EPFL DEDIS

École Polytechnique Fédérale de Lausanne

Hacking Sovereign Identity

by Alexandre Ivan Délèze
CS-Cybersecurity MA2

Master Project Report

Approved by the Examining Committee:

Prof. Bryan Alexander Ford
Thesis Advisor

Jeffrey Richard Allen
Thesis Supervisor

EPFL IC IINFCOM DEDIS
BC 210 (Bâtiment BC)
Station 14
CH-1015 Lausanne

June 5, 2020

Acknowledgments

I want to give a special thanks to my advisor Jeffrey Allen for his help throughout this project. Auditing the security of a system like Sovrin can be hard work requiring much experience in distributed and decentralized systems. He helps me during our weekly meetings to understand where are the most valuable targets, not to lose myself in non-important details, and keep my motivation high for my project during this COVID-19 period.

I also want to thank Prof. Bryan Ford for his supervision and all the DEDIS team for their feedback during the two midterm meetings.

Lausanne, September 22, 2020 Alexandre Ivan Délèze

Abstract

Electronic identities become a trendy topic for our governments and companies. In this report, we analyze the security of the decentralized identity system of the Sovrin Foundation.

We first analyze their organization's policies. In the second phase, we evaluate their components and threat model. Then we select the main targets and use industry-standard auditing, fuzzing, and other hacking attacks to find vulnerabilities.

This report presents theoretical attacks against the setup of a client and the node's upgrade procedure. We also find vulnerabilities in the consensus algorithm, which can simplify amplification attacks and also permit a user with no privilege to write illegitimate information on the blockchain.

Contents

Chapter 1

Introduction

Numeric identities are a significant challenge for the digitization of our governments and companies. Some countries already have a numeric identity solution, like Denmark [1], or want to acquire one in the future, like Switzerland [2]. Sovrin [3] is a foundation that offers a self-sovereign identity based on a blockchain. The government of British Colombia in Canada has decided to use Sovrin technology to implement their eID solution [4].

The Decentralized and Distributed Systems Lab (DEDIS) at EPFL develops the ByzCoin distributed ledger [5]. DEDIS is currently investigating using Sovrin decentralised identities in Byzcoin. This work is motivated by DEDIS' need to integrate only well designed and correctly implemented identity systems into their system.

The main challenge in these eID technologies is the trust of the population. To reinforce it, we need to ensure that these systems are secured and that privacy is respected.

In the best practices to ensure a good level of security, we have the requirement that we have to expect that an attacker knows the source code and that the project is regularly audited. Sovrin's source code and all its dependencies are open-source, and so we can make in-depth security analysis. In this project, we audit Sovrin's security by analyzing their components and their threat models. Then we target some components with industry-standard auditing, fuzzing, and other hacking techniques.

In this report, we present our findings on some vulnerabilities in their organization's policies and their technical implementation.

Chapter 2

Background

2.1 Sovrin

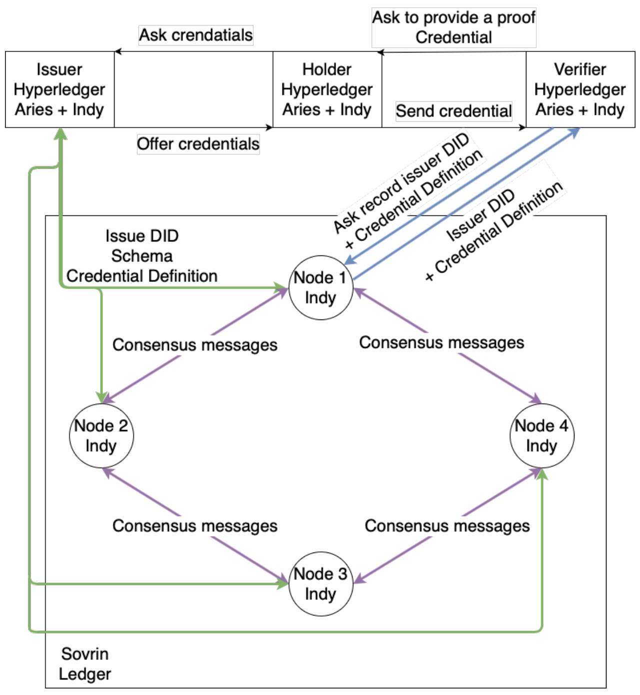
The sovereign identity system we will focus on during this project is Sovrin [3]. Three Hyperledger projects from the Linux Foundation: Aries [6], Indy [7], and Ursa [8] compose Sovrin. Sovrin is a public-permissioned blockchain.

Ursa is the cryptography library used by all other components for their cryptographic operations [9]. Indy implements interactions between a client and the blockchain, and it also implements all the behaviors of the ledger. Indy SDK [10], Indy Node [11], and Indy Plenum [12] compose Indy. A diagram of the custom RBFT implementation of Indy Plenum is in Appendix A. Indy SDK is the toolbox for interactions between a client and ledger's nodes. Indy Node is the library that implements the business logic of the blockchain and uses Indy Plenum for its custom PBFT consensus algorithm implementation. Aries [13] manages peer-to-peer interactions between agents based on decentralized identities (DID) and verifiable credentials.

Sovrin has three different ledgers: BuilderNet, StagingNet, and MainNet. MainNet is the ledger in production. Without contraindication, every time we will speak about Sovrin ledger, we will speak about Sovrin MainNet. We have four different levels of permission: Trustee (all rights), Steward (can manage a Validator Node and sign transactions), Endorser (can sign transactions), and no role with no write permission.

2.2 Practical Illustration

To describe the interactions between all these projects which build Sovrin, we will go through a practical example illustrated in Figure 2.1. Harry (Holder) wants to receive a numeric passport from his government (Issuer). He requests a new credential to the government using Aries. He



Ask crendatials
Ask to provide a proof Credential
Issuer Hyperledger Aries + Indy
Holder Hyperledger Aries + Indy
Verifier Hyperledger Aries + Indy
Send credential
Offer credentials
Ask record issuer DID + Credential Definition
Issuer DID + Credential Definition
Issue DID Schema Credential Definition
Node 1 Indy
Consensus messages
Consensus messages
Node 2 Indy
Node 4 Indy
Consensus messages
Consensus messages
Node 3 Indy
Sovrin Ledger

Figure 2.1: Sovrin Model

can know the DID of the issuer because this one already registered his identity on the blockchain.

To write his public DID on the ledger, the government asks an Endorser to sign his DID record, composed by an identifier, a public key, and by other extra information if needed. Then the Endorser sends the DID to all nodes composing the ledger. The nodes verify if the signature is correct and if the Endorser has the right to write on the ledger. If all conditions are fulfilled, nodes send an acknowledge message to the Endorser and write the DID record of the government on the blockchain. The issuer can use the same process to save a credential definition, which he will use to issue credentials to his clients.

One time that the government has a public DID and a credential definition for a numeric passport, it can create a credential for Harry using the credential definition registered on Sovrin Network. Then it sends the numeric passport to the holder, Harry.

When Harry wants to buy a flight ticket, the airline company (Verifier) asks him to provide proof of his identity. Harry sends his numeric passport or a part of it, so the company can control if this credential is legit. The verifier asks a node of the Sovrin Network to send the DID record of the government and the credential definition of a numeric passport issued by this government. The verifier checks if the credential's signature is correct and if all fields are identical to the credential definition. If all conditions are fulfilled, the company accepts Harry's numeric passport.

Chapter 3

Coverage

3.1 Sovrin's policies

We first want to analyze the internal Sovrin Foundation organization and to determine the responsibilities inside the Foundation. We also desire to see what are the risks inherited from their business architecture. Then, we will see how trust is distributed. To achieve this goal, we will look at the authorization rules and determine which are the most critical DID on the ledger and who is permissioned to write on Sovrin MainNet.

We will also go through the upgrade process and what are the policies of Sovrin to manage this task. Finally, in this part, we will speak about the setup of a new Sovrin client. We will see how a client can know how to connect to the MainNet and retrieve the address of all nodes involved in this blockchain.

3.2 Indy Ledger

We think that the most major surface of attack is the consensus algorithm because it implements a custom algorithm of PBFT represented in Appendix A. We will try to slow down the consensus process. We will also try to write illegitimate transactions on the ledger. To reach these goals, we want to launch the attack from the client side with and without a corrupted node controlled by the attacker.

One risk can also be how the nodes at their interface handle the messages. We will try to fuzz the methods which handle messages coming from the outside world.

3.3 Out of Scope

In this project, we will not work on the Aries project, because we have decided to focus on the blockchain and the communication with this one. We will also not focus on the revocation registry of credentials, which is a zero-knowledge proof. The reason is that revocation registries can use different cryptographic accumulators [14]. To have a general point of view on revocation, we have to study every accumulator separately, and it can cause a large amount of work. Because we have decided not to focus our research on Aries, which is the sharing of credentials, we have decided that the time needed to analyze the revocation of credentials is not worth it.

The last point we will not cover is the Decentralized Key Management System (DKMS) because this technology is for the moment not implemented, and only the wished architecture is published on GitHub [15].

Chapter 4

Attacks and Results

4.1 Sovrin's policies

We analyzed first the node's upgrade process. We saw in the Steward Technical Policies [16] that a Steward has to upgrade its node within three business days. Furthermore, to order an upgrade, the Board of Trustee (BoT), which is the overall governing body of the Sovrin Foundation, writes a POOL_UPGRADE on the sub-ledger Config and three Trustees have to sign this transaction. It contains the version number of the upgrade, an SHA-256 integrity check of the package, and a scheduling for each node to balance the connection to the upgrade server [17, 18]. Each node writes on the sub-ledger Config their state in the upgrade process (start, in_progress, complete) in a NODE_UPGRADE transaction. A risk is that the attacker may have an attack window of up to five days (if update on Friday) if a critical vulnerability exists in the nodes.

Another aspect we had to discuss was the composition of the Board of Trustee (BoT), composed by the Trustees. On their website [19], they say that BoT members are selected across multiple industries and geographies. We observed that height Trustees over the ten composing BoT live in the US. We noticed that the Sovrin Foundation is registered in Utah, US. We have seen in the FBI-Apple encryption dispute [20] that the US Supreme Court can use the All Writs Act of 1789 to force US citizens or companies to provide information and help them to solve cases [21]. Because the majority of the Trustees and the Foundation are under US regulation, it can create a risk that they will have to install back doors on the ledger to comply with a writ of the Supreme Court. Another issue is that Sovrin may not be compliant with EU GDPR Section 3 [22] because rectification and erasure of personal data on a blockchain are not possible. Sovrin mitigates this risk by not allowing to write personal data on the ledger. However, we can see in the sub-ledger Pool that node's transactions contain the name of the node, its IP address, its country, its geographic location, and its time zone [23].

We looked then at the policy of permission to write on the ledger and to run a Validator Node.

To obtain information about the authorization rules, we had to connect to MainNet with an Indy client and type the command *ledger get-auth-rule*. We observed that we need the signature of three Trustees to acknowledge a new Trustee or Steward. We also need the signature of one Trustee to add a new Endorser. We need a signature of a Trustee, a Steward, or an Endorser to write a new DID on the ledger (see the extract of auth-rule of MainNet in Appendix B).

Lastly, we wanted to analyze how an Indy client can retrieve information about nodes to be able to interact with the ledger. To connect to the Pool, we have to furnish to the Indy client a genesis block. We can find the one for Sovrin MainNet on GitHub [24]. The client will then retrieve information from the node listed in the genesis block. There are two possible issues with this implementation. The first is that the Sovrin Foundation does not sign the genesis block, and there is no integrity check. A malicious person can publish a fake genesis block online and can use a social engineering attack to redirect his victims to this fake genesis block. To avoid this issue, we can sign the genesis block with the private key of the Sovrin Foundation and hard code the certificate of the Sovrin Foundation in the Indy client. We can also add an SHA-256 hash of the package to check its integrity.

The second issue is that this genesis block contains the ten first nodes of the ledger. These nodes must be up and running to retrieve all pool transactions to find all of the ledger's nodes. It means that these nodes cannot change their IP address, because the client can then not find them, or be revoked in the case that a node becomes malicious. We can solve this problem by writing a DNSSec record signed by the Sovrin Foundation containing the IP address of all nodes permissioned to run the ledger. The client can then contact the addresses of the DNS record to retrieve all information about the nodes.

4.2 Technical Attacks

4.2.1 Test Environment

For all our experiments, we set up an environment with a host on Ubuntu 18.04 using Docker version 19.03.9. To install our test environment, we use the Dockerfile given in Hyperledeger Indy SDK [25]. This Dockerfile creates an image with four nodes running indy-plenum version 1.12.1 and indy-node version 1.12.1 in one Docker container. In this environment, we have n = 4 and f = 1.

4.2.2 Quorums

Our first question was, what is the quorum in the RBFT to go to the next step in the 3 phase commit. The messages listed in Table 4.1 are the messages described in the RBFT consensus protocol (see Appendix A for the diagram). We can find this quorum specification in plenum.server.Quorums [26].

We can see in plenum.server.Node [27] in the method setPoolParams that we have to call this method each time we add a node to the pool to update the quorum. We can have a risk at the moment of adding a new node that we can write a transaction with f malicious node because the quorum is not up-to-date in some nodes.

	Quorum
Propagate	f + 1
Prepare	n -f - 1
Commit	n - f
View Change	n - f
BLS signatures	n - f

Table 4.1: Quorums

4.2.3 Nodes switched down

We switched down one node, and we tried to pass a legitimate transaction to the ledger. We observed that the ledger had recorded the transaction correctly. Sometimes it took more time to register if the node we had switched down was the Master. In this case, the other nodes had to do a view change and continue the consensus process.

In the next experiment, we switched down two nodes and wrote a legit transaction on the ledger. We observed that we had a timeout, and nothing was written on the ledger. It was normal behavior because we need n-f COMMIT messages to write on the ledger (see Table 4.1). In our experiment, only two nodes sent this message instead of the three needed, and the ledger could not reach the consensus and never answer to the client.

4.2.4 Control Nodes

We wanted to write a new DID on the ledger without having the signature of an Endorser. For that, we controlled one node, which did not check the signature of the write request. This technique did not work because we need, according to Table 4.1, f+1 PROPAGATE messages to go to the next step, and here only the malicious node sent a PROPAGATE message.

In a second tentative, we modified two nodes out of four. Here our strategy worked because f+1 nodes validated the transaction's signature, and the signature is not checked at a later stage.

Then we made another trial with one node which did not check the signature and one crashed node. What we wanted to see was how the system handles a missing node in the consensus process. We observed that the nodes refused the write request because f did not change by the fact that one node was switched down. An Indy client considers a transaction as refused when it receives f+1 NACKs from the nodes [28].

The last experiment we did in this section was to create a loop in the Master node, which sent the PROPAGATE message indefinitely to the other nodes when it received a write request. This attack implied that we could not write anything on the blockchain for two minutes. The nodes could not go through the 3 phase commit if the Master did not send a PRE-PREPARE message like in Figure A.2. After this period, the other nodes asked a view change and voted for a new Master. It was possible then to write new transactions.

4.2.5 Fuzzing

The first fuzzing experiment we did was against two methods at the interface between plenum.server.Node [27] and the queue of messages which took input from ujson parser. It was *handleOneClientNodeMessage* and *handleOneNodeMessage*. We fuzzed these two functions with python-afl [29] for two hours. We noticed after this experiment that both methods catch all exceptions of type Exception. It means that plenum.server.Node caught all runtime errors, and so our fuzzer would never crash.

We decided after this trial to fuzz the function *load* from the library ujson [30]. This method is used to transform the payload of all exchanged messages to a JSON object and passed that objects to *handleOneClientNodeMessage* and *handleOneNodeMessage*. We also ran python-afl [29] for two hours against the *load* method. Because in state of the art we execute fuzzing during days or weeks, we found no bug within two hours [31]. Because testing ujson was not in the scope of our project, we have decided not to continue to search vulnerabilities in ujson.

4.2.6 Replay Attacks

We made two different replay attacks. In the first one, we built a write request with the signature of an Endorser. We sent it to the ledger to be registered. Then we sent this transaction again. We observed that this second order was acknowledged but with the sequence number of the previous write request. We concluded that our attack failed because we did not see a double write on the ledger.

Our second replay attack was with a read request. This time we checked if a malicious node can send an old record to the client. We observed that the client sent a read request to f+1, and so our attack failed. We can see in the read request specifications [32] that the client asks one node a DID record. If the reply contains a correct BLS signature and a fresh-enough timestamp, then the client accepts this DID record. If it is an old timestamp, the client sends a read request to f+1 nodes. In conclusion, a malicious node can only reply with a previous version of a DID if this one was written on the ledger recently.

4.2.7 Amplification Attack

We sent a transaction signed by a DID with no writing permission. We observed that nodes marked the message as incorrect after the PROPAGATE, but they still put this transaction in the batch for consensus for ordering purposes. The nodes sent NACKs for our transaction at the end of the 3 phase commit.

We can see some drawbacks in this role check process. In the implementation of plenum.server.Node, a client can be blacklisted if he sends fake write requests. Nevertheless, this mechanism works only if the request is malformed or if the signature is incorrect. The nodes checked these conditions before sending a PROPAGATE message (see Figure A.1). However, because the role is checked only after reaching the quorum of PROPAGATE, the client will not be blacklisted if he sends too many orders with wrong roles.

It allows us to imagine a kind of amplification attack. For this attack, we needed to have a DID record on the ledger with no particular role. It was a requirement because we needed that our public key was on the blockchain to check the signature field of our write requests. Then we sent a write transaction signed with our private key. Because the transaction was correctly signed, the ledger entered in the 3 phase commit. Each node sent to the other three messages (PROPAGATE, PREPARE, and COMMIT), and the Master node also sent a PRE-PREPARE to the other. We can see in Table 4.2 how many messages each node received for three kinds of inputs coming from the client. If the client sends an empty message, an incomplete one, or a message with a wrong signature, then each node will receive only this unique message for this transaction. However, if the client sends a write request signed by a regular DID, then each node will receive that message and ten other coming from the other nodes.

We can derive a formula to find how many messages each node receives depending on the number N of nodes involved in the consensus process. We find $\frac{\#messages\ received}{Node} = (N-1)*3+1+1$ because we have N-1 PROPAGATE/PREPARE/COMMIT, one PRE-PREPARE, and one message coming from the client (see Appendix A). We searched on Uniresolver [33] the DID *did:sov:bPTNiLzWPFHKr7mJGaump* to found out that 14 Stewards signed a single DID record on Sovrin MainNet. If we apply now the formula, we find that for a single message sent by the client, each node receives traffic of 41 messages. The definition of the amplification factor we found in a meta-analysis [34] was $Amplification\ factor = \frac{Size\ of\ response}{Size\ of\ request}$. Because each message in our experiment has approximately the same size, we can approximate our amplification factor as 41, which is comparable with the one of a DNS amplification attack [35].

Lastly, we measured the delay between the moment a client sends a write request, and when he receives an ACK from the ledger. Because we ran this experiment on our computer, we can consider the propagation and transmission delay as negligible. We ran ten times each experiment separately to take into account the OS queuing delay. With these precautions, we can estimate that the node's processing time is proportional to the answering delay. The time amplification factor was 120 between an incomplete message and a message with a wrong role (see Table 4.2).

	Messages received by each node	Time to receive answer (ms)
Incomplete payload	1	19.4
Wrong signature	1	42
Wrong role	11	2340

Table 4.2: Number of messages received by each ledger's node and time to receive an answer depending on the type of malicious request.

4.2.8 Update DID with Non-Privileged Signature

We have seen in the previous section that nodes checked the roles only at the end of the 3 phase commit. In our last experiment, we wanted to go further with this property. We wanted to see how the nodes check the role and how we can exploit this check.

In Sovrin, the Owner or the Endorser of a DID can update this DID [36], and an unprivileged user shall not be able to update the DID of another user [37]. We tried to violate this rule.

Firstly, we registered a regular DID on the ledger. Then we looked in the genesis block to find the DID of a TRUSTEE. According to the specs, the regular DID shall not be able to update the DID of the TRUSTEE. We sent a write request for the TRUSTEE signed by the regular DID, which is the standard operation to do an update according to Sovrin Specs [36]. We observed that our write request was accepted. When we made a read request, we saw that the signing DID of the TRUSTEE was the regular DID and that the sequence number and the timestamp were changed.

The risk of this attack is that an unprivileged user can write on the ledger. The consequence is that an attacker can write a large amount of illegitimate data on the ledger. The second consequence is that the traceability of who endorsed who can be more difficult, and we can not trust the last DID record to retrieve that information. If we want to find out who gives permissions to a particular DID, we have to look at all occurrences on the ledger to find the oldest one.

This bug was responsible disclosed to the Hyperledger Indy Security team. At the moment I write these lines, the security team has not yet fixed the bug, but they have created a draft security advisory on GitHub with CVE identifier *CVE-2020-11093*. They also have created unit tests corresponding to this bug.

A possible fix can be the usage of Sovrin Tokens. StagingNet and BuilderNet implement Sovrin Tokens. In the future, we will be supposed to spend a token to write on the ledger, which makes the above attack costly. However, it is not, for the moment, mandatory to spend a token to write on the ledger. We do not know if the token will also be spent if a write request fails. If it is the case, it will also mitigate the amplification attack. The Indy security team's fix proposition is to reject a DID update if the submitting DID is not the same as the DID being updated and if neither the role neither the verkey are changed.

Chapter 5

Related Work

The company Nettitude published in November 2018 a penetration test report about Hyperledger Indy [38]. They analyzed in particular Indy Node, Indy Plenum, and Indy SDK. Nettitute did a white box testing and found two medium severity bugs and six low severity bugs. They revealed vulnerabilities with the cryptographic libraries, deserialization, and access to the filesystem.

Our approach differs with the one of Nettitude because we focused our work on finding bugs in an end-to-end scenario and not to do a code review component by component.

Concerning the fuzzing, a member of Sovrin's technical governance board tried to fuzz the Indy client using libfuzzer for Rust [39]. In the Indy Contributors Call of September 23rd, 2019, he said that he was worried about the unsafe code of libindy, which is one of the main components of the Indy client. He said that more fuzzing was necessary because the code looked bad [40].

In our approach, we wanted to fuzz the interfaces of the Indy Plenum nodes. Nevertheless, to reach this goal, we need fuzzing tools for safe languages. High-level languages guarantee the absence of low-level flaws, like buffer overflows [31]. However, these languages have some similar vulnerabilities as the low-level languages such as crashing inputs, null pointer exception, excessive resource utilization. Fuzzing high-level languages like Python is a new field of research [41] and will take more importance in the future.

Chapter 6

Future Work

In the Indy Contributor Call of October 10th, 2019, the Indy contributors discussed the opportunity to migrate the consensus algorithm of Indy Node from their custom RBFT to Aardvark [42]. The reason is that RBFT is slower than Aardvark. When we get the new consensus algorithm in Indy Plenum, it can be interesting to test it and audit it to find possible new vulnerabilities.

Another new feature to follow is the Sovrin Token [43]. As explained in subsection 4.2.8, we have to see if the contributors of Sovrin will use them as security against a specific form of DOS by asking a token for each transaction, whatever it fails or succeeds.

We did not focus on the Schemas and Credential Definitions. These two concepts are essential in a decentralized identity system because they allowed organizations to create verifiable claims. It can be part of future work to check how these Schemas and Credential Definitions are created and sent to the ledger and also see how a verifier can use a Credential Definition to check the validity of a claim. It is also crucial to analyze how a credential's holder can create a proof and if this proof does not leak information about the original credential [44].

As said in subsection 4.2.5, it can be interesting to continue to fuzz the interfaces of the nodes. To fuzz efficiently, we have to adapt the code of plenum.server.Node by removing the catch of exceptions of type Exception. Furthermore, we have to adapt the fuzzers to allowed them to pass fuzzed tuples to the methods *handleOneClientMessage* and *handleOneNodeMessage*. It can also be worth to fuzz the methods which write the transaction on the ledger to see if we can corrupt the information stored on the blockchain, like the method *updateSeqNoMap* in plenum.server.Node.

Chapter 7

Conclusion

We have seen some weaknesses in the organization model of Sovrin. We have found some risks of independence in the Foundation's governance. It is due to the concentration of the majority of the Trustees and the Foundation in one country. Another point was the upgrade policy allowing a node to take three working days. If the Board of Trustees issues an upgrade transaction a Friday, then the upgrade process must be done on Tuesday. It means that an attacker can have a window of five days to exploit a vulnerability. The last problematic point is how a client can retrieve information about the ledger's nodes. Therefore, Sovrin must guarantee the authenticity and integrity of the genesis block.

On the technical side, we have seen some vulnerabilities in their implementation. The first one is that each node has to update the quorum of the RBFT manually each time a node is added or removed from the ledger. The most severe vulnerability is the check process of the authorization/role of the signing DID. It can imply two types of attacks. In the first one, we can have an amplification attack comparable to a DNS amplification attack. The second one allows a regular DID to have limited write permission. We can change the parent DID of every transaction with a regular DID and so write on the ledger.

Nevertheless, Sovrin has the advantage of being a fully open-source project as well as its dependencies. Another positive aspect is their good reactivity on the Sovrin chat to give explanations and to debate their architecture's choices. The last point is that their responsible disclosure procedure is efficient. We received a confirmation within one day, and a member of the security team reviewed the disclosure in three days. Then a GitHub security advisory was created, and the security team gave us the right to observe and comment on their fix process.

As a result of the different aspects of the work I described above, the overall analysis is that Sovrin is not mature enough to be used in sensitive applications like an eID infrastructure. Key features like the DKMS are not yet implemented, and the consensus algorithm is not standard and thoroughly audited. Aries needs to be reviewed before issuing official credentials for critical operations like e-banking, voting, or patient's consent for DNA research.

Bibliography

[1] Nem ID. *Introduction to NemID.* URL: https://www.nemid.nu/dk-en/about_nemid/introduktion_til_nemid/index.html (visited on May 31, 2020).

[2] egovernment. *Implementing eID.* URL: https://www.egovernment.ch/en/umsetzung/schwerpunktplan/elektronische-identitat/ (visited on May 31, 2020).

[3] Sovrin Foundation. *Sovrin.* URL: https://sovrin.org (visited on May 22, 2020).

[4] Sovrin Foundation. *Use case spotlight: The Government of British Columbia uses the Sovrin Network to take strides towards a fully digital economy.* URL: https://sovrin.org/use-case-spotlight-the-government-of-british-columbia-uses-the-sovrin-network-to-take-strides-towards-a-fully-digital-economy/ (visited on May 31, 2020).

[5] DEDIS. *ByzCoin.* URL: https://github.com/dedis/cothority/blob/master/byzcoin/README.md (visited on June 5, 2020).

[6] Linux Foundation. *Hyperledger Aries.* URL: https://www.hyperledger.org/use/aries (visited on May 22, 2020).

[7] Linux Foundation. *Hyperledger Indy.* URL: https://www.hyperledger.org/use/hyperledger-indy (visited on May 22, 2020).

[8] Linux Foundation. *Hyperledger Ursa.* URL: https://www.hyperledger.org/use/ursa (visited on May 22, 2020).

[9] Hyperledger. *Ursa.* URL: https://github.com/hyperledger/ursa (visited on May 22, 2020).

[10] Hyperledger. *Indy SDK.* URL: https://github.com/hyperledger/indy-sdk (visited on May 22, 2020).

[11] Hyperledger. *Indy Node.* URL: https://github.com/hyperledger/indy-node (visited on May 22, 2020).

[12] Hyperledger. *Indy Plenum.* URL: https://github.com/hyperledger/indy-plenum (visited on May 22, 2020).

[13] Hyperledger. *Aries.* URL: https://github.com/hyperledger/aries (visited on May 22, 2020).

[14] Hyperledger. *Indy Project Enhancements Documentation.* URL: https://readthedocs.org/projects/indy-hipe/downloads/pdf/latest/ (visited on May 22, 2020).

[15] Hyperledger. *DKMS (Decentralized Key Management System) Design and Architecture V3.* URL: https://github.com/hyperledger/indy-sdk/blob/5f9144feab5b9e969389e325da47d7218f87081e/docs/design/005-dkms/DKMS%5C%20Design%5C%20and%5C%20Architecture%5C%20V3.md (visited on May 22, 2020).

[16] Sovrin Foundation. *Sovrin Steward Technical Policies.* URL: https://sovrin.org/wp-content/uploads/2019/03/Sovrin-Steward-Technical-Policies-V1.pdf (visited on May 25, 2020).

[17] Hyperledger. *Requests.* URL: https://github.com/hyperledger/indy-node/blob/master/docs/source/requests.md (visited on May 25, 2020).

[18] Indyscan. *POOL UPGRADE TX 9315.* URL: https://indyscan.io/tx/SOVRIN_MAINNET/config/9315 (visited on May 25, 2020).

[19] Sovrin Foundation. *People.* URL: https://sovrin.org/team/ (visited on May 25, 2020).

[20] Amy Davidson Sorkin. "The Dangerous All Writs Act Precedent in the Apple Encryption Case". In: *The New Yorker* (2016). URL: https://www.newyorker.com/news/amy-davidson/a-dangerous-all-writ-precedent-in-the-apple-case.

[21] govinfo. *28 U.S.C. 1651 - Writs.* URL: https://www.govinfo.gov/content/pkg/USCODE-2011-title28/pdf/USCODE-2011-title28-partV-chap111-sec1651.pdf (visited on May 25, 2020).

[22] European Union. *REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation).* URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679&from=EN (visited on May 25, 2020).

[23] Indyscan. *NODE TX 88.* URL: https://indyscan.io/tx/SOVRIN_MAINNET/pool/88 (visited on May 25, 2020).

[24] Sovrin Foundation. *Pool Transactions Live Genesis.* URL: https://github.com/sovrin-foundation/sovrin/blob/master/sovrin/pool_transactions_live_genesis (visited on May 25, 2020).

[25] Hyperledger. *indy-pool.dockerfile.* URL: https://github.com/hyperledger/indy-sdk/blob/9a382f418b65b8ad7476f57f64d362f65686887f/ci/indy-pool.dockerfile (visited on May 25, 2020).

[26] Hyperledger. *quorum.py.* URL: https://github.com/hyperledger/indy-plenum/blob/master/plenum/server/quorums.py (visited on May 26, 2020).

[27] Hyperledger. *node.py.* URL: https://github.com/hyperledger/indy-plenum/blob/master/plenum/server/node.py (visited on May 26, 2020).

[28] Hyperledger. *Write Requests.* URL: https://github.com/hyperledger/indy-plenum/blob/master/docs/source/diagrams/write-requests.png (visited on May 26, 2020).

[29] Jwilk. *python-afl.* URL: https://github.com/jwilk/python-afl (visited on May 26, 2020).

[30] ultrajson. *ultrajson*. URL: https://github.com/ultrajson/ultrajson (visited on May 26, 2020).

[31] Mathias Payer. *Software Security: Principles, Policies, and Protection*. Apr. 2019, pp. 31, 32, 67. URL: http://nebelwelt.net/SS3P/softsec.pdf.

[32] Hyperledger. *Read Request*. URL: https://github.com/hyperledger/indy-plenum/blob/master/docs/source/diagrams/read-requests.png (visited on May 26, 2020).

[33] Universal Resolver. *Universal Resolver*. URL: https://uniresolver.io (visited on May 26, 2020).

[34] Ryba et al. "Amplification and DRDoS Attack Defense – A Survey and New Perspectives". In: *arxiv* (May 2016). URL: https://arxiv.org/pdf/1505.07892.pdf.

[35] Anagnostopoulos et al. "DNS Amplification Attack Revisited". In: *Computer & Security* 39 (Dec. 2013). URL: https://www.researchgate.net/profile/Marios_Anagnostopoulos/publication/258403326_DNS_Amplification_Attack_Revisited/links/59de6c8f458515376b29df28/DNS-Amplification-Attack-Revisited.pdf.

[36] Sovrin Foundation. *Sovrin DID Method Specification*. URL: https://sovrin-foundation.github.io/sovrin/spec/did-method-spec-template.html (visited on May 26, 2020).

[37] Hyperledger. *Default AUTH_MAP Rules*. URL: https://github.com/hyperledger/indy-node/blob/master/docs/source/auth_rules.md (visited on May 26, 2020).

[38] Nettitude. *Penetration Testing Technical Report*. 2018. URL: https://wiki.hyperledger.org/download/attachments/13862116/TECHNICAL_REPORT_Hyperledger_Indy_Linux_Foundation_2018-10-31_v1.0.pdf?version=1&modificationDate=1560353100000&api=v2 (visited on May 24, 2020).

[39] Axel Nennker. *Fuzzing Libindy*. URL: https://github.com/AxelNennker/indy-sdk/tree/fuzzing/libindy/fuzz (visited on May 24, 2020).

[40] Hyperledger. *2019-09-23 Indy Contributors Call*. URL: https://wiki.hyperledger.org/display/indy/2019-09-23+Indy+Contributors+Call (visited on May 28, 2020).

[41] Daniel Kroening. "Fuzz Testing Java and Other Managed Languages". In: *Medium* (Feb. 2019). URL: https://medium.com/javarevisited/fuzz-testing-java-and-other-managed-languages-289a0be0a9c5.

[42] Hyperledger. *2019-10-07 Indy Contributors Call*. URL: https://wiki.hyperledger.org/display/indy/2019-10-07+Indy+Contributors+Call (visited on May 29, 2020).

[43] Sovrin Foundation. *Sovrin Foundation Launches Test Token for Decentralized Identity Network*. URL: https://sovrin.org/sovrin-foundation-launches-test-token-for-decentralized-identity-network/ (visited on May 29, 2020).

[44] Hyperledger. *Negotiate Proof*. URL: https://github.com/hyperledger/indy-sdk/tree/master/docs/how-tos/negotiate-proof (visited on May 29, 2020).

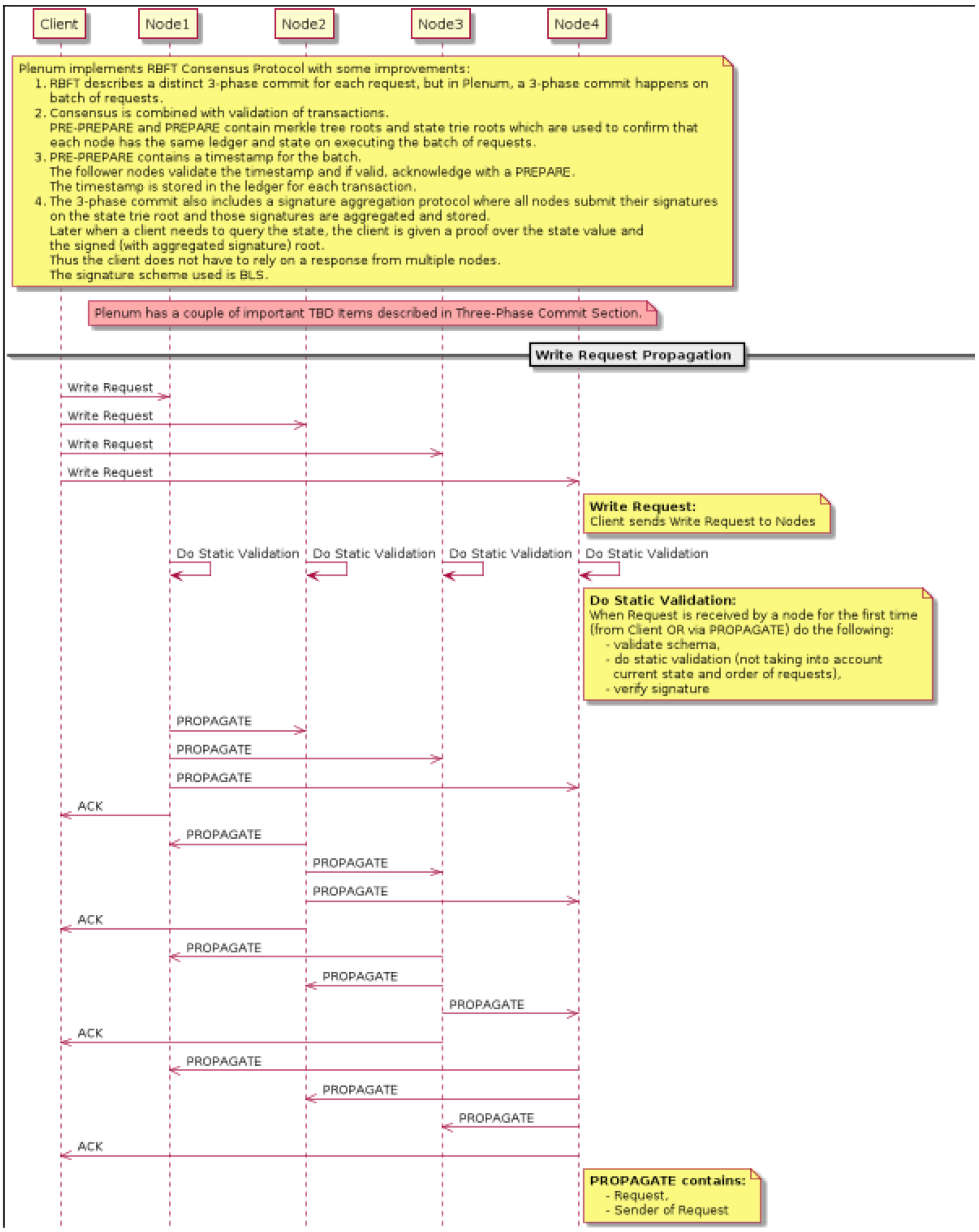
[45] Hyperledger. *Consensus Protocol*. URL: https://github.com/hyperledger/indy-plenum/blob/master/docs/source/diagrams/consensus-protocol.png (visited on May 25, 2020).

[46] DEDIS. *DEDIS - Decentralized and Distributed Systems.* URL: https://www.epfl.ch/labs/dedis/ (visited on May 31, 2020).

Appendix A

Consensus Diagram

Credit for consensus protocol diagram: indy-plenum [45].



Plenum implements RBFT Consensus Protocol with some improvements:
1. RBFT describes a distinct 3-phase commit for each request, but in Plenum, a 3-phase commit happens on batch of requests.
2. Consensus is combined with validation of transactions.
   PRE-PREPARE and PREPARE contain merkle tree roots and state trie roots which are used to confirm that each node has the same ledger and state on executing the batch of requests.
3. PRE-PREPARE contains a timestamp for the batch.
   The follower nodes validate the timestamp and if valid, acknowledge with a PREPARE.
   The timestamp is stored in the ledger for each transaction.
4. The 3-phase commit also includes a signature aggregation protocol where all nodes submit their signatures on the state trie root and those signatures are aggregated and stored.
   Later when a client needs to query the state, the client is given a proof over the state value and the signed (with aggregated signature) root.
   Thus the client does not have to rely on a response from multiple nodes.
   The signature scheme used is BLS.

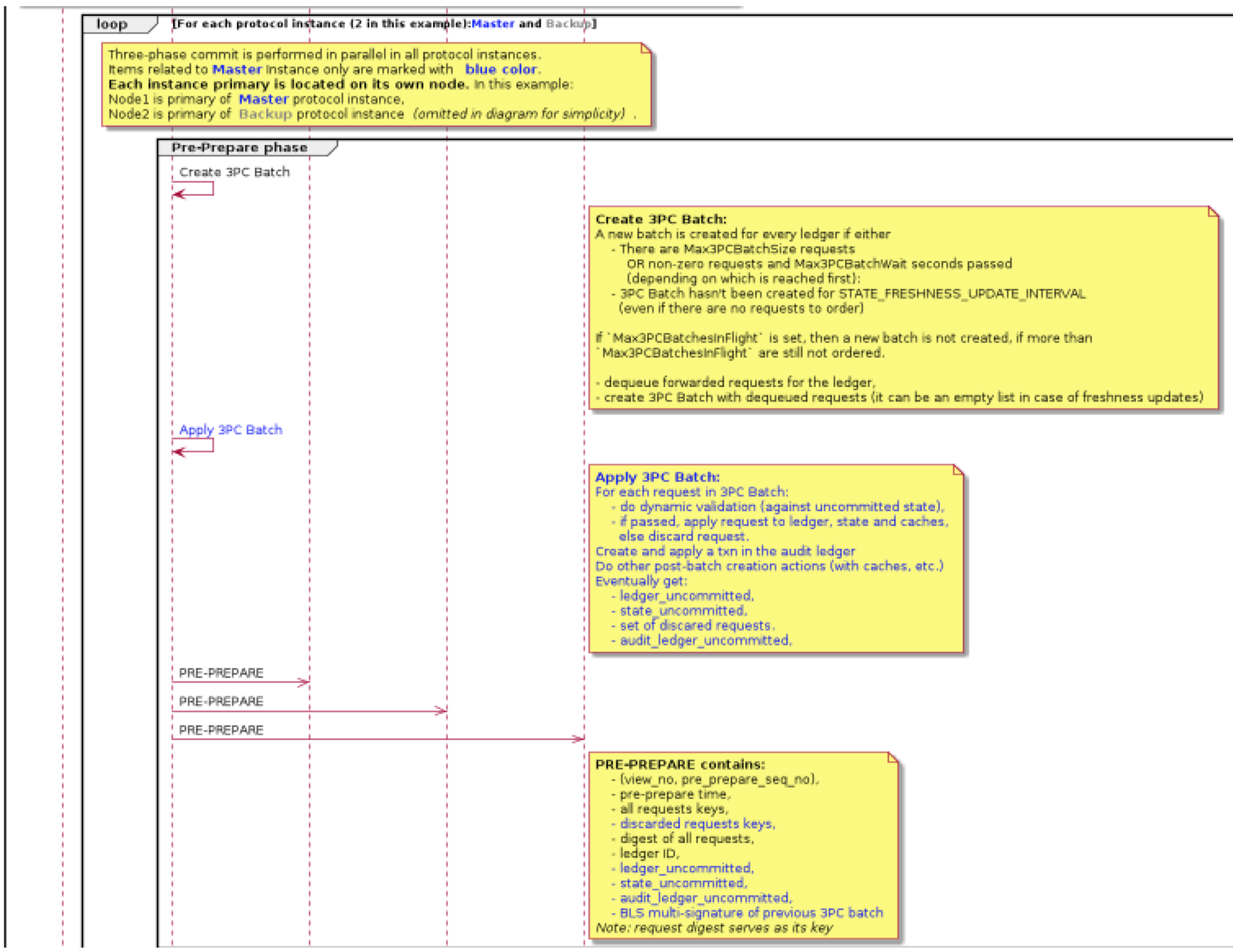
Plenum has a couple of important TBD items described in Three-Phase Commit Section.

Write Request:
Client sends Write Request to Nodes

Do Static Validation:
When Request is received by a node for the first time (from Client OR via PROPAGATE) do the following:
- validate schema,
- do static validation (not taking into account current state and order of requests),
- verify signature

PROPAGATE contains:
- Request,
- Sender of Request

Figure A.1: Consensus Protocol Propagate



Three-phase commit is performed in parallel in all protocol instances.
Items related to **Master** Instance only are marked with **blue color**.
Each instance primary is located on its own node. In this example:
Node1 is primary of **Master** protocol instance,
Node2 is primary of Backup protocol instance *(omitted in diagram for simplicity)* .

Create 3PC Batch:
A new batch is created for every ledger if either
- There are Max3PCBatchSize requests
  OR non-zero requests and Max3PCBatchWait seconds passed
  (depending on which is reached first);
- 3PC Batch hasn't been created for STATE_FRESHNESS_UPDATE_INTERVAL
  (even if there are no requests to order)

If `Max3PCBatchesInFlight` is set, then a new batch is not created, if more than `Max3PCBatchesInFlight` are still not ordered.

- dequeue forwarded requests for the ledger,
- create 3PC Batch with dequeued requests (it can be an empty list in case of freshness updates)

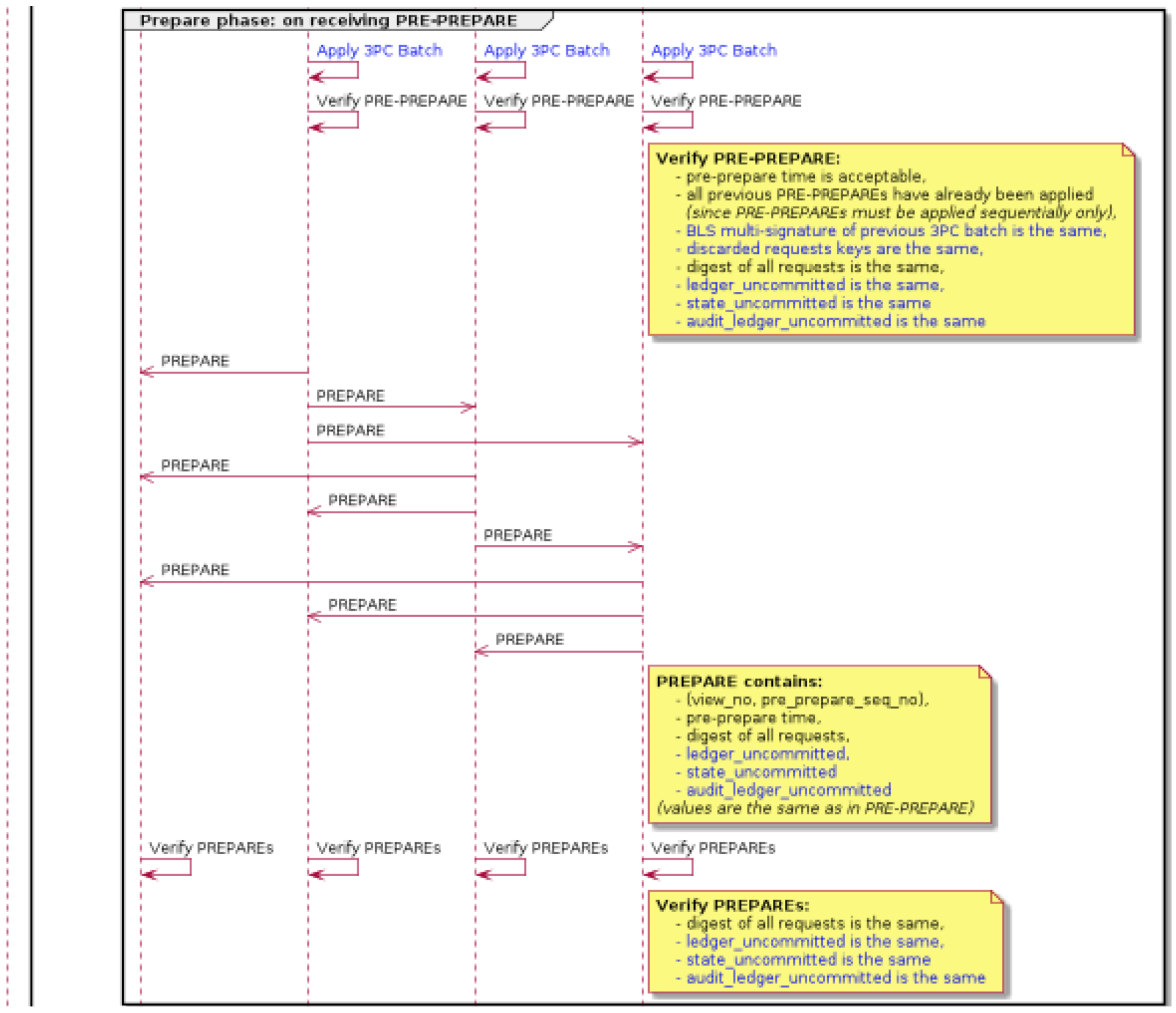
Apply 3PC Batch:
For each request in 3PC Batch:
- do dynamic validation (against uncommitted state),
- if passed, apply request to ledger, state and caches,
  else discard request.

Create and apply a txn in the audit ledger
Do other post-batch creation actions (with caches, etc.)
Eventually get:
- ledger_uncommitted,
- state_uncommitted,
- set of discared requests,
- audit_ledger_uncommitted,

PRE-PREPARE contains:
- (view_no, pre_prepare_seq_no),
- pre-prepare time,
- all requests keys,
- discarded requests keys,
- digest of all requests,
- ledger ID,
- ledger_uncommitted,
- state_uncommitted,
- audit_ledger_uncommitted,
- BLS multi-signature of previous 3PC batch

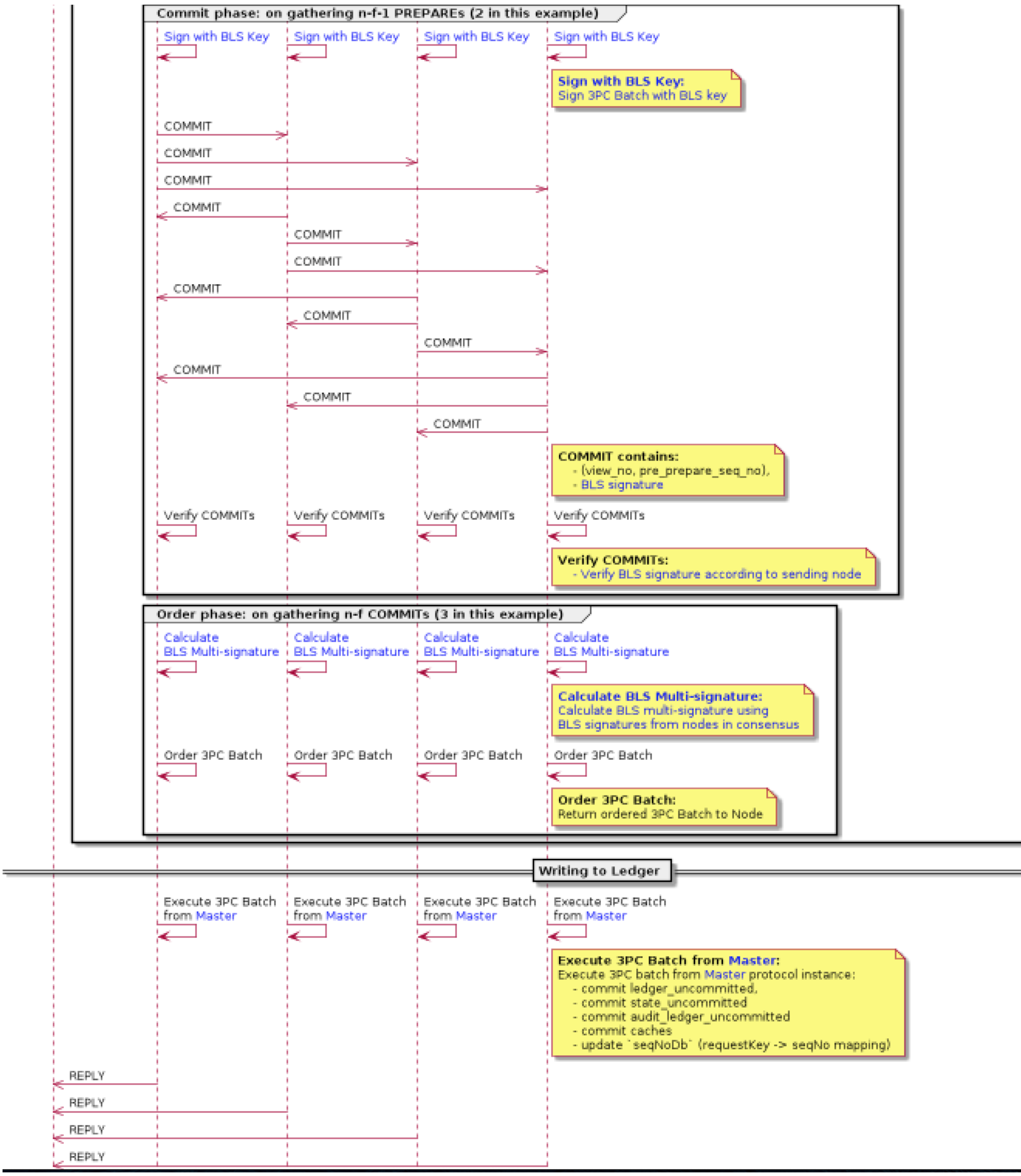
Note: request digest serves as its key

Figure A.2: Consensus Protocol Pre-Prepare



Prepare phase: on receiving PRE-PREPARE
Apply 3PC Batch
Apply 3PC Batch
Apply 3PC Batch
Verify PRE-PREPARE
Verify PRE-PREPARE
Verify PRE-PREPARE
Verify PRE-PREPARE:
- pre-prepare time is acceptable,
- all previous PRE-PREPAREs have already been applied (since PRE-PREPAREs must be applied sequentially only),
- BLS multi-signature of previous 3PC batch is the same,
- discarded requests keys are the same,
- digest of all requests is the same,
- ledger_uncommitted is the same,
- state_uncommitted is the same
- audit_ledger_uncommitted is the same
PREPARE
PREPARE
PREPARE
PREPARE
PREPARE
PREPARE
PREPARE
PREPARE
PREPARE
PREPARE contains:
- (view_no, pre_prepare_seq_no),
- pre-prepare time,
- digest of all requests,
- ledger_uncommitted,
- state_uncommitted
- audit_ledger_uncommitted
(values are the same as in PRE-PREPARE)
Verify PREPAREs
Verify PREPAREs
Verify PREPAREs
Verify PREPAREs
Verify PREPAREs:
- digest of all requests is the same,
- ledger_uncommitted is the same,
- state_uncommitted is the same
- audit_ledger_uncommitted is the same

Figure A.3: Consensus Protocol Prepare

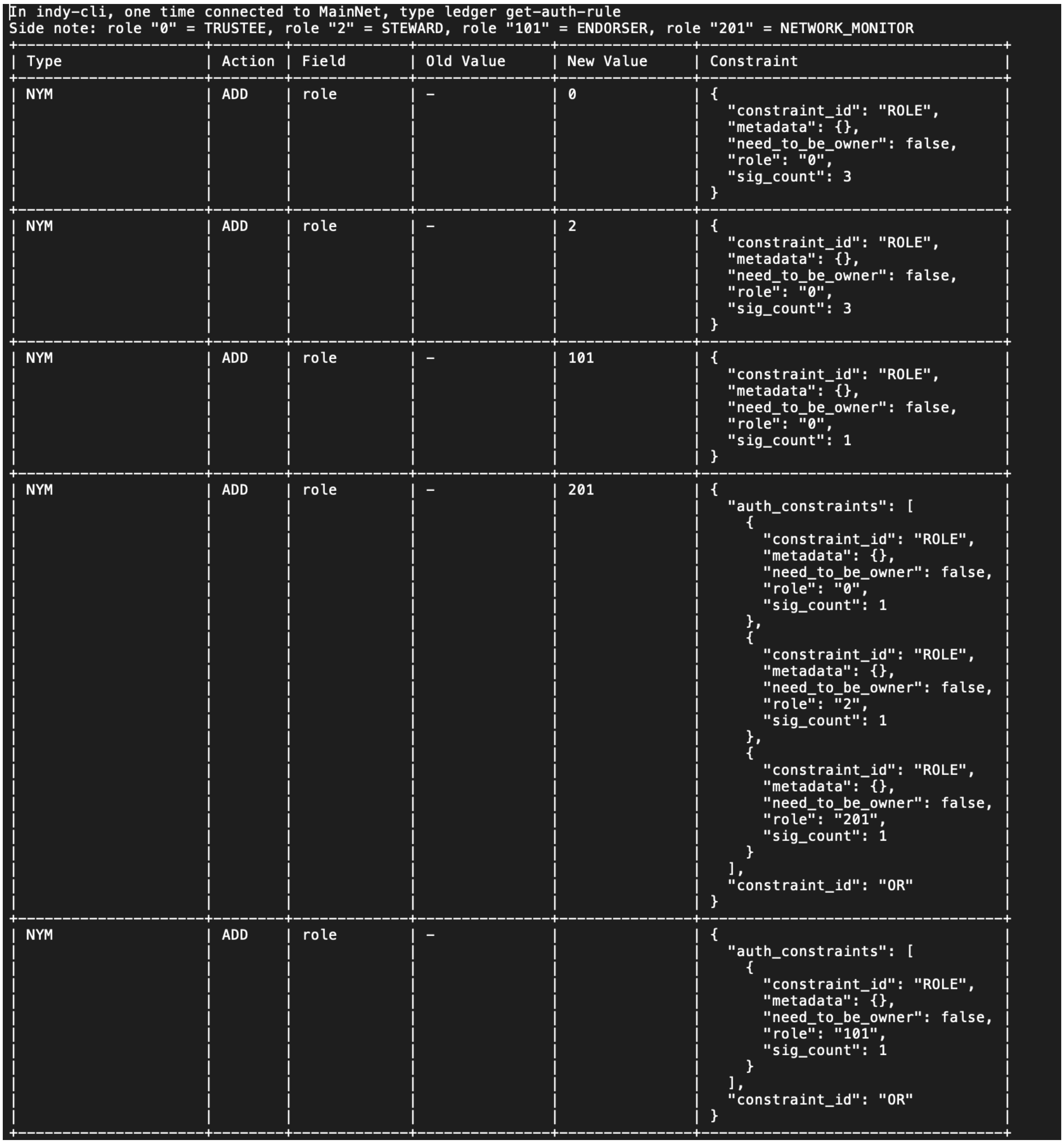


Commit phase: on gathering n-f-1 PREPAREs (2 in this example)
Sign with BLS Key
Sign with BLS Key
Sign with BLS Key
Sign with BLS Key
Sign with BLS Key:
Sign 3PC Batch with BLS key
COMMIT
COMMIT
COMMIT
COMMIT
COMMIT
COMMIT
COMMIT
COMMIT
COMMIT
COMMIT
COMMIT
COMMIT
COMMIT contains:
- (view_no, pre_prepare_seq_no),
- BLS signature
Verify COMMITs
Verify COMMITs
Verify COMMITs
Verify COMMITs
Verify COMMITs:
- Verify BLS signature according to sending node
Order phase: on gathering n-f COMMITs (3 in this example)
Calculate BLS Multi-signature
Calculate BLS Multi-signature
Calculate BLS Multi-signature
Calculate BLS Multi-signature
Calculate BLS Multi-signature:
Calculate BLS multi-signature using BLS signatures from nodes in consensus
Order 3PC Batch
Order 3PC Batch
Order 3PC Batch
Order 3PC Batch
Order 3PC Batch:
Return ordered 3PC Batch to Node
Writing to Ledger
Execute 3PC Batch from Master
Execute 3PC Batch from Master
Execute 3PC Batch from Master
Execute 3PC Batch from Master
Execute 3PC Batch from Master:
Execute 3PC batch from Master protocol instance:
- commit ledger_uncommitted,
- commit state_uncommitted
- commit audit_ledger_uncommitted
- commit caches
- update `seqNoDb` (requestKey -> seqNo mapping)
REPLY
REPLY
REPLY
REPLY

Figure A.4: Consensus Protocol Commit

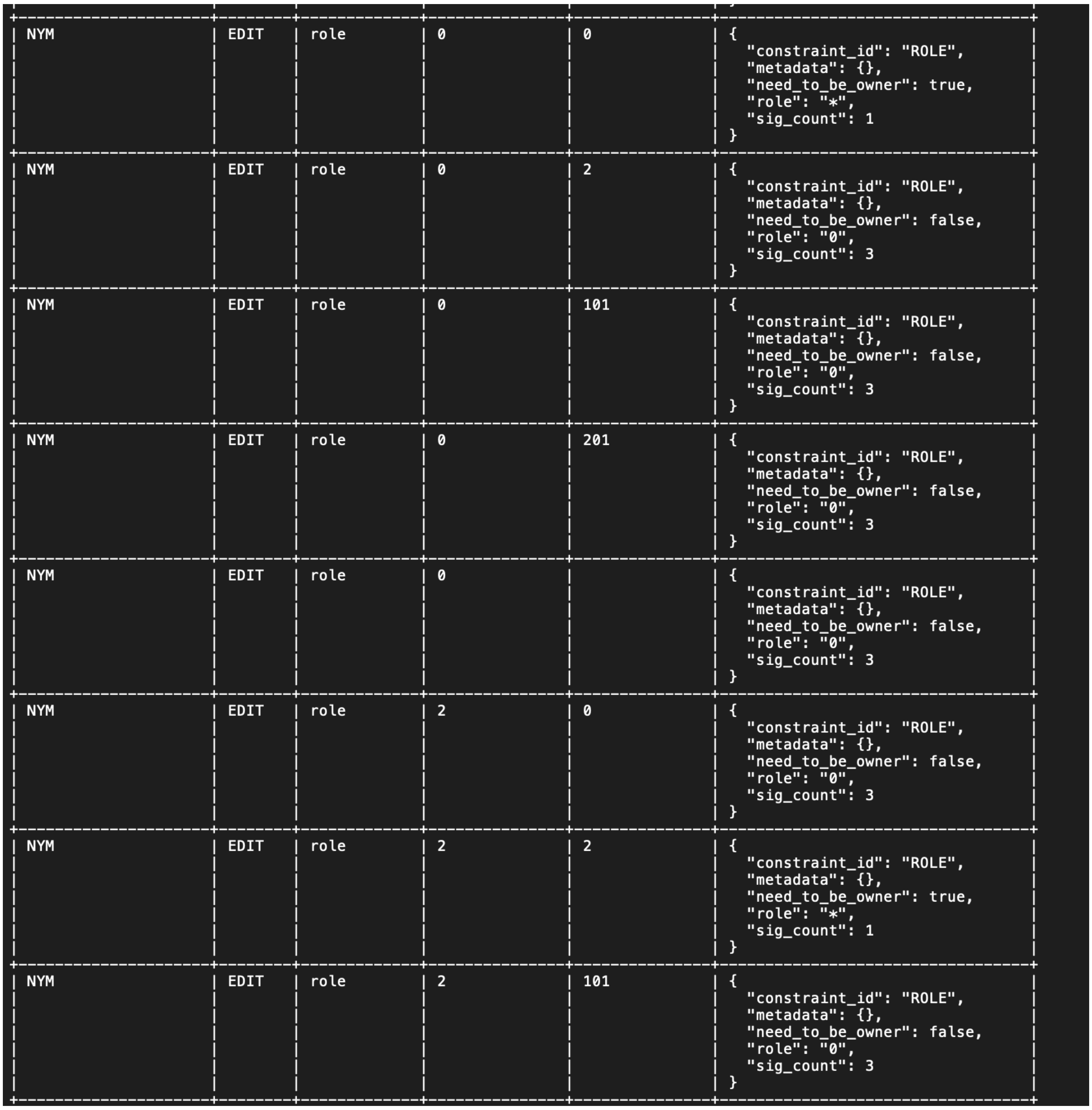
Appendix B

Auth-Rules MainNet



```
In indy-cli, one time connected to MainNet, type ledger get-auth-rule
Side note: role "0" = TRUSTEE, role "2" = STEWARD, role "101" = ENDORSER, role "201" = NETWORK_MONITOR
+----------------------+--------+--------------+----------------+----------------+----------------------------------+
| Type                 | Action | Field        | Old Value      | New Value      | Constraint                       |
+----------------------+--------+--------------+----------------+----------------+----------------------------------+
| NYM                  | ADD    | role         | -              | 0              | {                                |
|                      |        |              |                |                |   "constraint_id": "ROLE",       |
|                      |        |              |                |                |   "metadata": {},                |
|                      |        |              |                |                |   "need_to_be_owner": false,     |
|                      |        |              |                |                |   "role": "0",                   |
|                      |        |              |                |                |   "sig_count": 3                 |
|                      |        |              |                |                | }                                |
+----------------------+--------+--------------+----------------+----------------+----------------------------------+
| NYM                  | ADD    | role         | -              | 2              | {                                |
|                      |        |              |                |                |   "constraint_id": "ROLE",       |
|                      |        |              |                |                |   "metadata": {},                |
|                      |        |              |                |                |   "need_to_be_owner": false,     |
|                      |        |              |                |                |   "role": "0",                   |
|                      |        |              |                |                |   "sig_count": 3                 |
|                      |        |              |                |                | }                                |
+----------------------+--------+--------------+----------------+----------------+----------------------------------+
| NYM                  | ADD    | role         | -              | 101            | {                                |
|                      |        |              |                |                |   "constraint_id": "ROLE",       |
|                      |        |              |                |                |   "metadata": {},                |
|                      |        |              |                |                |   "need_to_be_owner": false,     |
|                      |        |              |                |                |   "role": "0",                   |
|                      |        |              |                |                |   "sig_count": 1                 |
|                      |        |              |                |                | }                                |
+----------------------+--------+--------------+----------------+----------------+----------------------------------+
| NYM                  | ADD    | role         | -              | 201            | {                                |
|                      |        |              |                |                |   "auth_constraints": [          |
|                      |        |              |                |                |     {                            |
|                      |        |              |                |                |       "constraint_id": "ROLE",   |
|                      |        |              |                |                |       "metadata": {},            |
|                      |        |              |                |                |       "need_to_be_owner": false, |
|                      |        |              |                |                |       "role": "0",               |
|                      |        |              |                |                |       "sig_count": 1             |
|                      |        |              |                |                |     },                           |
|                      |        |              |                |                |     {                            |
|                      |        |              |                |                |       "constraint_id": "ROLE",   |
|                      |        |              |                |                |       "metadata": {},            |
|                      |        |              |                |                |       "need_to_be_owner": false, |
|                      |        |              |                |                |       "role": "2",               |
|                      |        |              |                |                |       "sig_count": 1             |
|                      |        |              |                |                |     },                           |
|                      |        |              |                |                |     {                            |
|                      |        |              |                |                |       "constraint_id": "ROLE",   |
|                      |        |              |                |                |       "metadata": {},            |
|                      |        |              |                |                |       "need_to_be_owner": false, |
|                      |        |              |                |                |       "role": "201",             |
|                      |        |              |                |                |       "sig_count": 1             |
|                      |        |              |                |                |     }                            |
|                      |        |              |                |                |   ],                             |
|                      |        |              |                |                |   "constraint_id": "OR"          |
|                      |        |              |                |                | }                                |
+----------------------+--------+--------------+----------------+----------------+----------------------------------+
| NYM                  | ADD    | role         | -              |                | {                                |
|                      |        |              |                |                |   "auth_constraints": [          |
|                      |        |              |                |                |     {                            |
|                      |        |              |                |                |       "constraint_id": "ROLE",   |
|                      |        |              |                |                |       "metadata": {},            |
|                      |        |              |                |                |       "need_to_be_owner": false, |
|                      |        |              |                |                |       "role": "101",             |
|                      |        |              |                |                |       "sig_count": 1             |
|                      |        |              |                |                |     }                            |
|                      |        |              |                |                |   ],                             |
|                      |        |              |                |                |   "constraint_id": "OR"          |
|                      |        |              |                |                | }                                |
+----------------------+--------+--------------+----------------+----------------+----------------------------------+
```

Figure B.1: Auth-Rule ADD



```
+----------------------+---------+--------------+---------------+----------------+------------------------------------+
| NYM                  | EDIT    | role         | 0             | 0              | {                                  |
|                      |         |              |               |                |   "constraint_id": "ROLE",         |
|                      |         |              |               |                |   "metadata": {},                  |
|                      |         |              |               |                |   "need_to_be_owner": true,        |
|                      |         |              |               |                |   "role": "*",                     |
|                      |         |              |               |                |   "sig_count": 1                   |
|                      |         |              |               |                | }                                  |
+----------------------+---------+--------------+---------------+----------------+------------------------------------+
| NYM                  | EDIT    | role         | 0             | 2              | {                                  |
|                      |         |              |               |                |   "constraint_id": "ROLE",         |
|                      |         |              |               |                |   "metadata": {},                  |
|                      |         |              |               |                |   "need_to_be_owner": false,       |
|                      |         |              |               |                |   "role": "0",                     |
|                      |         |              |               |                |   "sig_count": 3                   |
|                      |         |              |               |                | }                                  |
+----------------------+---------+--------------+---------------+----------------+------------------------------------+
| NYM                  | EDIT    | role         | 0             | 101            | {                                  |
|                      |         |              |               |                |   "constraint_id": "ROLE",         |
|                      |         |              |               |                |   "metadata": {},                  |
|                      |         |              |               |                |   "need_to_be_owner": false,       |
|                      |         |              |               |                |   "role": "0",                     |
|                      |         |              |               |                |   "sig_count": 3                   |
|                      |         |              |               |                | }                                  |
+----------------------+---------+--------------+---------------+----------------+------------------------------------+
| NYM                  | EDIT    | role         | 0             | 201            | {                                  |
|                      |         |              |               |                |   "constraint_id": "ROLE",         |
|                      |         |              |               |                |   "metadata": {},                  |
|                      |         |              |               |                |   "need_to_be_owner": false,       |
|                      |         |              |               |                |   "role": "0",                     |
|                      |         |              |               |                |   "sig_count": 3                   |
|                      |         |              |               |                | }                                  |
+----------------------+---------+--------------+---------------+----------------+------------------------------------+
| NYM                  | EDIT    | role         | 0             |                | {                                  |
|                      |         |              |               |                |   "constraint_id": "ROLE",         |
|                      |         |              |               |                |   "metadata": {},                  |
|                      |         |              |               |                |   "need_to_be_owner": false,       |
|                      |         |              |               |                |   "role": "0",                     |
|                      |         |              |               |                |   "sig_count": 3                   |
|                      |         |              |               |                | }                                  |
+----------------------+---------+--------------+---------------+----------------+------------------------------------+
| NYM                  | EDIT    | role         | 2             | 0              | {                                  |
|                      |         |              |               |                |   "constraint_id": "ROLE",         |
|                      |         |              |               |                |   "metadata": {},                  |
|                      |         |              |               |                |   "need_to_be_owner": false,       |
|                      |         |              |               |                |   "role": "0",                     |
|                      |         |              |               |                |   "sig_count": 3                   |
|                      |         |              |               |                | }                                  |
+----------------------+---------+--------------+---------------+----------------+------------------------------------+
| NYM                  | EDIT    | role         | 2             | 2              | {                                  |
|                      |         |              |               |                |   "constraint_id": "ROLE",         |
|                      |         |              |               |                |   "metadata": {},                  |
|                      |         |              |               |                |   "need_to_be_owner": true,        |
|                      |         |              |               |                |   "role": "*",                     |
|                      |         |              |               |                |   "sig_count": 1                   |
|                      |         |              |               |                | }                                  |
+----------------------+---------+--------------+---------------+----------------+------------------------------------+
| NYM                  | EDIT    | role         | 2             | 101            | {                                  |
|                      |         |              |               |                |   "constraint_id": "ROLE",         |
|                      |         |              |               |                |   "metadata": {},                  |
|                      |         |              |               |                |   "need_to_be_owner": false,       |
|                      |         |              |               |                |   "role": "0",                     |
|                      |         |              |               |                |   "sig_count": 3                   |
|                      |         |              |               |                | }                                  |
+----------------------+---------+--------------+---------------+----------------+------------------------------------+
```

Figure B.2: Auth-Rule EDIT