Blockchain Security for IoT

Sean Zhong & Richard Johansson
Email: {seazh870,ricjo462}@student.liu.se
Supervisor: Andrei Gurtov, andrei.gurtov@liu.se
Project Report for Information Security Course
Linköping University, Sweden

ABSTRACT

Unmanned Aircraft (UA), such as drones, are becoming increasingly popular in society and need to be securely identifiable, and Drone Remote Identification Protocol (DRIP) with the Hyperlegder Iroha blockchain as registry could solve the problem. In this study, a small example network is set up and the security aspects are evaluated and it's concluded that location updates of UA:s can be securely stored and accessed in a blockchain, such as the Hyperledger Iroha blockchain.

1 INTRODUCTION

Internet of Things (IoT) devices, such as drones, are becoming increasingly popular, both in business and for personal use. This leads to many possible security vulnerabilities, such as malicious actors fetching sensitive data or even taking control of the devices. These vulnerabilities are exceptionally important to mitigate when using IoT devices in environments such as the medical, military, and other critical fields.

Drone Remote Identification Protocol (DRIP) is a newly proposed protocol that aims to securely identify Unmanned Aircrafts (UA) such as drones [1]. DRIP also enables trustful messages between UAs and other actors. DRIP introduces a way for observers, such as the military, to gather information about the presence of UAs in their airspace. Within the DRIP protocol, confidential information must be restricted to authorized personnel. The DRIP protocol is meant to be applied to all entities in an Unmanned Aircraft Systems (UAS) network and all communication between them.

The DRIP protocol has not yet determined the exact implementation of the registries and Network Remote Identification (Network RID) methods, i.e., how to identify a UA through the network. A solution to this problem is proposed by Y. Hashem, E. Zildzic, and A. Gurtov by using the Hyperlegder Iroha blockchain [2]. In their study, they present a proof-of-concept (POC) that Hyperlegder Iroha can be used as a permission-based blockchain network that both has good performance and scalability as registry in DRIP. The use of blockchain technology to secure drones and their networks is also covered in [3].

This project continues their study and a test environment based on DRIP and Hyperleger Iroha were created, where Raspberry Pi:s worked as UA devices and was connected to the network. The test environment was updated to the latest version of Hyperledger Iroha. Furthermore, the security aspects of the protocol were evaluated, and attacks, such as adding fake nodes, changing information of a node, and dropping nodes were tested and evaluated.

The paper is focused on answering the following questions: in a network of UAs connected with the DRIP protocol using Hyperledger Iroha as registry, can a malicious actor...

- change the information on a UA?
- remove other UAs from the network?
- add UAs to the registry?
- get details about other UAs on the network?

The rest of the paper is structured as follows, section 2 presents some background and theory needed to understand the study, section 3 consists of the methodology used to conduct the study, section 4 presents the results and section 5 discusses the results. A security analysis is conducted in section 6 and a performance evaluation is made in section 7. All of this is then concluded by the end of the paper in section 8.

2 BACKGROUND

This section presents the relevant background and theory needed to understand the study. Presented parts are the DRIP protocol, blockchains in general, and the Hyperledger Iroha blockchain.

2.1 Drone Remote Identification Protocol

Drone Remote Identification Protocol (DRIP) is a protocol published in 2022 that defines solutions to the problem of Unmanned Aircraft System Remote Identification and Tracking (UAS RID) [1]. The protocol is focused on security and safety and uses already existing Internet resources to verify that the information stored and sent is trustworthy, both online and offline.

DRIP states small UA such as drones to be "low observable" since they are small and highly maneuverable, quiet and typically fly at low altitudes. However, they can act as threats by carrying weapons, such as sensors and cyber weapons, and themselves can be used as weapons by flying into targets.

There are a set of requirements that apply to DRIP, which are separated into four subsets: general, identifier, privacy,

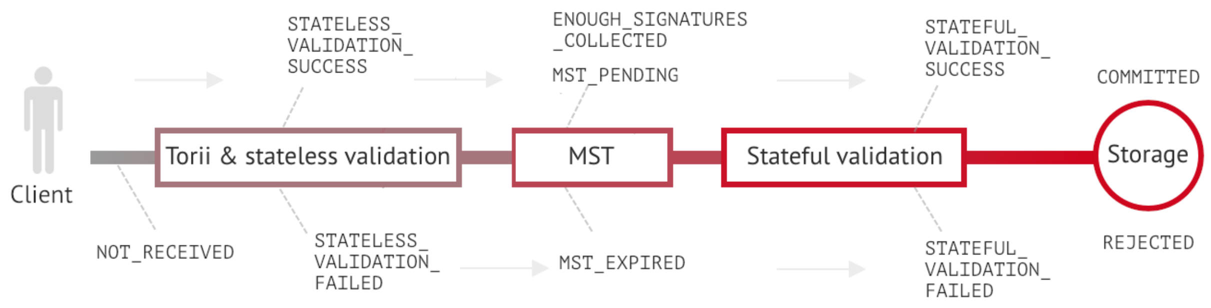


Figure 1: Event stream for each transaction made in Hyperledger Iroha [4].

and registries. There is no registry information model determined for DRIP, a part from choosing the UAS ID as a unique key, and since this study focuses on DRIP with Hyperledger Iroha as the register, the registries part is the most important. There are four normative requirements for a registry in DRIP:

- *REG-1: Public lookup* - Given a UAS ID, the register must give the public information regarding the UAS, regardless of the identity or role of the submitter.
- *REG-2: Private lookup* - Given a UAS ID, the register must give the private information regarding the UAS, if the submitter has the right to see such information in regards to their identity or role.
- *REG-3: Provisioning* - The registry must give static information about the UAS and its operator, as well as dynamic information about its current operation and the UAS Service Supplier (USS).
- *REG-4: AAA Policy* - Attestation, Authentication, Authorization, Access Control, Accounting, Attribution, Audit, or any subset thereof (AAA) must be specifiable by policies, accessible in the register and administration of such must be protected by AAA.

2.2 Blockchain

A blockchain is an example of a distributed ledger and consists of a sequence of transaction blocks, which together create a complete history of transactions in the system[5]. Each block store the hash of its "parent block" in its header and has one, and only one, parent block. The first block in the blockchain is called "genesis block" and has no parent block.

Each block consists of a block header and a body. The header stores information such as the hash of its parent block, the hash of every transaction in the block and a timestamp. The body includes a transaction counter and transactions, which are authenticated with an asymmetric cryptography mechanism, i.e., a key pair of a private and a public key.

The blockchain technology has some key characteristics that enforce security, these are the following:

- *Anonymity* - Users can interact anonymously with the blockchain using a different, generated, address, that does not reveal their real identity.
- *Auditability* - Every transaction has to some previous transaction, therefore each transaction is easy to verify and track.
- *Persistency* - Since all transactions are easy to verify, invalid transactions would not be admitted and it is very hard to delete already included transactions. Blocks containing invalid transactions could be discovered immediately.
- *Decentralisation* - Consensus algorithms are used to remove the need of third party validation of transactions and are responsible for data consistency.

2.3 Hyperledger Iroha

Hyperledger Iroha is an open-source Distributed Ledger Software framework that allows for a decentralized way of storing data and communicating between devices. A Distributed ledger is also known as a blockchain.

Iroha has a robust permission-based command system that allows permission-based access for joining networks, executing commands, and making queries. Iroha uses built-in commands instead of smart contracts to make it easier to do common tasks such as creating digital assets, registering accounts, and transferring assets between accounts.

Iroha uses transaction status codes to tell the user if a transaction was successful or not [4]. These are part of a stream of events for each transaction and can be seen in Figure 1. The transaction statuses used in this project are:

- *STATELESS_VALIDATION_SUCCESS*: The transaction passed the stateless validation step successfully.
- *STATELESS_VALIDATION_FAILED*: The transaction contained fields that violated some stateless validation constraints, hence failing.
- *STATEFUL_VALIDATION_SUCCESS*: The transaction successfully passed the stateful validation.

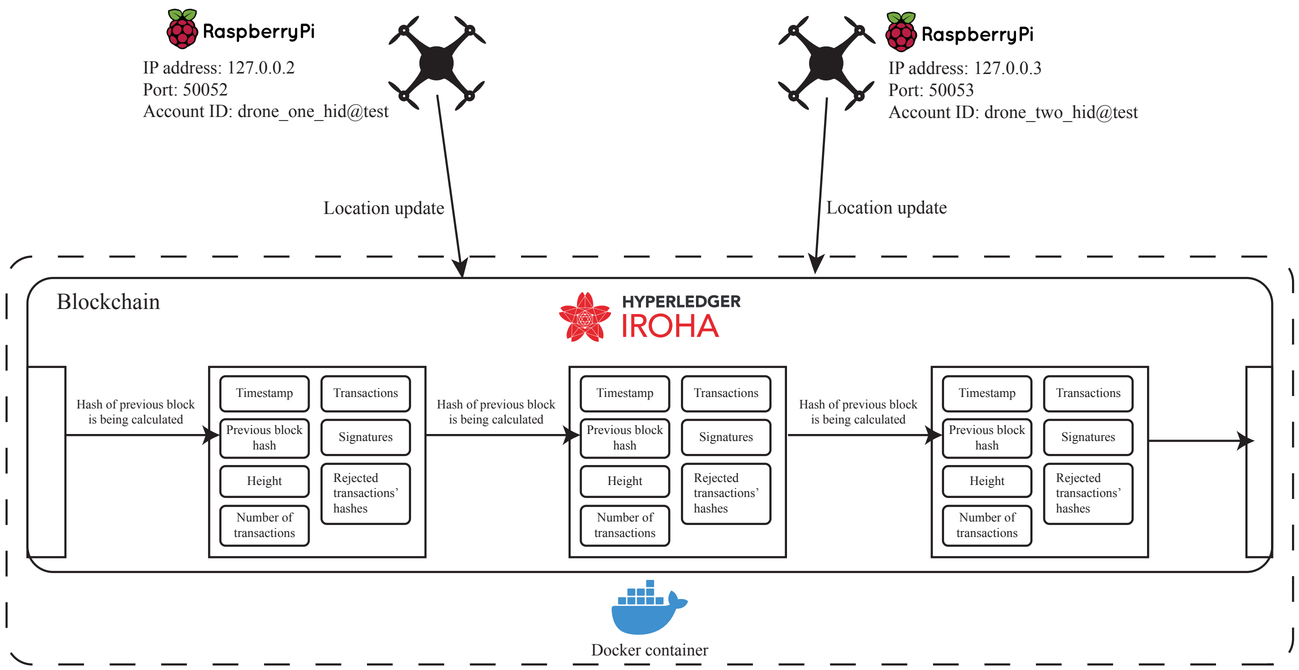


Figure 2: Network used in the test cases.

- *STATEFUL_VALIDATION_FAILED*: The transaction violates some rules checking the state of the chain, e.g., account permission.

Hyperledger Iroha fulfills the four requirements on a registry in the DRIP protocol, these requirements can be seen in subsection 2.1.

2.3.1 REG-1 & REG-2. Since the accounts in Hyperledger Iroha are created on domains, the same account name can be used in multiple domains. This leads to these accounts being treated as different, separate accounts, although they can use the same UAS ID. This leads to the possibility to create two accounts for each user, where one contains private information and one contains public information. Access to view the account details could then be restricted to certain accounts.

2.3.2 REG-3 & REG-4. The location updates are set using a key/value pair set in the account information of the corresponding UA. These key/value pairs cannot be removed or modified by another account than the one setting it in the first place. Furthermore, for every update of the value's information, the previous information is stored on the blockchain. This supports non-repudiation, traceability as well as trust in the UA:s own data. One drawback, however, is that Hyperledger Iroha does not store logs of who has access to what data, nor the purpose. To achieve this, the queries would have to work as transactions themselves and therefore be stored on the blockchain.

3 METHODOLOGY

The goal of this study was to analyze security risks with Hyperledger Iroha as a registry in the DRIP protocol. To conducts the analysis, a test network was set up using two Raspberry Pi:s and a host device, in this case, a laptop. A Hyperledger Iroha blockchain was set up on the host and all devices were connected to the same LAN using a mobile hotspot. A network schema can be seen in Figure 2.

The Raspberry PI:s worked as simulated UA:s and were given a set IP-address and port number in the LAN to make sure the transactions from the Hyperledger Iroha blockchain were sent to the right places. Drone #1 got IP 127.0.0.2 and port50052 and Drone #2 got IP address 127.0.0.3 and port 50053. The host device also got a static IP address and port, 127.0.0.1 and 50051 respectively.

Accounts were set up for both the drones and the host. The host got an admin account with more rights than the accounts connected to the drones. The admin account got the ID "admin@test", Drone #1 got "drone_one_hid@test" and drone #2 got "drone_two_hid@test".

A Docker container was set up to create a virtual environment where Hyperledger Iroha could be used safely without being affected by other programs on the system. In the container, a standard distribution of Iroha was installed, instead of building a customized version. Since no customization was done to the genesis block or the nodes, an environment file was created to overwrite the initial peers linked to the genesis block.

A separate Docker container with PostgreSQL was set up using Irohas default configuration provided in the documentation. PostgreSQL was then used as a database for Iroha.

A persistent Docker volume called Blockstore was also created in order to store the blockchain persistently between sessions on the host computer. Creating blockstore as a Docker volume enabled persistent storage between different sessions. Without a Docker volume, the stored data would be lost between sessions.

Two Raspberry Pi:s were installed with Raspberry Pi OS, a Debian-based Linux distribution designed for the Raspberry Pi. Python 3 and the Iroha Python library were then installed. The Pi:s were set up as Iroha peers in order to enable communication over the network between the host and Raspberry Pi:s, using Python scripts that created accounts for the peers and connected them to the Iroha blockchain.

Key pairs were created for each account, consisting of a private key and a public key, generated according to the SHA-3 cryptography algorithm. The algorithm used was included in the IrohaCrypto library, which first randomly creates a private key and then derives a public key from the already created private key.

The IP addresses of the Raspberry Pi:s and the host machine were saved in the environment file with the key pairs created earlier, as well as the different account names and domain ID. The exact values used can be seen in Figure 3. The environment file also allowed for adding initial peers to the Iroha network which was needed for communication between devices.

3.1 Sending transactions

Python scripts were written to send transactions between the peers and the host computer. These python scripts utilized Iroha's Python library to make transactions between the host and the Raspberry Pi. Several cases were tested and they are listed below, firstly as the account holder of Drone #1:

(1) Initialize location of Drone #1
(2) Change location of Drone #1

and then as the (malicious) account holder of Drone #2:

(1) Change location of Drone #1
(2) Remove Drone #1 from the network
(3) Add Drone #3 with stolen credentials from Drone #1.

The Iroha blockchain has several status codes that were printed to the terminal and analyzed to get assured about the success or failure of an operation. These status codes were the results of several stages and operations in Iroha. These stages were executed in a set order step by step and the results of those steps were output as the status codes.

4 RESULTS

Here, the result of each test case is presented.

4.1 Initialize location of Drone #1

The account holder of Drone #1 set the location of Drone #1 to some initial values in the format of "latitude:longitude:altitude", with the Iroha Python transaction command `SetAccountDetail`. Since the account holder himself set their own account details, no extra permission was needed and this returned the status code `STATEFUL_VALIDATION_SUCCESS`.

4.2 Change location of Drone #1

The account holder of Drone #1 updated the location of Drone #1 to the values "111:111:111" with the Iroha Python transaction command `SetAccountDetail`, which overwrites old values stored. As in the previous test, the account holder set their own account details, so no extra permission was needed and this returned the status code `STATEFUL_VALIDATION_SUCCESS`.

4.3 Change location of Drone #1 as malicious user

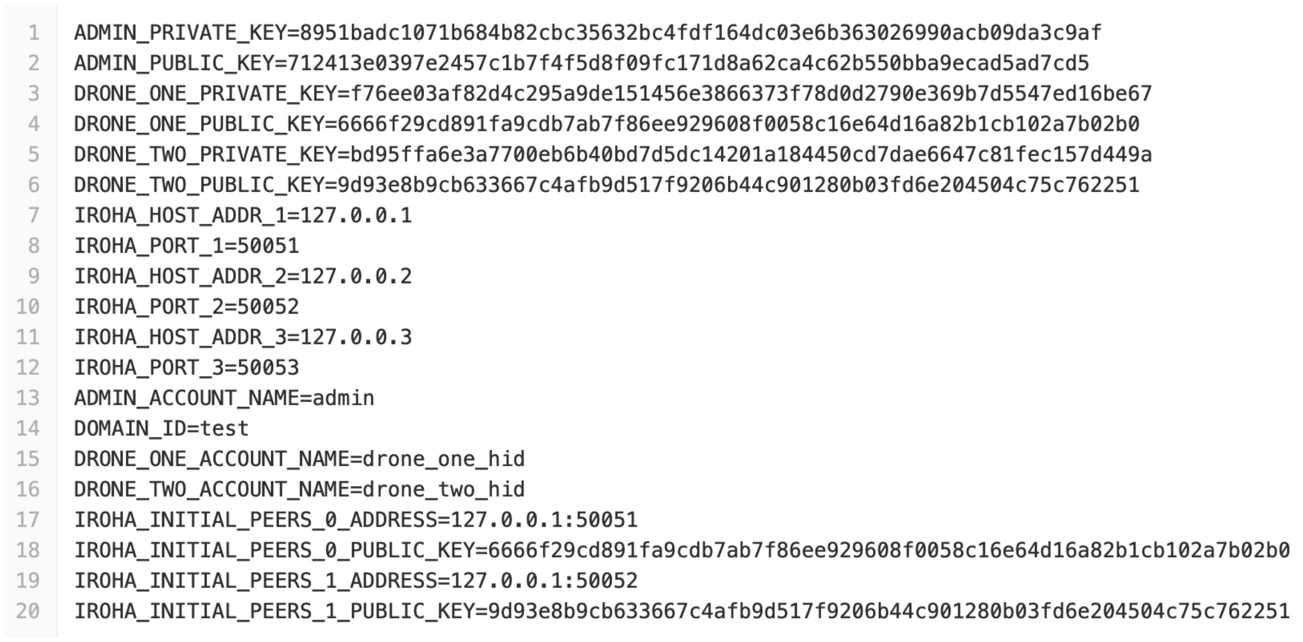
The account holder of Drone #2 updated the location of Drone #1 to the values "222:222:222", with the Iroha Python transaction command `SetAccountDetail`. The transaction was signed with the private key of Drone #2 and since no permission was granted to Drone #2 in beforehand, the transaction failed with status code `STATEFUL_VALIDATION_FAILED`.

4.4 Remove Drone #1 from the network as malicious user

The account holder of Drone #2 sent a transaction to remove the peer Drone #1 with the transaction command `RemovePeer` and the public key of Drone #1 as value. The transaction was signed with the private key of Drone #2 and the transaction failed with status code `STATEFUL_VALIDATION_FAILED`, with the additional information `No such permissions`.

4.5 Add Drone #3 with stolen credentials from Drone #1

The account holder of Drone #2 sent a query to get the location of Drone #1 with the query `GetAccountDetail`, to later be used in the creation of Drone #3. The transaction was successful and the location "111:111:111" was returned.



```
ADMIN_PRIVATE_KEY=8951badc1071b684b82cbc35632bc4fdf164dc03e6b363026990acb09da3c9af
ADMIN_PUBLIC_KEY=712413e0397e2457c1b7f4f5d8f09fc171d8a62ca4c62b550bba9ecad5ad7cd5
DRONE_ONE_PRIVATE_KEY=f76ee03af82d4c295a9de151456e3866373f78d0d2790e369b7d5547ed16be67
DRONE_ONE_PUBLIC_KEY=6666f29cd891fa9cdb7ab7f86ee929608f0058c16e64d16a82b1cb102a7b02b0
DRONE_TWO_PRIVATE_KEY=bd95ffa6e3a7700eb6b40bd7d5dc14201a184450cd7dae6647c81fec157d449a
DRONE_TWO_PUBLIC_KEY=9d93e8b9cb633667c4afb9d517f9206b44c901280b03fd6e204504c75c762251
IROHA_HOST_ADDR_1=127.0.0.1
IROHA_PORT_1=50051
IROHA_HOST_ADDR_2=127.0.0.2
IROHA_PORT_2=50052
IROHA_HOST_ADDR_3=127.0.0.3
IROHA_PORT_3=50053
ADMIN_ACCOUNT_NAME=admin
DOMAIN_ID=test
DRONE_ONE_ACCOUNT_NAME=drone_one_hid
DRONE_TWO_ACCOUNT_NAME=drone_two_hid
IROHA_INITIAL_PEERS_0_ADDRESS=127.0.0.1:50051
IROHA_INITIAL_PEERS_0_PUBLIC_KEY=6666f29cd891fa9cdb7ab7f86ee929608f0058c16e64d16a82b1cb102a7b02b0
IROHA_INITIAL_PEERS_1_ADDRESS=127.0.0.1:50052
IROHA_INITIAL_PEERS_1_PUBLIC_KEY=9d93e8b9cb633667c4afb9d517f9206b44c901280b03fd6e204504c75c762251
```

Figure 3: The environment file used to configure the Iroha blockchain.

The next step was to create a new peer with the public key from Drone #1, this was done with the transaction command `Peer`, with the public key and an IP address as fields. However, this failed with status code `No such permissions`.

5 DISCUSSION

Here, the results of each test case are discussed, as well as the project itself.

5.1 Initialize and update location of Drone #1

The account holder of Drone #1 has the right to set the location of their own drone, which is crucial for the location updates to work properly. In a real environment, the location data would come from GPS modules on the drone and then be forwarded to the blockchain via the user's account. The permission to update the location could also be given to another user, such as an admin, to do the initialization when an account is set up.

5.2 Change location of Drone #1 as malicious user

The Iroha transaction command `SetAccountDetail` needs to be signed with a valid private key, and the permissions of the user connected to the private key are checked before the request is sent, which is crucial to enforce the integrity of the system.

5.3 Remove Drone #1 from the network as malicious user and Add Drone #3 with stolen credentials from Drone #1

Logged in as the account holder of Drone #2, there was no way to remove the peer Drone #1 or add new drones to the network. This is a feature needed to make the system secure and ensured via the secure SHA-3 cryptography key exchange since the private key is needed to sign a transaction and is verified via the blockchain with the corresponding public key.

5.4 Methodology discussion

During the project, many problems occurred and this section will cover them and discuss the methodology in general. One major problem was to get the Iroha blockchain running, although Hyperledger Iroha has very good documentation, there were many small parts in the project setup that did not work.

The initial plan was to edit the source code of Hyperledger Iroha to fit the needs of the DRIP protocol, by changing the regex requirements for the peer ID to match a Drone HID needed by Iroha. This was not possible due to a lack of time and machine power needed to build the modified version of Hyperledger Iroha. During the build process, multiple packets were downloaded and unpacked, which took more than a day on the laptop used, hence the process was aborted. Instead, a precompiled version of Hyperledger Iroha was used, although this had some restrictions, such as the HID needing to be in the format "hid@domain", instead of just "hid". This precompiled version was a ready-made Docker image. This image had some limitations such as a predefined genesis block or the first initial block of the blockchain. This problem was solved by providing the Iroha node with some environment variables. The environment variables enabled the node to use variables from the environment variables instead of reading them from the genesis block that was not modifiable.

Another problem that occurred was the creation and storage of the key pairs. These were created on the host machine and both keys were stored in the environment file, which was pushed to a remote repository and accessible by all users in the network. This is of course not safe and something that needs to be done individually on each peer to ensure that the private key is kept private. However, since the network that was built as part of this project was a proof of concept and due to time constraints, this issue was left unsolved.

In general, the setup used in the project worked very well for the intended purpose. The intended purpose being to set up a proof of concept Iroha network and to be able to run some simple tests. Docker is highly recommended when setting up similar networks. An ARM64-based MacBook was used as the host computer for the project. The MacBook posed some compatibility issues with some of Irohas' dependencies due to it being an ARM64-based computer. However, running the environment in docker and using Apples built-in AMD64 to ARM64 conversion layer worked fine.

6 SECURITY ANALYSIS

Here, a high level security analysis of Iroha is conducted and discussed. A systematic examination of security risks associated with blockchain systems were conducted by Li et al. [6], and several attacks on blockchain systems are presented. Another systematic review were done by Deirmentzoglou et al. [7] as a literature review on long-range attacks on blockchains.

6.1 Potential Point of Failure Weakness in Iroha

The Iroha registry needs to be able to register new accounts and perform different operations and queries. Since Iroha is permission-based, there is always at least one administrator account that has all permissions. Those permissions could be the ability to create new accounts, change accounts, add new nodes, etc. Having only one single administrator account would introduce a single-point-of-failure to the whole system. A compromised admin account would compromise the rest of the system and enable adversaries to gain full control of the blockchain network.

This potential risk can be mitigated by using the concept of multi-signature transactions. Multi-signature transactions in this case would require several admin accounts to sign a transaction or query. Not only the admin account that created the transaction or query. For every operation in the blockchain network, a portion of the admin accounts would be required to sign the operation. By using this mitigation technique, an adversary would have to compromise multiple admin accounts in order to compromise the whole system which in turn greatly increases the difficulty of the attack and decreases the likelihood of its occurrence.

It is not recommended to require all admin accounts to sign every operation since that would enable an adversary to block all operations on a network by compromising only one admin account.

6.2 Man-in-the-middle Attacks

Iroha does not encrypt the actual contents of the data when it is being transmitted between actors in the network. This enables Man-in-the-middle attacks since the contents of the data are sent in plaintext and therefore can then be read. However, Hyperledger Iroha supports the use of Transport Layer Security (TLS), which can be used to mitigate man-in-the-middle attacks. TLS would encrypt and secure all communication taking place in the blockchain network, and encryption should be mandatory if there is a risk that the data being sent could give access to and compromise a node. This type of data could for example create node requests sent from an admin account. TLS is sufficient for blocking and mitigating most man-in-the-middle attacks.

6.3 Replay attacks

A replay attack is done by a malicious actor getting hold of data sent via a Man-in-the-middle attack. This could work out even if the malicious user e.g., gets hold of an encrypted password, they could replay it later when asked for authentication. This message would seem to be legit since it was an authentic message that was sent by a legitimate user and encrypted in the correct way. This sort of attack is mitigated in Iroha since all packets have a timestamp and all transaction hashes are stored in the block store, even if the transaction was rejected.

6.4 Logging

An important part of any critical system is to log actions taken by its users. Logs that are of interest are all actions taken by administrator accounts, as well as updates and changes to the peers. The logs need to be encrypted and stored in a secure place, with an authentication mechanism to prevent malicious actors from accessing and submitting logs.

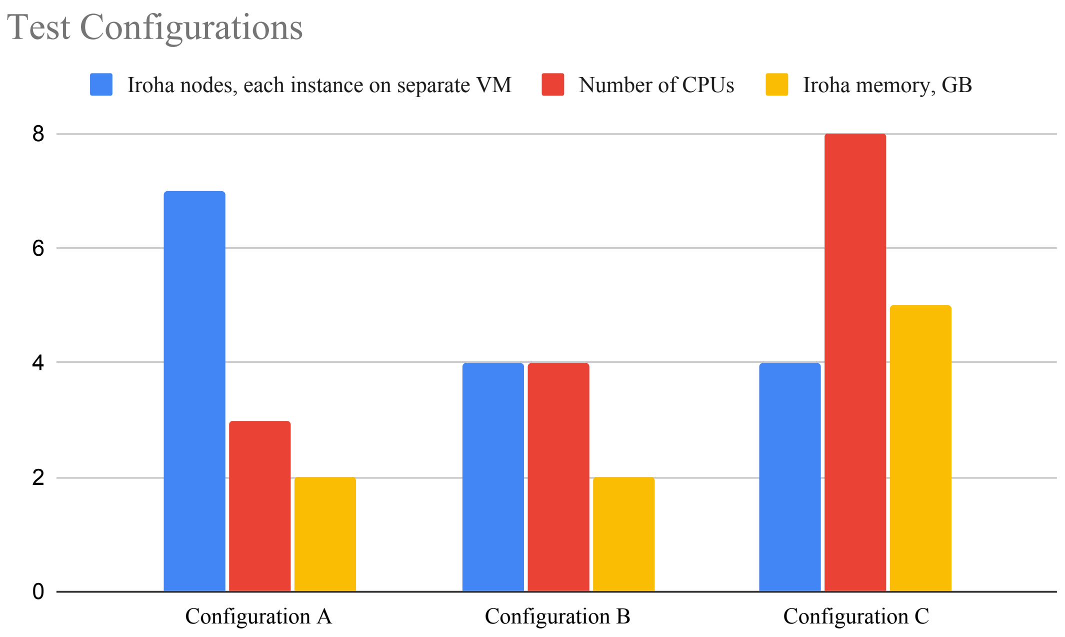


Figure 4: Configurations used in performance evaluation

6.5 Storage of private keys

Since an asymmetric cryptography algorithm is used, it is very critical to store the private key in a secure way to minimize the risk of it being compromised. Anyone having access to the private key could potentially create, sign and verify transactions in the name of the entity to which the private key belongs to. This is especially important for UA:s operating in public spaces where anyone could get a physical hold of the device and access the hardware. If a malicious actor gets hold of an admin private key, it would be catastrophic for the security of the whole network. The problem could be mitigated with hardware encryption on the UA where the private key is stored to make it harder for a malicious actor to access it.

7 PERFORMANCE EVALUATION

The Hyperledger Wiki provides a performance test comparing the performance between using Hyperledger Iroha with PostgreSQL and RocksDB. The performance test was made using Iroha 1.3.0 and featured a number of different configurations. All data points used in the performance test were gathered from the Wiki [8].

All tests were performed with similar Hyperledger Iroha configurations that were set in a configuration file, which can be seen in Figure 6. The maximum proposal size was 1000, this is the maximum amount of transactions that can be in one proposal. An increase of this parameter increases the performance up to a point where a degradation is present. The proposal delay was set to 1000. Vote delay was 500, this effects how much time is spent before sending a vote to the next peer, more nodes requires longer vote delay. Multi-signature transaction network transport was enabled (mst_enable was set to true). The time period before a not fully signed transaction is considered expired (mst_expiration_time) was set to the default 1440 minutes. The maximum delay between two consensus rounds (mst_expiration_time) was set to 500 milliseconds, lower idle period is more resource consuming, but requires less waiting time. Lastly, the maximum rounds to keep an open status stream while no status update is reported (stale_stream_max_rounds) was set to 5, an increase of this value leads to less reconnections to track a transaction that is updated with new rounds.

There are three different configurations in the performance evaluation, these can be seen in Figure 4. Configuration A has 7 Iroha nodes, 3 CPUs, 2GB RAM and 8GB SSD storage per node. Configuration B has 4 Iroha nodes, 4 CPUs, 2GB RAM and 17GB SSD storage per node. Configuration C has 4 Iroha nodes, 8 CPUs, 5GB RAM and 218GB SSD storage per node. The SSD storage of every test configuration is not present in Figure 4 since it would skew the graph and make it less legible, however, the amount of Iroha nodes, CPUs and Iroha memory for each configuration is shown.

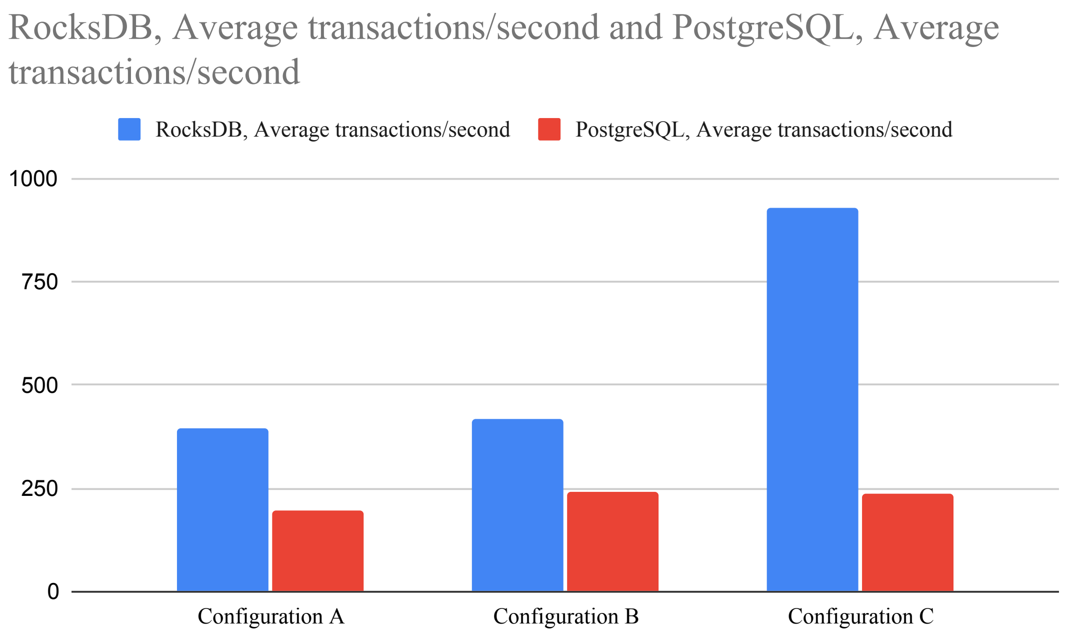


Figure 5: Performance evaluation comparing PostgreSQL and RocksDB

```
{
  "block_store_path" : "/tmp/block_store",
  "database": {
      ...
  },

  "max_proposal_size" : 1000,
  "proposal_delay" : 1000,
  "vote_delay" : 500,
  "mst_enable" : true,
  "mst_expiration_time": 1440,
  "proposal_creation_timeout": 500,
  "stale_stream_max_rounds": 5
}
```

Figure 6: Configuration file used in performance evaluation.

Figure 5 shows the average transactions per second for each of the three configurations, using PostgreSQL and RocksDB. Only already commited transactions were counted as transactions. The total duration of the tests were 2 hours for each configuration. The average value was taken in the middle of the test over a period of 30 minutes. Throughout the tests RocksDB performs better than PostgreSQL across all three configurations. There was a significant difference in performance between RocksDB and PostgreSQL in configuration C as seen in Figure 5.

8 CONCLUSION

Although the Hyperledger Iroha blockchain is both time and resource-consuming to configure, deploy and maintain, the security aspects of using a blockchain as a registry in DRIP are very relevant.

IoT devices are increasing in popularity and UA:s such as drones pose new challenges. One of those new challenges is the remote identification of drones, where reliable, fast and secure drone identification is vital. DRIP is a protocol that was built with secure drone remote identification in mind, but DRIP needs a secure way of storing and sending data between different actors on a drone network.

Using a permission based blockchain such as Hyperledger Iroha to store and send data enables many advantages but also some disadvantages. In this paper Hyperledger Iroha was used as a DRIP registry. One big advantage of blockchains is that previous blocks are very difficult to modify, this property makes storing certain data such as identification tags inherently more secure compared to traditional databases. Another advantage is that blockchains are decentralized, making it harder to compromise all the data stored by simply compromising a critical part of the system. Furthermore, making sure identifiers of drones are not modified by adversaries is critical to the security of the system. Iroha enables storage of drone identifiers on the blockchain, which is very advantageous.

Iroha's permission based structure is very useful for differentiating and allowing privileges between different actors on the network. This allows for good privilege separation and clear roles in the drone network.

Some challenges and disadvantages of using Iroha and blockchains in general is that they introduce higher computation requirements to the network. All decentralized ledgers or blockchains needs some sort of consensus algorithm in order to synchronize and agree upon data between all of the different nodes of the blockchain. When compared to a regular database that can simply be queried and updated, blockchains need more computational power to complete the same tasks.

Permission-based blockchains, such as Hyperledger Iroha, can be used as DRIP registries and provide many advantageous features. However there are still some disadvantages, mostly regarding performance, that needs to be considered for large drone networks.

The contributions made in this paper can be summarized by a thorough security analysis of Iroha when used as a DRIP registry, backed up by concrete tests. The findings can be used by future implementers of drone networks and can assist in creating a safer network. Different aspects of using Iroha specifically and also using blockchains with IoT devices in general is discussed. Those discussions can aid in the thought process behind choosing different technologies when building a security focused drone network.

REFERENCES

[1] S. W. Card, A. Wiethuechter, R. Moskowitz, and A. Gurtov, "Drone Remote Identification Protocol (DRIP) Requirements and Terminology," RFC 9153, Feb. 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9153

[2] Y. Hashem, E. Zildzic, and A. Gurtov, *Secure Drone Identification with Hyperledger Iroha.* New York, NY, USA: Association for Computing Machinery, 2021, p. 11–18, (Alicante, Spain). [Online]. Available: https://doi.org/10.1145/3479243.3487305

[3] A. Ossamah, "Blockchain as a solution to drone cybersecurity," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, 2020, pp. 1–9.

[4] L. Soramitsu Co. Hyperledger iroha documentation. (accessed: 2022-04-14). [Online]. Available: https://iroha.readthedocs.io/en/main/index.html

[5] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *2017 IEEE International Congress on Big Data (BigData Congress)*, 2017, pp. 557–564.

[6] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," *Future Generation Computer Systems*, vol. 107, pp. 841–853, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X17318332

[7] E. Deirmentzoglou, G. Papakyriakopoulos, and C. Patsakis, "A survey on long-range attacks for proof of stake protocols," *IEEE Access*, vol. 7, pp. 28 712–28 725, 2019.

[8] S. Lavrentev. (2021) Release 1.3.0. (accessed: 2022-05-05). [Online]. Available: https://wiki.hyperledger.org/display/iroha/Release+1.3.0