Technische Universität München

Department of Informatics

Bachelor's Thesis in Informatics

Address Resolution and Key Management for a Distributed Communication Underlay

cand. inf. Andreas Kammerloher

TECHNISCHE UNIVERSITÄT MÜNCHEN
DEPARTMENT OF INFORMATICS

BACHELOR'S THESIS IN INFORMATICS

Address Resolution and Key Management for a Distributed Communication Underlay

Adressauflösung und Schlüsselverwaltung für ein verteiltes Privatsphärenerhaltendes Kommunikations-Underlay

Author	cand. inf. Andreas Kammerloher
Supervisor	Prof. Dr.-Ing. Georg Carle
Advisor	Dr. Holger Kinkelin
	Marcel von Maltitz, M.Sc.
Date	August 17, 2015

Informatik VIII
Chair for Network Architectures and Services

I confirm that this thesis is my own work and I have documented all sources and material used.

Garching b. München, August 17, 2015

Signature

Abstract

Two important building blocks for every network infrastructure are identification/authentication of peers and address resolution, i.e., the mapping of a peer's ID to its IP address. Today's predominant solution for address resolution, DNS (Domain Name System), has various drawbacks regarding privacy. First, a user's DNS query can be evaluated by an observer at various places in the network and be abused to derive information about this user. Second, DNS responses reveal the IP address of a peer to everyone who knows this peer's DNS name. This is especially critical, if the peer is a home network or a mobile device, as an observer would be able to track the peer over a longer time.

This thesis introduces a distributed, privacy-preserving approach to address resolution based on a distributed hash table (DHT). Every participating user holds a DHT node. The DHT is used by the participants to store a mapping from a peer ID to the peer's IP. Vice versa, the DHT allows to resolve a peer's ID to its IP. IP addresses stored in the DHT are always encrypted by public keys of those peers that are authorized to perform the address resolution. This approach effectively prevents data leakage, as unauthorized peers are unable to decrypt and abuse a peer's IP address.

Besides this functionality that offers a privacy preserving, DynDNS-like service, our approach was specifically adjusted to act as an address resolution component for a framework that offers privacy preserving network communication. For this purpose, various adaptations were needed and further functionalities and precaution mechanisms had to be integrated into the solution. We finally perform an evaluation that shows the suitability of our found solution for the named application scenario.

Contents

List of Figures

Chapter 1

Introduction

As revelations of surveillance programs are growing in number and severity many people grow more conscious of their online privacy. While many existing technologies allow for increased privacy while browsing, none of them are focused on Voice over IP (VoIP). Such technology in form of a privacy-preserving communication underlay is to be created. This underlay requires a privacy-preserving method to resolve addresses. Technologies like DNS are not applicable since they don't provide the required authenticity and confidentiality. Improvements on this technology like DNSSec are also not enough as DNSSec provides authenticity but not confidentiality. On top of that the resolution of a certain address may need to be limited to a subset of users. Such limitations are not possible through DNS. The address resolution and key management module described in this thesis will be focused on solving these privacy problems. Separate theses describe the routing [1] and the integration of the actual VoIP service [2] for the underlay.

There are already many approaches to address resolution, some of which are eligible for usage as basis for the address resolution module, but none of them allow appropriate security and usability as required for the communication underlay.

1.1 Problem statement

The privacy-preserving communication underlay requires an address resolution module. This module has to allow a user to publish their address to the public or trusted peers only. It also has to resolve addresses confidentially. Third parties must not be able to see who resolves the address of whom. In addition, the address resolution has to be trustworthy, so the data source needs to be authenticated and the integrity of the data has to verifiable.

The module also has to provide a way to complete a trust exchange to broaden the range of trusted peers whose addresses can be resolved confidentially through the system itself. A way to hold the identity of trusted peers and their contact data is also required.

On top of that the module should allow users to offer their services as relays and resolve the address of potential relays to use to build tunnels as described in a separate thesis [1].

1.2 Goals of the Thesis

The goal of this thesis is to design, implement and evaluate a software module that provides privacy preserving address resolution and key management for the privacy preserving network underlay. The module's functionality will include publishing and resolving contact data in a privacy preserving and secure manner. Only trusted parties will be able to resolve a users address. In addition it will provide a way to complete a trust exchange through the module.

1.3 Research Questions

How can addresses be published and resolved privately and securely?

- What exact functionality is required of the address resolution module?
- Which existing address resolution technologies are suitable as a basis for this work?
- What can the architecture look like?
- What threats to security exist and how can they be counteracted?
- How can participants be identified and authenticated?

1.4 Outline

Chapter 2 will explain concepts and technologies that are used in the thesis and have to be grasped to understand the rest of the thesis. Chapter 3 provides information about related work and describes the differences and similarities to this work. Chapter 4 provides an analysis of requirements, necessary functionality, choice of base technology and security threats. Chapter 5 describes the design that was created based on the knowledge gathered in previous chapters. Chapter 6 elaborates on the implementation based on the design described in the previous chapter. This includes the definition of protocols that implement the functionality provided by the address resolution module. Chapter 7 details the evaluation of the implemented module. It evaluates how well requirements were met and security features. Chapter 8, the conclusion, wraps the thesis up in a quick summary.

Chapter 2

Background

2.1 Security

Information Security is based on the following security goals.

- Confidentiality
- Authenticity
- Integrity

Confidentiality means that only intended recipients are able to read a message. It is achieved by encrypting the data with a key that is only known to the author of the message and those whom the message is directed to.

Authenticity means that every party that participates in communication is who they say they are. This is accomplished through signatures using asymmetric key pairs. A private key, known only to the author of a message, is used to sign said message while an associated public key, known to everyone, is used to verify the signature.

Integrity means that messages can not be modified in an unauthorized or undetected manner. This can be ensured by using signatures over cryptographic hashes.

2.2 Privacy

To grasp the concept of privacy it is important to understand what information can breach privacy if it is not protected and in what context leaked information can threaten privacy.

2.2.1 Information to Protect

One definition of the information that can threaten privacy if unprotected is *personally identifiable information* (PII). PII is information "that can be related to an individual" [3]. This could be an IP address or a person's real name. Often times data is not a threat to a users privacy directly or on its own but can lead to other threatening information or can endanger a users privacy when combined with other data. For these cases, the term "privacy-relevant data" was coined. It includes data that can lead to a breach in privacy in ways that may not have been considered. Protecting privacy-relevant data is to protect all data than can be protected withing a reasonable scope since all unnecessarily leaked data can be relevant to a users privacy.

2.2.2 Unlinkability and Anonymity

Anonymity is the confidentiality of an entities identity [4]. *Unlinkability* means that no new information about the relation or origin of separate pieces of data can be gained by linking them. [5] Often a single piece of information is not a threat on its own but can breach a users privacy if it can be linked to other information. Preventing linkability entirely is incredibly complicated, if not impossible, so it is more sensible to focus on preventing linkage of data from breaching a users Anonymity. Anonymity is the confidentiality of an entities identity [4]. Anonymity in name resolution is achieved when it is not known who resolves the name of whom. Often true anonymity is very hard to achieve and pseudonymity is used instead. This means that the real identity of an entity is concealed, but the entity is known by a pseudonym. If actions can be linked to a single pseudonym it can threaten the entities privacy, which is especially dangerous with name resolution. This is because the relevant data that is updated by an entities actions contains its identity.

2.3 Name Resolution

DNS (Domain Name System) is a common name resolution protocol. It maps a name to an IP address through a a low amount of publicly accessible, centralized servers. DNS address resolution is based on domain names. When a user wants to resolve an address, the resolver, a local agent, takes the domain name as an argument and returns information related to that domain name. To access this data the resolver contacts a set of name servers. The resolver knows at least one name server which he addresses. He either receives the requested information or a referral to another name server until the information is found. [6]

DynDNS updates DNS servers information in real time to ensure that the mapping of names to IP addresses is correct at all times even in an environment where entities frequently change name or IP address.

DNSSec is a set of extensions for DNS that provides improved security. DNSSec authenticates the origin of resolved data to protect users from spoofing. It also allows to check the integrity of the data. However, it does not provide confidentiality.

2.4 Distributed Hash Tables

A *distributed hashtable* (DHT) is a network built from nodes that offer storage and bandwidth. Every participating user runs a DHT client that serves as a node. The client connects to a limited subset of other clients that are part of the network. It performs put and get operations. Put takes a key-value pair, saves it to the hash table and asks other nodes to do the same. Get takes a key and searches the network for the value associated with said key.

The way put or get messages are propagated through the network is either iteratively or recursively. If a DHT implementation uses iterative message propagation, a node that recieves a message responds with a list of other nodes for the author of the message to address. The author keeps addressing nodes until the desired node is found. In recursive message propagation a node that recieves a message sends messages to its own contacts. These messages form a chain until the desired node is found.

2.4.1 Kademlia

Kademlia [7] is a flexible DHT implementation. It minimizes redundant messages by spreading configuration information along with lookup operations and uses parallel, asynchronous queries to avoid timeout delays. Lookup-keys use the same format as the result of a SHA-1 hash. This allows strings that can be remembered by humans to be used as keys by hashing them.

Every node has a node-ID and a k-bucket. Node-IDs can be chosen by the node and are usually randomly generated. They have the same format and length as keys and have to be unique within the network. The distance between nodes or between keys and nodes is measured by applying the XOR metric. This distance is used in routing. Aside from that, node-IDs also serve as identifiers for a node.

The k-bucket contains a list of the k closest nodes to a nodes own node-ID for routing.

All lookup operations between Kademlia nodes are based on the find operation. It takes a key and a short string indicating the remote procedure call (RPC) and finds the nodes closest to the key. If the RPC is 'findNode' the key is equal to the desired nodes node-ID. A node that receives this message returns contact data of the node with the least distance to the desired node from its k-bucket until the requested node is found. If the RPC is ’findValue’ the key is equal to the key associated with the requested value. A node that receives this message returns contact data of the node with the least distance to the key from its k-bucket until a node that holds the value is found.

2.4.2 Entangled

Entangled is an augmentation for Kademlia that introduces a delete RPC and keywords with the "publish" operation.

The delete RPC is based on the find operation using the key associated with the value that is to be deleted and the string ’delete’ as RPC specifier. It behaves like ’findValue’ except instead of the value being returned it is deleted.

The keywords work as follows: Whenever a value is published to the DHT every separate word that is longer than two characters in the pre hash key string is used as a new key that is associated with the full pre hash key. So the pre hash key "this is my key" spawns the keywords "this" and "key" that are both keys that point to the full phrase "this is my key". Keyword entries work exactly like regular DHT entries. If several publications share the same keywords the full pre hash key the pre hash key is appended. Retrieving the keyword returns a list of all associated pre hash keys.

2.5 Authentication

The following existing technologies are used for authentication.

2.5.1 GPG

GNU Privacy Guard (GPG) is an implementation of the OpenPGP standard and an established method of exchanging certificates. It allows a user to bind an email address to their public key and allows to exchange these public keys through key servers, which take an email address and return the associated public key. Trust is established through a Web of Trust. Users sign other users certificates to show that they trust the person. The trust of a well trusted user is more valuable than the trust of an untrusted user.

2.5.2 X.509

X.509 is a public key infrastructure (PKI) standard for issuing certificates. They are signed by the issuing certificate authority (CA) and contain the certified parties public key. Certificates are only as trust worthy as the issuing entity.

Often certificate chains are formed with a certificate hierarchy. An entity is certified by a root CA and therefore certificates issued by this entity are trusted if the root CA is trusted.

2.5.3 Authentication in Unmanaged Network Domains

Previous work at the Chair for Network Architectures and Services targeted the question how standard X.509 certificates can be used for identification and authentication in and between so called unmanaged network domains. Unmanaged Domains exist in home networks or small companies and are identified by the missing of a computer expert that performs network and security management and therefore require an automated service to perform the task.

In Autonomous and Robust Components for Security in Network Domains [8] mechanisms are described that distribute X.509 certificates to devices and services that belong to the local Domain. Additional mechanisms described in [8] create so called Trust Relationships between Domains of befriended persons. Performing a Trust Exchange between two Domains A and B exchanges the Domain CA certificates of the participating Domains mutually. As a consequence, a service in Domain A is able to authenticate a device from Domain B (and vice versa).

Chapter 3

Related work

3.1 Namecoin

Namecoin [9] is a cryptocurrency that is based on the code of Bitcoin. It is also used as an address resolution service, using decentralized namespaces. A namespace in this context is "an online system that maps names to values" [9]. Namecoin has a block chain, which is a public, distributed data structure that stores public Namecoin transactions in chronological order. Namecoin stores key-value-pairs that are used for name resolution in this block chain; every block can hold 520 bytes of additional data.

To save the key-value-pairs to the block chain a Namecoin transaction has to be completed. There are three script operations that are used for publishing key-value pairs. The first is ”NAME_NEW”: A user turns a coin into a name token. Whoever holds this token can determine the value associated with the token's name. To publish the token, without a value for now, a transaction has to take place. To keep the token a user can set up a second name coin address and set up a transaction to said alternate address. This transaction is essentially a request for the token's name. After the block chain has reached consensus on the ”NAME_NEW” transaction the name is claimed and the second operation can be performed: The ”NAME_FIRSTUPDATE” operation. This operation is another transaction to an alternative address controlled by the same user that associates a value with the name. The final operation is the ”NAME_UPDATE” operation. This operation can be used to change the value associated with the name by completing a transaction to a self controlled alternative address with the new value or refresh the old value by republishing it. This is used to prevent the name-value pair from expiring. Lastly this operation can be used to transfer ownership of the name to another user by creating a transaction to the other users address.

To resolve a publish address the block chain has to be searched for the associated name. A copy of the complete block chain can be stored locally so resolving an address does not require trusting a third party.

3.1.1 Similarities

Decentralized name resolution that provides confidentiality and authenticity.

3.1.2 Differences

Name resolution is used to resolve website addresses using human readable domain names. All entries are publicly accessible. Registering a domain name is not free, albeit very cheap, and domain names can be sold. Names also have an expiration, but can not be deleted at will. Since saving a name to address mapping works through transactions changing a value is also not free. Publishing new addresses and updating already published ones is not automated and requires a fair amount of user input.

3.2 Freenet

Freenet [10] [11] is a decentralized peer-to-peer network designed to allow censorship resistant and anonymous publication of files. Every freenet node ist part of a DHT and stores key-value pairs, the values of which are encrypted with symmetric keys. The lookup-keys are either a simple hash of the content or a randomly generated key with a signature. The signed keys are called Signed Substapce Keys(SSK). They allow the original publisher to overwrite the value associated with them if he can provide another signature with the same private key. The files stored in the freenet DHT can be entire websites, called freesites.

To access content behind a freenet key the key used to encrypt said content has to be provided as well. Since the nodes hosting the content don't have access to the encryption key they can avoid responsibility for the content as their case may fall under plausible deniability.

Freenet can be used as an opennet, a darknet or a hybrid. In the opennet approach the user connects to random other nodes. With the darknet approach the user connects to a set of trusted nodes which in turn connect to nodes they trust, creating a large network where very node is adjacent only to trusted nodes. In the hybrid approach the user connects to both random users and trusted nodes.

The Freenet DHT could be used for publishing the users own contact data to trusted peers.

3.2.1 Similarities

A DHT stores encrypted values. The associated lookup keys and encryption keys can be shared only with trusted peers to limit access to the stored information. Retrieving information from the DHT is confidential and, if an SSK was used, authenticated. If used for address resolution, multiple aliases can be created as SSKs are randomly generated. Values behind SSKs can be updated.

3.2.2 Differences

Freenet requires a fair amount of user interaction to retrieve a value and also requires users to install the freenet client. Freenet startup is slow, uploading files is slow and searching freenet is slower than searching a network used specifically for address resolution purposes.

Since freenet allows publishing arbitrary data it could be used for address resolution and even trust exchange. However, these processes could hardly be automated and would be fairly slow.

3.3 TOR

TOR [12] is a distributed overlay network that allows users to connect to the internet anonymously. This is achieved by routing traffic through a circuit built from several nodes belonging to different users. These circuits end at an exit node that connects directly to the requested website and routes the traffic back to the originally requesting node. All traffic inside the circuit is encrypted. Since the only unencrypted traffic is between the exit node and the addressed website it is difficult to find out who the requesting user was.

While TOR uses DNS for address resolution for regular websites, it also offers hidden services [13] with anonymous address resolution in the form of .onion domains. Hidden services act like regular websites but they don't reveal their IP address. A .onion domain is set up by choosing a set of rendezvous points. These introduction points are regular TOR relays that are addressed by the hidden service through a TOR circuit and told the hidden services public key. They do not know the hidden service's IP address. Now a hidden service descriptor containing the public key and references to the introduction points is created and signed with the hidden service's private key. This descriptor is saved to a DHT. The key associated with the descriptor is the .onion domain name, derived from the service's public key.

To resolve a hidden service's contact data a user has to know the .onion address. With this address he can resolve the service's descriptor from the DHT. He now knows the service's introduction points and its public key. Next the user creates a circuit to another TOR relay and asks it to act as rendezvous point. Now the user composes an introduce message including the user's public key and a reference to the rendezvous point, encrypted with the hidden service's public key. He sends this message through another TOR circuit to the service's introduction points which relay it to the service. The service now connects to the rendezvous point, using another TOR circuit. The rendezvous point informs the user that a connection with the service has been established and the user and the service can now communicate through the rendezvous point. No party knows the other party's IP address.

3.3.1 Similarities

DHT technology is used for contact data resolution and contact data resolution is confidential. Publishing and resolving contact data is mostly automated and requires very little user input.

3.3.2 Differences

The resolved contact data is not an IP address and even after a service's contact data is resolved it remains anonymous to the resolving party. Access to contact data resolution can be controlled by choosing carefully with whom to the .onion address is shared, but access rights can not be removed after they have been given and everybody who has the .onion address can share it further.

3.4 EncDNS

EncDNS [14], which is short for encapsulated DNS, is an approach to achieve confidential address resolution in a way that not only the traffic between the client and the DNS resolver is encrypted, but the resolver itself doesn't know what name the client wants to resolve.

The EncDNS client encrypts the DNS query together with its public key for the EncDNS server. It then appends the EncDNS server's domain name and sends this query to a conventional resolver (CR). The CR reads the EncDNS server's domain name and relays the message to said server. The EncDNS server decrypts the message and uses the DNS name servers to resolve the requested address. Next the public key that was appended to the query is used to encrypt the resolved address which is sent back to the CR which relays it back to the requesting user. The traffic between the user and the CR and the traffic between the CR and the EncDNS server was encrypted and could not be read by an observer. All messages the CR received were encrypted as well, so the CR did not know what name was resolved to what address.

3.4.1 Similarities

The approach allows for confidential address resolution. The entity used for resolving the address does not know the address that was resolved. The address resolution is automated and requires a minimum of user interaction.

3.4.2 Differences

EncDNS doesn't allow to limit access to a published address and is based on DNS technology, hence it doesn't allow publishing data confidentially.

Chapter 4

Analysis

4.1 Scenario

The previous work provides a network of domains that can identify and authenticate each other after a trust exchange.

The related thesis ”A modular Framework for a collaborative solution for privacy-preserving network communication” by Christian Brosche [1], which specifies routing for the communication network, makes it clear that the address resolution network has to work with entities outside of the network of trusted domains. This is because a network with low amount of peers would not provide a meaningful protection of privacy. A global network that contains peers that can not be authenticated immediately is required.

4.2 Use Cases

The Scenario leads to the following use cases. For all of the following use cases the user is connected to the DHT through his own node. He has a set of other users he trusts and knows their public keys.

The node wants its trusted peers to know its contact data. At first its contact data is not saved to the DHT, it is unencrypted and the peers do not know the contact data. After the following procedure it will be encrypted for its trusted peers, saved to the DHT, read by said peers and decrypted. Initially the node encrypts its contact data individually for every trusted peer using the associated public key. Next it puts the encrypted contact data to the DHT using its ID as key. Following that the trusted peers get the encrypted contact data from the DHT and try to decrypt every instance until they get the one that was encrypted with their public key and they get access to the nodes contact data.

Now a node wants to establish trust with another node through the DHT. Before the procedure the node knows the other nodes public key and its peering ID; the other node does not trust the node and doesn't know its public key. Afterwards trust will be established and both nodes will know each others public key. At first the node compiles a trust request message that includes its own identity and public key. This messages is subsequently encrypted with the other nodes public key and saved to the DHT under the other nodes peering ID. The other node keeps polling for its peering ID so it receives the trust request message. It proceeds to decrypt said message and gain its public key and identity. It can now decide whether to trust the node or not. If it does not trust the node, it does nothing. If it does trust the node it will save the nodes contact data as a trusted peer and the trust exchange is done.

The node wants to offer its services as a relay for another node. At first there is no entry for the node as relay in the DHT and another node is looking for a relay. In the end the node will be entered as a relay in the DHT and the other node will be able to use it as a relay. Initially the node creates a new asymmetric key pair. The new key pair's public key is now saved to the DHT together with the nodes contact data, using the keyword "relay". A new public key is used so the relay identity can't be linked to the nodes alias in the DHT through the public keys. The other node searches the DHT for entries with the keyword "relay" and receives the data that was saved to the DHT. It can now use the node as relay.

4.3 Requirements

The following requirements will define how the system is specified.

4.3.1 Functional Requirements

- FR1: The system has to resolve the addresses of peers and authenticate them
- FR2: It has to allow the user to limit who can resolve his address.
- FR3: The system has to be capable of trust management
- FR4: It has to be possible to establish trust through the system as well as through side channels
- FR5: The system requires the ability to change a users alias or create multiple aliases
- FR6: It has to be possible for users to modify or delete data that they published previously

4.3.2 Non-Functional Requirements

- NFR1: The system needs to maintain the users privacy as defined in the background section
- NFR2: Address resolution has to be confidential
- NFR3: The authenticity of other nodes that publish, modify or access data has to be verifiable
- NFR4: Changing aliases or adding new ones has to be quick and without much delay
- NFR5: The system has to be distributed and work without a centralized authority
- NFR6: It should also be as automated as possible, requiring a minimum amount of user input
- NFR7: It has to be possible to switch the DHT module out for another address resolution module

4.4 Applicability of Established Name Resolution Protocols

We can not fulfill the requirements with DNS. It doesn't allow to limit resolution of a users name to those he trusts (FR2). It also doesn't provide authenticity or confidentiality for the resolved addresses (NFR2) and requires centralized servers (NFR5). Also, DNS caches update too slowly to allow for frequent and quick changes of a users alias (NFR4).

DynDNS can solve the last problem and DNSsec provides authenticity, but all other problems remain unsolved. Therefore DNS is not suited for our needs.

We use a DHT because it is inherently decentralized and allows for a lot of customization while providing a solid basis for distributed address resolution. This approach comes with a few security issues which are described in the next section.

4.5 Security and Privacy Implications of DHT Usage

This section examines the different types of attackers and how they could threaten the systems security.

4.5.1 Passive Observer

The *passive observer* can see all traffic in the network. He only reads the traffic but can not modify it.

Reading the messages that publicize a users contact data compromises their anonymity as it allows linking their IP address to their alias in the network. Reading the messages generated when a user wants to read another users contact data is also dangerous, as it allows the observer to see whom a user may be in contact with.

The passive observer can not be prevented from reading meta data such as when a message was sent, where it came from and which node it addressed. This information includes which RPC is called through a message as RPC calls can not be reasonably encrypted as that would require having a unique encryption key for every node.

4.5.2 Malicious Node

The *malicious node* is a node in the network that belongs to a malicious user that can modify the nodes behavior.

Passive attacks by a malicious node are similar to the passive attacks by a passive observer, but the malicious node only has access to the traffic that is routed through it. More importantly, the malicious node can interact with other nodes to perform active attacks by modifying data.

4.5.2.1 Malicious Node: Attacks on Security

The malicious node poses a threat to integrity and authenticity of data. If a node requests the data associated with a key the malicious node can return a modified version of the correct data or new data it created by the malicious node itself. This can be used for spoofing; making the victim believe the malicious node was a different, trustworthy entity. It can also be used to simply disrupt the DHTs functionality.

The malicious node can also insert false data into the DHT. This could be used to publish incorrect contact data under a key that another user uses for publishing his contact data which can also be used for spoofing. Inserting false data is especially dangerous if the attacker can overwrite legitimate data, as that would break the address resolution functionality entirely.

In addition the malicious node can save a delete or overwrite message and resend it later to delete or overwrite the value behind the key again. This is called a replay attack.

4.5.2.2 Malicious Node: Attacks on the DHT

Malicious nodes can also disrupt the networks functionality. Spamming data inserts can fill up the DHTs storage, spamming requests can use up bandwidth and slow down responses for legitimate nodes.

On top of that there's the *eclipse attack*. With an eclipse attack malicious nodes controlled by a single attacker surround another node or part of the network in such a way that all requests between the eclipsed part and the rest of the network have to travel through the malicious nodes and can be deleted by the attackers. This can be done by choosing node-IDs close to the target nodes node-ID. Information on how to avoid eclipse attacks can be found in the papers "Avoiding Eclipse attacks on Kad/Kademlia: an identity based approach" [15] and "Eclipse Attacks on Overlay Networks: Threats and Defenses" [16]. Unfortunately, these solutions harm the decentralization of the DHT.

Lastly there's the *Sybil attack*. A large number of malicious nodes controlled by a single attacker enter the network so that most requests arrive at a malicious nodes, giving the attackers control over the DHT. [17] [18]

These last three attacks are aimed at DHTs in general and not specific to this implementation. Defending against them is not part of this thesis.

4.5.3 Trusted Malicious Node

The trusted malicious node is a malicious node that has successfully completed a trust exchange with a targeted user. It has the same active attack capabilities as a regular malicious node but it can gain more information from passive attacks. Since the malicious node is trusted it can access a trusting users contact data at all times. Obviously this can not be prevented since trusting users are supposed to be able to read this information. Users need to be careful whom they trust.

Chapter 5

Design

This chapter outlines the concepts and ideas that guided the implementation of the DHT module.

5.1 Architecture

The DHT module displayed in figure 5.1 consists of the node that is connected to the DHT, a node interface that allows communication between the node and the rest of the system, a petname table that maps nick names of trusted users to their contact data locally and a cryptography module that is used for all cryptographic operations.

5.1.1 Components

The DHT module is built from the following components.

5.1.1.1 Node

The *node* is based directly on the entangled node and contains all functionality. It interacts with the DHT directly and is a part of the network. It also executes put and get operations and the operations described in the use case section. In addition, it has direct access to the petname table and the cryptography module.

5.1.1.2 Node Interface

The *node interface* allows communication between the node and the rest of the system. It maintains the node in a separate thread to allow it to act independently of other system functions. It also provides an array of methods the system can use to interact with the

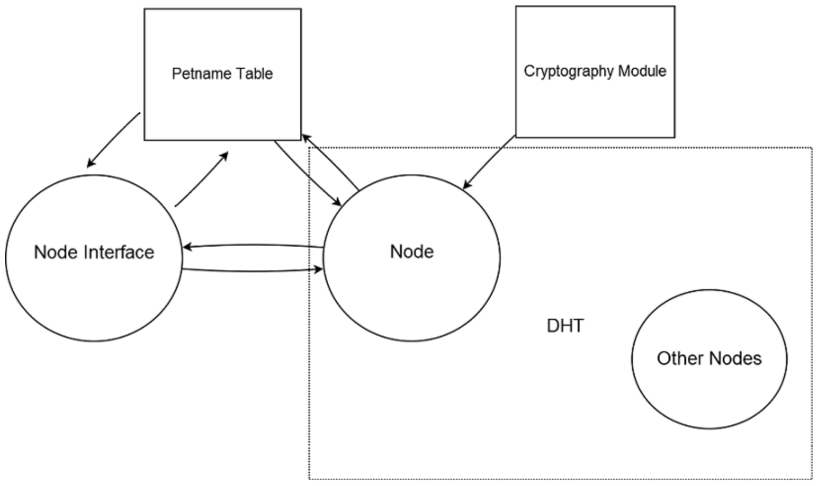

Figure 5.1: Components of the DHT Address Resolution Module

node. These methods produce commands for the node which are read and executed by the node. The commands include put and get operations on the DHT, joining a network and the functionality described in the use cases section. In turn the node creates results, like the result from a get operation on the DHT, which are read by the node interface and relayed to other parts of the system upon request.

5.1.1.3 Petname Table

The *petname table* maps the names of a users trusted peers to their contact data, including their public key, locally. It contains two data structures: One that does the aforementioned mapping and one that holds trust request messages from DHT trust exchanges that are awaiting user approval.

5.1.1.4 Cryptography Module

The *cryptography module* is designed to allow switching out hash or encryption algorithms easily. It is used for all cryptographic operations in the DHT module and currently employs AES for symmetric cryptography, RSA for asymmetric cryptography and sha-1 for cryptographic hashing.

5.1.2 IDs

The system uses the following IDs: The regular *ID* that is used as the key associated with which the nodes address will be published. This ID is a hash of the public key associated with the used alias. Then there's the *peering-ID* which is used by other users to publish trust exchange requests. The peering-ID can be a hash of a random password or the hash of a PGP-key. The user can decide what peering-ID to publish and, just like regular IDs, it is possible to have more than one. The node-ID is used to identify nodes in the DHT. It can be chosen by the node itself, but it has to be unique in the network. It uses the same format as lookup-keys and is used for routing.

5.2 Security Features

The following features uphold the security goals specified in the requirements.

5.2.1 Encryption

To ensure confidentiality the following data is encrypted when publicized: the contact data that is published for trusted peers and trust exchange requests. Both are encrypted using hybrid encryption. That means that asymmetric encryption is used to encrypt a symmetric-encryption-key which in turn is used to encrypt the data. The used algorithms are RSA for asymmetric encryption and AES for symmetric encryption.

When a node offers its service as a relay it publishes its contact data unencrypted. However this is not a threat to confidentiality as the relay identity can not be linked to the nodes other identities in the DHT and doesn't reveal meaningful information about the node on its own.

5.2.2 Signatures

Signatures are used to ensure the authenticity and integrity of a message. They are generated using a private key and can be verified with the associated public key. When data is put into the DHT it includes a public key. This public key is used to authenticate messages that try to modify this published data. Deletion and overwrite request messages include a signature over a timestamp and the DHT key associated with the value they intend to change. Nodes that receive such messages check their authenticity and integrity using the public key that was attached to the value.

5.2.3 Timestamps

Timestamps are attached to a message to limit its validity to a small window of time. This prevents attackers from recording the message and resending it later (replay attack), for example to delete a value behind a key. The DHT module uses timestamps with deletion and overwrite messages.

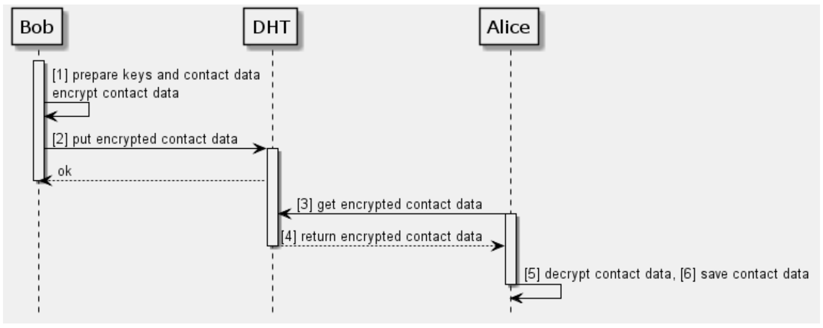
Bob
DHT
Alice
[1] prepare keys and contact data
encrypt contact data
[2] put encrypted contact data
ok
[3] get encrypted contact data
[4] return encrypted contact data
[5] decrypt contact data, [6] save contact data

Figure 5.2: Publish contact data to trusted peers

5.3 Protocols

In the following descriptions Alice and Bob are both users that own a node connected to the same DHT network. Unless specified otherwise, the names Bob and Alice refer to the user's nodes. The DHT is not a single entity but a large network of nodes.

Figure 5.2 displays the protocol for publishing a nodes contact data to trusted peers through the DHT. At step [1] Bob generates a symmetric key and encrypts his IP address and port with it. The symmetric key is then encrypted using the public keys of the contacts in Bob's petname table. At step [2] Bob saves his encrypted contact data and the encrypted symmetric-key-list under his ID to the DHT. At the [3]rd step Alice requests the data under Bob's ID from the DHT. At step [4] she receives Bob's IP address and port encrypted with a symmetric key and a list consisting of said symmetric keys encrypted with several public keys, one of which belongs to Alice. At the [5]th step, Alice tries to decrypt all the encrypted symmetric key blocks until she finds the one that was encrypted with her public key. She then uses the decrypted symmetric key to decrypt Bob's IP address and port. Finally, at the [6]th step, Alice saves Bob's decrypted contact data to her petname table.

Figure 5.3 displays the protocol for establishing trust through the DHT. At step [1] Bob generates a symmetric key and composes a messages consisting of the following elements: Bob's name, IP address, port, public key and a personal message. This message is encrypted with the symmetric key, which in turn is encrypted with Alice's public key. At step [2] Bob puts his encrypted message and the encrypted symmetric key into the DHT under Alice's peering ID. Alice regularly polls her peering ID and eventually receives Bob's request at step [3]. At step [4] Alice decrypts the symmetric key with her private key and uses the decrypted symmetric key to decrypt Bob's message. She now has Bob's name, his IP address and port, his public key and his personal message. Finally, at step [5], Alice, the user, manually reviews the information she received and decides whether she wants to accept Bob's request and add his contact data to her petname table or not.

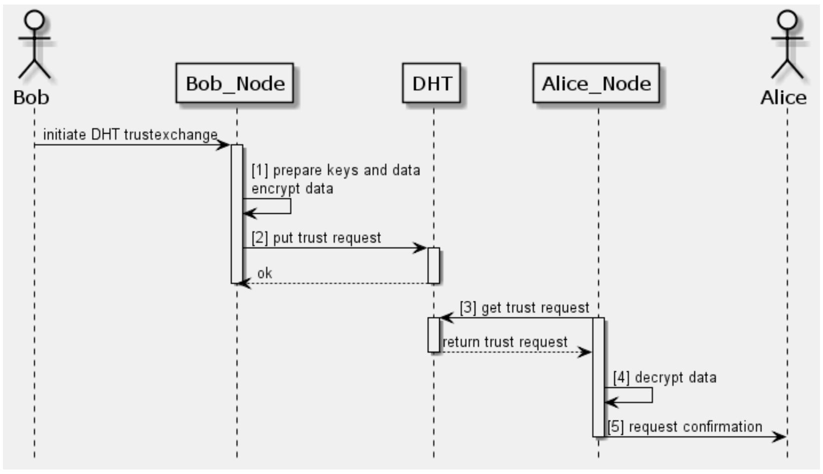

Figure 5.3: Trust Exchange through the DHT

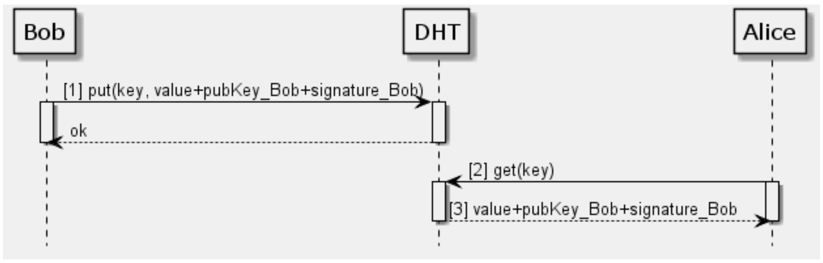
Bob
DHT
Alice
[1] put(key, value+pubKey_Bob+signature_Bob)
ok
[2] get(key)
[3] value+pubKey_Bob+signature_Bob

Figure 5.4: Put and Get DHT entries

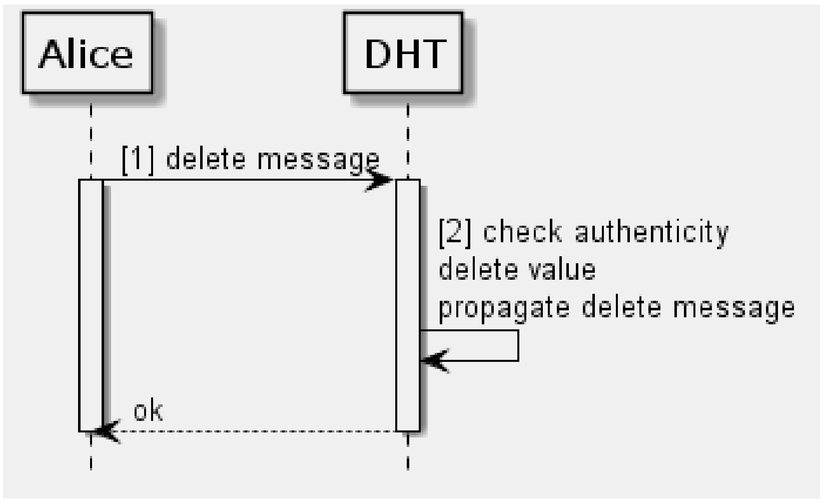

Figure 5.5: Delete a DHT entry

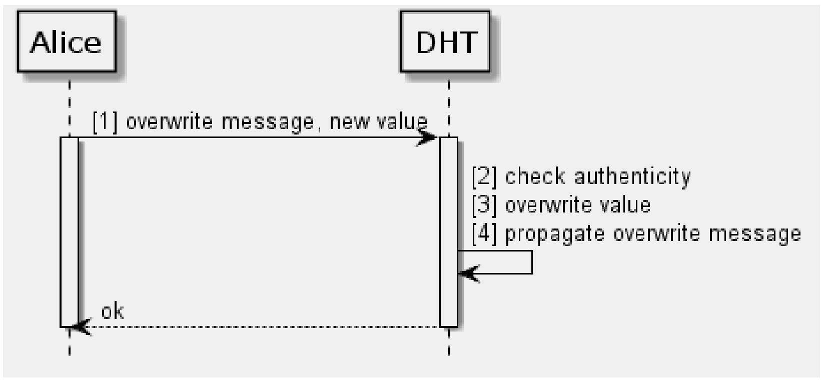
Alice
DHT
[1] overwrite message, new value
[2] check authenticity
[3] overwrite value
[4] propagate overwrite message
ok

Figure 5.6: Overwrite a DHT entry

Figure 5.4 displays the protocol for putting a value into the DHT and retrieving it through the get operation. At step [1] Bob composes a message from the value he wants to insert, his public key and a signature over both. This messages is put into the DHT. Inserting the public key is optional but recommended as it is used for authenticating deletion and overwrite requests. At step [2] Alice performs the get operation and receives the message Bob composed earlier at step [3].

Figure 5.5 displays the protocol for deleting a DHT entry created by the same user. At step [1] Alice composes a delete message that contains a timestamp and a signature over the DHT-key and the timestamp. This message is sent to the DHT. At step [2] the DHT nodes receive Alice's request. If the value behind the DHT-key has a public key attached to it, Alice's signature and the timestamp are verified and the value is deleted. If there is no public key the value is deleted without verification of the requests authenticity.

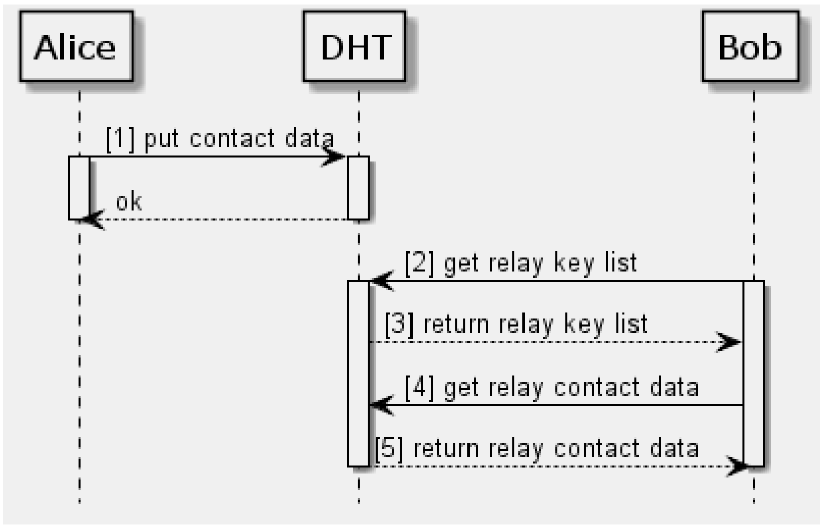
Alice
DHT
Bob
[1] put contact data
ok
[2] get relay key list
[3] return relay key list
[4] get relay contact data
[5] return relay contact data

Figure 5.7: Offer relay services and find a relay

Figure 5.6 displays the protocol for overwriting a DHT entry created by the same user. At step [1] Alice composes an overwrite message that contains the new value, a timestamp and a signature over the DHT-key, the new value and the timestamp. This message is sent to the DHT. At step [2] the DHT nodes receive Alice's request. If the value behind the DHT-key has a public key attached to it, Alice's signature and the timestamp are verified and the value is overwritten. If there is no public key the value is overwritten without verification of the requests authenticity.

Figure 5.7 displays the protocol for offering ones service as a relay and finding a node to act as relay. At step [1] Alice volunteers to act as a relay and puts her IP address, port and a newly generated public key under the keyword "relay" in the DHT. At step [2] Bob needs a random relay node and requests DHT-keys behind the keyword "relay" from the DHT. At step [3] Bob receives a list of DHT-keys under which he can find the contact data of relay nodes. At step [4] Bob chooses one or more of the DHT-keys he received and requests the contact data associated with these DHT-keys. Finally, at step [5], Bob receives the IP address, port and public key of the requested relays and has found the relay nodes he needed.

Chapter 6

Implementation

6.1 Architectural Overview

The implementation of the DHT module is based on Entangled, which in turn is based on the Kademlia DHT implementation. This chapter will illustrate the basis that the DHT module is built upon, the changes to functionality and the addition of new functionality. It also explains additional parts of the module that are not based directly on existing implementations.

The DHT module node uses the Entangled node for basic DHT tasks like inserting and retrieving values, Remote Procedure Calls and storage and routing methods. It interacts with the rest of the system through the node interface. This includes calling high level methods that implement the use cases, which are part of the DHT module node. For cryptographic operations the DHT module node relies on the cryptography module. The petname table is used to store and retrieve the contact data of trusted users. It interacts directly with the DHT module node.

6.2 Base Technology: Kademlia and Entangled

Kademlia is a DHT implementation. It provides basic DHT functionality like storage, routing and put and get methods. Kademlia also implements Remote Procedure Calls (RPCs). RPCs are messages that ask another node to run an RPC method. RPC methods are marked as such in the code. RPCs are used through the iterativeFind method. This method takes a key or a node-ID and a string to represent the requested RPC. The message is then routed to the who's IDs are closest to the key or node-ID used as argument for the method call. The RPCs allow to find or ping a node, find a value or store a value.

Entangled adds a basic delete RPC that simply deletes the value behind the key used as argument. It also introduces keywords through the "publish" method. Whenever a value is published through Entangled the pre-hash-key, which is a regular string, is searched for words with three or more letters. These words are then used as keys to publish the entire key for the original value. The DHT can now be searched for those keywords to retrieve the complete key for the value.

6.3 Adaptations and Functional Improvements for the Entangeld DHT

The following additions to the DHT are based on Kademlia and Entangled.

6.3.1 Improvements on Existing Functionality

The ”iterativeFind” method now takes an extra argument that allows the delete, overwrite and append RPC methods to be executed with said additional argument when called through ”iterativeFind”. This additional argument is used to send a signature and a timestamp along with the message.

When a value is inserted published into the DHT the publisher’s public key is appended to the value. This public key is used to verify signatures of RPC messages that ask the node to alter or delete a value in its storage.

The delete RPC now has a signature and a timestamp as arguments. These are checked by the delete RPC methods of nodes that receive a delete RPC message before the value is deleted.

6.3.2 New Functionality

The DHT module node adds an overwrite RPC and an append RPC. The overwrite RPC works similarly to the improved delete RPC. The overwrite RPC contains the RPC keyword ”overwrite” with the new value, a timestamp and a signature as RPC arguments. The overwrite RPC method checks the signature and the timestamp before overwriting the value.

6.4 Higher Level Functionality

All of the following methods are part of the DHT module node.

The ”publish_address_trusted” method implements the protocol for publishing a nodes contact data to trusted peers through the DHT. It retrieves the public keys of trusted peers from the petname table and creates a new cryptography module instance with a newly generated symmetric key. This cryptography module instance is used to encrypt the node’s IP address and port. It then encrypts the symmetric key with its trusted peer’s public keys and adds the encrypted contact data and the encrypted symmetric keys to a dictionary. This dictionary is serialized and published under the node’s ID.

The ”resolve_address_trusted” reads the IP address and port of trusting peers from the DHT. It is called for every peer who’s address is to be resolved using the peer’s ID to retrieve the serialized dictionary with the encrypted symmetric keys and the encrypted contact data. The method then attempts to decrypt the encrypted symmetric keys until the correct one is found, decrypts the contact data and relays it to the petname table where it is stored in the contact dictionary.

The ”post_addrequest” method takes a peering-ID, a name, a personal message and optionally a public key as parameters. It composes a trust exchange request message from the name and the personal message. If a public key was given, it generates a symmetric key, encrypts the trust exchange request message with the symmetric key and encrypts the symmetric key with the public key. The trust exchange request message is now published to the DHT with the peering-ID as key.

The ”poll” method is called periodically and retrieves the value behind the users peering-ID. The retrieved data is decrypted and relayed to the petname table where it is saved to the request dictionary until the user manually reviews the request.

6.5 Additional Entities

The following entities are also part of the DHT module but are not based on Kademlia or Entangled at all.

6.5.1 Petname Table

The petname table contains two dictionaries: The contact dictionary that maps petnames to the IP address, port and public key of a user and the request dictionary that contains DHT trust exchange request messages. These messages contain the same contact information that are saved in the contact dictionary in addition to a personal message intended to convince the user to accept the trust exchange.

The most notable methods are ”append_from_dht”, ”map_name” and ”add_AddRequest”. ”append_from_dht” takes the resolved addresses of contacts and updates the contact dictionary accordingly. ”map_name” takes a petname and returns the associated contact data. ”add_AddRequest” takes a trust exchange request message and adds it to the request dictionary.

6.5.2 Cryptography Module

The cryptography module is responsible for all cryptographic operations in the DHT module. It can be instantiated with just a public key, an asymmetric key pair, a symmetric key or with the instructions to generate a new asymmetric key pair or symmetric key. It can not hold an asymmetric key pair and a symmetric key at once. If both are required, two instances of the module are used. The methods of ”encrypt”, ”decrypt”, ”hash”, ”sign_value” and ”verify_signature” work differently internally depending on what type of keys the module was instantiated with. ”sign_value” and ”verify_signature” only work with a private key or a public key respectively. If the module was instantiated with a symmetric key it uses AES, if it was instantiated with asymmetric keys it uses RSA. It also provides a hash method that works independently of its key. The hash method uses sha-1. It uses sha-1 because sha-1 is used by Kademlia to generate keys from strings and it is not a secure cryptographic hash.

6.6 Node Interface to Node Interaction

The system instantiates a Node Interface Object which in turn creates a new thread in which a node is run.

The node interface communicates with the node through two queues. A task queue and a result queue. The node interface fills the task queue with command objects that tell the node which method to execute, including arguments. The node checks the task queue periodically and executes the methods requested by the command objects. If the method has a return value it is saved to the result queue.

Chapter 7

Evaluation

7.1 Performance

7.2 Security and privacy

This section evaluates how vulnerable the system is to the attacks defined in section 4.5.

7.2.1 Passive Observer Attacks

All data that is not intended to be public is encrypted. Metadata, including the type of RPC that is part of a message, could not be encrypted as this would require a successful key exchange. The authenticity of messages from arbitrary nodes of the DHT can not be verified and therefore a key exchange algorithm like Diffie-Hellman would be vulnerable to a man in the middle attack. Therefore metadata is readable for a passive observer.

7.2.2 Malicious Node Attacks

Malicious nodes could return false data upon receiving a request. This was solved for address resolution by adding a signature to the data to ensure authenticity. Since verifying signatures requires a prior trust exchange so the necessary public keys are available, trust exchange requests and relay service offers can be spoofed.

Malicious nodes could also overwrite another users data to disrupt functionality or for spoofing. This was successfully prevented by appending a public key to inserted data, which is used to check the signature of an overwrite or delete RPC before it is executed. Timestamps were used successfully to prevent replay attacks from circumventing the need for producing a valid signature.

7.3 Meeting Requirements

This section examines how well the requirements specified in section 4.3 were met.

7.3.1 Meeting Functional Requirements

To test if functional requirements have been met automated use case test scripts were created. These scripts mostly run the protocols defined in section 5.3. Every use case is executed by two scripts; one for each node. The first steps are always the same. Two nodes called Alice and Bob are initialized and create a network with them as members. The initialization includes the generation of an asymmetric key pair and the initialization of a petname table.

The first set of scripts implements the address resolution for trusted peers. In this scenario, Bob and Alice have already completed a trust exchange. Bob publishes his contact data for Alice, Alice retrieves the data from the DHT and decrypts it. She can check Bob's signature to authenticate him. Bob's contact data is saved to Alice's petname table, which already contains Bob's public key as he is a trusted peer. This test shows that the following requirements have been met: The system can resolve addresses of peers and authenticate them (FR1), it allows the user to limit who can resolve his address (FR2) and the system is capable of trust management (FR3). Alice can create an arbitrary amount of asymmetric key pairs to use as basis for different aliases and publish her address to users with whom she completed a trust exchange using one of these other key pairs. This shows that the system allows to change an alias or create multiple aliases (FR5).

The second set of scripts implements the DHT trust exchange. Bob and Alice have not completed a trust exchange, but Alice has either publicized her peering-ID and public key or shared them with Bob. Bob puts a trust request, encrypted with Alice's public key, into the DHT and Alice retrieves and decrypts it. The request is saved to the request table of the petname table, awaiting manual review by Alice. This test shows that trust exchange through the system (FR4) is possible.

The third set of scripts implements the overwrite and delete functionality. Bob saves a value to the DHT, Alice attempts to delete it but fails. Bob attempts to change or delete it and succeeds. This shows that the system allows users to modify or delete data they published previously (FR6).

7.3.2 Meeting Non-Functional Requirements

Since all published data contains a signature by default the authenticity of nodes that publish, modify data can be verified (NFR3). Confidential data, like published addresses, is encrypted and can only be decrypted by authorized users. If a user can read this data he is implicitly authenticated as trusted peer as he would not possess the necessary key otherwise.

Changing aliases can be done at will and adding new ones is as quick as generating a new key pair and performing a put operation. Therefore adding aliases is very quick (NFR4).

Since the module is based on a distributed hash table it is inherently distributed. It doesn't require a centralized authority(NFR5).

The system requires very little user input (NFR6) as resolving the address of trusted users and publishing the users own address to trusted peers can easily be automated and performed on start-up or in arbitrary intervals. The only actions that require user input are initiating a trust exchange through the DHT and reviewing a trust exchange request message.

The address resolution module can be replaced with another address resolution module (NFR7) as long as it implements the methods that can be called through the Node Interface.

Chapter 8

Conclusion and Outlook

To develop the address resolution module a detailed analysis regarding expected functionality, requirements and security threats was conducted. The choice of established technology to be used as a basis for the module fell on a distributed hash table. Use cases defined the expected functionality.

A design for the address resolution module was established. First all necessary components were introduced and their interaction was defined. The chosen DHT implementation was Entangled, which is based on Kademlia. The DHT module expanded on the functionality provided by the base technology by building a DHT module node from the Entangled node and was supported by several other components: A petname table to hold the contact data of trusted peers, a cryptography module for cryptographic operations and an interface that handles communication between the DHT module node and the rest of the system.

To provide security and privacy a set of security features were specified and protocols to implement the functionality specified in the use cases during analysis were created with security in mind.

Based on this design a prototype was implemented, tested and evaluated based on the requirements specified earlier.

Not all features are working as of yet. RPCs can not be called with an additional argument to hold the signature and timestamp yet as the relevant method, the iterativeFind method taking the arguments key and rpc, is deeply interconnected with many other areas of the code and runs a high risk of breaking the module if modified. An alternative approach is to send RPCs that require additional arguments as part of the value in the store RPC call. The local store RPC could be modified to work with a list of arguments saved in the value-variable.

Saving a list of arguments to other variables than the store RPC calls' value-variable would break functionality as the key-variable is used for routing and the rpc-variable determines which RPC call will be used. As such these variables can not be modified.

In addition to that, the DHT address resolution module has yet to be tied into the greater system.

Chapter 9

Glossary

- *DHT*: Distributed Hash Table. A peer-to-peer network of nodes that offer storage and routing functionality to the network.
- *ID*: The ID is used as the DHT key for publishing the users address to trusted peers. It is a hash of the nodes public key.
- *Peering-ID*: The peering-ID is used by other users as the DHT key for publishing trust exchange request messages to the peering-ID's owner. It is a hash of an arbitrary string or the hash of a GPG public key.
- *Node-ID*: The node ID is used to identify nodes in the DHT. It can be chosen by the node itself, but it has to be unique in the network. It uses the same format as lookup-keys and is used for routing.
- *RPC*: Remote Procedure Call. A message sent from one node of the DHT to another to make the other node call a method.

Bibliography

[1] C. Brosche, “A modular framework for a collaborative solution for privacy-preserving network communication,” Bachelor Thesis, Technische Universität München, 2015.

[2] B. Schöntag, “Integrating voice over ip into a privacy friendly network,” Bachelor Thesis, Technische Universität München, 2015.

[3] G. Danezis, J. Domingo-Ferrer, M. Hansen, J.-H. Hoepman, D. L. Métayer, R. Tirtea, and S. Schiffnerr. (2015) Privacy and data protection by design. Visited 2015-8-9. [Online]. Available: https://www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/privacy-and-data-protection-by-design

[4] M. Rost and A. Pfitzmann. (2009) Datenschutz-schutzziele – revisited. Visited 2015-8-9. [Online]. Available: http://maroki.de/pub/privacy/DuD0906_Schutzziele.pdf

[5] M. Hansen and S. Meissner. (2007) Verkettung digitaler identitäten. Visited 2015-8-9. [Online]. Available: https://www.datenschutzzentrum.de/uploads/projekte/verkettung/2007-uld-tud-verkettung-digitaler-identitaeten-bmbf1.pdf

[6] P. Mockapetris. (1987) rfc1035. Visited 2015-8-9. [Online]. Available: https://tools.ietf.org/html/rfc1035

[7] P. Maymounkov and D. Mazières. (2002, May) Kademlia: A peer-to-peer information system based on the xor metric. Visited 2015-8-9. [Online]. Available: http://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf

[8] H. Kinkelin. (2013) Autonomous and robust components for security in network domains. Visited 2015-8-9. [Online]. Available: http://www.net.in.tum.de/fileadmin/bibtex/publications/theses/NET-2013-10-1.pdf

[9] H. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan. (2015) An empirical study of namecoin and lessons for decentralized namespace design. Visited 2015-8-9. [Online]. Available: http://randomwalker.info/publications/namespaces.pdf

[10] I. Clarke, O. Sandberg, M. Toseland, and V. Verendel. (2010) Private communication through a network of trusted connections: The dark freenet. Visited 2015-8-9. [Online]. Available: https://freenetproject.org/papers/freenet-0.7.5-paper.pdf

[11] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. (2001) Freenet: A distributed anonymous information storage and retrieval system. Visited 2015-8-9. [Online]. Available: http://homepage.cs.uiowa.edu/~ghosh/freenet.pdf

[12] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Visited 2015-8-9. [Online]. Available: https://svn.torproject.org/svn/projects/design-paper/tor-design.html

[13] Tor: Hidden service protocol. Visited 2015-8-9. [Online]. Available: https://www.torproject.org/docs/hidden-services.html.en

[14] D. Herrmann, K.-P. Fuchs, J. Lindemann, and H. Federrath. (2014) Encdns: A lightweight privacy-preserving name resolution service. Visited 2015-8-9. [Online]. Available: https://svs.informatik.uni-hamburg.de/publications/2014/2014-07-30-HFLF14-ESORICS14-EncDNS.pdf

[15] L. Maccari, M. Rosi, R. Fantacci, L. Chisci, L. M. Aielloy, and M. Milanesioy. (2009) Avoiding eclipse attacks on kad/kademlia: an identity based approach.

[16] A. Singh, T.-W. J. Ngan, P. Druschel, and D. S. Wallach. (2006) Eclipse attacks on overlay networks: Threats and defenses. Visited 2015-8-9. [Online]. Available: http://www.csd.uoc.gr/~hy554/papers/eclipse-infocom06.pdf

[17] X. Yue, X. Qiu, Y. Ji, and C. Zhang. (2009) P2p attack taxonomy and relationship analysis. Visited 2015-8-9. [Online]. Available: http://www.minelab.cn/downfile/20125217544488063.pdf

[18] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach. (2002) Secure routing for structured peer-to-peer overlay networks. Visited 2015-8-9. [Online]. Available: http://zoo.cs.yale.edu/classes/cs426/2013/bib/castro02secure.pdf