University of Zurich[UZH]

BACHELOR THESIS – Communication Systems Group, Prof. Dr. Burkhard Stiller

Design and Development of a Decentralized Access Control Tool for BloSS

Mervin Cheok
Zurich, Switzerland
Student ID: 07-194-392

Supervisor: Bruno Rodrigues, Dr. Thomas Bocek
Date of Submission: March 20, 2018

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Bachelor Thesis
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Das ist die Kurzfassung...

each 1 sentence

- intro problem, ddos bloss, problem of bloss, - this thesis solve the proble, - how. using, result are

Acknowledgments

I thank my supervisor Bruno Rodriques for his help and guidance through the BloSS/IPFS labyrinth and the continuous motivational words. They were much needed.

A big thank you also to Rowena Raths and my family for their everlasting support and the understanding of my constant absence.

I want to mention and thank Ms. Pearl La Marca-Ghaemmaghami for helping me focus on the important things.

Last but not least, a big shoutout to Andreas Gruhler and Lukas Eisenring.

Contents

Abstract i

Acknowledgments iii

1 Introduction 1

1.1 Description of Work . . . 2

1.1.1 Problem Description . . . 2

1.1.2 Objective . . . 2

1.2 Methodology . . . 2

1.3 Thesis Outline . . . 3

2 Background and Related Work 5

2.1 Distributed Denial of Service (DDoS) . . . 5

2.2 DOTS . . . 5

2.3 Blockchain . . . 6

2.4 InterPlanetary File System (IPFS) . . . 6

2.5 Related Work . . . 6

3 System Design 7

3.1 Requirements . . . 7

3.1.1 BloSS Data Sharing Scenario . . . 8

3.1.2 Technological Considerations . . . 10

3.2 Access Control for BloSS Data Sharing using IPFS . . . 10

3.3 Design . . . 11

4 Implementation **15**

4.1 BloSS Decentralized Application 15

4.1.1 Sender and Receiver App Scripts 15

4.1.2 Classes and their Functionality 16

4.1.3 Passwords and Key Store 18

4.1.4 Local Application Data 19

4.1.5 Libraries and Tools 19

4.2 Cryptographic Tools 20

4.3 Encountered Problems 21

5 Experimental Evaluation **23**

5.1 Test Setup 23

5.2 Test Run 24

5.3 File Encryption 25

6 Discussion **27**

6.1 Existing Tools and Concepts 27

6.2 Requirements 27

6.3 System Design and Implementation 28

6.3.1 Requirement Fulfillment 28

6.3.2 Smart Contract 29

6.4 Evaluation 29

7 Final Considerations **31**

Bibliography **33**

Abbreviations **37**

List of Figures **39**

List of Tables **41**

A Encryption Keys and Cipher Texts **43**

A.1 Sender Public Key 43

A.2 Plain IP Address List 44

A.3 Encrypted IP Address List 45

A.4 Encrypted File Key 46

B Installation Guidelines **47**

C Contents of the CD **49**

Chapter 1

Introduction

DDoS (Distributed Denial-of-Service) attacks have the simple goal of interrupting or suspending services available on the Internet and its motivations range from personal grudges over blackmail to political reasons [21]. A notable example is an attack conducted against Domain Name System (DNS) servers responsible for domains such as Twitter, PayPal, and Spotify [31] in October 2016. As a consequence, those services became unavailable to many US (United States) users for several hours. Besides the frequency, also the strength and growing duration of DDoS attacks increase their threat. One reason for the increasing volume of attacks is the availability of many weakly secured or configured IoT (Internet of Things) devices or home gateways [31].

The distributed nature of DDoS attacks suggests that a distributed mechanism is necessary for a successful defense. In this regard, coordinated protection efforts have become an attractive alternative to extend defense capabilities of a single system. Among existing alternatives, the blockchain technology offers an out-of-the-box solution that not only reduces the complexity of signaling DDoS attack information but could also provide means of establishing financial incentives, for cooperation at a reduced operational cost [24]. Blockchain and smart contracts [11] combined provide an out-of-the-box solution that can be used for signaling DDoS attacks information across multiple domains. Thus, existing approaches using gossip-based protocols can be simplified and remove the need to build specialized signaling protocols [26].

The Blockchain Signaling System (BloSS) [26] leverages blockchain capabilities to define individual agreements or conditions in a smart contract to perform a mitigation service in cooperative network defense. The signaling data is stored off-chain using the InterPlanetary File System (IPFS) [10, 23]. The IPFS link for the signaling data is stored in a smart contract. However, BloSS requires a security mechanism to manage the access to the addresses signaled in the blockchain, by using an off-chain access control mechanism.

Many concepts evolve around blockchain and private distributed data storage. Some concepts are making or made the transition to a usable system. None of the systems satisfies the needs of BloSS with IPFS. Encryption is the only mechanism available to control access with IPFS. Technological incompatibility or immaturity are the main reasons. Thus, a simple concept based on common encryption tools exchanging public keys via smart

contracts was designed and implemented. This simple system fulfills most of the BloSS access control requirements.

1.1 Description of Work

1.1.1 Problem Description

An autonomous system (AS) can join the cooperative defense system by creating a blockchain account and registering at a central smart contract. A central smart contract holds the management information of the defense system, such as IP networks of an AS, the address for the blockchain account of an AS and the address for the smart contract of an AS. Is an AS registered, it can retrieve the reported addresses of other ASes or can request help of other ASes by submitting IPs belonging to an attack. Has an AS received the newest information, it then can use common DDoS mitigation tools and techniques to react to an attack [1]. However, BloSS requires a security mechanism to manage the access to the addresses signaled in the blockchain, by using an off-chain access control mechanism.

1.1.2 Objective

The analysis of related works is an important factor to reveal which methods and tools can be implemented. The design and development stage consists in the modeling of a system capable of a decentralized authentication and authorization mechanism, using smart contracts and a decentralized application (dApp). Access control systems regulate the access to resources defining policies expressing the rights of subjects to access resources. Therefore, the main goal of this thesis is:

- **Design and develop and an off-chain mechanism to ensure that only authorized participants of a cooperative network defense can access data stored off-chain.**

Access control systems restrict access to critical or valuable resources defining policies expressing the rights of subjects to access resources. Therefore, this thesis formalizes and propose the design, operation, and evaluation of the access control mechanism. The output is a working prototype consisting of the integration of the designed and implemented elements into the whole system, producing results and material to be analyzed and contrasted with the previously specified requirements.

1.2 Methodology

Based on the description of work, the following research steps need to be accomplished:

- **Problem analysis and planning**: understanding of the BloSS requirements and analysis of storage platforms based on the previously listed requirements.
- **Requirements and System Design**: formalization or proposal of requirements to the selected platform to operate with BloSS.
- **Implementation**: design of an architecture and its components and development and integration with the BloSS system. Source code needs to be open source, well documented, and readable. Use appropriate testing practices to test the prototype.
- **Experimental Evaluation**: concerning the time required to process, store and fetch lists of IP addresses reported by BloSS in scale.
- **Report Writing**: involves the documentation process and motivation, background information, related work, design decisions, implementation details, evaluation, and conclusions.

1.3 Thesis Outline

Chapter 2 Background and Related Work acquires the needed knowledge about the used technologies such as blockchain, IPFS, access control and encryption. The chapter also gives an overview of the concepts and tools that can be used for private distributed storage.

Chapter 3 System Design gathers the requirements for the BloSS access control mechanism. Based on these requirements possible tools and concepts are evaluated. At last, a new system is designed using the newly gained information.

Chapter 4 Implementation provides the details on the developed proof of concept. The code architecture, important building blocks such as the encryption tools and the Ethereum and IPFS interfaces.

Chapter 5 Experimental Evaluation explains the used test setup for privte blockchain and the installed BloSS dApps and then elaborates the result of the test run.

Chapter 6 Discussion provides a critical examination about the design and the implementation of the proof of concept. Potential for improvement is shown.

Chapter 7 Final Considerations does a final wrapup of the work of this thesis and recommends next steps.

Chapter 2

Background and Related Work

2.1 Distributed Denial of Service (DDoS)

A Denial of Service (DoS) attack is a cyber attack, first occurred in the 1980's. A DoS attack aims to stop legitimate users from accessing a certain network resource. In 1999 the first Distributed Denial of Service (DDoS) attack had been reported and most of the DoS attacks from 1999 until today were of distributed nature [34]. One form of DoS attack over the Internet is to generate a massive amount of traffic to occupy all the resources, hindering them to provide service to legitimate traffic. This form of attack is difficult to prevent; targets are vulnerable just because they are connected to the internet. When the malicious traffic comes from multiple attacking entities, it is called a Distributed DoS attack. Using multiple attack entities amplifies the DDoS attack and makes the defense more complicated [22, 26]. The frequency of DDoS attacks is increasing. One reason for the rising volume of attacks is the availability of many weakly secured or configured Internet of Things (IoT) devices [31, 20].

The defense against a DDoS attack is severe. It proposes that a distributed mechanism is necessary or advantageous for a successful defense. Hence coordinated protection is becoming popular [24, 27]. Two example for coordinated DDoS signaling systems are DOTS and BloSS.

2.2 DOTS

The IETF is currently proposing a protocol called DOTS (DDoS Open Threat Signaling) covering both intra-organization and inter-organization communications to advertise attacks. The protocol requires servers and client DOTS agents, which can be organized in both centralized and distributed architectures to advertise black or whitelisted addresses. A DOTS client should register to a DOTS server in advance sending provision and capacity protection information and be advertised of attacks. Then, the DOTS protocol is used among the agents to facilitate and coordinate the DDoS protection service [**?**].

2.3 Blockchain

A blockchain system is a so-called distributed ledger. The ledger is replicated to a large number of identical databases. Different parties can host these. Happens a change in one of the copies, all the other copies are simultaneously updated. If a transaction occurs, it is permanently stored in all the ledgers. Communication happens directly between peers. No central node has to be involved. Every record is visible to anyone with access to the system. Once a record is stored in the blockchain, the record cannot be changed. Bitcoin is the first blockchain system [17]. In the meanwhile, there are other systems ie. Ethereum. Ethereum uses smart contracts [13]; these are contracts that can map executable logic onto the blockchain [12, 25]. A smart contract is a programmable account.

Storing data on a blockchain can be expensive. Off-chain storage is used for better scaling [30, 28]. The off-chain storage should be distributed as well.

2.4 InterPlanetary File System (IPFS)

The InterPlanetary File System (IPFS) is a distributed file system. IPFS is peer-to-peer and integrates ideas such as Distributed Hash Tables, BitTorrent, versioning like Git and the Self-Certified Filesystems (SFS). IPFS is a fully distributed system and has no single point of failure. The data of a node is stored locally. Nodes can connect to each other and transfer data [10].

2.5 Related Work

There are decentralized and distributed identity or key management concepts available. For example decentral PKIs [15, 18, 14] or Self-Sovereign Identity. Most of the concepts are not ready to use.

There are tools for sharing data using blockchain and off-chain storage. Unfortunately, there is no tool available yet that suits the needs of BloSS. After an inspection of possible tools, the conclusion is, that there are two camps: 1) of storage data sharing and 2) tools for secure off-chain data stores protecting privacy. BloSS needs a combination of the two. Some candidates that come near to what is searched are Enigma [35], Tahoe-LAFS [7], StoreJ [33], and Peergos [4].

Enigma should work with Bitcoin and Ethereum, but only Bitcoin applications were found. Tahoe-LAFS does not work with IPFS. StoreJ is distributed private storage that does not yet support sharing.

Chapter 3

System Design

3.1 Requirements

A system able to comply with the goals described in Chapter 1, has to consider requirements of aspects such as the BloSS scenario and its use cases, general security considerations as well as Blockchains with Smart Contracts and IPFS specific considerations. Taking these aspects, BloSS should account the following requirements:

Functional Requirements:

1. Control which entity can access a specific IP address list stored on the off-chain storage.
2. Share information between multiple parties.
3. Share different information types (file formats and file sizes).
4. Update the shared information timely.
5. Revoke access to shared information.
6. Ensure the availability of the shared data during an attack.
7. Ensure the integrity of the shared data.

Non-functional Requirements:

1. Consist only of parts that work in a distributed manner.
2. Use a private Ethereum Blockchain.
3. Use IPFS for off-chain storage.
4. Use Python 3.

5. Run on Ubuntu.

The list entries are ordered by the sequence they were acquired. The following thow Subsection guide through the development of the requirements for the Bloss Data Layer.

3.1.1 BloSS Data Sharing Scenario

For this work, the BloSS data transfer and sharing mechanism is in focus. For identifying the corresponding requirements, the BloSS scenario has to be described in detail and then examined.

BloSS is using a private Ethereum blockchain with smart contracts. All BloSS participants have to be registered in a smart contract. A smart contract is used for coordination and data is shared via IPFS.

A high-level consideration is that BloSS as a tool to fight DDoS attacks, should not be susceptible to such attacks. BloSS is using a blockchain, a distributed technology, to reduce the attack surface respectively to reduce the impact of a DDoS attack. If one system cannot respond anymore, the remaining systems can still proceed with their work. Therefore the used data sharing mechanism must comply with the property of a distributed system.

Non-functional requirement 1: **BloSS consist only of parts that work in a distributed manner.** Independent of a central management entity.

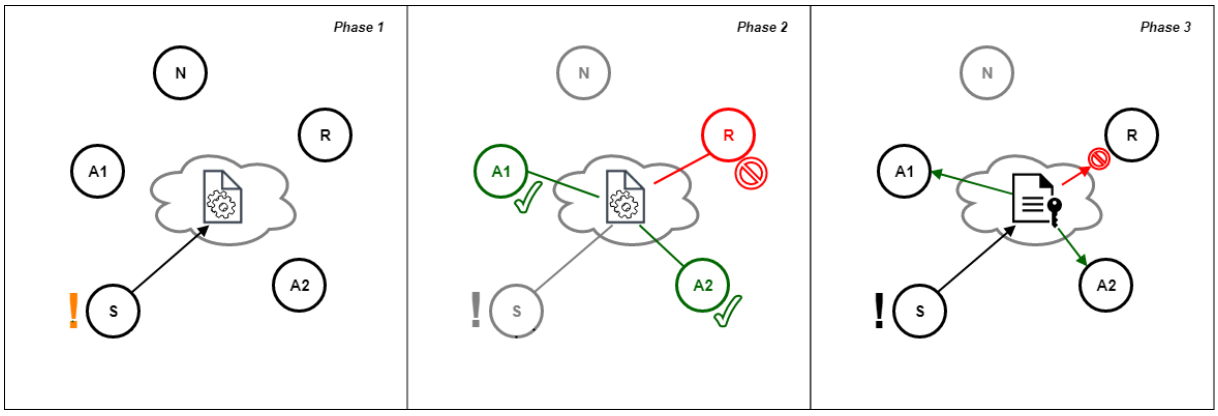

Figure 3.1: BloSS Data Sharing Scenario

Figure 3.1 visualizes the BloSS signaling and data sharing scenario. The circles are autonomous systems (ASs) representing a node in the BloSS network. “S” for sender is the AS that is attacked by a DDoS attack. “A<Nr.>” for accepted are the ASs whos attack mitigation offer has been accepted by the sender, these ASs are participating in the DDoS attack mitigation. “R” for rejected is the AS whos attack mitigation offer has been rejected by the sender. “N” for neutral is an uninvolved AS. The smart contract and the files stored on IPFS are placed in the “Cloud”. The scenario is split into three phases:

Phase 1: The Sender is under attack. It sends a help request in the form of a smart contract on the blockchain. Other ASs are checking the blockchain for requests and find the request from the Sender. The other ASs then can offer their help in the attack mitigation.

Phase 2: The three ASs A1, A2, and R decide to support the sender in the attack mitigation. They notify the sender by registering in the smart contract. AS neutral does not participate in the mitigation. The sender decides which ASs can help. The offer of A1 and A2 are accepted, the offer of R gets rejected.

Phase 3: The Sender processes the signaling data and stores it on IPFS. The corresponding IPFS address hash is then stored on the blockchain again. The other AS can retrieve the information from the blockchain, but only the approved AS will be able to read the information. A1 and A2 can access the signaling data. The Rejected and Neutral AS cannot access the information.

The scenario shows what is mentioned in Chapter 2. The IP information is sensitive. The attacked AS does not want full disclosure about the ongoing attack, therefore, it wants to share information as little as possible. Hence the first two functional requirements are:

Functional requirement 1: **BloSS can control which entity can access a specific IP address list stored on IPFS.**

Functional requirement 2: **BloSS can share information between multiple parties.**

In the scenario, the data shared with IPFS are IPv4 addresses in a text file. The exact format of this file is not defined yet. The question of how many files are used - one file per participating AS or only one file per attack mitigation or other possibilities - is not defined either. The IP information is the most important data that needs to be shared, but not the only one. According to V.Revuelto [32] incident reviews and information disclosure is part of the last stage of a DDoS response procedure. For reviewing and analysis purposes logfiles and reports must be exchanged between the involved AS. Thus the third functional requirement:

Functional requirement 3: **BloSS can share different information types(file formats and file sizes).**

DDoS attacks can last for hours, and the attacking entities can change during an attack. The IP addresses that have to be blocked can change during an attack. The involved AS have to be notified in a short period so that the mitigation targets the correct attacker as long as they are attacking.

Functional requirement 4: **BloSS can update the shared information timely.**

At a certain point of the mitigation, some involved AS might have to be excluded from the mitigation process. No further malicious request are recorded from the domain of an AS; there is no reason to provide that AS with more information. Review reports and analysis data might want to be shared for a specified time only. As a consequence, the fourth functional requirement is:

Functional requirement 5: **BloSS can revoke access to shared information.**

An additional consideration is that the shared data must be available at any time during an attack so that the participating AS can take mitigation actions. The data must be tamper proof as well so that an attack can not manipulate the defense coordination actions.

Functional requirement 6: **BloSS has to ensure the availability of the shared data during an attack.**

Functional requirement 7: **BloSS has to ensure the integrity of the shared data.**

The scenario can be summarized as such: BloSS is using a private blockchain with participating AS that share a similar goal (the uninterrupted provision of their services) but do not fully trust each other. The participants build a community of purpose to defend against DDoS attacks. In this context, the participants are willing to share data, if needed.

3.1.2 Technological Considerations

The same technological conditions as in the BloSS proof of concept from Rodrigues et al. [24] are applied for this work. The BloSS proof of concept is using a private Ethereum blockchain and IPFS for off-chain storage. The BloSS dApps are running on an Ubuntu server.

Non-functional requirement 1: **BloSS uses a private Ethereum Blockchain.**

Non-functional requirement 2: **BloSS uses IPFS for off chain storage.**

The BloSS concept dApp is written in Python 2 and uses web3.py to interact with the Ethereum blockchain and the Python IPFSAPI to interact with IPFS. Since the beginning of 2018 the web3.py module only supports Python 3. Thus the BloSS dApp components have to be upgraded to Python 3.

Non-functional requirement x: **BloSS uses Python 3.**

Non-functional requirement x: **BloSS runs on Ubuntu.**

3.2 Access Control for BloSS Data Sharing using IPFS

This section elaborates how the previous high-level requirements can be met and why the most appropriate solution is a simple public key encryption mechanism.

First, some vital information about the available building blocks. The Ethereum blockchain with smart contracts and IPFS are the base technologies. Both share the property that stored data is accessible by everyone. Plain data stored on IPFS or Ethereum is considered public data. Every party that gets hold of the address of a smart contract or the address hash of an IPFS resource can access this data directly. Even without address information, it is possible to find and access the data by exploring the blockchain or IPFS.

As mentioned in Chapter 2 IPFS offers no possibility to restrict access to resources except encryption. The IPFS specification mentioned built-in encryption functionality [10]. The API and ongoing discussions in the IPFS user group show that the encryption feature is not yet implemented [3]. For both IPFS and Ethereum sensitive data has to be encrypted by the user manually.

Combining the need for encryption with the requirement for a distributed system based on a blockchain and IPFS is the starting point for the search for suitable tools. Is there a distributed tool that allows sharing data and at the same time protect it from unauthorized access using encryption? The evaluation of possible tools concludes "No". Either there is a mismatch in technologies, the tool is not ready, or it contains a centralized component. Chapter 2 shows more information on this decision.

The lack of suitable tools forces the author to design a new concept. This concept builds on the findings of literature research focusing distributed identity and key management. Similar to the distributed private storage with sharing possibility, no suitable concept seems ready for use. Chapter 2 lists the examined concepts such as decentralized PKI, Self-Sovereign Identity or Attribute-Based Encryption. Although there are some promising ideas, the adaption and implementation for the BloSS scenario exceed the scope of this thesis.

Since building new secure applications is delicate - it is easy to neglect critical security details - the design should be as simple as possible. Symmetrical encryption allows encrypting bigger data file. The shared file is encrypted with a generated random file key using symmetrical encryption. The cipher file is then stored on IPFS. The file key is encrypted using public key encryption. Together with the IPFS hash, the encrypted file key is then stored in a smart contract. The public key encryption allows encrypting data for a group of parties so that BloSS can encrypt data once and share it with a group of participants. Every BloSS node has its public and private key pair.

3.3 Design

Due to the experimental state of BloSS and the growing IPFS ecosystem, the designed solution depends on the following assumptions:

- It is assumed, that the shared public keys in the smart contracts are legitimate AS known to and trusted by BloSS. It is not checked if a public key belongs to a trusted node. All keys from the contracts are trusted.

- It is assumed, that IPFS can provide a resource all the time. Since resources have to be provided by multiple nodes to be accessible during a DDoS attack. The incentives to improve resource availability is not part of this work.

The following sequence describes the steps depicted in Figure 3.2. The sender is the BloSS dApp that is attacked and asks other BloSS dApps called the receiver for support. This sequence depend on the starting situation, where the sender has created a smart contract (for simplicity named as the contract in the rest of the document) on the blockchain. The

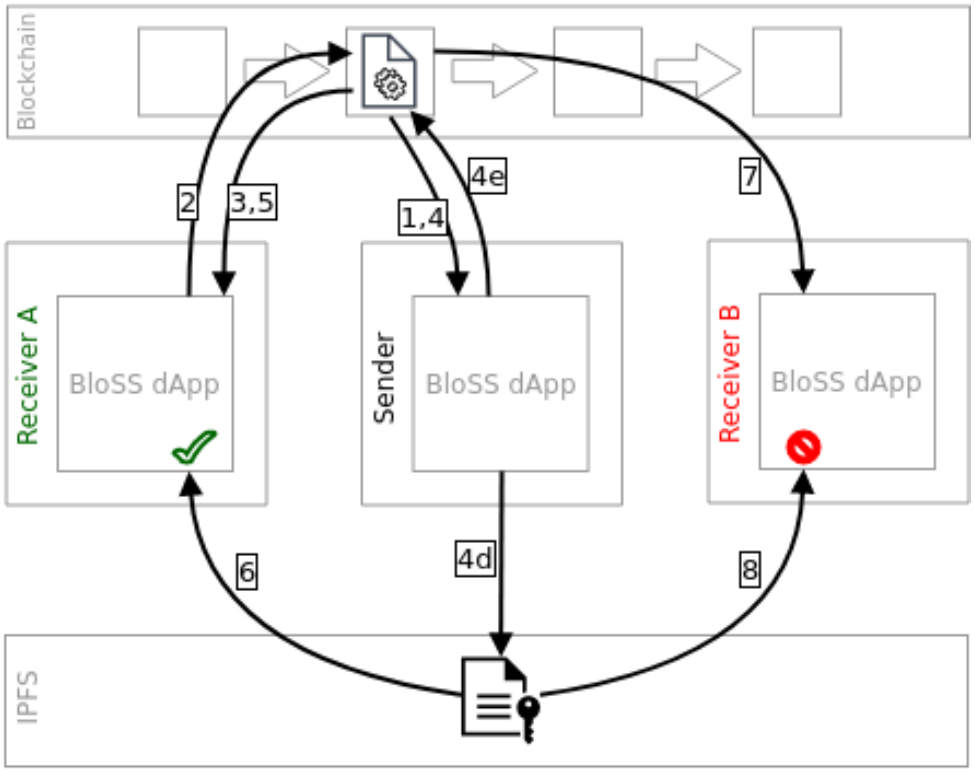

Figure 3.2: BloSS Sequence

address of that contract is known by all involved BloSS dApps. The receiver is known waiting for other BloSS dApp to provide their public key information.

1. The sender polls the public key information from the contract. Currently, the contract holds no keys and returns an empty set. The sender waits for a defined interval until it starts the next polling request.

2. Receiver A (for accepted) registers his public key information in the contract.

3. Afther receiver A has successfully registered his public key information, the receiver transitions into the polling phase. In the polling phase, the receiver checks the contract for new IPFS information. Since currently there is no IPFS information, the receiver waits for the next poll.

4. The sender polls for the public key information once more. This time he finds the public key from receiver A. The sender uses this information to encrypt his ip address list and to store the encrypted list on ipfs. The sender puts the decryption information and the IPFS address in the contract.

   (a) The sender creates a new file key (the sender generates a random key).

   (b) The new file key is used to encrypt the current ip address list.

   (c) The file key will be encrypted using the public keys of the receivers.

   (d) The encrypted ip addresses are than stored on IPFS.

   (e) The IPFS address hash and the encrypted file key are stored in the contract.

5. Receiver A polls again for new IPFS information. The receiver finds the new IPFS address hash and the encrypted file key.

6. With this information the receiver can get the encrypted ip address list from IPFS and use the file key to decrypt the ip address list.

   (a) The receivers gets the IP address list from IPFS.

   (b) The receivers uses its own private key to decrypt the file key from the contract.

   (c) Using the plain file key, receiver A can decrypt the IP address list.

   (d) With the plain IP address list, receiver A can start with possible attack mitigation actions.

7. Receiver B (for blocked) polls for IPFS information. The receiver finds the IPFS and the encrypted file key information.

8. With this information, receiver B can geht the encrypted IP address list but is not able to decrypt the file, respectively is not able to decrypt the file key without the IP address list can not be decrypted.

This sequence assumes, that the contract address is provisioned beforehand. This could change and a mechanism for identifying the correct contracts on the blockchain could be implemented in a following step.

The decision of which BloSS DApp is trusted and accepted for the DDoS mitigation is pre defined for this scenario. Logic to make this decision dynamicly in real time can be added in a next iteration. The decision making logic could for example use a reputation system as information basis.

Chapter 4

Implementation

4.1 BloSS Decentralized Application

Altough there is no software library or tool for the BloSS functionality, the code of the BloSS prototype setup was available and could be used to understand the concept on how to interact with Ethereum and IPFS. Based on this information and code snippets it was possible to build Ethereum and IPFS adapters. The implementation effort is spent on file handling, the cryptographic functionality and the public key exchange. Other BloSS functionalities are mocked if needed or omitted if not needed at all.

Figure 4.1 shows the used python files. Elements marked as "script" are Python scripts, the other elements represent classes. Two scripts sender_app.py and recevier_app.py are the two entry points and are used to simulate the BloSS dApp behaviour, using the provided BloSS dApp functionality. Together the two scripts emulate the in Section **??** proposed procedute.

4.1.1 Sender and Receiver App Scripts

The sender_app script simulates the actions of an attacked AS, which are the polling for public keys of participating AS's and the encryption and provisioning of the IP address list. The receiver_app script simulates the actions of an AS that participates in a mittigation process, which are the publishing of its own public key, the polling for new information from the smart contract, the fetching of the IP address lists from IPFS and the encryption of that list.

Each script has a corresponding class object encapsulating the needet BloSS functionality. The Sender class provides the functionality for the sender_app script and the Receiver class provides the functionality for the receiver_app script. The scripts gets the path to a node configuration file passed as argument. On startup, the script loads the configuration file, that can be used to create the Sender or Receiver class.

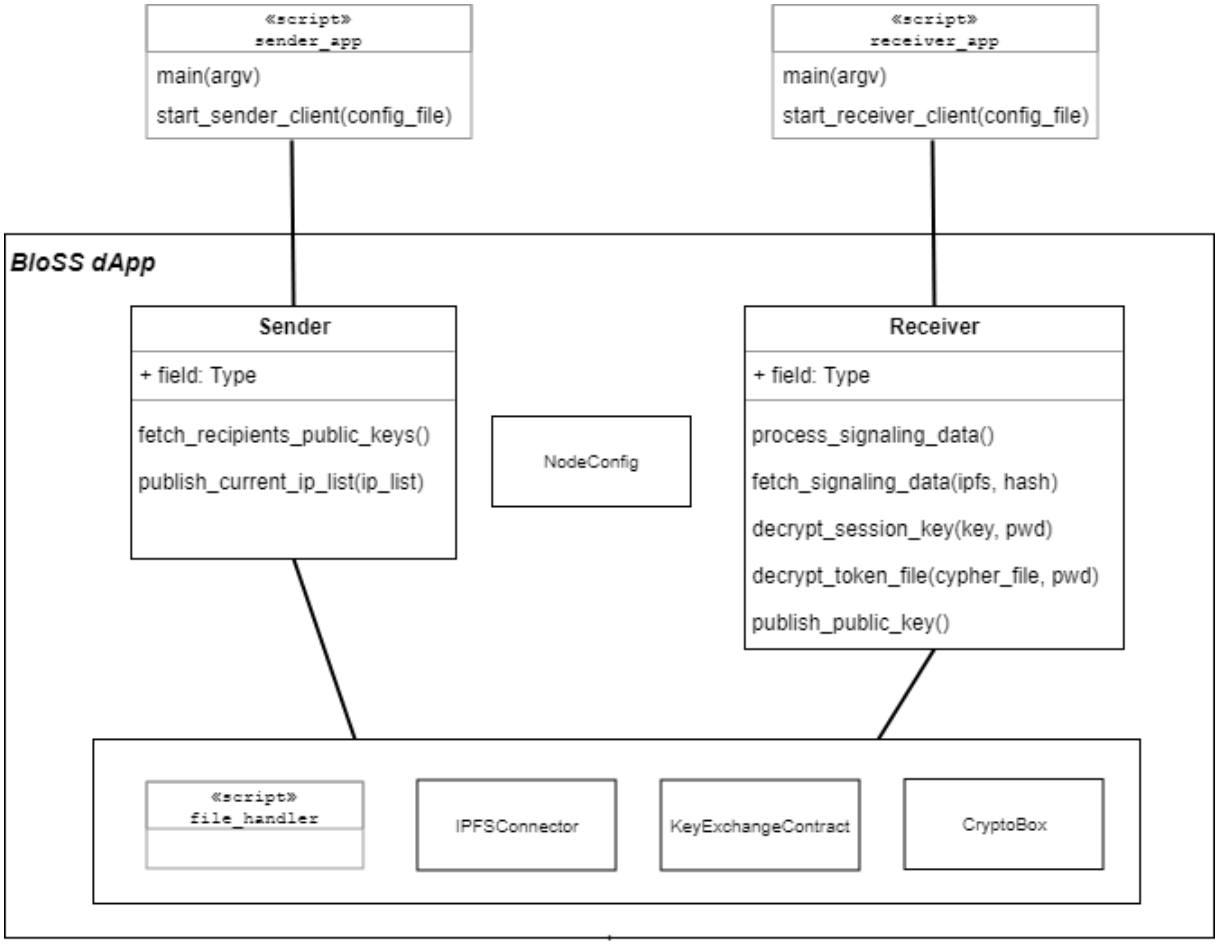

Figure 4.1: BloSS dApp Components and Classes

On startup, the scripts also load and configure the Python logging module. This module provides a flexible event logging system that allows to record the activities of the BloSS dApp. The scripts log to the standard output console, but logging to a file or database could be configured simply by adujsting the config definition in the two scripts.

The scripts exit with exit code “0” if the execution was error less. Should an error occure, the script prints the corresponding error message and exits with code “10”. All exceptions and errors that could occure during execution are caught in the scripts.

4.1.2 Classes and their Functionality

This Subsection describes the used class objects and their functionality of the BloSS dApp also shown in Figure 4.1. When ever information should be logged, the Python logging framework is used. Every class defines its own logger.

Sender: The Sender class gathers all needet functionality that is used for an AS that initializes an attack mittigation. The Sender class is used by the sender_app script. During the initialization of this object, the connection to the Ethereum and IPFS

client is established. Can the needed connections not be established, the initilization fails with en exception.

Methods:

fetch_recipients_public_keys(): Gets the published public keys of participating ASs from the KeyExchangeContract.

publish_current_ip_list(ip_list_path): Encrypts the current IP address list with the public keys of the participating ASs, stores the encrypted list on IPFS and publishes the IPFS hash and the file key in the KeyExchangeContract.

Receiver: Analog to the Sender class, this Receiver class gathers the needet functionality for an AS that wants to participate in an ongoing mittigation. The Receiver class is used by the receiver_app script. Same as with the Sender initialization process, the Receiver tries to establish connection to Ethereum and IPFS and the initialization fails if the connections cannot be established.

Methods:

process_signaling_data(): Checks the KeyExchangeContract for new IPFS hashes. If new hashes are found, the files are fetched from IPFS and decrypted.

fetch_signaling_data(): Get the corresponding file for a hash from IPFS and saves the file on the local file system.

decrypt_session_key(message, pwd): Decrypt a cypher message using the CryptoBox object.

decrypt_token_file(file_path, pwd): Decrypt a file with encrypted content using the CryptoBox object.

publis_public_key(): Store this nodes public key to the KeyExchangeContract.

EthereumConsole: The EthereumConsole class represents the Ethereum interface for BloSS. This class holds a connected instance of the Ethereum web3.py api. At creation, the EthereumConsole takes an Ethereum RPC node host address and the corresponding port number. An exception is thorwn, if web3.py can not connect to the given host and port combination.

Methods:

unlock_account(account, pwd): This method unlocks the given account on the connected Ethereum node.

load_key_exchange_contract_for_account(address, account): This method creates a connected KeyExchangeContract object for the given contract address and the account that should be used to interact with the contract.

KeyExchangeContract: The KeyExchangeContract class represents the Solidity Smart Contract for the BloSS key exchange. This class implements all the method calls to the Smart Contract. The class knows the location of the corresponding contract ABI JSON file.

CryptoBox: All the cryptographic functionality is encapsulated in the CryptoBox class (symetric encryption with Fernet and asymetric encryption with GnuPG). When creating an new instance of this class, the GnuPG home directory has to be declared. The public keys of the participating AS are imported by the CryptoBox into GnuPG.

Methods:

import_trusted_recipients(trusted_recipients): Imports a list of public keys into the GnuPG key ring. Returns a list of fingerprints of the imported public keys.

encrypt_plain_file_for_recipients(file_path, trusted_recipients): Encrypts a plain text file for the given recipients. This is done by generating a new file key that is used to encrypt the given file with AES 128. The generated file key is than encrypted for the public keys of the trusted recipients using GnuPG's encryption function.

gpg_decrypt_message(messge, pwd): Decrypts a given cypher text with GnuPGs private key.

decrypt_token(file_path, pwd): Decrypt the given encrypted file with the given key.

IPFSConnector: Encapsulation for the IPFS API library. During the initialization of the connector a connection to IPFS must be possible, otherwise the initialization fails. The connector can add a file to IPFS and get the corresponding content for an IPFS address hash.

Methods:

get(adr_hash): Gets a file from IPFS.

add(file_path): Adds a file to IPFS.

NodeConfig: Utility class for holding the different configuration parameters for the dApp, GnuPG, Ethereum and IPFS. Uses the Python configparser to handle INI configuration text files.

file_handler: This file gathers utility functions for working with the local file system.

Methods:

init_node_home(): Creates a new home directory for the dApp, including the gpg_home and the data directory.

save_content_to_file(file_path, file_content): Saves the content localy to the given path.

4.1.3 Passwords and Key Store

The BloSS dApp must have access to account and password information to connect to Ethereum and IPFS. The dApp also needs write access to the GnuPG key store. Authentication data is sensible and requires careful treatment. The data should be accessible only by legitimate entities. The combination of these three tools leave two options to handle the authentication data: 1) enter the data manually at application startup or 2)

load the data from a protected source. For the proof of work setup in this work, both options would be viable. There are only a few nodes that need to be managed, and the uptime of the nodes is not critical. Considering a more realistic deployment where a more significant number of nodes needs to be managed and operated, option 2 is more suitable.

The BloSS dApp configuration file stores the authentication information. The configuration holds the Ethereum account and password, the IPFS account and password as well as the password for the GnuPG private key. To restrict access to this data, the correct OS access rights for the configuration file have to be set. Only the OS user used for the BloSS dApp (the configuration file owner) shall have access. The Linux file access rights are set to "-rwx—— / 0700" which means that only the owner of the file can read, write, and execute it. A file protected like this does not guarantee as failsafe protection. A user with administrator privileges will still have access to the file. If the OS is compromised, access to the data could be gained.

4.1.4 Local Application Data

The local BloSS installation has a home directory with the following structure:

```
/NodeHome
    /config   - Configuration
    /data     - Application data such as IP address lists.
    /dApp     - Python scripts
    /gpg_home - GnuPG home directory
```

As in the previous section mentioned, the gpg_home directory access rights have to be set to "-rwx—— / 0700" to ensure that only the owner can use the configuration data. Since the data directory will contain the IP address lists or other sensitive report data and the Python scripts have to be protected from modification by third parties it is recommended, to restrict access for the NodeHome directory.

4.1.5 Libraries and Tools

The used Python modules that had to be installed additionally are:

web3 The Python Ethereum console implementation.

ipfsapi The Python IPFS API implementation.

cryptography Pythons crypto library, used for the symmetric encryption.

gnupg The Python wrapper for GnuPG.

Further, GnuPG has to be installed on the local Ubuntu machine.

4.2 Cryptographic Tools

Pythons ”cryptography” library is used to encrypt the IP address list file with a random key. This random key will be encrypted with GnuPG public-key encryption functionality. To interact with GnuPG via Python, ”python-gnupg” a Python wrapper for GnuPG is used. All these tools are free or open source software (FOSS) for Python 3.

The tools were selected after an evaluation with the following criteria: 1) the tool must support Python 3, 2) it must be free or open source, 3) it is in active development, 4) it is used by other projects and 5) it runs on Ubuntu, 6) it has undergone security reviews with public reports.

The security vulnerability portal cvedetail.com [1] is used to check the tools for known security flaws. According to CVE Detail, the selected tools have no severe issues or flaws. Some minor security vulnerabilities are known, but none tangent the field of application of this work. This does not prove, that there are no vulnerabilities, but it shows that the tools at least undergo some reviews.

PyCrypto is a collection of hash functions and encryption functions. The library is in version 2.6.1 and supports Python version 2.1 to 3.3. PyCrypto is an often used library; the development is stopped. The last commit for this project is from June 2014 [19].

PyCryptodome is a PyCrypto active developed fork. It supports the same functionality as its original and Python version 2.6 and all Python 3 versions. The newest version 3.5.0 dates back to March 2018. PyCryptodome is partially under public domain and partially under BSD 2 released [6].

Cryptographic is a Python package providing cryptographic recipes and primitives for Python developers. The developers of aim to make it the cryptographic standard library for Python. Supported Python versions are 2.7 and 3.4+. Supported cryptographic algorithms are symmetric ciphers, message digests, and key derivation functions. The projects latest stable release 2.1.4 is from the 29.11.2017. Used licenses are the Apache Software License, version 2.0 and the BSD license [5].

Deadlock is file encryption tool after miniLock in and for Python 3. Contrary to miniLock, Deadlock is not audited and no longer in development. The last commit of the project was in August 2014. The tool is published under the GNU Affero General Public License [16].

GnuPG is a free implementation of the OpenPGP standard. Its primary purpose is the end-to-end encryption of email communication; it lets encrypt and sign data and communication in general. It also features versatile key management. In 2017 GnuPG is around for 20 years. The newest version is released at the 22.02.2018. GnuPG is released under the GNU General Public Licence [2]. ***Python-GnuPG*** is a GnuPG wrapper for Python. The latest version is 0.4.1 dating from 15.11.2017 and supports GnuPG version 2.1 and later [29].

4.3 Encountered Problems

The public key exchange via smart contract encountered an encoding problem during the development process. Some nodes could not decode the public key from the contract correctly. Other nodes could decode as expected. All the affected nodes were located on the same physical machine (Desktop 2). After the reinstallation of the machine, the encoding problem has vanished. It is not clear why this problem occurred. Further investigation is recommended.

Chapter 5

Experimental Evaluation

This Chapter shows the verification procedure for the design described in Chapter 3. First comes an explanation of the test setup, followed by the data gathered during a test run.

5.1 Test Setup

The test setup uses a local network consisting of three computers. These three computers build a private Ethereum blockchain with five nodes. Two dedicated nodes for mining and three nodes for the BloSS dApp: one Sender dApp that provides the IP address list, one dApp that is accepted by the Sender to participate in the attack mitigation and one dApp that is rejected by the Sender. The three computers are two more powerful desktop machines and one notebook. Two Ethereum nodes are deployed on each of the machines with better performance, one mining node, and one dApp node. The notebook hosts only one dApp. The described setup is depicted in Table 5.1. Miner nodes run with port 30303 and RPC port 8545. DApp nodes run under port 30304 and RPC port 8546.

For the setup, all dApps use the same source code version. The Rejected Receiver does not publish a public key to ensure that the Sender encrypts the data for the accepted receiver. Each dApp has a new GnuPG directory and a freshly generated key pair. A new Key Exchange Contract is created on the blockchain, and the address is provisioned to all three dApps.

Node	**Node_11**	**Node_12**	**Node_21**	**Node_22**	**Node_33**
Purpose	Miner	Sender dApp	Miner	Accepted dApp	Rejected dApp
Port	30303	30304	30303	30304	30304
RPC Port	8545	8546	8545	8546	8546
Network	12344	12344	12344	12344	12344
Machine	Desktop 1	Desktop 1	Desktop 2	Desktop 2	Notebook
IP Address	192.168.1.100	192.168.1.100	192.168.1.101	192.168.1.101	192.168.1.102

Table 5.1: Ethereum Node Setup

5.2 Test Run

A test run is initiated by starting the three dApps (Sender, Accepted Receiver, and Rejected Receiver) manually.

1. The Accepted Receiver publishes his public key to the key exchange contract. The public key is accessible in the Appendix A.1.

2. The Sender gets the Receivers public key from the contract and starts the IP list encryption.

   (a) The sender generates a random file encryption key.

   ```
   oAX5IC1vFddSuKlGRd4ascGFQaHcFMnETxAAQiSKCjU=
   ```

   (b) A dummy IP list file gets encrypted with the file encryption key. The plain file is accessible in Appendix A.2 and the encrypted list is accessible in the Appendix A.3.

   (c) The sender stores the encrypted IP list on IPFS and gets an IPFS hash in return.

   ```
   QmRnCZ4GE8dJjBe9HQCVksUxrrRDtCbwjzxGVGbi8h2FNi
   ```

   (d) The file encryption key is encrypted using the Receivers public key. The encrypted file key is accessible in the Appendix A.4.

   (e) The IPFS hash and the encrypted file key are stored on the key exchange contract.

3. The Accepted Receiver finds the new IPFS hash and the encrypted file key.

   (a) The Accepted Receiver gets the corresponding file for the hash from IPFS. The Receiver gets the encrypted IP list form Appendix A.3.

   (b) The encrypted file key can be decrypted using the Receivers own private key. The receiver holds now the plain file encryption key.

   ```
   oAX5IC1vFddSuKlGRd4ascGFQaHcFMnETxAAQiSKCjU=
   ```

   (c) With the plain file key, the receiver can decrypt the encrypted IP list. The receiver holds know the plain IP list from Appendix A.2.

4. The Rejected Receiver finds the new IPFS hash and the encrypted key.

   (a) The Encrypted Receiver gets the corresponding file for the hash from IPFS. The Receiver gets the encrypted IP list form Appendix A.3.

   (b) The encrypted file key cannot be decrypted using the Rejected Receivers own private key. The decryption call fails with an error message:

   ```
   Could not decrypt session key. Abort data update process.
   ```

   The GnuGP decryption attempt fails with following error message:

   ```
   gpg: encrypted with RSA key, ID 4C4B9316
   gpg: decryption failed: secret key not available
   ```

5. The Sender could decrypt the encrypted file key with his private key if needed.

5.3 File Encryption

BloSS must share data in a reasonable time so that the participating AS can react timely. Otherwise, the mitigation efforts could be in vain, since they target obsolete attackers.

The symmetric encryption with fernet was tested with different file sizes:

- 20 IPv4 addresses in JSON format (< 1kB)
- 1000 IPv4 addresses in JSON format - 32 kB
- 10 MB
- 100 MB

All of the files could be encrypted within under 1 second. The goal is to handle 10MB files.

Actions like adding data to IPFS and the transaction with a smart contract take much more time than the encryption.

Chapter 6

Discussion

This Chapter challenges this thesis and illustrates the positive as well as the negative aspects of this systems design and its implementation.

After a brief roundup of the existing tools and concepts for this domain, the acquired requirements get reviewed by comparing to other requirements of a related signaling system. The system design and implementation are then checked against this works requirements, followed by a discussion about the used evaluation method. Last, some thoughts about the general approach of this thesis.

6.1 Existing Tools and Concepts

The development of use cases and tools for blockchain in combination with a distributed off-chain storage is very lively. As addressed in Chapter 2 the development of concepts and tools for sharing private data progresses. Due tue the young but broad field of research, it was difficult to discover compatible tools for an application with the BloSS technology stack (Ethereum and IPFS). Hence the decision to design a new system. The new system should be as simple as possible.

6.2 Requirements

The new requirements mainly rest upon a high-level scenario with some additional technical considerations. The resulting requirement s are correspondingly high-level. A cross-comparison with the DOTS data channel requirements highlights this [8]. Although the requirements head in the same direction and cover similar ground, the DOTS requirements are much more specific. The security aspect shows this discrepancy very noticeable. To mitigate this imprecision, some requirements have been revised during the design phase.

6.3 System Design and Implementation

6.3.1 Requirement Fulfillment

The goal was to design a simple system that uses common, proven and easy to use tools but satisfies as many requirements as possible. The proof of concept implementation should cover all the requirements but in a most minimal way. Therefore the system is validated against all requirements:

1. **Control which entity can access a certain IP address list stored on the off-chain storage.**
   Fulfilled, encryption is used for access control. Public key encryption allows encrypting the file key for a single participant or a group. Although everyone can get the encrypted file from IPFS, it is useless without the file encryption key.

2. **Share information between multiple parties.**
   Fulfilled, a sender can create multiple smart contracts, each contract for a participating receiver. Another way would be one smart contract that can hold multiple public keys.

3. **Share different information types(file formats and file sizes).**
   Fulfilled, the encryption mechanism can handle different file formats and works tested with file sizes until 100MB.

4. **Update the shared information timely.**
   Partially fulfilled, the symmetric encryption worked fast with encryption times for under 1 Second for files up to 100MB. New information can be encrypted for the known participants, and the new encrypted data can be stored on IPFS again. The contract gets updated with the new IPFS hash. A bottleneck in the process is storing information on IPFS. This aspect was neglected in this work. A task for future work.

5. **Revoke access to shared information.**
   Partially fulfilled, the sender can decide for which known participants the new information should be encrypted. A party that got access to the old data must not necessarily get access to the updated data. However, the access to the old data cannot be revoked. This is categorized as a minor issue. The participant had access to the data before and could have stored it locally already.

6. **Ensure the availability of the shared data during an attack.**
   Theoretically fulfilled, IPFS and Ethereums distributed nature ensures that the data is available, even if a node stops operating.

7. **Enusre the integrity of the shared data.**
   Theoretically fulfilled, IPFS and Ethereums storage should be tamper proof.

The above statements assume 1) that IPFS does distribute stored files in its network so that the file can be accessed even if one node dies. 2) The statements assume that Ethereum guarantees the integrity of stored data.

6.3.2 Smart Contract

The key exchange contract consists mainly of getter and setters for the public key, file encryption key, and the IPFS hash. Due to its simplicity, it offers little attack surface. Currently, the contract can handle only one public key. The contract could be upgraded to handle multiple keys and store them together with the wallet address of the corresponding Ethereum account. This upgrade would make the interaction between the participants even simpler.

Smart contracts security vulnerability are explored very little. Still, some best programming practices and attacks are known [9]. This information should be applied to the key exchange contract.

6.4 Evaluation

The experimental evaluation covers only the essential function of the concept. Aspects such as transaction costs, IPFS performance and distribution should have been part of the evaluation as well.

Chapter 7

Final Considerations

This chapter gives a summary of this thesis work and findings. Recommendations for next steps follow on the wrapup.

BloSS needs a decentralized access mechanism to control the access for data on the off-chain storage. After literature research and evaluating new concepts and tools, no satisfying candidate was found. In consequence, a simple system based on common and proven tools was designed and implemented. The evaluation of that proof of concept showed that the combination of this simple tools could indeed satisfy the basic BloSS access control needs.

Recommendations for the next upgrade steps for the BloSS access mechanism are: 1) use Ethereum accounts linked to published public keys to decide which AS is a trusted receipient. 2) Apply best practices for smart contract programming and check the known vulnerabilities. 3) Create a mechanism to handle smart contracts with the application. Currently, only one configured contract is in use. To handle multiple contracts and to identify them on the chain would be a useful extension.

The blockchain ecosystems undergo rapid expansion. The development pace of concepts and system is fast. During this work, no suitable system was found for the BloSS access mechanism. This can change quickly in such a thriving ecosystem.

Bibliography

[1] CVE Details. https://www.cvedetails.com/. Accessed: 17.03.2018.

[2] GnuPG. https://www.gnupg.org. Accessed: 09.03.2018.

[3] IPFS FAQ. https://discuss.ipfs.io/t/file-encryption-built-into-ipfs/323. Accessed: 18.03.2018.

[4] Peergos, an end-to-end encrypted, peer-to-peer file storage, sharing and communication network. https://peergos.org/. Acessed: 19.03.2018.

[5] pyca/cryptography. https://cryptography.io, . Accessed: 09.03.2018.

[6] Pycryptodome. https://www.pycryptodome.org, . Accessed: 10.03.2018.

[7] Tahoe-lafs documentation. http://tahoe-lafs.readthedocs.io/en/latest/. Acessed: 19.03.2018.

[8] T. R. A. Mortensen, R. Moskowitz. Distributed denial of service (ddos) open threat signaling requirements. https://tools.ietf.org/html/draft-ietf-dots-requirements-14, Feb. 2008. Version: 14.

[9] N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on ethereum smart contracts (sok). In *International Conference on Principles of Security and Trust*, pages 164–186. Springer, 2017.

[10] J. Benet. Ipfs - content addressed, versioned, p2p file system. jul 2014.

[11] T. Bocek and B. Stiller. Smart Contracts - Blockchains in the Wings. In C. Linnhoff-Popien, R. Schneider, and M. Zaddach, editors, *Digital Marketplaces Unleashed*, pages 1–16. Springer, Berlin, Heidelberg, Germany, 2017. URL https://goo.gl/M6LWrH.

[12] V. Brühl. Bitcoins, blockchain und distributed ledgers. *Wirtschaftsdienst*, 97(2): 135–142, 2017.

[13] V. Buterin. Ethereum: a next generation smart contract and decentralized application platform (2013). *URL {http://ethereum. org/ethereum. html}*, 2017.

[14] V. B. J. C. D. D. C. L. P. K. J. N. D. R. M. S. G. S. N. T. Christopher Allen, Arthur Brock and H. T. Wood. Decentralized public key infrastructure. Technical report, Web of Trust, 2015.

[15] C. Fromknecht, D. Velicanu, and S. Yakoubov. A decentralized public key infrastructure with identity retention. *IACR Cryptology ePrint Archive*, 2014:803, 2014.

[16] C. Garvey. Deadlock on github. `https://github.com/cathalgarvey/deadlock`. Accessed: 09.03.2018.

[17] M. Iansiti and K. R. Lakhani. The truth about blockchain. *Harvard Business Review*, 95(1):118–127, 2017.

[18] K. Lewison and F. Corella. Backing rich credentials with a blockchain pki. Technical report, Tech. Rep, 2016.

[19] D. Litzenberger. Pycrypto on github. `https://github.com/dlitz/pycrypto`. Accessed: 09.03.2018.

[20] S. Mannhart, B. Rodrigues, E. Scheid, S. S. Kanhere, and B. Stiller. Toward Mitigation-as-a-Service in Cooperative Network Defenses. 3001forthcoming.

[21] S. Mansfield-Devine. The Growth and Evolution of DDoS. *Network Security*, (10): pp. 13–20, 2015.

[22] T. Peng, C. Leckie, and K. Ramamohanarao. Survey of network-based defense mechanisms countering the dos and ddos problems. *ACM Computing Surveys (CSUR)*, 39(1):3, 2007.

[23] B. Rodrigues, T. Bocek, D. Hausheer, A. Lareida, S. Rafati, and B. Stiller. Blockchain-based Architecture for Collaborative DDoS Mitigation using Smart Contracts. In *ForDigital Blockchain*, pages 1–4, Karlsruhe, Germany, feb 2017. KIT. URL `https://files.ifi.uzh.ch/CSG/staff/rodrigues/extern/publications/cooperativeblockchainddos.pdf`.

[24] B. Rodrigues, T. Bocek, A. Lareida, D. Hausheer, S. Rafati, and B. Stiller. *A Blockchain-Based Architecture for Collaborative DDoS Mitigation with Smart Contracts*, pages 16–29. Springer International Publishing, Cham, 2017. ISBN 978-3-319-60774-0. doi: 10.1007/978-3-319-60774-0_2. URL `https://doi.org/10.1007/978-3-319-60774-0_2`.

[25] B. Rodrigues, T. Bocek, A. Lareida, D. Hausheer, S. Rafati, and B. Stiller. A Blockchain-based Architecture for Collaborative DDoS Mitigation with Smart Contracts. In *IFIP International Conference on Autonomous Infrastructure, Management, and Security (AIMS 2017). Lecture Notes in Computer Science Vol. 10356*, pages 16–29. Springer, July 2017. Zürich, Switzerland.

[26] B. Rodrigues, T. Bocek, and B. Stiller. Enabling a Cooperative, Multi-domain DDoS Defense by a Blockchain Signaling System (BloSS). In *Demonstration Track*, pages 1–3, Singapore, Singapore, Oct 2017. IEEE. URL `https://goo.gl/5TMFUt`.

[27] B. Rodrigues, T. Bocek, and B. Stiller. Multi-Domain DDoS Mitigation Based on Blockchains. In *IFIP International Conference on Autonomous Infrastructure, Management and Security*, pages 185–190, Zürich, Switzerland, July 2017. Springer.

[28] B. Rodrigues, T. Bocek, and B. Stiller. The Use of Blockchains: Application-driven Analysis of Applicability. In *Advances in Computers*, volume 111, pages 163–198. Elsevier, 2018.

[29] V. Sajip. Python-GnuPG. `https://bitbucket.org/vinay.sajip/python-gnupg/overview`. Accessed: 09.03.2018.

[30] E. Sixt. *Bitcoins und andere dezentrale Transaktionssysteme: Blockchains als Basis einer Kryptoökonomie*. Springer-Verlag, 2016.

[31] The Associated Press. Hackers Used 'Internet of Things' Devices to Cause Friday's Massive DDoS Cyberattack. `http://www.cbc.ca/news/technology/hackers-ddos-attacks-1.3817392`, Oct 2016. [Online, accessed 2017-1-10].

[32] K. V.Revuelto, S.Meintanis. Ddos overview and response guide. Technical report, CERT-EU, Mar. 2017. CERT-EU Security Whitepaper 17-003.

[33] S. Wilkinson, T. Boshevski, J. Brandoff, and V. Buterin. Storj a peer-to-peer cloud storage network. 2014.

[34] S. T. Zargar, J. Joshi, and D. Tipper. A survey of defense mechanisms against distributed denial of service (ddos) flooding attacks. *IEEE communications surveys & tutorials*, 15(4):2046–2069, 2013.

[35] G. Zyskind, O. Nathan, and A. Pentland. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471*, 2015.

Abbreviations

AES	Advanced Encryption Standard
AS	Autonomous System
dApp	Decentralized Application
DDoS	Distributed Denial of Service
DNS	Domain Name System
DOTS	DDoS Open Threat Signaling
FOSS	Free and Open Source Software
IoT	Internet of Things
IP	Internet Protocol
IPFS	The InterPlanetary File System
JSON	JavaScript Object Notation
OS	Operating System

List of Figures

3.1 BloSS Data Sharing Scenario 8

3.2 BloSS Sequence 12

4.1 BloSS dApp Components and Classes 16

List of Tables

5.1 Ethereum Node Setup 23

Appendix A

Encryption Keys and Cipher Texts

A.1 Sender Public Key

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1

mQENBFqcNA8BCADH4ug+Wie3gjn5IYIGQ1+lk/rMqTEU1IoS3qT4L6kZx70xmKZR
s4BtcDvVoC7e58i21b+RC8yBhF2fIW0B091GLOzC8p1RdHKUgzocQne4wjM7ELBx
L5znuGRu8q7qQJMYdmnk/lbNpytyCdXNvi/E6EwDrLu1Qc1SGpOkonHH2Afm/Dx3
uwE3gOdVpqAeSthIqybH0Z88q1qgTFXzAXRyjEYlpKyMOUIaiIfDh/lwYGa6QJsU
H+gOIwPYocYTAsq947rHEaj3d/w9VVn5n72BrEql6Wqygpaoz9Vg8Cp0t7ooz0RZ
8iqjVQ5UNZ79z0YCFTus4C+xrpLzWUDQ1PBHABEBAAG0QE5vZGUgMTIgUmVjZWl2
ZXIgKE5vZGUgMTIgUmVjZWl2ZXIpIDxub2RlLTEyLXJlY2VpdmVyQGJsb3NzLm9y
Zz6JAT4EEwECACgFAlqcNA8CGwMFCQB2pwAGCwkIBwMCBhUIAgkKCwQWAgMBAh4B
AheAAAoJEBHuJq8e85jVsgoH/0dXJ0wUKfXyypCSh51NV0v+JWUOrmtLRALC9mUK
5eCqnmMyAEP4nrPfeqpfII9CqxGXsWdxg75Dt2+r+e8cEPdyzzMQF+gh2reATHaM
bsMsw5i7uK32h53hSoNmT6l9faibKsqvYFIdvgswLHmxQRRhxBZdLWavyoBT3CJw
0uEPmhJa70NrtN97RAvHvpLgB5VIWWhtoG6KLXoHNf0u4uYLfV9s2pCXhlDWkI5Q
6Eji7dC31w/yGT92RGk2mZXZTdIqhGZq3skgCxzQIQeFykl9grQ1HjY2qG888tk9
tO43Tl1ofG+fcW/PTzFYQ7QiVBUQO65o4Hu6OuVYBy3LGLS5AQ0EWpw0DwEIALfF
3s4RU+kzKohfb4fi4VF7tCWfHP9wj4egwfhTNETFM/87ci7Pplo6WJ4d1yOBnSej
5QLTMWfEXzJKNDma7NKVpwJ/mnP/Dm0FD7hIFey5vFrh17VCchYtSSkl8UUR+5R4
RQQ8nkQveFljol0LU9WeEbi4sBzJHy+XilhE4JFNKygGzOLTe4bpe4jr/Q2TqV2z
8ol8N/7PdigzlMVcOUQ5R1VpG5WTrNfFDZv5qnNsBe+MlVD9Pz5t7prnJeGJebe4
Q7d2/GNCFVepsGt3iDicpyvNg/eS/k5mCB7qSpCtAaedV0VSqKHVLU0b1gr6KVCJ
zcmZGigyrRiKuFOgt28AEQEAAYkBJQQYAQIADwUCWpw0DwIbDAUJAHanAAAKCRAR
7iavHvOY1QfSCACoqzvklGmKqOwYAdLfp2Q4naI9yfVxh7emDV4GEdhYAL8Mo3W0
Tm60j95gDcl0Q5OhodCdb8sgnXxje8Q/aWz+6AF/xe+cffETqnK1yE6f+P7cqNkg
MJji5UevSX4gn/tmJEr/ahybmE0SWLy2p+pIxt1ne9EI0+wqbjT4LgXcs+vjVlcJ
m5hzgkVE36WsR63hyLq5DK+CTuWJoq7pyby3+vvmEgBC+VVTnxv1jCEumXoHHWbt
jk3cM7SLPH/vHZuNAUiTKOOsT64q8hlJfnJ74jeAfBBgbekL4uEs/zmys1cIt5Xe
LwIANgjncxeP56WXtftWe5EZ5VM0rqO03q1B
```

```
=7W/v
-----END PGP PUBLIC KEY BLOCK-----
```

A.2 Plain IP Address List


```
[
  {
    "ip_address": "139.250.139.232"
  },
  {
    "ip_address": "203.82.136.126"
  },
  {
    "ip_address": "153.98.150.227"
  },
  {
    "ip_address": "0.57.174.170"
  },
  {
    "ip_address": "149.202.188.67"
  },
  {
    "ip_address": "10.31.178.183"
  },
  {
    "ip_address": "250.187.19.221"
  },
  {
    "ip_address": "5.103.198.238"
  },
  {
    "ip_address": "9.44.35.3"
  },
  {
    "ip_address": "195.88.95.142"
  },
  {
    "ip_address": "244.64.69.49"
  },
  {
    "ip_address": "209.71.110.165"
  },
  {
    "ip_address": "222.45.45.215"
  },
```

```
  {
    "ip_address": "83.160.228.129"
  },
  {
    "ip_address": "60.100.193.137"
  },
  {
    "ip_address": "87.37.108.46"
  },
  {
    "ip_address": "155.150.185.86"
  },
  {
    "ip_address": "199.65.221.243"
  },
  {
    "ip_address": "219.68.152.20"
  },
  {
    "ip_address": "2.192.38.29"
  }
]
```


A.3 Encrypted IP Address List


```
gAAAAABar5yzHZJIckTTsESIjfYRh0l6mNKDs4UZzKc25IfQQ3UKB6KBuZUZkcAnb-
   ↪ ROeFXmr5pxXKuS_50Jkd-7FfTtrPQRzLGe2JSystttAska48tTdKZb5XECdV3tFf0
   ↪ xBqLASfICSdptoFWgm2AkS4K9J2HrJseCMNhI4e5MIBNsjXL4bsBIxPdcB-
   ↪ _F_xmcU_dw_bSHYLyLnDPtvRB9pl1D4SehMjnjC4JCyQm4DqD5cdB2fpf_xz_SBFEz0
   ↪ SFAEg7sVnLTYIPJoJLRvtgrzzpT7GU6OXuF29oVyII0
   ↪ KYLHqN3a6TNaDlGUfOQFnoDCXh1jX6o3uBYN5GBDCtZu-3G3gKcG3t8inG0DGqi
   ↪ KUw9bTUD0_eXI2olwAPmtPQfVmHFv40YU6jl-lIsoBDQn2xgU7cxv8cj5t0
   ↪ ajpC4_fELuAO9GTKSAHhS3OzdpaVMTezqDHXHzDj2mFUzv0ziujI0
   ↪ XCpXSU3qYwlkQvs-MYLpigb4O3fUZMabdy8SFhnwmfaMQVLcIwlvu3hzb8wM8Iebja4
   ↪ iHP6d71lCe-pKaW9k1wme34xvqVM9WRj4Zvo5aN1cQnDyy0LLiYxMSS3LNRxr5zGGOi
   ↪ 7y8Z2GwXdfJfvMxJxFXS9FgCW-
   ↪ b4eHZbElskC6dsvzNgnG69rLkFHJE6YlzbeW3NlEkHdnzYjiFk42v-Sf78WdF4x-
   ↪ MFl2wCFNgu2Q9Oc5RnYL1G6YLs6nfjP8eTPtC7CDYVXLXfDFn0
   ↪ VqC5qT41ZIzqVDMH3FLexdvAzDDTfrekv_rx9YT9OkWBW3xz5UACaSRDr6jQ_fCkOBSb
   ↪ iiMbFp7PSozjbjXe6NzHzRGE2a0adcmwAanGq12va_c7a6Q7gL_TUjk-579
   ↪ EHzeTvoa7dZro7HJDJP2Tl3AwDtE0zxarN478UBI08HDbdvco4KiosRiwEHTFnT_bBW
   ↪ -3E36_L-kq5uZ6wscyICj0xLz2GbvWq8mFgLt-OJle4Q-_0a3NjE51kAN7kgous-
   ↪ ekyCARkpVAWXrenwqoO4t7ek5URYWnjoAVqi
   ↪ yFFfdItcLoQbGYyufU89l9SaeLOZAvE6XjinR55FB-VfQGrHsAWxcl5WN45qymiRf30
   ↪ -EOtKAVrVBhf_tm0u8k8DISohYQ7L2f4tZmWCo9lbiW0aT8MCqN97j8lei
```

```
    ↪ MzaAFu2DhRJL37lKFxDwhQmRoF832Wh91EIiJGGglSUqq1SHnMhrWCgchPvsTz-
    ↪ H6Flp2cuKFeOeorDtow==
```

A.4 Encrypted File Key

```
-----BEGIN PGP MESSAGE-----

hQEMA2i5Ut5MS5MWAQf/eSTpCfCIK7cHtk3xxa20QaPxe4KUbFXSWo9aFRgv3qbn
t9fbs9RbOeg6kgHB5wXKO7QG9RQD/rLEpILB4G7hKkgz/vNCDPsgZOlFNGlNKVhw
fC2ZJqUFfgurVLiMTb4Xt+cBUYIXh+70zR9v22lpTk3gHRlDBLuw3IMk3QWCMkep
JV8TFUo6fCtvOuabxSnNWvhZQcLKMobjyW9UWOzUd57z8nPFbg2BPglzVksVkmJh
BVtgjaPdrOAS4jDFjJit53FQSED1nvAF3GNqL/AR1TvDklNF6WXhTSLCzIYUARbe
4fKPjZLDKTD5U95l7pyBAHFE+kT+dI7ld545E04dLNJnAapv1LdP2Z+qVqn8Y4N5
TjZTkXJHfaTKfOpeBzhHORcBHVlSlx2UtUyk24+7Nif7iMceC7DUuamkWnT+hfno
XL0LhFZnUzdkfbG4Pf+BC6nvmFcN8zlG8chrc8LOaBr2M70r4WiWCA==
=D3lC
-----END PGP MESSAGE-----
```

Appendix B

Installation Guidelines

Appendix C

Contents of the CD

Bachelor Thesis The LaTeX files for the Bachelor Thesis document.

Source Code The Python source code for the BloSS decentralized application.

References The in the Thesis used references in PDF form.