Secure Smart Contract-Based Data Aggregation using Homomorphic Encryption

Floor Joosen
Supervisor: Kaitai Liang
EEMCS, Delft University of Technology, The Netherlands

January 23, 2022

Abstract

Privacy and security are important topics in data aggregation, as companies are aggregating large amounts of sensitive data from their users. The aim of this paper is to improve the security and privacy of data aggregation using smart contracts, homomorphic encryption and post-quantum encryption methods on Hyperledger Fabric. After a literature study, the protocol discussed in [1] was taken as the starting point for an adapted protocol. The adapted protocol is implemented and tested on performance and the security is discussed. The protocol is improved in terms of security by resolving or lowering the chance of security flaws that were present in the protocol in [1], and introducing post-quantum encryption. However, the adapted protocol in its current implementation is not scalable.

1 Introduction

The expression "knowledge is power" has never been more appropriate than in the current day and age, as organisations aggregate large amounts of data from people to improve their practices. For example, in the power grid, intelligent distribution of power and proactive utility placements can be implemented because of aggregated power consumption data [2], [3].

Security and privacy play a critical role in these data aggregation applications since the aggregated data can be sensitive. Revealing sensitive data can have severe consequences for the user and their privacy. For example, the daily routines of users can be deduced based on power usage data alone [4]. Existing system design is currently being challenged due to a rise in events where systems were compromised [1].

The unique properties of Homomorphic Encryption, smart contracts, and blockchain when combined could present the solution to the problem of data privacy and security in data aggregation. Blockchain technology offers data security, traceability and robustness due to its decentralised nature and immutable transaction ledger [1], [5], [6]. Smart contracts remove the need for trust because of automatic execution and enforcement of agreed-upon terms [1]. Homomorphic Encryption adds privacy and confidentiality as it allows for encrypted data aggregation without ever exposing the plaintext [1], [2], [4].

There have already been numerous papers on applying data aggregation to smart contracts on blockchain networks with the added feature of homomorphic encryption [2], [4], [7]. These papers have in common that they covered one specific use case. [7] applies blockchain, smart contract, and homomorphic encryption for the aggregation of votes, whilst [4] applies it to the Internet of Things and [2] for the aggregation of power consumption. Since the setup of data aggregation is generally the same, it should be possible to propose one generalised protocol. In [1], one such protocol was proposed which is not only applicable to different use cases, but is also platform-independent.

The generality of the protocol in [1] does come with a cost. Its platform independence means complex solutions were used, which could be solved by platform-specific features instead. For example, the paper includes a proposal to implement a whitelist to authenticate participants, whilst with Hyperledger Fabric, built-in features could be used to achieve the same. Complex solutions due to platform independence are not its only flaw.

The protocol in [1] can further be improved in terms of security. First of all, the protocol has five security flaws that are covered in the paper. These could be resolved by using the proposed solutions, or Hyperledger Fabric specific features. Furthermore, the protocol can easily be adapted to use Post-Quantum Encryption methods. This would ensure the security of the protocol in a post-quantum era as well.

In this paper, the focus will be on introducing a protocol and tool for data aggregation utilising Homomorphic Encryption and smart contracts on the Hyperledger Fabric platform to increase privacy and security. The main contribution of this paper is the introduction of a private and secure data aggregation protocol that is generalised to fit many use case requirements. The protocol was also implemented and was made available for public use and future development.

The rest of the paper is structured as follows. Section 2 starts with the necessary background information. This is followed by the introduction of the protocol in section 3. In section 4 the implementation of

the protocol is discussed. Section 5 details the set-up for the performance analysis, and is followed up by the results in section 6. Section 7 contains the discussion of the results and overall security of the protocol and implementation. Next, section 8 contains a discussion on Responsible Computer Science. Finally, the conclusion and future recommendations are discussed in section 9.

2 Background Information

This section describes the required technologies for the proposed protocol. First blockchain is discussed in section 2.1. Next, in section 2.2 Homomorphic Encryption is covered. In section 2.3 Post-Quantum Encryption is touched on. Finally, in section 2.4 an overview of how the concepts fit together is given.

2.1 Blockchain

A blockchain platform, at its core, is a peer-to-peer network utilising a distributed ledger and transactions [6], [8]. Peers can interact with the network by submitting transactions. These transactions are either queries or updates to the ledger. When the transaction is an update, the peers on the network must reach consensus on the submitted transaction using the consensus protocol.

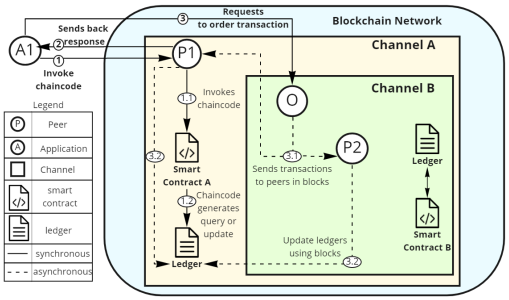

Figure 1: The simplified flow of a transaction on the Hyperledger Fabric network.

The distributed ledger is what makes blockchain platforms robust, immutable, and makes transactions traceable and verifiable [1], [5], [6]. First of all, the decentralised nature of the ledger eliminates a single-point of failure making the network more robust [6]. Furthermore, the ledger consists of two parts, the world state and the blockchain [8], which can be seen in Figure 2. The world state is a database containing the current state of the network, whilst the blockchain is a record of all transactions which led to the current world state. The hashes on the blockchain ensure that it is tamper-resistant, as changing or removing blocks would show a discrepancy in the recorded hashes [6]. Whilst the chain of blocks itself enables traceability [1]. Finally, any peer can verify the world state by starting at the first block in the blockchain and applying all the transactions recorded on the blockchain. The combination of these properties is how blockchain platforms eliminate the need for trust between peers, something which smart contracts contribute to even further.

Smart contracts are trusted distributed applications [8]. It is a mean with which the life cycle of the assets on the world state can be controlled. It is trusted because of its automatic execution and enforcement of agreed-upon terms [1].

For this paper specifically the focus is on the Hyperledger Fabric[1] (HF) blockchain network. HF is a permissioned blockchain network [9], meaning peers need to be authorised. HF is also very modular where several components can be adapted depending on needs [8]. Finally, channels can be utilised which split off a part of the network to keep transactions hidden from unauthorised peers [1]. Channels in essence almost create an entirely new network on the existing network using only authorised parties. This can be seen in Figure 1 where P1 can access the ledger on channel A, but not on channel B.

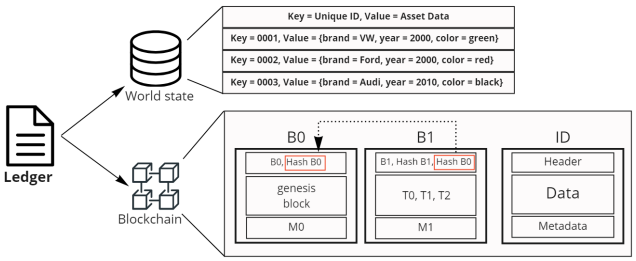

Figure 2: The two parts that make up the ledger, the blockchain and world state.

Transactions on the Hyperledger Fabric network need to consider concurrency, since its ledger is distributed. In Figure 1 the transaction flow of a single transaction can be seen. Step 2 returns a proposal response to the application. This proposal response includes read and write sets which specify the assets that were read from and written to. If two transactions try to edit the same value at the same time, one will succeed. The data read by the second will be registered as outdated, making it fail. This is defined as a Write-Write Concurrency Conflict [10]. This poses a problem for data aggregation, since this involves many devices trying to update a single value.

In order to avoid Write-Write Concurrency Conflicts, Composite keys can be utilised. Composite keys are keys made up out of multiple parts [11]. For data aggregation keys, two parts could be used. The first is the same for all devices, the second is unique per device. The combination is used to commit data to the ledger. Finally, the data having the first similar part can be merged, and forms the result. This method avoids all devices trying to edit the same value at the same time.

2.2 Homomorphic Encryption

Homomorphic Encryption (HE) is an encryption method that has as a unique characteristic that computations can be performed on the ciphertext without intermediate decryption [12]. When decrypting the ci-

[1]For more information on Hyperledger Fabric their documentation can be consulted at hyperledger-fabric.readthedocs.io/en/latest/whatis

phertext the plaintext has the right value as if the computations were performed on the plaintext. An example of this is given in Figure 3. In this figure the encryption method is additive homomorphic, as addition can be performed on the ciphertext without having to decrypt the numbers in between. However, addition is not the only way in which a scheme can be homomorphic.

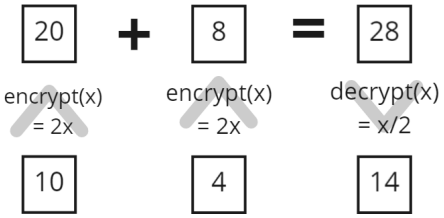

Figure 3: An example of homomorphic encryption.

HE schemes are divided into one of the following three groups, Partial HE (PHE), Somewhat HE (SWHE) and Full HE (FHE) [1], [13]. Which group a scheme belongs to depends on the number of functions it can apply to the ciphertext and how often (See Table 1).

Table 1: The properties of the different HE groups shown by homomorphism and frequency with which these can be applied.

Group	#functions	Frequency
PHE	Only one	Unlimited
SWHE	More than one	Limited
FHE	All functions	Unlimited

Other than homomorphic characteristics, the homomorphic encryption schemes also have general encryption characteristics. First of, the scheme could either be symmetric or asymmetric. Symmetric schemes generally have a better performance, whilst asymmetric schemes are more secure [14]. Next, the expansion rate of an encryption scheme is also important to consider. The expansion rate is a value that indicates how many bits of ciphertext are produced for every encrypted bit [1]. This is an important metric for data aggregation which often involves devices with limited computation power and memory. This also means performance is an important factor to consider.

2.3 Post-Quantum Encryption

The development of quantum computers is increasing the interest in post-quantum cryptography. Quantum computers are still under development, but already the threat to widely-used public-key cryptosystems (PKCs) is becoming apparent. It is approximated that quantum computers will effortlessly be able to compromise secure PKCs in two decades [15]. This threat has led to the increase in development of post-quantum cryptography algorithms, which are secure against attacks from quantum computers.

The development of post-quantum cryptography is crucial for blockchain technology, especially for the application of private and secure data aggregation. It is important that the sensitive data that is aggregated stays secure in the post-quantum era. Therefore, the application of post-quantum encryption was included when developing the protocol.

2.4 Combining Key Techniques

Figure 4 visualises the concepts involved in the implementation, how they fit together and what properties apply. As important traits privacy, security, memory efficiency, automation and user-friendliness are present in the figure. Other traits, for example performance, were also considered during each part of the process.

Automation was also considered as one of the important traits as many data aggregation processes are embedded. For example, data aggregation of power consumption in a region is generally automated.

3 The Protocol

The protocol in this paper includes three types of actors: the Asker, Operator and Participant. The Asker is requesting the aggregated data, and the data gets collected from the Participant. The Operator is also a Participant, but they have the added task of reporting the aggregated data to the Asker.

The protocol makes use of two channels, Channel A and Channel P. Channel A includes all actors, whilst Channel P excludes the Asker. Both Channel A and P have one smart contract deployed on it, the Data Query Contract and the Aggregation Contract respectively. The Data Query Contract functions for the Asker to initialise the process, and the Operators to report back the data, while the Aggregation Contract executes the actual data aggregation process.

In the protocol, any plaintext is always encrypted in the local application of the user. Sending plaintext from the application to the peer on the network risks interception of the plaintext. Another option is having one private key and store it on the ledger, but this would severally compromise the security of the private key. Therefore, the safest option is to have the

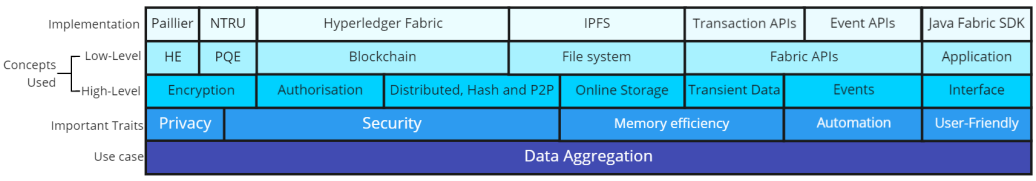

Figure 4: Overview of the important traits of data aggregation and how to achieve those through concepts.

plaintext encrypted in the local application with a locally stored private key.

The different steps of the protocol can be viewed in Figure 5. The protocol can be divided into three phases: Selection, Aggregation and Retrieval. In the Selection phase the process is initialised and the Operators are selected. This phase includes the following steps:

1. The Asker initialises the data aggregation process with the number of Operators (o), the time limit of the process (t) and the public keys of the homomorphic and post-quantum encryption methods (HK_A and QK_A, respectively). This triggers an event that notifies all participants of the new data query.
2. The notified Participants try to volunteer as Operators by first starting the process on Channel P and then committing their own QK_{o_i}s. Only the first o Participants succeed in this and become Operators, the rest stay Participants. The Operators receive back an index corresponding to the index of their QK_{o_i} as stored in the list on the ledger. This list has length o. When the final Operator is selected, an event is triggered which starts the Aggregation phase.

The Aggregation phase is the data aggregation part of the protocol which has two steps:

3. The Participants check their number of Operator threshold (r). This threshold dictates if the Participant wants to participate given o. If $o >= r$ or the Participant is also an Operator, the Participant participates.
4. The remaining Participants and all Operators add their encrypted, obfuscated data ($E_P(DN_i)$) to the total data on the ledger ($E_P(DN_n)$), and add a set of nonces ($E_N(N_i)$). The data is obfuscated by the nonces and encrypted using homomorphic encryption. There are o nonces per Participant and the nonces are encrypted using post-quantum encryption.

The Retrieval phase is started when t is reached or when another optional rule applies. An example of an optional rule is number of Participants. The Retrieval phase consists of the following steps:

5. The Operators retrieve $E_P(DN_n)$ and $E_N(N)$.
6. The Operators decrypt, sum and re-encrypt their respective nonces using the index they received at the start. During re-encryption the QK_A is used.
7. The Operators commit their summed, re-encrypted nonce to the ledger on Channel A. The first Operator commits the aggregated data and number of Participants, whilst the other Operators check these values. If an Operator notices an inconsistency they turn on the corresponding flag. The final Operator triggers an event that notifies the Asker that the process has ended.
8. The Asker retrieves $E_P(DN_n)$ and the re-encrypted nonces. By decrypting both, and then subtracting the nonces from the obfuscated data, the actual data is reinstated.

4 The Implementation of the Protocol

In this section the most important design choices and implementation details will be discussed. Section 4.1 gives an overview of the implemented smart contracts and the available transactions. Next, in section 4.2 the dependencies required in the implementation are mentioned.

4.1 Smart Contracts

As discussed, the protocol contains two smart contracts, the data query contract and the aggregation process contract. An overview of the available transactions in the data query contract and aggregation process contract is given in Table 2. The tables also specify the step in the protocol that the transaction is used in.

The Remove and Exists transactions in both tables do not correspond to a step in the protocol. The Remove transaction can be used by the Asker to remove a data query process when this is wanted, in this case an event triggers the removal of the corresponding aggregation process as well. The Exists transaction is useful in a practical setting to check if a certain asset is present in the world state.

4.2 Implementation Dependencies

In order to implement the protocol discussed in the previous section, several dependencies were combined.

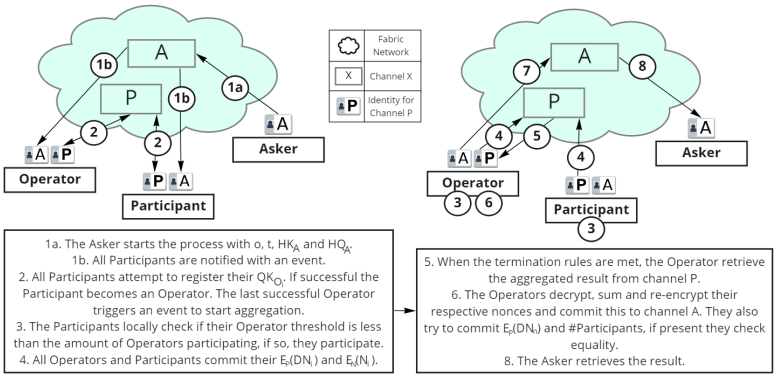

Figure 5: An abstraction of the protocol workflow on the network. The applications are left out for simplicity.

Table 2: The transactions available in the Data Query and Aggregation Process contracts.

Contract	Transaction	Type	Description	Event	Actor	Transient Param	Input Param	Step
Data Query	Start	submit	Initialise data query	StartQuery	A	$HK_P, QK_{N,A}$	id, #O, t	1
	Add	submit	Check results, commit nonces	DoneQuery	O	$E_P(DN_n), E_Q(N_i)$	id, #P, i	7
	Close	submit	Closes data query	-	A	-	id	8
	Retrieve	eval	Returns data query	-	A	-	id	8
	Remove	submit	Removes data query	RemoveQuery	A	-	id	-
	Exists	eval	Checks existence of data query	-	A	-	id	-
Aggregation Process	Start	submit	Initialise agg. process, select Os	StartAggregating	P	$QK_{N,O}$	id,IPFS hash	2
	Add	submit	Adds data and nonces	-	P/O	$E_P(DN_i), E_Q(N_i)$	id	4
	Close	submit	Closes agg. process	-	O	-	id	5
	Retrieve	eval	Returns agg. process	-	O	-	id	5
	Remove	submit	Removes agg. process	-	O	-	id	-
	Exists	eval	Checks existence of agg. process	-	P/O	-	id	-

In this section these dependencies are discussed.

Homomorphic Encryption

Homomorphic encryption is utilised to sum the obfuscated encrypted data in the protocol. The homomorphic encryption scheme was selected based on functionality, performance, memory usage and security. In Appendix A.1 the selection process is elaborated on.

The homomorphic encryption scheme selected for the implementation is Paillier. Paillier is a assymetric, additive PHE scheme. Its expansion rate is limited to around twice the bit-size of the public key regardless of the plaintext size [16]. In [1] a bit size of 2048 was used. In the implementation 4096 will be used to ensure security on a longer term [17]. Paillier is not post-quantum safe. Post-quantum safe Paillier-based schemes do exist, but this reduced the performance severely making it unusable [18].

In terms of dependencies, the library Javallier[2] was used. Several libraries implementing Paillier were found, and from the Java libraries Javallier was judged to be most reliable.

Post-Quantum Encryption

The National Institute of Standards and Technology (NIST) of the United States announced the Post-Quantum Cryptography Standardization (PQCS) competition in which, among other things, PKC were tested against attacks to determine their security level against quantum attacks [15]. This competition will include four rounds, and, as of writing, has published the results of the third round. The third round finalists for PKC are CRYSTALS-KYBER, NTRU, SABER and Classic McEliece [19], and were thus far deemed most secure in the post-quantum era.

The NTRUEncrypt algorithm implemented by the Bouncy Castle package[3] was used and implemented in both smart contracts and the application. Their specific key parameters "APR2011_743_FAST" was utilised, which is running NTRUEncrypt-743. This, for example, uses a public key size of 8184 bits and a private key size of 9384 bits [15].

Online Storage

In order to limit the storage needed for the process on the blockchain, online storage was utilised in the implementation to store the ciphertexts and public keys. Since a distributed ledger is used, every peer has to store the data for each transaction. Since this is also permanently stored on the blockchain of each peer, it is important to limit data stored on the ledger if possible. Especially for ciphertexts and public keys as these often have large bit sizes.

The InterPlanetary File System[4] (IPFS) was used for online storage because of its security benefits. The pointer to a file is a cryptographic hash of its contents, and referred to as Content Identifier (CID) [20], [21]. Since a CID is made from the contents of a file, the CID changes when the file contents change. This makes the files immutable. The CIDs will be stored on the ledger, which is also immutable [6]. Furthermore, the generation and storage of the CIDs will be managed in the smart contracts, and submissions of transactions by default are signed in Hyperledger Fabric by the sender. This combination of blockchain, IPFS, smart contracts, and signed transaction submissions makes it impossible to change the ciphertext and public keys that will be stored in the IPFS files.

Fabric APIs

In order to ensure the memory advantage of IPFS and reduce the number of places the ciphertext is stored in, the transient data API was used to send the ciphertext and public keys to the smart contract. When applying a transaction to the ledger, the input of the smart contract is also stored on the ledger. Transient data can be used by the smart contract, but is not stored on the ledger as input to the transactions [22].

Furthermore, the event API is used to notify the peers on the network of the different aggregation stages. The events are set in the smart contracts, which can be seen in Table 2. When the transaction is committed to the ledger, the listeners are notified [11]. These listeners are set in the client applications.

Unfortunaly, Composite keys were not included in the implementation due to time restrictions. The Key APIs would have been utilised for this. Instead, in order to mitigate the Write-Write Concurrency Conflicts a back-off algorithm is implemented when submitting a transaction. The back-off time is dependent on the amount of times a transaction failed, f, and a random number between 0-10, r. The resulting formula to calculate the back-off time is: $r * 200 + i * 1000$.

[2]For more information on the Javallier package their GitHub page can be found at github.com/n1analytics/javallier
[3]For more information on the Bouncy Castle package their GitHub page can be found at github.com/bcgit/bc-java
[4]For more information on the ipfs view https://ipfs.io/

High-Level Overview

In Figure 6 a high-level overview of the implementation of the protocol is shown. The smart contracts and client application were both implemented using Java. In order to connect to the Hyperledger Fabric network, the Fabric's SDK for Java[5] was utilised. The test-network from the fabric-samples repository[6] was used to set up a network for the implementation. Furthermore, version 2.4 of Hyperledger Fabric was utilised. The implementation was made public on GitHub[7].

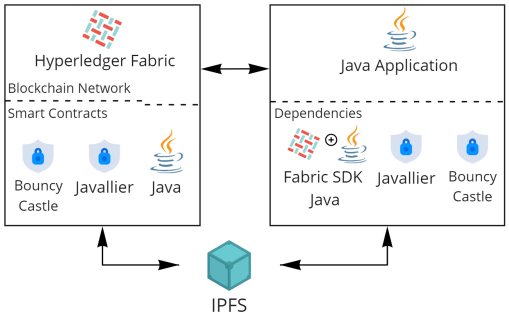

Figure 6: High level overview of implementation.

5 Performance Analysis Setup

In order to test the protocol, a performance analysis is performed on the smart contracts discussed in the implementation. For this analysis, the following steps will be timed:

1. The individual steps of the protocol discussed in section 5. This will be ran with three different configurations, of 1 or 5 Participant with 1 Operator, and 5 Participants with 5 Operators. This will be measured 10 times to produce an average. The average back-off time will also be measured. This will be run with and without encryption of Paillier and NTRUEncrypt-743.
2. The total cycle time over 100 cycles in 10 second intervals. This will be ran with 1 to 5 participant applications and Operators set, in increments of 2. The average back-off time will also be measured. This will be run with and without encryption of Paillier and NTRUEncrypt-743.

The setup of the experiment is as follows. The network will have one peer on channel A, and four on both channels. The network is restarted for every unique configuration. For all the aforementioned tests the data will be retrieved from a dataset containing hourly power consumption measurements in Megawatts [23]. Which entry is selected will be randomized. The nonces that are used in the process will be randomly generated positive integers. The implementation of the four test cases can also be found on GitHub. The tests were ran on Windows in the WSL2 terminal. The computer has 16GB RAM and an Intel Core i7-8750H CPU with 2.20 GHz and 6 cores.

The first test functions to test the individual steps of the protocol and how these are affected by increasing numbers of Participants and Operators. The second test is meant to put the protocol in a scenario that is closer to its real use case, where more than one cycle is required on a set interval.

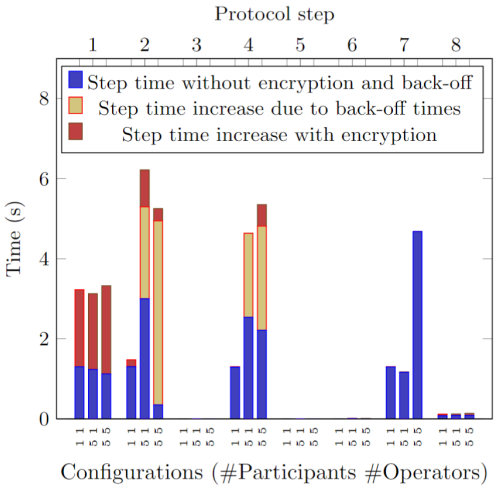

Figure 7: The individual steps of the protocol with different configurations of Participants and Operators.

6 Performance Analysis Results

In Figure 7 the results of the first test are displayed. This shows that step 1 is heavily impacted by encryption. Whilst steps 2 and 4 mostly impacted by the back-off time. Steps 1, 3, 5, 6 and 8 are not impacted by the different configurations of Participants and Operators. Steps 2 and 4 seem most impacted by the number of Participants, whilst step 7 is most impacted by the number of Operators.

An average run time could be determined using the steps in the protocol. However, due to how the steps were measured, the time it takes for events to reach listeners was not measured. Therefore, this data will not be used to draw conclusions on the total running time of the protocol.

Table 3: The average slowdown per Participant, Operator and with Paillier and NTRUEncrypt-743 over 100 cycles.

Factor	Avg. Slowdown (s)	Avg. Back-off (s)
#O	4.19	0.14
#P	1.66	1.26
Enc.	1.38	0.88

In Table 3 the average slowdown per added Operator, Participant and with encryption is shown for the 100 cycles test case. The biggest slowdown occurs when a new Operator is added, 4.19 seconds, next is per added participant, 1.66 seconds, and finally 1.38 seconds by adding encryption. For the average back-off time, the number of Participants have the largest impact, then the encryption and only finally the number of Operators, with 1.26, 0.88, and 0.14 seconds respectively.

[5]For more information on the Fabric SDK for Java their GitHub page can be found at github.com/hyperledger/fabric-sdk-java
[6]The repository can be found here: https://github.com/hyperledger/fabric-samples
[7]The full implementation of the protocol can be found on GitHub at https://github.com/Floor97/fabric-samples

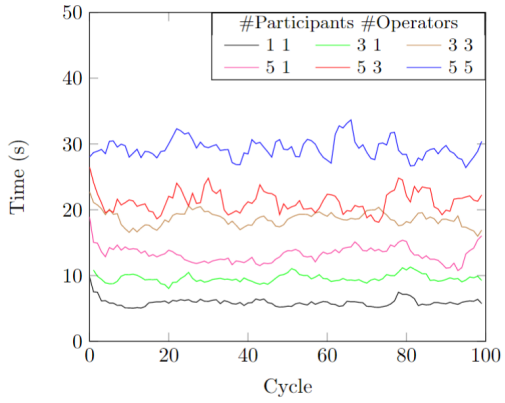

Figure 8: Time of a cycle of the protocol, run for 100 cycles with 10 second intervals. Different configurations were applied for Participants and Operators. A simple moving average was applied over five points.

In Figure 8 the results of running 100 cycles with varying amounts of Operators and Participants can be seen. A simple moving average over five points was applied to better visualise the trend in the data. Here it can be seen that the amount of Operators had a greater impact on the run time of a single cycle than the amount of Participants.

In this section the most relevant results were shown. The rest of the results, including a raw data visualisation of the 100 cycles, can be found in Appendix A.2.

7 Discussion

In this section the protocol and implementation will be discussed based on the results of the performance analysis and the literature study. First, in section 7.1, the performance results will be discussed. Next, in section 7.2 the security of the protocol and implementation will be discussed.

7.1 Performance Discussion

From Table 3 and Figure 8 it can be seen that the protocol in its current implementation is not scalable. Per Operator the average slowdown is 4.19 seconds, and per Participant 1.66 seconds. If data was being aggregated from 100 Participants with 10 Operators, in ideal settings and taking 1 Operator and 1 Participant as a starting point, it would take $7.43 + (4.19 \times 10) + (1.66 \times 100) = 215.33$ seconds, or approximately 3.59 minutes. For 1000 Participants and 100 Operator this increases to about 35 minutes. This means the protocol in its current implementation is not usable on a large scale. However, the results also reveal areas where performance improvements could be made.

The average slowdown per step when adding Participants can be seen in 7, and is likely due to Write-Write Concurrency Conflicts . The current implementation includes all Participants trying to commit an update with the same asset id in steps 2 and 4. This leads to an increase in transactions that are rejected and have to be repeated after a certain back-off time has elapsed. Furthermore, the more often a transactions fails, the higher the back-off time gets. Therefore, an implementation that avoids Write-Write Concurrency Conflicts could significantly improve the performance of the protocol. As previously mentioned, one way in which this could be achieved is by utilising Composite keys.

When the individual steps in Figure 7 are considered the performance of the protocol overall can be improved by generating key pairs during idle time. The step that suffers most in performance from encryption is step 1. This is where the Asker generates the key pairs, and commits those and the rest of the parameters on the ledger. After this step the Asker is idle until the last step. Generating the key pairs in this interval can reduce the time of a single protocol cycle by approximately 2 seconds. The same could be said for Operators which generate a NTRUEncrypt key pair in step 2. However, the performance benefit is not only less high, it is also less certain, since the Operator is not necessarily idle after step 2.

7.2 Security Discussion

In this section the security of the protocol and implementation will be analysed. First the encryption methods will be analysed. Then the five security flaws of the protocol covered in [1] will be visited for the protocol proposed in this paper.

7.2.1 Security of Encryption

As promising as the technology of HE is, there are still security issues that need to be addressed. The same characteristics that define HE, also cause it to be limited to security level IND-CPA [24]. This is not secure enough for sensitive data. In the same way, the chosen HE scheme for implementation, Paillier, is not post-quantum safe. This makes the data insecure both in the pre- and post-quantum era. However, other concepts were applied to mitigate this low security level.

In the protocol, the data security largely depends on the security of the nonces. The data is obfuscated by the nonces, and therefore without the nonces the obfuscated data is meaningless. In the implementation NTRUEncrypt-743 is used to encrypt the nonces. NTRUEncrypt-743 specifically is equivalent to AES-192 for classical security, and claims 159 bits of quantum security [15]. This means it is secure for now, but this needs to be improved in the future.

Apart from the strength of the encryption schemes, another important factor is managing the plaintext, ciphertext and key pairs. The keys are generated locally in the application, and only the public key is shared on the network. The plaintext is also only exposed in the local applications. This makes both secure, unless the application is compromised. Furthermore, the ciphertexts and public keys are sent as transient data and stored in IPFS. These cannot be tampered with due to the earlier discussed synergy of smart contracts, blockchain, IPFS and signed transactions.

7.2.2 Security Improvements and Flaws

Five security flaws of the protocol were discussed in [1], these include the following together with a status indication of the flaw in the protocol discussed in this

paper:

1. False Participant injection: resolved. This flaw occurred when unauthorised Participants participated in the aggregation process. The use of channels has resolved this issue.
2. Participant re-participation: not resolved. This can be resolved by registering identities of Participants when participating, either in the Context of the smart contract or on the ledger. This was not implemented due to time constraints.
3. False Participant contribution: not resolved. This occurs when Participants send false data to the process. In [1] a solution is proposed by letting Participants sign the data. However, this does risk the privacy of the Participants.
4. Collusion Asker Operator: reduced. The Asker and Operator could share their private keys to gain access to all data. The ability to have multiple Operators per aggregation process was introduced which reduces the chance of collusion by increasing the number of Operators that need to be involved. The Asker chooses the number of Operators used, but Participants set a threshold at which they participate. This forces the Asker to use a reasonable number of Operators.
5. False Operator data: moved. In [1], the Operator could return wrong data to the Asker. In the adapted protocol this is not possible as Operators check each others input. However, it is still possible for an Operator to return wrong nonces. To solve this, multiple data aggregation processes can be run in parallel as was proposed in [1], where all processes should return the same data.

Apart from the five security flaws discussed in [1], the implementation contains one more security flaw: Participant data request. Participants can currently request data from a data aggregation process, as if they were an Asker. This was not resolved due to the time constraint. However, this could easily still be added by adding identity checks or roles in the implementation. This would only allow Askers to submit data aggregation requests.

8 Responsible Research

This section discusses responsible research in relation to this paper. In section 8.1 the reproducibility of the research in this paper will be reflected on. In section 8.2 the integrity used while performing the research will be discussed.

8.1 Reproducibility

Reproducibility of experiments is crucial for scientific discovery, as is skepticism. If reproducibility is not promoted, it is hard to test points of skepticism, which hampers future discovery. I experienced this problem for myself as I was performing my literature study. Therefore, I tried to disclose as much as I could about the implementation and performance analysis setup.

In section 4 all relevant details about the implementation were disclosed. The implementation was also made public on GitHub so the exact implementation is clear. In section 5 the setup of the experiments run on the implementation was disclosed. Furthermore, the GitHub repository of the implementation also contains four branches which were used to run the four performance analysis experiments. The system environment was disclosed and in the README in the repository it is explained how the implementation can be ran, and how to run the experiments. Therefore, it can be exactly seen what experiments were ran and how and it should be possible for anyone with a Computer Science background to reproduce the experiments performed.

8.2 Integrity

Integrity is just as, if not more, important than encouraging reproducibility. Without a strict conduct of integrity anyone could make any wild claimes and publish them as fact without consequence. For this research, integrity as defined by the Netherlands Code of Conduct for Research Integrity[8] were upheld by upholding five principles defined within it: honesty, scrupulousness, transparency, independence and responsibility. Furthermore, the TU Delft Code of Conduct[9] was upheld.

One dataset being presented in section 6, namely the 100 cycles dataset, has been averaged using a simple moving average over 5 points. This has been mentioned with the Figure containing the dataset, and in the text discussing the dataset. The reason was to better visualise the trend in the dataset by smoothing out some of the variation. However, the dataset without simple moving average is still displayed in Appendix A.2. This Appendix also includes the rest of the data that was not shared in section 6, both of which are mentioned in the section itself.

9 Conclusion

The aim in this paper was to propose a private and secure protocol and tool for data aggregation utilising homomorphic encryption and smart contracts on Hyperledger Fabric. In order to do so, first a literature study was performed into the necessary concepts and the existing protocols for data aggregation utilising smart contracts and homomorphic encryption. From the existing protocols, the protocol in [1] was chosen as a starting point and improvements were applied to increase the security and privacy. Finally, a performance analysis was applied to the new protocol and the security was discussed.

The performance test showed that the current implementation of the protocol is not scalable, but that further improvements can be applied to increase the overall performance. For every Participant added to

[8]The Netherlands Code of Conduct for Research Integrity can be found here

[9]The TU Delft Code of Conduct can be found here

the process the runtime increases by on average 1.66 seconds, for each Operator this increases to 4.19 seconds. This shows the protocol is not scalable as for the case where there is 1000 Participants and 100 Operators it would already take approximately 35 minutes to complete. However, the performance of the protocol can still be improved by avoiding Write-Write Concurrency Conflicts and by generating the key pairs when the actors are idle.

The security discussion revealed the protocol was improved in terms of security and privacy, but still has other security flaws that need to be resolved. False Participant injection attacks are resolved by applying channels which block unauthorised Participants from participating. The usage of channels also increased the privacy by blocking the Asker from being able to see who participated in the data aggregation process. Collusion between the Asker and Operators was mitigated by introducing multiple Operators to the data aggregation process, and allowing Participants to define their own number of Operators threshold. Furthermore, Post-Quantum Encryption was introduced to the protocol to make the protocol secure in a post-quantum era.

Limitations

The performance analysis has the following limitations that need to be considered when viewing the results:

- The implementation was run locally on one computer. Data aggregation generally involves many devices on a global network. The performance could change when run in a real world scenario.
- Due to memory constraints the implementation was run with maximally four peers and six applications. Again, data aggregation usually involves a lot more devices and requires a lot more peers on the network. The results were therefore limited and the performance difference per thousand Participants might differ from what was recorded per Participant in this paper.

Future Recommendations

Due to the time constraint of the project, some features that were conceptualised could not be included. These are still recommended for future research and include the following:

- Research into factors affecting the scalability of the protocol. An example that was covered was the factor of Write-Write Concurrency Collisions. A solution is using the previously mentioned Compound keys per Participant, before merging these into the final result.
- Research into the performance of the protocol in a real world environment, i.e. a larger scale and a global network.
- Research on including roles or identities in the current implementation to resolve existent security flaws. Using either roles or identities would resolve the false Asker requests security flaw, whilst registering identities during the process resolves Participant re-participation.

References

[1] C. Regueiro, I. Seco, S. de Diego, O. Lage, and L. Etxebarria, "Privacy-enhancing distributed protocol for data aggregation based on blockchain and homomorphic encryption," *Information Processing & Management*, vol. 58, no. 6, p. 102 745, 2021, ISSN: 0306-4573. DOI: https://doi.org/10.1016/j.ipm.2021.102745. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306457321002272.

[2] Y. Wang, F. Luo, Z. Dong, Z. Tong, and Y. Qiao, "Distributed meter data aggregation framework based on blockchain and homomorphic encryption," *IET Cyber-Physical Systems: Theory & Applications*, vol. 4, no. 1, pp. 30–37, 2019. DOI: https://doi.org/10.1049/iet-cps.2018.5054. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-cps.2018.5054. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-cps.2018.5054.

[3] P. Singh, M. Masud, M. S. Hossain, and A. Kaur, "Blockchain and homomorphic encryption-based privacy-preserving data aggregation model in smart grid," *Computers & Electrical Engineering*, vol. 93, p. 107 209, 2021, ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2021.107209. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790621002032.

[4] F. Loukil, C. Ghedira-Guegan, K. Boukadi, and A.-N. Benharkat, "Privacy-preserving iot data aggregation based on blockchain and homomorphic encryption," *Sensors*, vol. 21, no. 7, 2021, ISSN: 1424-8220. DOI: 10.3390/s21072452. [Online]. Available: https://www.mdpi.com/1424-8220/21/7/2452.

[5] J. Chen and F. You, "Application of homomorphic encryption in blockchain data security," in *Proceedings of the 2020 4th International Conference on Electronic Information Technology and Computer Engineering*, 2020, pp. 205–209. DOI: 10.1145/3443467.3443754.

[6] R. Awadallah, A. Samsudin, J. S. Teh, and M. Almazrooie, "An integrated architecture for maintaining security in cloud computing based on blockchain," *IEEE Access*, vol. 9, pp. 69 513–69 526, 2021. DOI: 10.1109/ACCESS.2021.3077123.

[7] B. Yu, J. K. Liu, A. Sakzad, *et al.*, "Platform-independent secure blockchain-based voting system," in *Information Security*, L. Chen, M. Manulis, and S. Schneider, Eds., Cham: Springer International Publishing, 2018, pp. 369–386, ISBN: 978-3-319-99136-8.

[8] L. J. K. Marija S. Kristić, "Hyperledger frameworks with a special focus on hyperledger fabric," *MILITARY TECHNICAL COURIER*, vol. 68, 3 2020. DOI: 10.5937/vojtehg68-26206. [Online]. Available: https://doi.org/10.5937/vojtehg68-26206.

[9] P.-C. Chen, T.-H. Kuo, and J.-L. Wu, "A study of the applicability of ideal lattice-based fully homomorphic encryption scheme to ethereum blockchain," *IEEE Systems Journal*, vol. 15, no. 2, pp. 1528–1539, 2021. DOI: 10.1109/JSYST.2021.3064053.

[10] L. Xu, W. Chen, Z. Li, J. Xu1, A. Liu, and L. Zhao, "Solutions for concurrency conflict problem on hyperledger fabric," *World Wide Web*, vol. 24, pp. 463–482, 2021. [Online]. Available: https://doi.org/10.1007/s11280-020-00851-6.

[11] A. O'Dowd, M. Sykes, B. Logan, Svetoslav, J. Taylor, and D. Enyeart, *Transaction context*, [Online; accessed 10-01-2022], 2020. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/latest/developapps/transactioncontext.html.

[12] C. Fontaine and F. Galand, "A survey of homomorphic encryption for nonspecialists," *Journal on Information Security*, no. 1, 2007. DOI: 10.1155/2007/13801. [Online]. Available: https://sci-hub.se/10.1155/2007/13801.

[13] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Comput. Surv.*, vol. 51, no. 4, Jul. 2018, ISSN: 0360-0300. DOI: 10.1145/3214303. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/3214303.

[14] M. Al-Shabi, "A survey on symmetric and asymmetric cryptography algorithms in information security," *International Journal of Scientific and Research Publications (IJSRP)*, vol. 9, p8779, Mar. 2019. DOI: 10.29322/IJSRP.9.03.2019.p8779.

[15] T. M. Fernández-Caramés, "From pre-quantum to post-quantum iot security: A survey on quantum-resistant cryptosystems for the internet of things," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6457–6480, 2020. DOI: 10.1109/JIOT.2019.2958788.

[16] R. S. Chakraborty, J. Mathew, and A. V. Vasilakos, *Security and fault tolerance in Internet of things*. Springer, 2019.

[17] D. Giry and P. Bulens, *Cryptographic key length recommendation*, [Online; accessed 19-December-2021], 2017. [Online]. Available: https://www.keylength.com/.

[18] S. D. Galbraith, "Elliptic curve paillier schemes," *Journal of Cryptology*, vol. 15, pp. 129–138, 2002. [Online]. Available: https://doi.org/10.1007/s00145-001-0015-6.

[19] N. I. of Standards and Technology, *Pqc standardization process: Third round candidate announcement*, [Online; accessed 19-December-2021], 2020. [Online]. Available: https://csrc.nist.gov/News/2020/pqc-third-round-candidate-announcement.

[20] *Ipfs powers the distributed web*, [Online; accessed 10-01-2022]. [Online]. Available: https://ipfs.io/.

[21] M. Vashistha and F. A. Barbhuiya, "Document management system using blockchain and inter planetary file system," 2020. DOI: 10.1145/3384943.3409443.

[22] D. Enyeart, M. Sykes, joealewine, *et al.*, *Private data*, [Online; accessed 10-01-2022], 2021. [Online]. Available: https://hyperledger-fabric.readthedocs.io/en/latest/private-data-arch.html.

[23] R. Mulla, *Hourly energy consumption*, [Online; accessed 17-01-2022], 2018. [Online]. Available: https://www.kaggle.com/robikscube/hourly-energy-consumption.

[24] P. Martins, L. Sousa, and A. Mariano, "A survey on fully homomorphic encryption: An engineering perspective," *ACM Comput. Surv.*, vol. 50, no. 6, Dec. 2017, ISSN: 0360-0300. DOI: 10.1145/3124441. [Online]. Available: https://doi-org.tudelft.idm.oclc.org/10.1145/3124441.

[25] B. Alaya, L. Laouamer, and N. Msilini, "Homomorphic encryption systems statement: Trends and challenges," *Computer Science Review*, vol. 36, p. 100 235, 2020, ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2020.100235. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013719303429.

[26] M. Song, Y. Sang, Y. Zeng, and S. Luo, "Blockchain-based secure outsourcing of polynomial multiplication and its application in fully homomorphic encryption," *Security and Communication Networks*, 2021. DOI: https://doi.org/10.1155/2021/9962575.

Appendices

A Appendices

A.1 Selection of the Homomorphic Encryption Scheme

Numerous HE schemes are available, all with their own upsides and downsides. Therefore in this section the types of HE schemes will first be narrowed down based

on overall characteristics before going deeper into the specific HE schemes.

For the application of data aggregation, PHE is preferred. FHE is functionally ideal, however the low performance of FHE schemes makes them unusable for practical applications [1], [9], [13], [25], [26]. The SWHE group performs better, but still worse than the PHE group. Moreover, the frequency with which the functions can be applied in SWHE schemes is limited, which is impractical for data aggregation schemes where a large number of data points need to be aggregated which could possibly exceed this limitation. Meanwhile, PHE schemes can only perform either addition or multiplication, but this is enough for most data aggregation use cases. Furthermore, the function that the PHE can perform can be applied an unlimited amount of times meaning aggregation will not be limited by the encryption method. Therefore, PHE will be used in the data aggregation protocol.

Additive homomorphism is the preferred option for data aggregation over multiplicative homomorphism [1]. It is simply more likely that addition is required than multiplication. It is possible to convert certain multiplicative homomorphic schemes to be additive, but the decrease in performance [1] and infrequency of use cases makes this unnecessary. Therefore an additive homomorphic encryption scheme is used.

Then there is still the selection of the characteristics applicable to encryption schemes in general. For the purposes of this use case, the scheme should be asymmetric as this increases their overall security. This eliminates schemes like Hill. Another important requirement is the expansion rate of the encryption scheme. These requirements combined led to Paillier being chosen.

As mentioned, Paillier is a additive PHE scheme. It is also asymmetric. Its expansion rate is limited to around twice the bit-size of the public key [16] regardless of the plaintext size. The bit-size of the public key must be at least 2048 bits for adequate security [1], in the proposed protocol the key size of 4096 will be used based on the information given in [17] which showcases until which year an encryption method is deemed secure by different security organisations based on the bit size. Paillier is also the most well-known PHE scheme and in the literature study for this paper the only one used when a PHE scheme was required. However, Paillier is not perfect as it is not post-quantum safe. It has been attempted to make Paillier post-quantum safe [18], but this reduced the performance severly making it unusable.

A.2 Results of Performance Test

In this appendix all results from the performance test are shown. First all results for the individual steps will be shown. Then the results of the 100 cycles test are displayed.

Individual Steps

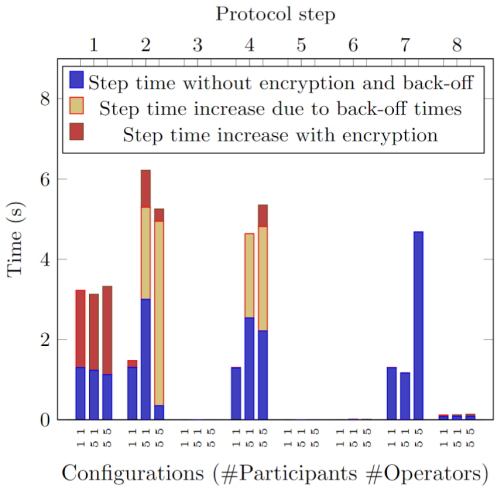

Figure 9: The individual steps of the protocol with different configurations of Participants and Operators.

Timing of 100 Cycles

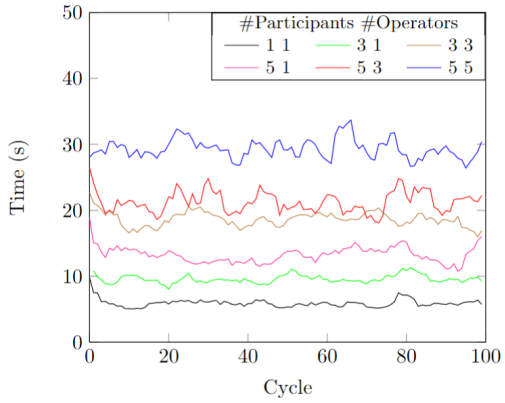

Figure 10: Time of a cycle of the protocol, run for 100 cycles with 10 second intervals. Different configurations were applied for Participants and Operators. A simple moving average was applied over five points.

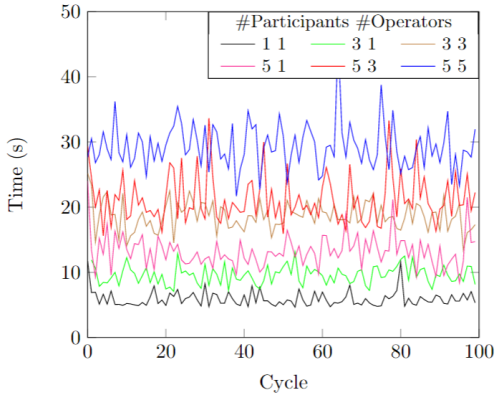

Figure 11: Time of a cycle of the protocol, run for 100 cycles with 10 second intervals. Different configurations were applied for Participants and Operators.

Back-off of 100 Cycles

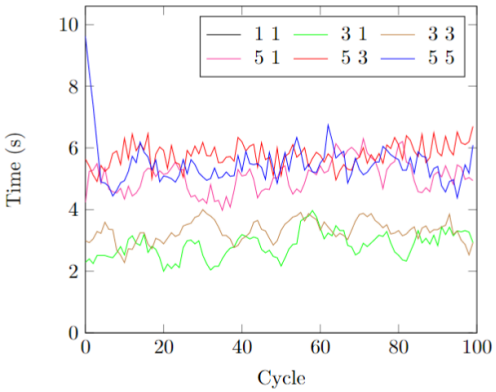

Figure 12: Back-off in a cycle of the protocol, run for 100 cycles with 10 second intervals. Different configurations were applied for Participants and Operators. A simple moving average was applied over five points.

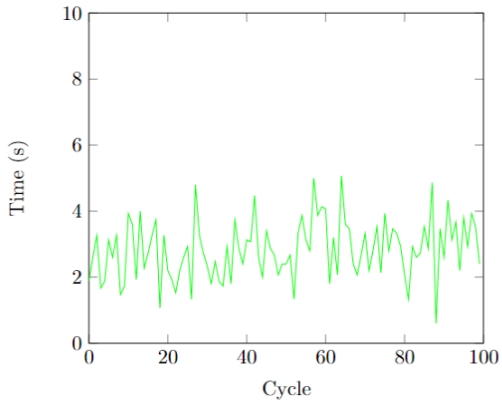

Figure 13: Back-off in a cycle of the protocol, run for 100 cycles with 10 second intervals. This is the configuration with 3 Participants and 1 Operator.

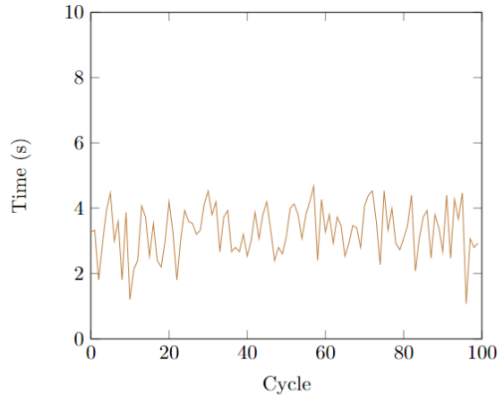

Figure 14: Back-off in a cycle of the protocol, run for 100 cycles with 10 second intervals. This is the configuration with 3 Participants and 3 Operator.

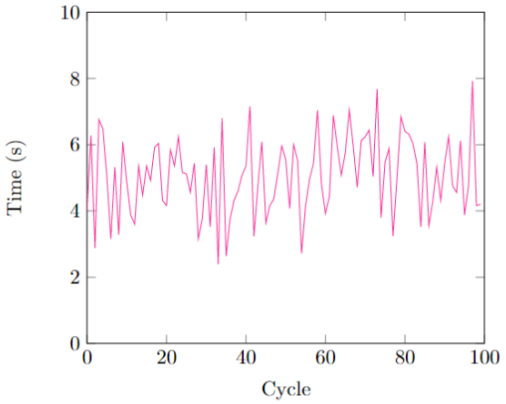

Figure 15: Back-off in a cycle of the protocol, run for 100 cycles with 10 second intervals. This is the configuration with 5 Participants and 1 Operator.

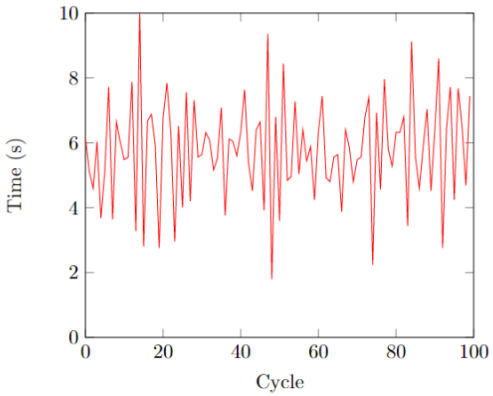

Figure 16: Back-off in a cycle of the protocol, run for 100 cycles with 10 second intervals. This is the configuration with 5 Participants and 3 Operator.

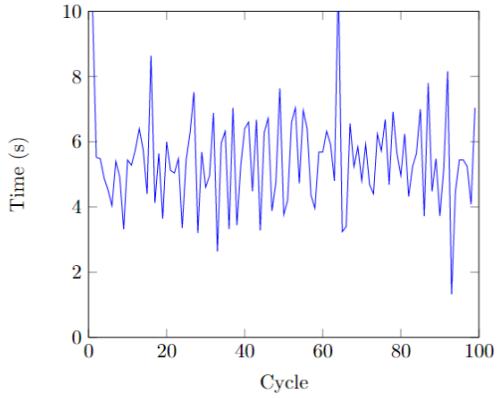

Figure 17: Back-off in a cycle of the protocol, run for 100 cycles with 10 second intervals. This is the configuration with 5 Participants and 5 Operator.

Time of 100 Cycles Without Encryption

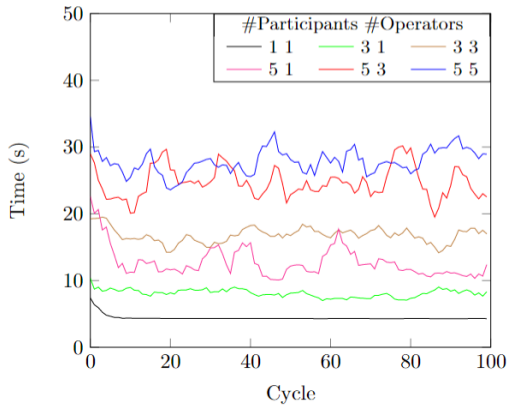

Figure 18: Time of a cycle of the protocol without encryption, run for 100 cycles with 10 second intervals. Different configurations were applied for Participants and Operators. A simple moving average was applied over five points.

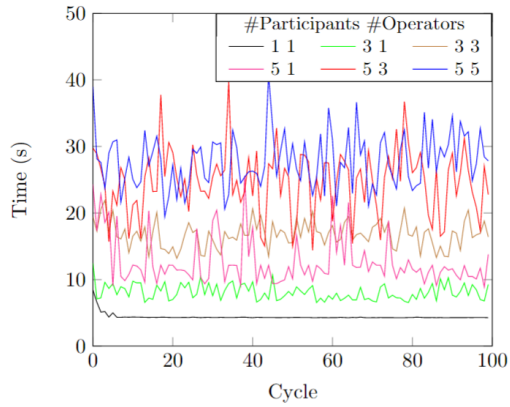

Figure 19: Time of a cycle of the protocol without encryption, run for 100 cycles with 10 second intervals. Different configurations were applied for Participants and Operators.

Back-off of 100 Cycles Without Encryption

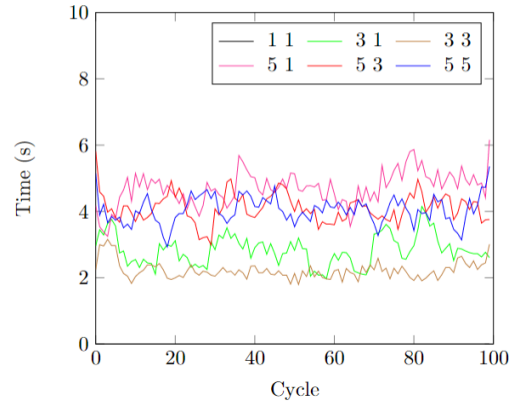

Figure 20: Back-off in a cycle of the protocol without encryption, run for 100 cycles with 10 second intervals. Different configurations were applied for Participants and Operators. A simple moving average was applied over five points.

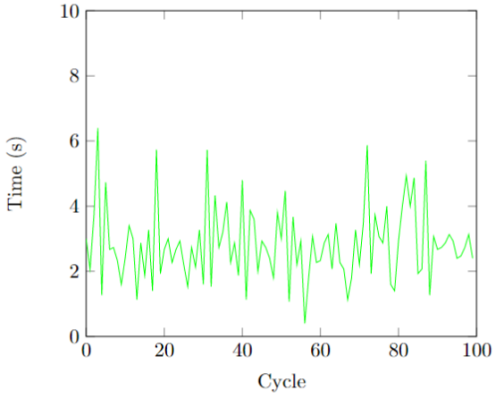

Figure 21: Back-off in a cycle of the protocol without encryption, run for 100 cycles with 10 second intervals. This is the configuration with 3 Participants and 1 Operator.

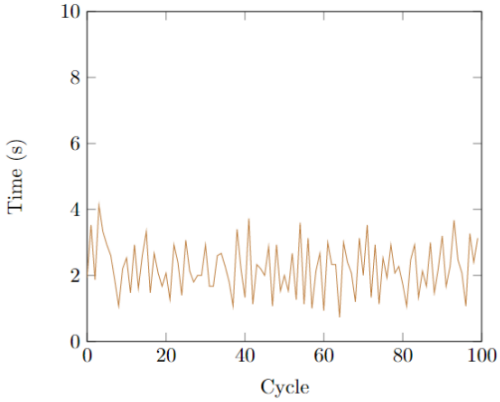

Figure 22: Back-off in a cycle of the protocol without encryption, run for 100 cycles with 10 second intervals. This is the configuration with 3 Participants and 3 Operator.

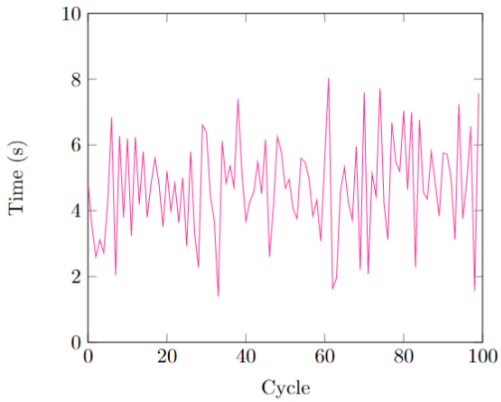

Figure 23: Back-off in a cycle of the protocol without encryption, run for 100 cycles with 10 second intervals. This is the configuration with 5 Participants and 1 Operator.

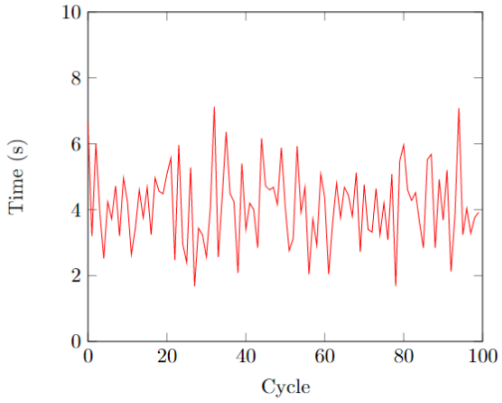

Figure 24: Back-off in a cycle of the protocol without encryption, run for 100 cycles with 10 second intervals. This is the configuration with 5 Participants and 3 Operator.

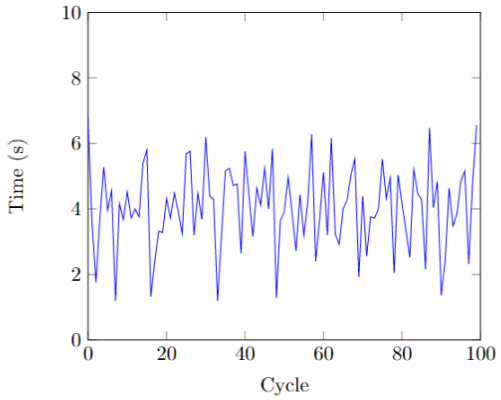

Figure 25: Back-off in a cycle of the protocol without encryption, run for 100 cycles with 10 second intervals. This is the configuration with 5 Participants and 5 Operator.