. Keeping the key secure is of major importance, as even if only one organisation gets compromised and the key becomes accessible to an attacker, the state database and ledger will become readable for the attacker if they successfully gain access to them. This means that organisations need to be extra cautious on how to store the key, and how to pass the key to the smart contracts.

Several possibilities for storing the symmetric encryption key are the following.

- Storing the key **inside the chaincode docker container**. From the Fabric documentation, "Chaincode runs in a secured Docker container isolated from the endorsing peer process" Hyperledger [28]. Each peer contains a separate docker container for each chaincode that is installed. By default, the chaincode docker containers come with a directory `/etc/hyperledger/fabric`, where TLS-related private keys and certificates are stored. These keys are used to maintain a secure connection when the container needs to communicate with the peer or the orderer. The symmetric key can be placed manually or through a script inside this folder. A big benefit of this approach is that the chaincode can access it directly, as it is deployed in the same container. Listing 5 demonstrates how the `getSymmetricKey()` function used in the decrypt and encrypt methods can be implemented, using this approach. Because of the high levels of security that Fabric ensures for the chaincode docker container and the easy implementation, this method of storing the encryption key is recommended.
- Storing the key **inside the peer docker container**. This method is similar to the previous option. However, by placing the symmetric key in the peer container, the chaincode does not have direct access to it. The peer would need to invoke the chaincode as usual, and after a response is returned, encrypt it using the symmetric key before the transaction proposal is returned to the invoking application. For this to be implemented, an

endorsement plugin needs to be used. An additional level of security, also mentioned by Putz and Pernul [11] is to encrypt the symmetric key using the private key of the peer. This method is not part of the current implementation shown in this study, but it is a possible extension. [23].

- Storing the key **inside the applications**. This is the third physical place where the symmetric keys can be stored. In order to achieve encryption, the symmetric key would need to be passed as an argument each time a chaincode function is invoked from the application. This option, however, has many drawbacks. Applications are usually more close to the end-users, meaning that they have a higher risk of being targeted by attackers. In addition, applications are large in size, compared to smart contracts, which are usually composed of only a single file. The final drawback is that by allowing applications to have access to the symmetric keys, the ledger data can be fully read by the application, whereas if the key is accessible only from the smart contract, the logic inside it can restrict certain fields from being visible to the application.
- Storing the key **inside the smart contract**. This is the most straightforward way to keep the key. This method is highly unsafe, as in case the attacker gains access to the chaincode, they will be able to see the key and decrypt the data. To increase the security, the key can be passed as an argument to the chaincode.
- Other options include storing the key in a **central server** that acts as an oracle. Chaincode can access the key via an HTTP call. Although this method is easy to implement, it creates other security vulnerabilities and relies on the security of the webserver implementation. This means that if the server itself gets compromised, the symmetric key might be leaked.

```
const { readFileSync } = require("fs");

const getSymmetricKey = () =>
  readFileSync("/etc/hyperledger/fabric/ledger_encryption.key")
  .toString();
```

Listing 5: Retrieve the symmetric encryption key from the corresponsing directory

Additional security improvements

Besides simply encrypting the ledger information, additional methods can be used to further protect the privacy of the data store in Fabric. There exist tools that help blockchain developers create more secure smart contracts. *Tineola* [29] is one such tool created for Hyperledger Fabric smart contracts. One of the suggestions that the tool gives is to hash private information that does not need to be accessible in raw format after it has been stored in the ledger. An example is passwords, as the chaincode can always verify if a password matches a stored password by hashing it. By hashing the data, the level of protection increases, as even if the symmetric key is compromised, any hashed information in the ledger will still be inaccessible by the attackers.

Another additional improvement to the methodology is increasing the security level by making use of encryption algorithms that are post-quantum resistant. Post-quantum algorithms are encryption methods that assume that the attackers have access to a large quantum computer. Research from Andrushkevych et al. [30] showed that AES encryption, together with other symmetric encryption algorithms, can be vulnerable to quantum attacks. Bernstein and Lange [31] showed possible post-quantum asymmetric encryption algorithms, such as Lattice-based encryption and Code-based encryption. They can be used to encrypt the symmetric key and protect it from quantum attacks.

Performance analysis

This encryption mechanism adds an additional layer of computation, and it will always perform worse than not having encryption. To calculate what exactly is the impact, two different performance analysis were performed. The first one measures how slower is the execution of a smart contract with encryption. The second analysis calculates what is the impact of the encryption in terms of storage. Details on how the experiment was performed can be seen in section 4. The source code used for the performance analysis can be found in GitHub[2].

The performance analysis of the smart contract execution time for both encryption and decryption can be seen in Figure 5. Clearly, the additional layer of computation makes the execution much slower. On average, read operations take around 466ms with encryption. The read time without encryption is measured to be 20ms. For writing, the average is 623ms with encryption, and 30ms without. It can be seen that adding encryption results in a significant performance penalty. However, the experiment measures only the pure smart contract execution time. As shown in the paper from Thakkar, Nathan, and Viswanathan [32], there are multiple operations that Fabric performs, which together add up to the total transaction time. The smart contract execution is only a small part of it. In addition, the performance of this implementation can be increased by optimising the algorithm. However, this is not the main focus of this research.

In terms of storage, Figure 5 shows the relation between the size of the raw input and the encrypted version. As it can be seen, there is a linear dependency between the input and the output for AES encryptions. The encrypted variant is 1.4272 times larger than the non-encrypted version. This increase of storage required is quite acceptable, as the increase rate is only linear.

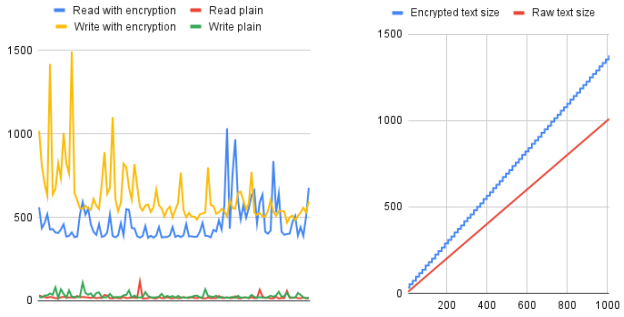
Both increases in speed and storage size need to be taken into account when using this encryption approach. For applications that need low transaction time or are executed in limited hardware scenarios, like IoT devices, this encryption method might be too expensive to execute. A possible optimization is to use a faster encryption algorithm, or a faster library. Other algorithms can also be used to reduce the required memory. As shown by Kansal and Mittal [33], AES requires more memory than DES and 3DES.

[2] https://github.com/radostefanov/fabric-samples

Recommendation for use in practice

Developers must take into account that this method has drawbacks and does not guarantee the overall security of the Fabric network. It rather only provides an enhancement that can protect the ledger information in case an attacker gains access to the network. The encryption method does not protect the ledger from internal attackers, as everyone in the network must know the symmetric key that will be used to encrypt and decrypt the data. In addition, in the case that the key is compromised, the organisations will need to relaunch their network, using a different symmetric key, or re-encrypt all entities in the ledger. Both options are hard to execute and will cost time and resources. Other limitations are present. For example, executing range queries using CouchDB will be more complicated, and in some cases not possible, as CouchDB on its own cannot read the values of the encrypted data.

However, implementing this encryption solution increases the security in a relatively cheap way, as the modifications required in the chaincode consist of making sure to decrypt the information when read, and encrypt it when it is stored back to the ledger.



(a) Performance comparison of read and write, with and without encryption in milliseconds

(b) A comparison between the size of the raw input and the size of the encrypted result

Figure 5: Performance analysis of symmetric encryption implementation

6 Protect stored information from internal parties

This section will present an overview of possible methods that can be used to encrypt and protect the information in a Fabric network in such a way that members in the network cannot read the information, but still execute smart contract logic for verification and auditing.

Zero Knowledge Proofs in Fabric

The main problem with protecting information from internal members is that by default, everyone has access to the same ledger. This dilemma is stated by Benhamouda, Halevi, and Halevi [17] "If everyone sees the same ledger, how can we have private data that some can see but others cannot?".

A solution to this problem can be achieved by using Zero Knowledge Proofs (ZKPs) [34]. The key principle is that information is encrypted before it is sent to the chaincode. The chaincode will not be able to read the information, but it can derive properties of the fields. Useful ZKPs can be range proofs, which can be used to convince a network peer that a number is within a certain range. There are multiple tools developed for Fabric. Examples are FabZK [35] and zkrpChain [36].

A potential use case where ZKPs can be useful is voting [37]. Validity of votes can be checked by using range proofs. This research performed a possible implementation[3] of a voting smart contract using Fabric and Paillier [38] encryption. Because of the additive homomorphism property, votes can be added together without they being decrypted. Thus, only the peer that has the corresponding private key can see the final result.

Limitations and comparison with symmetric encryption

Most of the ZKP research is concentrated on public blockchains, as encrypting them is much more important because everyone has access to the ledgers. Although Fabric is a private blockchain network, ZKPs can also be applied for encrypting the data. ZKPs, however, have some drawbacks. Implementing them takes a considerable amount of time, as they are highly complex and require developers to understand complicated mathematical structures. In addition, ZKPs are relatively new and are still under research, which means that potential vulnerabilities within them might be present. Swihart, Winston, and Bowe [39] show a found zero-knowledge proof vulnerability in Zcash, which is a permissionless blockchain technology.

7 Responsible research

This chapter will present an ethical analysis of the obtained results from this research paper. The first part is a discussion regarding the moral and ethical implications of this research. In the second part, an analysis of the reproducibility will be presented.

Research Integrity

Parts of this paper reflect on potential security vulnerabilities in Hyperledger Fabric. The report contains results and recommendations for developers and organisations using Fabric to enhance the security of smart contracts. However, the presented exploits can be utilised by malicious parties to execute attacks on existing smart contracts. This risk is considered in the paper. The list of attacks is already known, as it is extracted from existing research papers. The conclusions formed by the performed security analysis only show additional fixes to already known security vulnerabilities. Thus, this research gives more opportunities for developers to enhance the security than it gives floor for malicious members to perform attacks.

Another part of this research considers enhancing the privacy of Fabric by introducing more effective encryption

[3] https://github.com/radostefanov/fabric-samples

methods in the data stored in the ledger and state database. The solutions presented can be applied to any smart contract. The enhanced privacy of Fabric can allow more organisations to make use of the tool in cases when privacy issues have been a barrier for those organisations. By itself, enhanced encryption does not pose immediate threats. However, as explained by Falk [40], tools alone do not have ethical implications, but it rather depends entirely on what is the purpose of the products developed using this tool. In the case of Fabric privacy, unethical smart contracts can be developed by organisations using the encryption methods discussed in this paper.

Several smart contracts were developed during the research period and used for demonstration and showcasing security and privacy vulnerabilities. All smart contracts, although representing possible real scenarios, are not real. No personal data was used to produce the paper, and all of the code used in the smart contracts are open source and listed in the references section. During the research process, the potential conflict of interest and bias were mitigated as much as possible. The research was not performed by members that participate in the Hyperledger Fabric development, nor own or maintain products using Fabric. The research is done in the most objective way possible. The absence or presence of vulnerabilities does not influence the researchers. Finally, the research is not financed by anyone.

Reproducibility

To facilitate reproducibility, the following measurements are taken in this research. All versions of the tools used are listed in the respective sections. The general methods shown in this paper are applicable to Hyperledger Fabric v2.3. Future versions of Fabric might not be vulnerable to the presented attacks, and the privacy enhancement methods might not be applicable. Besides the code shown in the actual paper, the full version of the code is linked in an open-source GitHub[4] repository.

8 Conclusions and future work

This research was executed to analyse the current state of security and privacy in Hyperledger Fabric and propose a method to enhance the security. The demonstrated method of encrypting the ledger data and state database using symmetric encryption can increase the level of protection in the case external malicious attackers gain access to the data stored in a Fabric network. The presented security enhancement is recommended for smart contracts that do not perform complex CouchDB range queries and can afford an increase in execution time for more security.

The main issues of this symmetric encryption approach are worse performance for both time and space, slightly higher code complexity, and limitations when executing CouchDB range queries. More research can be performed to measure the performance implications in real scenarios, improve the performance, and study possible additional drawbacks of this encryption method when used in production-level smart contracts.

[4] https://github.com/radostefanov/fabric-samples

More research on the area of ZKPs can significantly improve the encryption possibilities, also mitigate the situation where attackers are already inside the network. The existing tools show promising results, but the range of applications does not cover all possible smart contract scenarios. The high levels of complexity are also a significant drawback. Thus more comprehensive libraries need to be developed with good documentations.

Unfortunately, symmetric encryption methods are not present in the documentation of Hyperledger Fabric, and there is a general lack of research in this direction. This paper recommends Fabric to include in the documentation an explanation for executing symmetric encryption. Implementing this encryption method as a plugin directly in the system chaincode will also avoid having encryption and decryption logic in the smart contracts.

References

[1] Hyperledger. *Introduction*. 2020. URL: https://hyperledger-fabric.readthedocs.io/en/release-2.3/whatis.html (cit. on pp. 1, 4).

[2] Jingming Li et al. "Energy consumption of cryptocurrency mining: A study of electricity consumption in mining cryptocurrencies". In: *Energy* 168 (2019), pp. 160–168. ISSN: 0360-5442. DOI: https://doi.org/10.1016/j.energy.2018.11.046. URL: https://www.sciencedirect.com/science/article/pii/S0360544218322503 (cit. on p. 1).

[3] Hyperledger. *Smart Contracts and Chaincode*. 2020. URL: https://hyperledger-fabric.readthedocs.io/en/release-2.3/smartcontract/smartcontract.html (cit. on p. 1).

[4] HG Insights. *Companies Currently Using Hyperledger Fabric*. 2021. URL: https://discovery.hgdata.com/product/hyperledger-fabric (cit. on p. 1).

[5] Elli Androulaki et al. "Hyperledger fabric: a distributed operating system for permissioned blockchains". In: *Proceedings of the thirteenth EuroSys conference*. 2018, pp. 1–15 (cit. on p. 1).

[6] Chaoqun Ma et al. "The privacy protection mechanism of Hyperledger Fabric and its application in supply chain finance". In: *Cybersecurity* 2.1 (2019), pp. 1–9 (cit. on p. 1).

[7] Baddepaka Prasad and S Ramachandram. "Decentralized Privacy-Preserving Framework for Health Care Record-Keeping Over Hyperledger Fabric". In: *Inventive Communication and Computational Technologies*. Springer, 2021, pp. 463–475 (cit. on p. 1).

[8] Hyperledger. *Membership Service Provider (MSP)*. 2020. URL: https://hyperledger-fabric.readthedocs.io/en/release-2.3/membership/membership.html (cit. on p. 2).

[9] Hyperledger. *Endorsement policies*. 2020. URL: https://hyperledger-fabric.readthedocs.io/en/release-2.3/endorsement-policies.html (cit. on p. 3).

[10] Hyperledger. *The Ordering Service*. 2020. URL: https://hyperledger-fabric.readthedocs.io/en/release-2.3/orderer/ordering_service.html (cit. on p. 3).

[11] Benedikt Putz and Günther Pernul. "Trust Factors and Insider Threats in Permissioned Distributed Ledgers". In: *Transactions on Large-Scale Data- and Knowledge-Centered Systems XLII*. Ed. by Abdelkader Hameurlain and Roland Wagner. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019, pp. 25–50. ISBN: 978-3-662-60531-8. DOI: 10.1007/978-3-662-60531-8_2. URL: https://doi.org/10.1007/978-3-662-60531-8_2 (cit. on pp. 3, 7).

[12] Nitish Andola et al. "Vulnerabilities on Hyperledger Fabric". In: *Pervasive and Mobile Computing* 59 (2019), p. 101050. ISSN: 1574-1192. DOI: https://doi.org/10.1016/j.pmcj.2019.101050. URL: https://www.sciencedirect.com/science/article/pii/S157411921830720X (cit. on p. 3).

[13] Ahaan Dabholkar and Vishal Saraswat. "Ripping the Fabric: Attacks and Mitigations on Hyperledger Fabric". In: Nov. 2019, pp. 300–311. ISBN: 978-981-15-0870-7. DOI: 10.1007/978-981-15-0871-4_24 (cit. on p. 3).

[14] Yongfeng Huang et al. "Smart Contract Security: A Software Lifecycle Perspective". In: *IEEE Access* 7 (2019), pp. 150184–150202. DOI: 10.1109/ACCESS.2019.2946988 (cit. on p. 3).

[15] Kazuhiro Yamashita et al. "Potential Risks of Hyperledger Fabric Smart Contracts". In: *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. 2019, pp. 1–10. DOI: 10.1109/IWBOSE.2019.8666486 (cit. on p. 3).

[16] Sotirios Brotsis et al. "On the Security and Privacy of Hyperledger Fabric: Challenges and Open Issues". In: *2020 IEEE World Congress on Services (SERVICES)*. 2020, pp. 197–204. DOI: 10.1109/SERVICES48979.2020.00049 (cit. on p. 3).

[17] F. Benhamouda, S. Halevi, and T. Halevi. "Supporting private data on Hyperledger Fabric with secure multiparty computation". In: *IBM Journal of Research and Development* 63.2/3 (2019), 3:1–3:8. DOI: 10.1147/JRD.2019.2913621 (cit. on pp. 3, 8).

[18] Huru Hasanova et al. "A survey on blockchain cybersecurity vulnerabilities and possible countermeasures". In: *International Journal of Network Management* 29.2 (2019). e2060 NEM-18-0162.R1, e2060. DOI: https://doi.org/10.1002/nem.2060. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nem.2060. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.2060 (cit. on p. 4).

[19] Benedikt Putz and Günther Pernul. "Detecting Blockchain Security Threats". In: *2020 IEEE International Conference on Blockchain (Blockchain)*. 2020, pp. 313–320. DOI: 10.1109/Blockchain50366.2020.00046 (cit. on p. 4).

[20] Hyperledger. *Fabric Frequently Asked Questions*. 2020. URL: https://hyperledger-fabric.readthedocs.io/en/release-2.3/Fabric-FAQ.html (cit. on p. 4).

[21] Penghui Lv et al. *Potential Risk Detection System of Hyperledger Fabric Smart Contract based on Static Analysis*. Tech. rep. EasyChair, 2021 (cit. on p. 4).

[22] Hyperledger. *Using the Fabric test network*. 2020. URL: https://hyperledger-fabric.readthedocs.io/en/release-2.3/test_network.html (cit. on p. 4).

[23] Hyperledger. *Pluggable transaction endorsement and validation*. 2020. URL: https://hyperledger-fabric.readthedocs.io/en/release-2.3/pluggable_endorsement_and_validation.html (cit. on pp. 5, 7).

[24] Prerna Mahajan and Abhishek Sachdeva. "A study of encryption algorithms AES, DES and RSA for security". In: *Global Journal of Computer Science and Technology* (2013) (cit. on p. 6).

[25] Er Shikha Atwal and Umesh Kumar. "A Comparative Analysis of Different Encryption Algorithms: RSA, AES, DSS for Data Security". In: (2021) (cit. on p. 6).

[26] Mansoor Ebrahim, Shujaat Khan, and Umer Bin Khalid. "Symmetric algorithm survey: a comparative analysis". In: *arXiv preprint arXiv:1405.0398* (2014) (cit. on p. 6).

[27] BRIX. *Crypto-JS*. 2021. URL: https://github.com/brix/crypto-js (cit. on p. 6).

[28] Hyperledger. *Fabric chaincode lifecycle*. 2020. URL: https://hyperledger-fabric.readthedocs.io/en/release-2.3/chaincode_lifecycle.html (cit. on p. 6).

[29] *Tineola*. URL: https://github.com/tineola/tineola/blob/master/docs/TineolaWhitepaper.pdf (cit. on p. 7).

[30] Alina Andrushkevych et al. "The block symmetric ciphers in the post-quantum period". In: *2016 Third International Scientific-Practical Conference Problems of Infocommunications Science and Technology (PIC S&T)*. IEEE. 2016, pp. 43–46 (cit. on p. 7).

[31] Daniel J Bernstein and Tanja Lange. "Post-quantum cryptography". In: *Nature* 549.7671 (2017), pp. 188–194 (cit. on p. 7).

[32] Parth Thakkar, Senthil Nathan, and Balaji Viswanathan. "Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform". In: *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. 2018, pp. 264–276. DOI: 10.1109/MASCOTS.2018.00034 (cit. on p. 7).

[33] Shaify Kansal and Meenakshi Mittal. "Performance evaluation of various symmetric encryption algorithms". In: *2014 International Conference on Parallel, Distributed and Grid Computing*. 2014, pp. 105–109. DOI: 10.1109/PDGC.2014.7030724 (cit. on p. 7).

[34] Oded Goldreich and Yair Oren. "Definitions and properties of zero-knowledge proof systems". In: *Journal of Cryptology* 7.1 (1994), pp. 1–32 (cit. on p. 8).

[35] Hui Kang et al. "FabZK: Supporting Privacy-Preserving, Auditable Smart Contracts in Hyperledger Fabric". In: *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2019, pp. 543–555. DOI: 10.1109/DSN.2019.00061 (cit. on p. 8).

[36] Shiwei Xu et al. "zkrpChain: Privacy-Preserving Data Auditing for Consortium Blockchains Based on Zero-Knowledge Range Proofs". In: *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. 2020, pp. 656–663. DOI: 10.1109/TrustCom50675.2020.00092 (cit. on p. 8).

[37] Mohammed Khaled Mustafa and Sajjad Waheed. "An E-Voting Framework with Enterprise Blockchain". In: *Advances in Distributed Computing and Machine Learning*. Springer, 2021, pp. 135–145 (cit. on p. 8).

[38] daylightingsociety. *Paillier Cryptosystem*. URL: https://paillier.daylightingsociety.org/about (cit. on p. 8).

[39] Josh Swihart, Benjamin Winston, and Sean Bowe. "Zcash Counterfeiting Vulnerability Successfully Remediated". In: (). URL: https://electriccoin.co/blog/zcash-counterfeiting-vulnerability-successfully-remediated/ (cit. on p. 8).

[40] Courtney Falk. "The Ethics of Cryptography". PhD thesis. May 2005 (cit. on p. 9).

[41] Hyperledger. *Channels Architecture*. 2020. URL: https://hyperledger-fabric.readthedocs.io/en/release-2.3/channels.html (cit. on p. 11).

[42] Hyperledger. *Private Data*. 2020. URL: https://hyperledger-fabric.readthedocs.io/en/release-2.3/private-data/private-data.html (cit. on p. 11).

[43] Hyperledger. *Gossip Protocol*. 2020. URL: https://hyperledger-fabric.readthedocs.io/en/release-2.3/gossip.html (cit. on p. 11).

A Existing built-in data protection tools in Fabric

Fabric offers two built-in mechanisms to protect private information in the ledger: *channels* and *private data*. The first one, which is the most commonly used one, is the concept of *channels* [41]. Channels allow part of the organisations in a Fabric network to create a separate sub-network, with a specified part of the members in the original network. This concept allows organisations to protect private data from other organisations in the network. A simple example is a network of three organisations consisting of one manufacturer and two resellers. In case that the manufacturer wants to create a special offer to one of the resellers, they can create a sub-channel with only those two organisations being part of it. The other reseller will not be able to see the data in the channel. The party will not even be able to know that such a channel exists [41].

The second tool provided by Hyperledger Fabric is *private data* [42]. Using this tool, organisations from the network can avoid saving private information in the ledger. Instead of saving the plain data into the ledger, peers will store only the hash of the data, while the actual data will be distributed between all peers that should access it using a peer-to-peer communication protocol named gossip [43]. This method protects the data, as other organisations that just have access to the ledger cannot compute the original data. In addition, in case an additional party needs to be added to the list of allowed members, they can easily verify if the data is correct by using the stored hashes in the ledger and recomputing the hashes of the data when the data is provided.