Bernd Prünster, BSc

Griđ
—
Secure Information Exchange Based on an Inverted Trust Model

Master's Thesis

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Computer Science

submitted to

Graz University of Technology

Supervisor

O.Univ.-Prof. Dipl.-Ing. Dr.techn. Reinhard Posch

Institute for Applied Information Processing and Communications
(IAIK)

Advisor
Dipl.-Ing. Dr.techn. Peter Teufl

Graz, January 2016

Abstract

The technological foundation of today's global communication stack was designed decades ago without security in mind. Nowadays however, digital certificates and systems like Pretty Good Privacy enable everyone to confidentially transmit information by means of public-key cryptography. Yet, even such sophisticated frameworks still do not put the sender in control of the most important aspects of a confidential information exchange. Today's established systems rely on a rigid trust model which forces the sender to trust the recipients' public keys. This can result in accidental exposure of sensitive information through compromised keys or insecure cryptographic algorithms. Additionally, while a transmission's payload may be easy to protect, meta data can still yield valuable information to eavesdroppers. Although anonymity networks such as Tor can help remedy this issue, the rigid trust model remains.

This work introduces Grið, a system aimed at secure information exchange based on an inverted trust model which enables communication parties to remain anonymous to the network. Instead of relying on trust in recipients' public keys, Grið lets the encrypting entity create its own key material to encrypt information for others. Recipients must subsequently contact the sender and authenticate themselves in order to gain access to this key material. By relying on Tor as its communication layer, Grið enables this information exchange to be carried out while remaining anonymous to the network. The incorporation of a highly flexible authentication framework based on CrySIL results in senders being able to freely define authentication mechanisms. This way, the encrypting entities can control the security level of an information exchange. By evaluating the developed system with respect to its security properties and against existing designs, the utility of its novel approach has been demonstrated. It can therefore be concluded that by putting the sender in full control Grið constitutes an improvement over established systems in certain situations.

Zusammenfassung

Die Technologien, auf deren Fundament die globalen Kommunikationsnetze von heute aufbauen, bieten von sich aus keine der mittlerweile üblichen Sicherheitsmechanismen. Seit einigen Jahren lassen sich jedoch basierend auf asymmetrischer Kryptografie gezielt Daten für einzelne Empfänger verschlüsseln. Meist wird dabei auf digitale Zertifikate oder Systeme wie Pretty Good Privacy zurückgegriffen. Allerdings haben diese Ansätze eine entscheidende Schwäche: Das maximal erreichbare Sicherheitsniveau wird vom öffentlichen Schlüssel des Empfängers bestimmt. Daher können sensible Daten auch ohne Fehlverhalten des Senders beispielsweise durch kompromittiertes Schlüsselmaterial oder unsichere kryptografische Verfahren in falsche Hände gelangen. Unabhängig davon lassen sich mit derartigen Verfahren sehr wohl die Nutzdaten, jedoch nicht die Metadaten, verschlüsseln. Anonymisierungsnetzwerke wie Tor können hier Abhilfe schaffen; die Schwächen des zugrundeliegenden Vertrauensmodells können dadurch allerdings nicht ausgeräumt werden.
Diese Arbeit stellt mit Grið ein System zum sicheren und anonymen Informationsaustausch basierend auf einem invertierten Vertrauensmodell vor. Anstatt den öffentlichen Schlüsseln von Empfängern vertrauen zu müssen, ist der Versender von Informationen selbst für das Schlüsselmaterial verantwortlich. Empfänger müssen sich dem Sender gegenüber authentifizieren und erhalten so Zugriff auf dieses Schlüsselmaterial. Um außerdem die Anonymität aller Kommunikationspartner zu garantieren, wird Tor als Transportschicht eingesetzt. Zusätzlich wird durch die Integration von CrySIL ein hochgradig flexibles Authentifikationsframework bereitgestellt, um entsprechende Sicherheitsniveaus erreichen zu können. Das vorgestellte System wurde des Weiteren einer Sicherheitsanalyse unterzogen um dessen Praxistauglichkeit zu evaluieren. Durch anschließende Vergleiche mit bestehenden Systemen konnten die Vorteile des neuartigen Ansatzes eines invertierten Vertrauensmodells dargelegt werden.

Affidavit

Acknowledgements

First, I would like to thank my advisor Peter Teufl for his continuous support through all the hours of productive discussions and long sessions of tackling conceptual issues, for his guidance, and his invaluable feedback.

I would also like to thank Christopher Liebmann for taking the time to read through certain chapters of this thesis individually and providing me with detailed feedback about almost every single line of argument. Furthermore, I owe my thanks to him for his willingness to thoroughly test the implemented prototype which made it possible to discover and subsequently fix many issues.

My biggest thanks, however, go to my other half, Stephanie Hüttl, who brought me back on track and encouraged me to really put my heart and soul into my studies. Without her support and encouragement over the past years I certainly would not have finished my studies in time and I probably never would have discovered my passion for information security.
In addition, I feel obliged to thank her for the countless hours she was willing to invest in proofreading this thesis and for not holding my sometimes horrible time management against me.

Contents

List of Figures

(Unless explicitly declared otherwise, all figures were created by the author.)

List of Tables

List of Listings

1 Introduction

Today, the Internet is more important than ever. E-mail and instant messaging have been established as communication media years ago and increasingly more information is being exchanged electronically – a trend which is expected to continue in the years to come [Cisco Systems, Inc., 2015]. Unimaginably large amounts of data containing an enormous amount of information are being transmitted on a daily basis. The underlying technologies enabling this fast-paced information exchange, however, were designed at a time where the digital landscape looked significantly different. For instance, the *Internet Protocol* (IP), one of the Internet's fundamental building blocks, was developed decades ago [Postel, 1981]. When examining the technological standards from that era, it becomes apparent that security was not a priority when designing such important parts of today's global communication stack. The Internet protocol does not have any security mechanisms built-in [Postel, 1981, p. 3] – a separate protocol aimed at securing IP was proposed for standardisation more than two decades later [Atkinson, 1995].

Nowadays, various ways of securing information exchange carried out over insecure channels are at everyone's disposal: The Tor anonymity network [Dingledine, Mathewson and Syverson, 2004], for example, enables users to anonymously browse the web, anonymise arbitrary TCP traffic and even anonymously host web services. However, the simple task of transmitting some secret information to another party is often still problematic. While the discovery of public-key cryptography, originally published in 1976, proved to be revolutionary by making it possible to selectively encrypt files for individual entities [Diffie and Hellman, 1976, p. 648], it also does not solve all problems: Given a party's public key, how can the encrypting entity make sure that the recipient has really kept the corresponding private key (required to recover some previously encrypted data) secret? Even under the assumption that a private key is never disclosed to a third party, a secure information exchange might not be possible due to the properties of the public/private key pair: It is easily conceivable that the cryptographic algorithm or the length of the key does not provide a security level adequate for the transmission of some highly sensitive data. In such a case, the sender cannot make sure that the encrypted data is safe from being recovered by unauthorised third parties. This trust model, based on senders expressing trust in a recipient's public key, however, forms the basis of both *Pretty Good Privacy* (PGP) [Garfinkel, 1995, p. 235] as well as *public key infrastructures* (PKIs) involving trusted third parties [Stallings, 2013, pp. 431–435]. In a situation where an information exchange has to be carried out anonymously, additional factors become relevant, which further complicates such endeavours: Ideally, familiar infrastructures should be used, while at the same time no traces should be left which could enable the de-anonymisation of communication parties. Due to the number of relevant factors, successfully exchanging encryption keys and transmitting a cryptogram while remaining anonymous poses a considerable challenge – especially if multiple services which poorly integrate with each other need to be used.

This work seeks to offer a novel solution to the problem of enabling secure information exchange by relying on an inverted trust model. A prototype illustrating this concept, called *Grið*, has been implemented. In scenarios involving traditional public-key cryptography, the security level of any encrypted data addressed to an entity is essentially dictated by this entity's public key. The aforementioned inverted trust model overcomes this issue as follows: Instead of expressing trust in recipients' public keys, the sender encrypts some to-be-delivered data using a randomly generated key and transmits it to the intended recipients. These are then given access to this key by authenticating themselves to the sender. Consequently, the sender not only always remains in total control over all involved key material, but is also able to freely define the security level for such an information exchange by opting for an appropriate authentication challenge.
In order to evaluate possible authentication methods, key transport approaches, and established trust models need to be examined. This also encompasses the very fundamentals of public-key cryptography, as well as the underlying principles of digital certificates which make up the foundation of *Public Key Infrastructures using X.509* (PKIXs). However, basic transfer formats to ease key management are also considered relevant prerequisites to designing a system aimed at secure information exchange.

Another goal of Grið is to implement secure information exchange while enabling communicating parties to remain anonymous to the network. Naturally, recipients must be able to contact the sender through some secure channel in order to perform the previously outlined authentication procedure in private. To be able to achieve this goal, the sender must essentially be reachable in the same way as a web-server, regardless of the network situation. *Peer-to-peer* (P2P) approaches come to mind as a possible solution to this problem. Therefore, P2P networking principles, different classes and implementations of such networks and their properties need to be examined. Most importantly, possible threats originating from the peer-to-peer approach and corresponding countermeasures need to be considered. It should be obvious why this is of great importance for a design aimed at secure information exchange. Given the fact that an anonymous communication is aspired, ways of anonymising traffic (especially in the P2P context) have to be explored as well. Apart from research in this area and general concepts, existing anonymity networks (preferably deployed on a global scale) shall also be evaluated. An important aspect of such an evaluation is the ability of a network to be integrated into other systems. In the case of Grið, a network layer directly providing the ability to establish anonymous P2P connections is desired. Ideally, this should be achievable by integrating a widely used, thoroughly scrutinised anonymity network.

In the best case, the developed prototype will also perform well in real-world network topologies. Essentially, this means that it should be possible to connect to another party regardless of its network situation, while enabling both participants to remain anonymous to the network. However, typical high-level network designs (including P2P networks) often do not deal with issues such as *network address translation* (NAT). This is understandable, given the sheer complexity of the matter in a real-world scenario on real-world hardware. As previously mentioned, the Tor anonymity network enables users to anonymise traffic and anonymously host services. It is therefore considered a prime candidate for providing a solution to these problems. Ideally, through the incorporation of Tor as a transport layer, Grið will be able to offer secure communication channels to perform authentication procedures and subsequent key transfers. Relying on Tor will of course also result in all communication being anonymous to the network.

In order to transfer further control to the encrypting communication party, a flexible authentication framework needs to be incorporated into Grið. The *Cryptographic Service Interoperability Layer* (CrySIL) [Reimair, Teufl and Zefferer, 2015] offers such an authentication framework. Enabling the encrypting entity to freely define an authentication procedure further empowers the sender. As already discussed, this approach can remedy many issues with established trust models. In this situation, the only remaining trust issue is the trustworthiness of the recipient. In other words: the human factor. However, such procedures need to be carefully designed in order to fulfil all requirements. The communication workflow between sender and recipient as well as a sensible container format for the actual information exchange need to be defined. In addition, key management issues have to be overcome in order to reach satisfactory results.

The desired feature-set is considered an improvement over established approaches to secure information exchange such as PGP as well as niche solutions like *OnionShare* [Lee, no date]. Naturally, the actual performance of the resulting system needs to be thoroughly evaluated in order to reach a verdict about its utility. Lastly, Grið also needs to be compared to existing solutions aimed at secure information exchange in order to determine its significance.

2 Key Distribution, Public Key Infrastructures, and Trust

Today's digital information exchange mechanisms heavily rely on cryptography to provide security services such as confidentiality and integrity. The growing security-awareness of users and continuously increasing global surveillance further foster research and development focusing on secure information exchange. Consequently, access control is becoming more important and thus, choosing appropriate authentication mechanisms is crucial.
In order to be able to reason about such complex concepts, more fundamental ones such as the key distribution problem and authentication methods need to be discussed, as these concepts are universal. However, before exploring the aforementioned issues, a terminology of the most basic concepts shall be defined based on Stallings [Stallings, 2013].

Firstly, the properties of *symmetric* and *asymmetric* encryption schemes shall be clarified: Symmetric encryption algorithms (also called *symmetric-key* algorithms, sometimes *private-key–* or *secret-key cryptography*) make use of only a single cryptographic key for both encryption and decryption. Therefore, this key needs to be kept secret to keep unauthorised third parties from being able to recover encrypted data.
Asymmetric algorithms are commonly referred to as *public-key cryptography* or *asymmetric cryptography*, since these encompass much more than simple encryption and decryption of data. Public-key cryptography algorithms do not employ a single secret key, but a *public/private key pair*. As the name implies, this key pair consists of a *public key* and a *private key* where the private key needs to be kept secret and must not be shared. In fact, every entity has its own personal key pair; the public key is used by others to encrypt data designated for an entity, who can then use its private key to decrypt and recover the previously encrypted data. Obviously, the public key needs to be known in order to encrypt data for an entity and is therefore made public.

Secondly, *digital signatures* shall be explained briefly. A digital signature can be used to provide non-repudiation similar to hand-written signatures and must therefore hold up to similar constraints. Most importantly, it must be computationally infeasible to forge a digital signature, but at the same time it must be possible to efficiently verify a given signature. Digital signatures are built upon asymmetric cryptography: Through clever application of the underlying algorithms, it is possible to swap the use of private and public keys and thus enable the creation of electronic signatures derived from the data and a private key [Stallings, 2013, p. 394]. Anyone can verify a signature by performing the complementary operation under the corresponding public key. If verifiers know to whom a public key belongs, they can be certain that the data was actually signed by the proclaimed signatory and that it has not been altered.

Finally, the notion of a *cryptographic hash function* and the concepts related to it shall be explained. Hash functions map inputs of arbitrary length to *hash values* (also called a *hash* or *digest*) of fixed length. A cryptographic hash function can be defined as "an algorithm for which it is computationally infeasible [. . .] to find either (a) a data object that maps to a pre-specified hash result (the one-way property) or (b) two data objects that map to the same hash result" [Stallings, 2013, p. 314].

Having defined the necessary cryptographic fundamentals, the concepts upon which secure communication techniques build can be elaborated on: the problem of distributing secret keys, public key infrastructures seeking to solve it, and the deep-seated issue of establishing trust in other entities.

2.1 The Key Distribution Problem

Whenever information is to be exchanged securely, various cryptographic algorithms are applied on this information prior to transmitting it in order to ensure its security and integrity. While this observation is an obvious one, the question of how to distribute the cryptographic keys needed to encrypt and decrypt information is far from trivial. This is commonly referred to as the *key distribution problem.*
Even the conceptually simplest approach, letting all parties share the same key, does not answer the question of how to actually distribute the required key material to all parties.[1] Therefore, agreeing on a key management– and key distribution scheme is essential.

According to Stallings [Stallings, 2013], ways for two communicating parties A and B to exchange secret keys fall into four categories:

1. A can select a key and physically deliver it to B.
2. A third party can select the key and physically deliver it to A and B.
3. If A and B have previously and recently used a key, one party can transmit the new key to the other, encrypted using the old key.

[1]It should be obvious, that sharing a single key between all communication parties is impractical in most situations.

4. If A and B each has an encrypted connection to a third party C, C can deliver a key on the encrypted links to A and B. [Stallings, 2013, p. 419]

Depending on the situation, any one of the above workflows can be useful. If, for example, A and B meet on a daily basis, a physical exchange of encryption keys can be feasible. Such a scenario also has a noteworthy advantage: It requires considerably more effort for an attacker to compromise a physical key exchange compared to any telecommunication-based key exchange protocol. Basic wire-tapping techniques can be enough to gain knowledge about a key which is transmitted over electronic communication channels. In contrast, a manual hand-over of a storage medium (such as a USB flash drive, for example) containing encryption keys can only be subverted if an attacker is able to physically interfere. In addition, authentication is usually a non-issue whenever parties meet in person.
The same properties also hold for physical key distribution through a (trusted) third party. However, such procedures are obviously only applicable if it is possible for the involved parties to meet in person. Furthermore, any time overhead and potentially long delays between key exchange and initiation of communication must also be acceptable. In addition, purely secret-key-based schemes don't scale well in general: A communication network of n parties requires $\mathcal{O}(n^2)$ keys in case any two communication parties need to exchange keys.

Key exchange based on previously used encryption keys is not subject to the aforementioned limitations and can therefore also be accomplished for many communicating parties which may be separated by large geographic distances. However, "if an attacker ever succeeds in gaining access to one key, then all subsequent keys will be revealed. Furthermore, the initial distribution of potentially millions of keys still must be made" [Stallings, 2013, p. 419].

To combat the challenges faced with the distribution of secret keys, public-key cryptography can be employed. Diffie and Hellman [Diffie and Hellman, 1976, p. 649] proposed a scheme based on the *discrete logarithm problem* (DLP), which is now known as the *Diffie-Hellman key exchange* (DHKE) in which both parties are able to contribute to and agree upon a shared key over an public communication channel. Using such methods does not require any key material to be exchanged prior to key agreement and thus eliminates the key distribution problem.

Due to the public/private key pairs being bound to entities, public-key cryptography scales exceptionally well: A communication network of n parties requires only $\mathcal{O}(n)$ keys. However, asymmetric encryption algorithms are typically orders of magnitude slower than symmetric ones [Diffie, 1988, p. 566]. Luckily, this drawback can easily be worked around: Public-key cryptography is typically used to securely exchange secret keys which are subsequently used for bulk encryption. This is referred to as a *hybrid* cryptosystem [Buchmann, Karatsiolis and Wiesmaier, 2013, pp. 10-11]. Additionally, *(perfect) forward secrecy* can be achieved by combining this approach with the DHKE using ephemeral keys.
However, common sense already suggests that without authentication mechanisms in place,

an attacker can easily impersonate a legitimate party and, by extension, mount *man-in-the-middle* (MITM) attacks. Therefore, authentication has to take place prior to the key-agreement/key-exchange procedure. This can either be done manually, as it is the case with *Pretty Good Privacy* (PGP), where parties are obliged to explicitly express trust in other entities' public keys [Garfinkel, 1995, p. 235], or by means of a trusted third party, which vouches for the authenticity of the communication parties [Stallings, 2013, pp. 431–435]. The latter approach is typically realised using a *public key infrastructure* (PKI) based on digital certificates.

2.2 Public Key Infrastructures

Shirey [Shirey, 2000] defines a (X.509-based) *public key infrastructure* (PKI) as "The set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography" [Shirey, 2000, p. 137]. Digital certificates enable entity authentication by means of trusted third parties, called *certificate authorities* (CAs) without the need to contact these during the verification process. However, in order to illustrate the reason why certificate-based authentication has become the predominant way of automatically authenticating entities, public key directories need to be examined first.

Common to all public-key-based authentication methods is the binding of an entity's public key to the entity's identifier. As soon as it is possible for a communication party to verify that the public key announced by another party actually belongs to this party, any insecure channel can be used for key agreement. Parties can thus authenticate themselves to others. One of the simplest approaches to achieve this goal is to introduce a public key directory trusted by all communication parties: Entities can register their public keys with the directory (requiring some form of identity proof) and by simply checking the announced public keys against the directory entries, man-in-the-middle attacks can effectively be prevented [Stallings, 2013, pp. 431–432]. While this already solves the problem of attackers being able to impersonate other entities, it also suffers from a serious flaw: If the public key directory becomes unavailable at any point in time, authentication processes cannot be carried out anymore. Digital certificates eliminate this availability constraint by relying on digital signatures.

2.2.1 Digital Certificates

Digital certificates are the predominant concept used for entity authentication as defined by the International Telecommunication Union [International Telecommunication Union, 2012] in the form of *Recommendation X.509*. Versions of this recommendation have already been standardised by the *Internet Engineering Task Force* (IETF) as early as 1999 [Housley et al., 1999] and have since become widely accepted. PKIs based on these standards are commonly referred to

as *Public Key Infrastructures using X.509* (PKIXs).
As part of a PKI, digital certificates enable entity authentication through a trusted third party (the certificate authority), by verifying a digitally signed statement of an entity's public key and its identifier against the public key of a CA. Thus, once an entity expresses trust in a CA's public key, others can prove their identity to the entity by presenting a certificate containing their public key and identity signed by the CA.
Buchmann, Karatsiolis and Wiesmaier [Buchmann, Karatsiolis and Wiesmaier, 2013] summarise this as follows:

> Suppose that user Alice wishes to verify a digital signature issued by user Bob. She queries a directory service and obtains Bob's public key. [. . .] Alice needs to convince herself of the authenticity of that public key. This is what certificates are used for. Certificates are data structures that bind public keys to entities and that are signed by a third party. If Alice has a certificate for Bob's public key and if Alice trusts the third party that signed the certificate and also trusts the signature verification key of the third party, then verifying the signature of the certificate convinces Alice of the authenticity of Bob's public key. In this way, certificates reduce the trust in a public key of an entity to the trust in some authority. [Buchmann, Karatsiolis and Wiesmaier, 2013, p. 21]

Not only does the CA not need to be available during certificate verification, it is also irrelevant how the verifier obtained the certificate, since it is a signed statement which can be verified in any case. Figure 2.1 illustrates the process of certificate creation as well as certificate validation based on Stallings [Stallings, 2013, p. 436].
Realistically, a signed statement containing only a public key and an entity identifier is not enough. Most importantly, validity periods are also part of any certificate and revocation mechanisms exist in order to be able to account for the possibility of fraud, theft of private keys, or simply distrust in an entity. The structure of a digital certificate is illustrated in Section 2 of *Recommendation X.509* by the International Telecommunication Union [International Telecommunication Union, 2012, pp. 12–13].

Although authentication and key distribution have been discussed, the actual exchange of signed and/or encrypted data has not been touched, since these issues can be considered application-specific. Yet, widely used, standardised ways exist since 1999 in form of the *Cryptographic Message Syntax* (CMS) [Housley, 1999] to perform such operations in an application-independent manner.

2.2.2 Cryptographic Message Syntax

The *Cryptographic Message Syntax* (CMS) "is used to digitally sign, digest, authenticate, or encrypt arbitrary messages. The Cryptographic Message Syntax describes an encapsulation

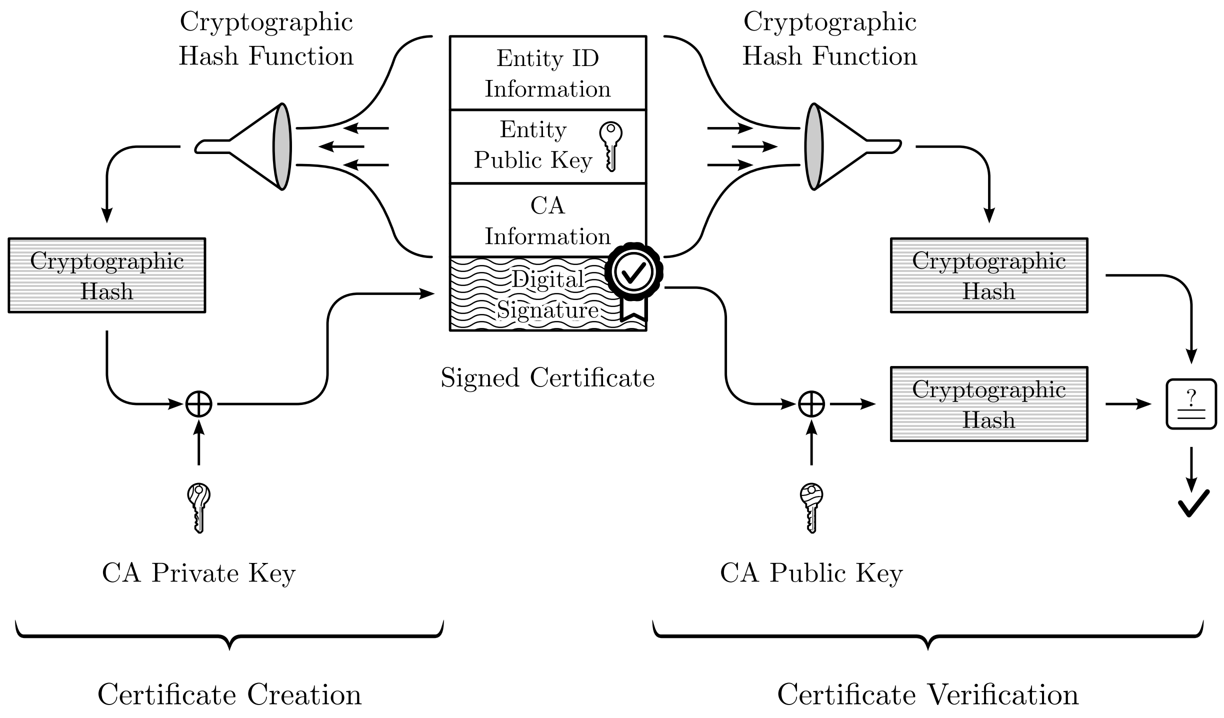


Figure 2.1: Creation and verification of digital certificates based on Stallings [Stallings, 2013, p. 436]

syntax for data protection" [Housley, 1999, p. 4] and is closely related to the PKIX framework. It defines an application-independent way of performing cryptographic operations on data and encapsulating the result into a well-defined container structure. Perhaps the best-known applications of the CMS standard are *Secure / Multipurpose Internet Mail Extensions* (S/MIME) and certificate life cycle management within PKIXs [Buchmann, Karatsiolis and Wiesmaier, 2013, p. 109].

A CMS-compliant data structure consists of two parts: a content type declaration and the actual content itself. The original specification defines six different content types (three of which are optional) for CMS containers which can be used to encapsulate and transfer arbitrary data. Due to its general nature, its versatility, and wide adoption, the structure of a CMS container and the mandatory content types shall be explained in more detail. These are: *data*, *signed-data* and *enveloped-data* [Housley, 1999, p. 4].

The *data* and *signed-data* content types are mostly self-explanatory. *Data* does not impose any constraints or force any preprocessing on the content. It is therefore not particularly useful on its own. *Signed-data*, on the other hand, includes the original data, a signed cryptographic hash of this data and the certificate of all signatories who signed the data [Housley, 1999, p. 6].

CMS also supports encapsulating data encrypted using hybrid schemes illustrated in Section 2.1 through the *enveloped-data* content type:

The enveloped-data content type consists of an encrypted content of any type and

> encrypted content-encryption keys for one or more recipients. The combination of the encrypted content and one encrypted content-encryption key for a recipient is a "digital envelope" for that recipient. Any type of content can be enveloped for an arbitrary number of recipients [. . .].
> The typical application of the enveloped-data content type will represent one or more recipients' digital envelopes on content of the data or signed-data content types. [Housley, 1999, pp. 12–13]

Thus, enveloped-data CMS containers make it possible to encrypt arbitrary data under a one-time key using efficient symmetric encryption schemes, while only granting select entities access to this key without concerning oneself with key management and access control: In this context the one-time key is the content-encryption key and will be encrypted under the select entities' public keys. Consequently, no records of content-encryption keys need to be maintained by the creator of such an enveloped-data CMS container while only the holders of the private keys corresponding to the public keys used will be able to recover the content-encryption keys. Figure 2.2 illustrates this scenario in its simplest form (involving only a single recipient).

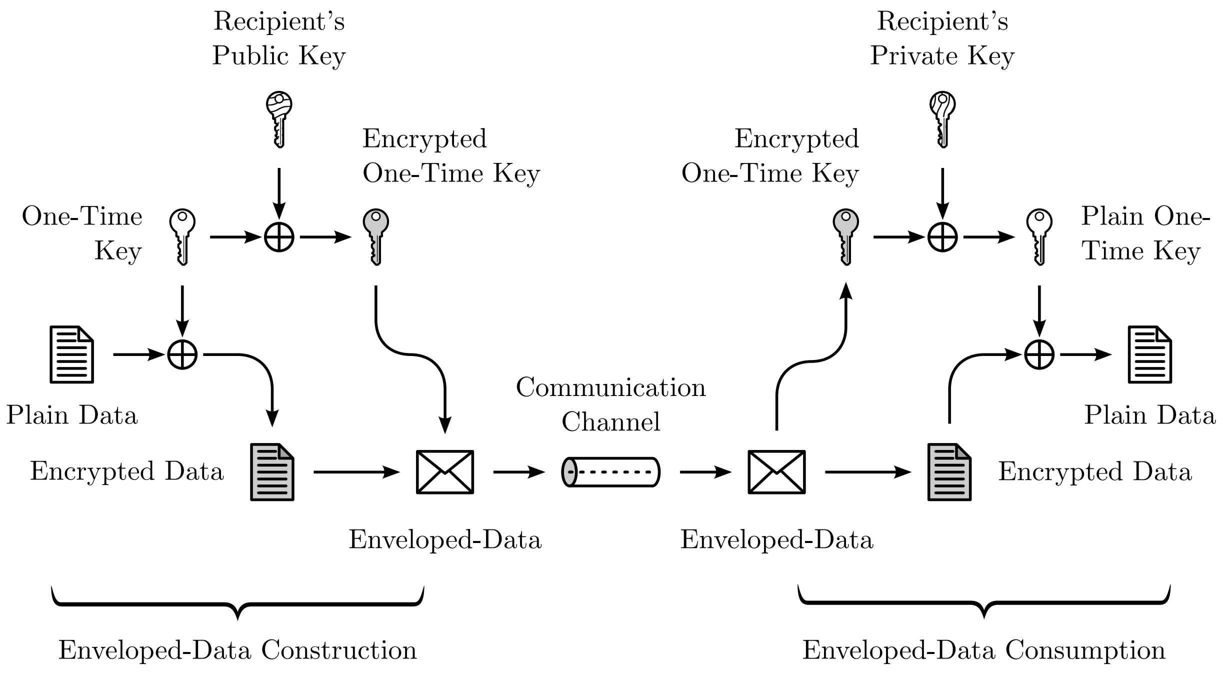


Figure 2.2: Construction and consumption of an enveloped-data CMS container for a single recipient

Up to now, no assertion has been made as to how trust is established. In the case of CAs, it has been shown that it is not necessary anymore to trust billions of entities and their public keys, but merely a small number of certificate authorities. The problem of establishing initial trust is still an open one for which no universally accepted solution exists, especially since trust is a highly subjective and deeply personal matter. Obviously, this issue is not limited to CAs but is also highly relevant on an application level and is therefore handled differently both depending on the application and its context. However, well-defined trust models and authentication schemes exist and shall therefore be discussed in more detail.

2.3 Trust

Trust can be established in many different ways. While technology can help ease the process, establishment of trust remains a non-technical, yet crucial issue. Still, well-defined technical protocols exist to (dis)prove the authenticity of data or identity claims. Although closely related, both issues can be discussed separately. Due to the general applicability, different trust models shall be elaborated on before exploring authentication methods, as the latter are somewhat application-specific.

2.3.1 Trust Models

In general, trust can either be established directly or through intermediaries, who vouch for the trustworthiness of others. Buchmann, Karatsiolis and Wiesmaier [Buchmann, Karatsiolis and Wiesmaier, 2013, pp. 39–50] discuss *direct trust*, *web of trust* and *hierarchical trust* as the main types of trust models. These models are universal and thus independent of authentication methods (obviously, ways of proving authenticity are necessary to establish trust). This, however, is precisely the highly subjective aspect of the overarching issue mentioned previously. Usually, trust is expressed in other entities' public keys. More precisely: Trust is expressed in the binding of a public key to an entity's (typically the recipient's) identifier regardless of trust model.

2.3.1.1 Direct Trust

Firstly, *direct trust* needs to be discussed, since it is needed to establish initial trust relations required by all other trust models [Buchmann, Karatsiolis and Wiesmaier, 2013, p. 39]. In essence, trust is considered to be established directly, if the information to be trusted is obtained from its owner. This does not mandate physical interaction; any (sufficiently trustworthy) communication channel can be used, however, no intermediaries who vouch for the trustworthiness of one of the parties must be involved during the trust-establishment process for it to hold up to the notion of direct trust. This can be illustrated by means of the following example:

> Like other operating systems, many Linux variants allow the installation of additional software such as updates or services from servers located on the Internet. The authenticity of those software packages is established by a digital signature. The verification of the signature requires a public key. Frequently, this key is provided on the original Linux distribution CD or DVD [...]. The authenticity of this key

> is guaranteed by the authenticity of the CD or DVD and by the difficulty of over-writing it. In such a situation, the trust in the public key is called *direct* since it is directly obtained from its owner (the Linux distributor). No third party is involved in authenticating the key. [Buchmann, Karatsiolis and Wiesmaier, 2013, p. 39]

Having discussed direct trust, indirect trust models involving third parties can be explored which rely on direct trust only for initialisation of a network of trust.

2.3.1.2 Indirect Trust

Direct trust is in general highly impractical: As already explained in Section 2.2.1, trusting intermediaries eliminates the need to trust each communicating party individually. Thus, indirect trust models are widely used. *Hierarchical trust* and the notion of a *web of trust* fall into this category. Perhaps the most prominent application of a web of trust is PGP, where users express *validity* and a *degree of trust* in other users' keys:

> The idea behind validity and trust is relatively simple: if you believe that a key is valid, you believe that only the person who created the key will be able to read a message encrypted with the key. If you believe that a person who creates a key is trustworthy and you find that person's signature on a third person's key certificate, you are likely to believe that the third person's key is itself valid. [Garfinkel, 1995, p. 235]

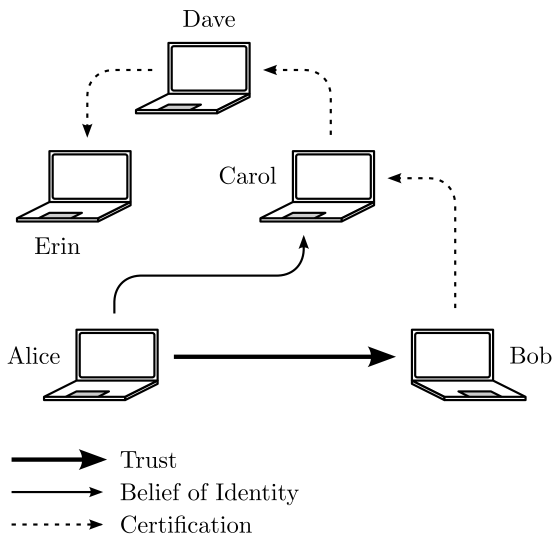
Applying this concept on a larger scale creates a directed graph of trust relations known as a *web of trust*. No central authority governs who to trust. Obviously, direct trust and manual user action are required to establish initial trust relations and to maintain the degree of trust expressed in other user's keys. While this potentially results in a considerable overhead, it also gives users complete control over who to trust and to what extend. This degree of trust also results in trust not being unconditionally transitive. A simple example of a web of trust is shown in Figure 2.3a.

In contrast to the web of trust, which is based on a social approach, *hierarchical trust* relies on a trusted central authority which vouches for the identity of all communicating parties. This reduces the overhead in creating and maintaining trust relations at the price of having less control over which individuals to trust. Most importantly, however, hierarchical trust models deal with the issue of liability:

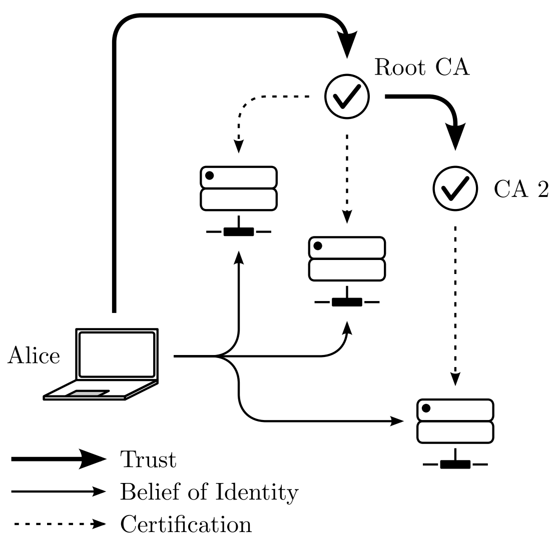
> Signers of public keys typically do not accept (legal) liability for the authenticity of the public keys which they sign. Therefore, the web of trust appears to be inadequate in a business context. For example, consider a user who verifies the authenticity of a home banking Web page before entering a secret password. He does this by verifying a digital signature of the Web page. In such a context it would be preferable that the public verification key for the signature of the Web page is certified by some authority

> that can be made liable if the key turns out not to be authentic. In the hierarchical trust model, certificate signers accept such liability. [Buchmann, Karatsiolis and Wiesmaier, 2013, p. 48]

PKIXs are the prime example of such a trust hierarchy: In order to keep the issuing, revocation and maintenance of certificates manageable, so-called *certificate chains* and *certification paths* are introduced. CAs can not only issue certificates to authenticated entities, but also for intermediate CAs, which are in turn allowed to issue certificates on behalf of the root CA [Buchmann, Karatsiolis and Wiesmaier, 2013, p. 50]. The graph formed by hierarchical trust models employing certificate chains is usually tree-like, as can be seen in the example illustrated in Figure 2.3b.



(a) A simple web of trust: Alice trusts Bob. Bob certifies Carol's public key and therefore vouches for her identity. Alice thus also believes that Carol's key actually belongs to her. Since Alice has not explicitly expressed trust in Dave, she has no trust relation to either Dave or Erin.



(b) A simple trust hierarchy: Alice trusts the root CA and thus also trusts any intermediate CAs. Consequently, she does not doubt the identities of any entity authenticated to the root CA or any intermediate CA. Trust is inherently transitive in hierarchical models.

Figure 2.3: Comparison of a web of trust and hierarchical trust

Regardless of which trust model is applicable to a specific environment or application, authentication is essential to establish trust in the first place. Different well-defined authentication methods and authentication factors exist to achieve this goal.

2.3.2 Authentication

Whenever communication is not carried out directly, but over an information channel, communicating parties need to be certain about the identity of their counterpart. While confidentiality is not always important, in almost all cases it is essential that any sent information reaches its intended destination. In case confidentiality is desired, this requirement becomes even more important: In addition to being certain that nobody is eavesdropping on the transmission of information, it is vital that the transmitted information reaches only the intended recipients. Obviously, cryptography helps achieve this. However, cryptography alone is of limited use when it comes to authenticating entities.

Well-defined methods and factors exist to establish entity authentication. Most of these rely on cryptography and trust models to secure communication channels. However, this is typically unidirectional or limited to provide an authenticated communication channel over which the actual entity authentication is then carried out. Entity authentication is typically categorised by so called *authentication factors* and achieved through *authentication methods*.

2.3.2.1 Authentication Factors

Authentication can be achieved by relying on different factors. These factors can be categorised and defined as follows:

- Something the individual knows: Examples include a password, a personal identification number (PIN), or answers to a prearranged set of questions.
- Something the individual possesses: Examples include cryptographic keys, electronic keycards, smart cards, and physical keys. This type of authenticator is referred to as a *token*.
- Something the individual is (static biometrics): Examples include recognition by fingerprint, retina, and face.
- Something the individual does (dynamic biometrics): Examples include recognition by voice pattern, handwriting characteristics, and typing rhythm. [Stallings, 2013, p. 452]

These different authentication factors are sometimes also referred to as *knowledge factors*, *possession factors*, and *inherence factors*. All of these are obviously able to provide authentication, however, their properties, strengths, and weaknesses differ significantly.

Knowledge and possession factors enable entities to authenticate themselves without revealing personal information. In some circumstances, these factors can be used for pseudonymous or even anonymous authentication, where the real identity of an entity can remain secret. For example, if a group of individuals is issued smart cards which are not personalised it is possible to implement access control without revealing an individual's identity: The only information known to the system implementing access control is that someone who is part of a group is interacting with the system. Due to inherence factors obviously being inherently personal, individuals will always present personal information during authentication based on inherence factors.

Of course, in case a user's identity (such as name, social security number, ...) is not recorded, authentication factors are not mapped to legal entities. However, user actions can be recorded across system borders, thus making it possible to trace individuals using inherence-based authentication. Additionally, such approaches usually require sophisticated equipment in order to map physical properties to their digital equivalents.
Finally, authentication mechanisms can also rely on multiple factors, which is commonly referred to as *multi-factor authentication*.

2.3.2.2 Authentication Methods

Although not fully disconnected from authentication factors, different authentication methods exist and can be employed somewhat independently. Perhaps the most widely-known authentication method is *password-based* authentication where users have to enter a password to gain access to some resource. An authentication method based on possession factors has already been touched in Sections 2.1, 2.2, and 2.3.1.1 (albeit from different abstraction levels): *public-key-based* authentication including certificate– and digital-signature-based authentication.

With respect to password-based authentication methods, Kizza [Kizza, 2015, pp. 212–214] defines (amongst others) methods based on reusable passwords, one-time passwords and challenge-response approaches and reaches a devastating conclusion about reusable passwords:

> Because these types of authentication are the most widely used authentication methods, they are the most abused. They are also very unreliable because users forget them [the passwords, author's note], they write them down, they let others use them, and, most importantly, they are easy to guess because users choose simple passwords. They are also susceptible to cracking and snooping. In addition, they fall prey to today's powerful computers, which can crack them with brute force through exhaustive search. [Kizza, 2015, p. 212]

One-time passwords, on the other hand, do not suffer from as many flaws as their reusable counterparts. Most importantly, (perfect) forward secrecy can be achieved, just as it can be done with session keys as mentioned in Section 2.1. However, in case passwords are tied to time-frames, synchronisation issues can arise. Still, one-time passwords are generally considered an advancement over reusable ones, given sensible password management is applied. One example of a multi-factor approach relying on one-time passwords are security tokens which generate one-time passwords for the user as it is often implemented for Internet banking systems. In this case, a knowledge factor (such as a customer ID and a password) and a possession factor (the security token, generating one-time passwords) are required for authentication.

Challenge-response protocols are another way to ensure timeliness of a user-provided password (or, in general, a user-provided value required for authentication) if implemented accordingly. From an abstract point-of-view, however, such protocols simply work as the name implies and

impose no requirements on timeliness: “Challenge-response, as a password authentication process, is a handshake authentication process in which the authenticator issues a challenge to the user seeking authentication. The user must provide a correct response in order to be authenticated” [Kizza, 2015, p. 213]. Even the simple procedure of asking the user for a reusable password could be characterised as challenge-response authentication.
Which form the challenge or the response takes depends on the circumstances and also on the required security level, but also on the motivation behind opting for a challenge-response protocol. Such schemes are thus not tied to using passwords and may mandate multiple communication channels to achieve a certain security level. An example of such a challenge-response scheme involving multiple communication channels (and multiple authentication factors) is the *qualified mobile server signature*:

> Upon receipt of a signature creation request, the user is first prompted for a decryption-PIN [. . .]. A text message containing an authorization code is sent to the user’s mobile phone. Only after the user has entered this authorization code on the server signature’s web interface, the requested signature is created. [Orthacker, Centner and Kittl, 2010, p. 108]

In this case the PIN sent over the Internet is a knowledge factor and the *subscriber identity module* (SIM) receiving the authorisation code is a possession factor. Thus, two distinct communication channels and two separate authentication factors are mandated by this challenge-response protocol.
Stallings [Stallings, 2013, p. 453] lists challenge-response authentication also as a countermeasure against replay attacks. Implemented accordingly, it guarantees that any user responses to previous challenges cannot be reused by an attacker to respond to challenges issued in the future. This is precisely the case whenever cryptographic nonces or one-time passwords are involved in the authentication process.

Of course, various other authentication methods exist and authentication factors can be employed and combined in different ways. Password-based authentication is just one of the most prominent examples.
Naturally, when relying on a certain authentication scheme, the properties of different methods do matter However, the underlying principles remain the same. Therefore, being familiar with the fundamentals as discussed by means of password-based authentication is crucial. In essence, being fully aware of the implication of the underlying details makes it possible to implement or choose an appropriate framework.

3 Peer-to-Peer Networks

When striving for a decentralised network topology, *peer-to-peer* (P2P) networks need to be discussed in detail, since various types of P2P networks exist, each providing its own set of features, strengths, weaknesses, and guarantees. Different designs cater to different needs; systems aimed at file-sharing may not be suited for streaming data, and anonymity networks may perform poorly when it comes to exchanging large files like it is common for file-sharing networks.

For this work, anonymity networks which build upon the P2P approach are of interest. Therefore, the same basic principles which make up the foundation of any P2P network apply to these systems as well. Additionally, these fundamentals can also be viewed as an abstraction, focusing on the way anonymity networks relay messages without concerning oneself with the security mechanisms when assuming certain security guarantees provided by such a system.
For these reasons, the basic principles of P2P networks will be elaborated on prior to exploring the security properties of such systems and, subsequently, discussing anonymity networks based on the P2P approach.

3.1 Peer-to-Peer Network Fundamentals

Traditional client-server network topologies dominate internet-based day-to-day activities such as web browsing or sending and receiving e-mails. Even perceived direct user-to-user activities such as instant-messaging typically rely on a central server to route messages between users. In contrast to this traditional client-server model, P2P networks consist of nodes (also called *peers*) which act both as client and server forming a network of equals each contributing resources to the network. Buford, Yu and Lua [Buford, Yu and Lua, 2009] define a P2P network as follows:

> A *peer-to-peer overlay* is a distributed collection of autonomous end-system computing devices called *peers* that form a set of interconnections called an *overlay* to share resources of the peers such that peers have symmetric roles in the overlay for both message routing and resource sharing. [Buford, Yu and Lua, 2009, p. 29]

The term *overlay* is used, since P2P networks are typically logical constructs which operate on top of physical networks [Buford and Yu, 2010, p. 3]. This structure eliminates a single server being a potential bottleneck, as it can happen in client-server architectures when it comes to distributing large quantities of data. However, the above definition does not explicitly exclude the need for a central authority. This aspect raises the question of classification of P2P networks.

3.1.1 Classification of Peer-to-Peer Networks

P2P networks, as already mentioned, are not necessarily fully decentralised, and may also employ one or more central authorities.
Typically, such a central authority acts as a middleman, informing peers of each other's existence but also about the resources peers are advertising to the network [Liu and Antonopoulosu, 2010, p. 75]. Types of P2P networks which operate in this manner are called *centralised peer-to-peer networks*[2]. While in such systems the exchange of resources is still performed directly between nodes, the network is not self-supporting and shutting down the central authority will render the whole network non-operational [Risson and Moors, 2006, p. 3490]. This property defies one key advantage of P2P networks over client-server architectures: the lack of a single point-of-failure. On the other hand, a central authority has full control over the network and is able to prevent malicious parties from compromising the network (given these can be identified).
At the other end of the spectrum, *decentralised peer-to-peer networks* exist. Such systems do not rely on a central instance governing and organising the network, leaving it up to the peers to collectively do so [Korzun and Gurtov, 2013, p. 5]. This, however, not only raises the question as to how individual peers can be trusted, but also inherently opens up the network to threats originating from inside the network since no central authority is responsible for ensuring correct operation.
Finally, a third category, *hybrid peer-to-peer networks*, exists, featuring different kinds of nodes, some of which have more power/bandwidth than others and possibly guaranteed uptimes and thus contribute more to the network. Such hybrid systems try to combine the strengths of decentralised and centralised designs, while trying to avoid the weaknesses of both approaches [Liu and Antonopoulosu, 2010, pp. 77–78].

Another way of classifying P2P networks is based on the way in which the network is organised and routing, peer–, and resource-discovery are performed. From this point-of-view, P2P networks fall into two categories, *structured* and *unstructured.* Unstructured P2P networks, as the name implies, do not impose any well-defined structure on the organisation of nodes. Each node simply knows about some of its neighbours, but not about the overall structure of the network. If any node whose location is unknown is to be contacted, a query for this node is recursively

[2]In this work, the terminology defined by Liu and Antonopoulosu in *From Client-Server to P2P Networking* [Liu and Antonopoulosu, 2010] is used to classify P2P networks based on their degree of dependence on a centralised authority.

forwarded to a node's set of neighbours until some node is reached which knows about the node in question [Korzun and Gurtov, 2013, p. 7]. Of course, optimisations which are superior to flooding the network in such a way exist. These more sophisticated search algorithms are either based on graph-theoretical models of the network and/or incorporate heuristics based on additional knowledge locally available at nodes [Fletcher, Sheth and Börner, 2005, pp. 18–19]. However, no strong guarantees about the success or failure of a query can be made; there is no clear a-priori upper bound on the number of hops or on the time it takes to contact any given node – at least not from the querying node's point-of-view, since it does not have any global information about the network. In this situation, even a provable upper bound on the number of hops based on the network size does not help, since nodes do not know about the network size. Yet, unstructured overlays can be worth considering, whenever guarantees on the success or failure of queries are of low priority. The lack of constraints and the loose nature of the network enables simple implementation, as there is obviously no need to maintain a globally coherent network structure. Of course, this comes at the cost of potentially poor scalability and well-defined upper bounds on number of hops needed for any operation.
Structured P2P networks, on the other hand, impose a well-defined *structure* on the organisation of nodes. This entails that each node does not only have local information about the network, but is able to put this information into context. Thus, search strategies for locating nodes can be employed which do not stress the network as much, since queries can be issued to specific nodes which are known a-priori to be closer[3] to the node in question [Ratnasamy et al., 2001, p. 162]. In short, structured P2P networks feature a well-defined routing behaviour. This makes it possible to give strong guarantees about the maximum number of hops it takes to locate any node (or to realise that the node in question is not part of the network). Therefore, it is possible to give a very sharp upper bound for such operations, which makes it easier to analyse the performance of structured P2P networks. In addition, this well-defined structure also makes for a very simple location of resources within the network, as these can directly be related to nodes. The *Content-Addressable Network* (CAN) defined this mechanism and the underlying structure, a *distributed hash table* (DHT) as it is now commonly called, which can not only be used to store, locate, and retrieve sources, but also directly for path-finding and routing [Ratnasamy et al., 2001, pp. 161–164]. This results in a simple implementation of routing protocols within structured P2P overlays, abstracting away the physical network topology. Due to this fact, it is (in theory) easy to deploy and operate such overlays.

3.1.2 Routing Based on Distributed Hash Tables

As per the original *Content-Addressable Network* design, the structured nature of DHT-based networks used for distributing and retrieving content inherently forms the base for routing al-

[3]The exact semantics of closeness are implementation-specific. However, the (very general) notion of distance being the number of hops between nodes suffices at this point.

gorithms within such networks. While the exact organisation of the network is implementation-specific, the underlying principles apply to many different designs. The original CAN design, for example, organised nodes within a d-dimensional Cartesian coordinate space. Later approaches to the DHT design such as *Chord* simplified this aspect drastically by introducing a one-dimensional, circular space for organising a network's nodes [Stoica et al., 2001, p. 150]. However, *Chord* actually pursues a different primary goal, focused solely on routing with distributed data storage being just one possible application: "The *Chord protocol* supports just one operation: given a key, it maps the key onto a node. Depending on the application using Chord, that node might be responsible for storing a value associated with the key." [Stoica et al., 2001, p. 149]
With the primary focus of this section being on routing and not distributed data storage, *Chord* already presents a simple concept suitable for illustrating these routing mechanisms. As routing is application-independent, it better demonstrates the concepts without concerning itself with the specifics of distributed data storage. Yet, while *Chord* already simplifies the organisation of nodes, the concept of a distance function is more elaborate (in contrast to the well-known Euclidean distance applicable to the original CAN design). As shown by designs such as *Kademlia* the distance function can be simplified as well, leading to an overall simple routing algorithm [Maymounkov and Mazières, 2002, pp. 54–56].
The building blocks which make up a DHT will be discussed based on the strengths of these different designs to best illustrate the principal steps which make up a routing protocol within a structured P2P network in form of a distributed hash table.

3.1.2.1 Identification and Location of Nodes

When following the approach to mount a structured P2P overlay, logical identifiers, *keys*, need to be defined. Using a circular, one-dimensional key space, each node must have a globally unique identifier. One possible way to fulfil this constraint is to simply let nodes pick a random number from the key-space; if the key-space is sufficiently large, the laws of probability ensure that collisions are a non-issue.[4] A node's position within this space is determined by its identifier. For example, given a key-space ranging from $0 \ldots n$, a node having the ID zero will be assigned the position zero, while a node having an ID of $\frac{n}{2}$ will be assigned a position at the middle of the key-space. It has to be noted, that since the key-space is circular, position zero (or any other position for that matter) does not mark the "start" or the "end" of the key-space. Figure 3.1 illustrates a key-space populated by nodes and the position of these nodes in the key-space.

The second important parameter, a distance metric, is required to elevate the key-space to a metric space. According to the mathematical definition of a metric space the following properties must hold:

[4]This statement only holds exactly, if every node's source of randomness is truly random (and not a *pseudorandom number generator* (PRNG)) and if the individual creations of identifiers are independent of each other. However, ignoring these facts makes it possible to focus on the essentials, while being formally correct would just needlessly overcomplicate the discussion of this matter.

- An item's distance to itself is zero: $dist(A, A) = 0$
- The distance function is symmetric: $dist(A, B) = dist(B, A)$
- The triangle inequality holds:
  Let A, B, C be vertices on a triangle, $dist(A, B) \leq dist(A, C) + dist(C, B)$

In layman's terms, the fact that a direct path from A to B is a straight line and this line is also the shortest path from A to B also holds for metric spaces.
Kademlia employs the binary *exclusive or* (XOR) operator as the distance function due to its symmetry and simplicity:

> Given two 160-bit identifiers, x and y, Kademlia defines the distance between them as their bitwise exclusive or (XOR) interpreted as an integer, $d(x, y) = x \oplus y$. We first note that XOR is a valid, albeit non-Euclidean metric. It is obvious that that [sic] $d(x, x) = 0$, $d(x, y) > 0$ if $x \neq y$, and $\forall x, y : d(x, y) = d(y, x)$. XOR also offers the triangle property: $d(x, y) + d(y, z) \geq d(x, z)$. The triangle property follows from the fact that $d(x, y) \oplus d(y, z) = d(x, z)$ and $\forall a \geq 0, b \geq 0 : a + b \geq a \oplus b$. [Maymounkov and Mazières, 2002, p. 56]

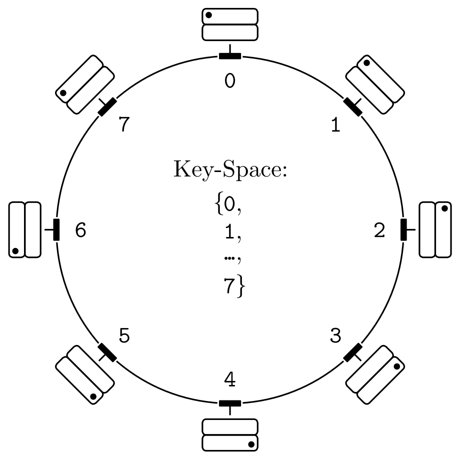


Figure 3.1: Schematic of a distributed hash table: Even distribution of nodes based on their identifier in a one-dimensional key-space

Due to the simplicity of the XOR metric, the process of locating nodes (which defines a structured P2P network's routing mechanism) will be illustrated based on the routing protocol employed by Kademlia. While nodes are assigned positions on a circular, one-dimensional space based on their identifier, "Kademlia effectively treats nodes as leaves in a binary tree, with each node's position determined by the shortest unique prefix of its ID" [Maymounkov and Mazières, 2002, p. 54]. This, in turn, perfectly maps a node's position in the tree to the position in the one-dimensional key-space, if the tree is traversed in-order. The location of nodes works as follows:

> For any given node, we divide the binary tree into a series of successively lower subtrees that don't contain the node. The highest subtree consists of the half of the binary tree not containing the node. The next subtree consists of the half of the remaining tree not containing the node, and so forth. [. . .] The Kademlia protocol ensures that every node knows of at least one node in each of its subtrees, if that subtree contains a node. With this guarantee, any node can locate any other node by its ID. [Maymounkov and Mazières, 2002, p. 54]

These definitions entail that a node knows more nodes which are close to it than nodes farther away. A prefix-based search algorithm then yields logarithmic runtime of node lookups, which results in a high level of scalability, as performance does not degrade much even for large networks [Fletcher, Sheth and Börner, 2005, p. 21].

3.1.2.2 Mapping Logical Identifiers to Underlying Network Identifiers

Two aspects have not been discussed yet which are crucial to the understanding of how P2P overlays operate on top of physical networks: The mapping of logical identifiers to identifiers of the underlying network topology and the process of *bootstrapping* the network.
The issue of mapping identifiers to (commonly) IP addresses is done in more or less the same way throughout implementations: A node's key is mapped to a node's IP address and port which is then used to communicate with this node. In *Chord*'s terminology, for example, these mappings are called *finger tables* [Stoica et al., 2001, p. 152]. Naturally, each node publishes its IP address and port itself. This causes problems when *network address translation* (NAT) is used or firewalls are in place which prevent entities from other networks to reach a node. Different approaches to solve these problems such as relaying and various kinds of *hole punching* have been proposed, however, the success rate of such techniques still highly depends on the concrete networking hardware and –software used at each node [Ford, Srisuresh and Kegel, 2005, pp. 181–188, 190–191]. Consequently, deploying a working P2P network in a real-world setting imposes lower-level constraints (such as a certain degree of hardware-support) and/or introduces workarounds such as some level of centralisation within the network[5].

The final issue of getting a P2P network up and running and making it possible for nodes to join an existing network is called *bootstrapping* and directly follows from the employed routing mechanism. While the details are, again, implementation-dependent, the underlying principle is always the same: If a node wants to join a network, it needs to know how to reach at least one node which is already part of the network. In the simplest case, the IP address and port of some node is known. This node, called a *seed node*, is then contacted and the connecting node advertises its presence to the network. In the case of Kademlia, the symmetry of the XOR metric makes it possible that simply querying for other nodes is enough to make a node advertise its

[5]In this context, relaying is considered a form of centralisation.

presence to other nodes and thus integrate itself into the network [Maymounkov and Mazières, 2002, pp. 53–54].

3.2 High-Level Security and Trust

Network security can be discussed and evaluated at many levels, from manifold points-of-view. With P2P networks being overlays operated on top of existing networks, additional factors need to be considered when discussing security. On one hand, overlays need to be able to rely on the underlying networks. Therefore it can be sufficient to attack these underlying networks to disrupt the overlay running on top of it.
On the other hand, since P2P networks typically leverage the sources of many individual parties to attain a common goal, the overlays themselves can become a threat if this common goal is to wreak havoc. The reason for this is obvious: The joint power of many can pose a much greater danger than an even larger number of individual, uncoordinated attackers.
Both these aspects are practically relevant, however, this section focuses on security issues which arise from the decentralised nature of P2P networks and which directly concern the overlays themselves. In short: threats to the network which originate from within the network itself. Therefore, the aforementioned general network security aspects will be discussed only briefly.

3.2.1 Network Security Considerations

As already mentioned, P2P overlays can also be viewed as an abstraction of the underlying, possibly highly complex, network topology. However, since P2P networks build upon lower-level technologies, network security in general and concepts like unilateral or mutual authentication become relevant: The *overlay layer*, introduced by P2P networks, relies on the correct operation of the underlying *network layer*[6]. Therefore, security properties, especially flaws in the network layer, need to be kept in mind when it comes to evaluating the (security) properties of an application based on P2P networks. In order to discuss these issues, a network security model is needed. Here, the model shown in Figure 3.2, based on Stallings [Stallings, 2013, p. 22], is used as it is both general and easily applicable to many practically relevant scenarios.

Ideally, the information channel (as depicted in Figure 3.2) provides all desired security services such as confidentiality, integrity, availability, but also others, such as authentication of communication parties. This allows for designing overlays without needing to re-implement already existing security mechanisms. However, the *Internet Protocol* (IP) was developed only with "provisions for timestamps, security, and special routing" [Postel, 1981, p. 3], but without any built-in security mechanisms. In essence, security was only an afterthought in the design of one

[6]Here, the term *network layer* encompasses all layers below the overlay layer introduced by P2P networks.

of the most important building blocks of the protocol stack employed to realise global communication networks. This means that without additional security mechanisms, securing messages sent over an IP information channel may not be enough, since even routing information used to build this information channel cannot be trusted [Stallings, 2013, p. 630]. "Even if data is encrypted, routing information is still sent in the clear because routers need to know packets' destinations, in order to route them in the right direction" [Goldschlag, Reed and Syverson, 1996, pp. 137–138]. While a standardised way to secure the IP-layer was proposed as early as 1995 by Atkinson [Atkinson, 1995], it has not been deployed on a global scale. The industry standard *Transport Layer Security* (TLS) protocol operates on a higher layer, on top of the *Transmission Control Protocol* (TCP), "to provide privacy and data integrity between two communicating applications" [Dierks and Allen, 1999, p. 3]. A variant of TLS, adapted for the use with datagram protocols such as the *User Datagram Protocol* (UDP), called *Datagram Transport Layer Security* (DTLS) has also been standardised. "The DTLS protocol is based on the Transport Layer Security (TLS) protocol and provides equivalent security guarantees" [Rescorla and Modadugu, 2006, p. 1]. Both TLS and DTLS are available as cross-platform open-source implementations as part of *OpenSSL* [Modadugu and Rescorla, 2004, p. 1].

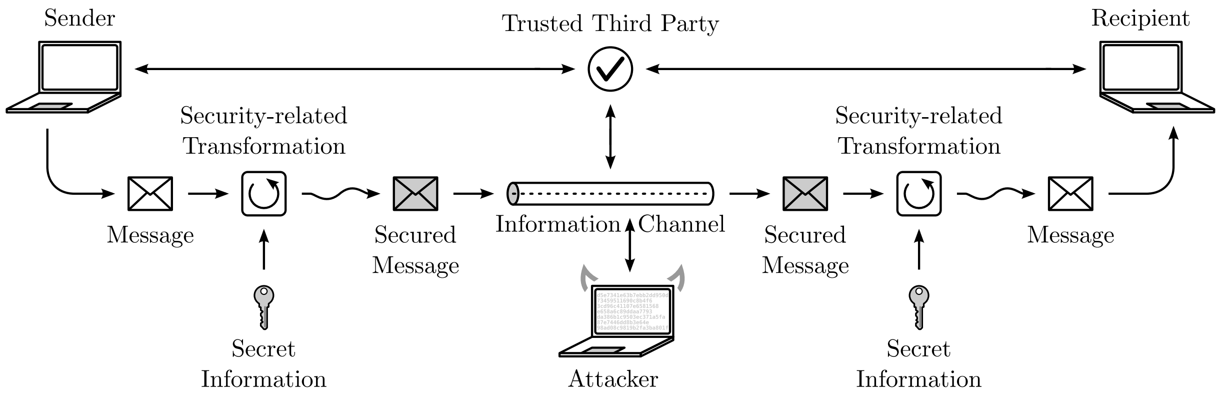


Figure 3.2: Network security model based on Stallings [Stallings, 2013, p. 22]

Unfortunately, the availability of standardised security protocols does not alleviate all security concerns. While TLS is designed to prevent eavesdropping, to provide data integrity and mutual– as well as unilateral authentication [Dierks and Allen, 1999, pp. 1, 4], this only covers the content of the messages transmitted over an information channel:

> While encrypting communication to and from web servers [. . .] can hide the content of the transaction from an eavesdropper [. . .], the eavesdropper can still learn the IP addresses of the client and server computers, the length of the data being exchanged, and the time and frequency of exchanges. Encryption also does little to protect the privacy of the client from the server. [Reiter and Rubin, 1998, p. 66–67]

Furthermore, the use of cryptography introduces new problems: How can the key material (or,

more generally, secret information, as depicted in Figure 3.2), which is needed to secure the messages exchanged between communication parties, be shared? How can sufficient trust be achieved, which is needed to authenticate communication parties? Chapter 2 deals with such issues, which is why these shall not be reiterated here.
Regardless of how trust is established, authentication of communication parties can either be mutual or unilateral or combined into a hybrid approach. A typical scenario for this is the use of a web service: First, the web server authenticates itself to the client and a secure connection is established by means of TLS; this step typically involves a PKI. Once a secure communication channel is established the client authenticates itself to the service; this happens on a higher layer, decoupled from the PKI.

To summarise, standardised protocols and courses of operation exist to provide secure communication over IP networks with TLS being "One of the most widely used security services" [Stallings, 2013, p. 525]. However, the security properties provided by such protocols are the result of a complex set of interdependencies (for example, untrusted routing, unilateral authentication and encrypted messages sent over public communication channels, building upon a variety of different cryptographic primitives and a wide array of cryptographic algorithms). This makes it difficult to judge the level of security of a given setup. This becomes even more difficult to assess when evaluating the resilience of a setup against *denial-of-service* (DoS) attacks, since no general way to prevent such attacks exists [Stallings, 2013, p. 524].
On the one hand, P2P networks add additional complexity to this already intricate situation. On the other hand, many aspects cannot be controlled whenever existing infrastructure has to be relied on. Therefore, the already available security mechanisms can be employed to secure the network layer and the provided security guarantees can be relied on by the overlay layer of P2P networks.

3.2.2 Peer-to-Peer Network Security and Trust-Based Systems

Depending on the degree of centralisation, a P2P network may be more or less vulnerable to a DoS attack. Since this is highly implementation specific and therefore not generally applicable, attacks on a central authority will not be discussed separately. The intuitive notion that a high degree of decentralisation correlates with a higher immunity against DoS attacks should, however, be kept in mind.
Generally more relevant are threats to the network originating from within the network itself. While in this case the opposite may seem to be obviously true when it comes to the degree of decentralisation, this issue cannot be dismissed as easily, since it is far more complex than, for example, a network device being overloaded. Reputation systems and, more generally, trust are relevant in such cases. A specific scenario focusing on these issues is described by the *Sybil attack*, where – in extreme cases, as a consequence of it – a single malicious entity can become powerful enough to take over an entire P2P network. Due to the general applicability and the

potentially devastating consequences of the Sybil attack, it is practically relevant to all P2P designs and will therefore be discussed in more detail.

3.2.2.1 The Sybil Attack

The *Sybil attack*, as defined by Douceur [Douceur, 2002], describes an attack where a single malicious entity creates many identities (an *identifier* or *key* in the DHT terminology) and is therefore able to present itself as multiple distinct entities to the network [Douceur, 2002, p. 251]. This is strongly related to other attacks based on (missing) identifier generation constraints, but also factors into a variety of other attacks specific to P2P networks, such as DoS attacks targeted at nodes of the overlay. Even the most basic tasks such as routing, as discussed in Section 3.1.2, rely on the cooperation of all members of the overlay. "Most designs against [...] malicious behavior rely on the assumption that a certain fraction of the nodes in the system are honest. For example, virtually all protocols for tolerating Byzantine failures assume that at least 2/3 of the nodes are honest" [Yu et al., 2006, p. 267]. Therefore, a single malicious entity posing as many can disrupt the network. In general, every task performed by means of a P2P network which can be subverted by the collusion of multiple nodes is a potential target of a Sybil attack. The key aspect of the Sybil attack, however, boils down to the following simple observation: "If the local entity has no direct physical knowledge of remote entities, it perceives them only as informational abstractions that we call *identities*." [Douceur, 2002, p. 251]

Due to the simplicity of the Sybil attack's underlying principle (which results from the very nature of P2P networks) and at the same time its enormous attack potential, different countermeasures have been proposed. One category of countermeasures focuses on identifier generation and the assignment of identifiers to nodes. Approaches include the integration of a PKI where identifiers are validated by a central authority which, at the same time, can limit the overall number of identifiers issued to an entity [Dinger and Hartenstein, 2006, p. 759]. Obviously, this introduces a single point-of-failure, a single point-of-trust[7], and also makes joining the network more elaborate. Depending on the identity proof required by the central authority, this may also lead to a non-automated process of joining the network, which can be a deal-breaker for some applications.
Another approach described in the identifier assignment taxonomy by Dinger and Hartenstein [Dinger and Hartenstein, 2006] is based on the use of external identifiers and does not require any central authority. If node identifiers are derived from external identifiers and the assignment of these external identifiers can be secured, Sybil attacks can be effectively prevented. As an example of external identifiers, IP addresses are mentioned, as these are both limited and unique [Dinger and Hartenstein, 2006, p. 759]. Aside from possible technical obstacles of this approach, it may conflict with certain other goals, depending on the situation: If, for example, anonymity of entities is relevant, the simple act of verifying an identifier by mapping it back to the external identifier can, in the case of IP addresses, uniquely identify an entity. This obviously conflicts

[7]The PKI as a whole, however many entities it may be composed of, is here considered a single point-of-failure.

with entity anonymity. In addition, approaches relying on an external identifier "implicitly rely on the authority of a trusted agency [. . .] to establish identity" [Douceur, 2002, p. 251]. This can again be a single point-of-failure.
Other techniques based on social networks have also been proposed to combat Sybil attacks. However, a social network, by definition, depends on manual human action. Therefore, even a sophisticated approach against Sybil attacks, such as *SybilGuard*, which relies on the structure of social networks, does not provide a solution to fully automatically single out Sybil nodes. [Yu et al., 2006, pp. 268–269].
Finally, even if it were possible to present a fool-proof, general-purpose countermeasure to Sybil attacks, any real-world P2P network usually caters to a specific need and therefore must observe domain-dependent constraints. Common sense suggests that these constraints, whatever these may be, can potentially conflict with such a general-purpose solution. For this reason, combating Sybil attacks in a general manner is considered an open problem and shall therefore not be discussed further at this point.

3.2.2.2 Other Peer-to-Peer-Specific Security Concerns

While the Sybil attack itself has been elaborated on, specific attack scenarios have not been discussed yet. The main reason for this is that it factors into a variety of attacks focusing on many kinds of different targets. Often, attacks are made possible largely due to the ability to mount a Sybil attack. Here, the following generalisation is made: Multiple colluding entities are considered equivalent to a single entity capable of successfully executing a Sybil attack if both situations result in multiple nodes working in concert against (some other nodes of) the network.
Two P2P-specific implementation– and application-independent attacks are *distributed denial-of-service* (DDoS) attacks and overpowering of the network.

(Distributed) denial-of-service attacks in P2P networks have the same goal as in any other setting. However, due to the fact that nodes are collectively responsible for operating the network, these attacks can also originate from within the network. A basic example of such a DDoS attack is described in the form of a message forwarding attack: Malicious "nodes may mis-route, corrupt, or drop messages and routing information" [Castro et al., 2002, p. 299].
Another related attack is directly targeted at the routing tables: "Adversaries could take advantage of routing table updates to feed false updates and thus introduce faulty routing table entries to other peers. The effect cascades after subsequent updates. This could potentially trigger many problems in the P2P network" [Buford, Yu and Lua, 2009, p. 327].
In both cases, basic fault-tolerance and fault-recovery mechanisms can still guarantee the (mostly) correct operation of the network as a whole, if the number of malicious nodes stays small enough [Maymounkov and Mazières, 2002, pp. 57–59, 65]. However, even in cases where the collusion of many malicious entities is unrealistic, the Sybil attack still applies.

Overpowering the network can be considered a generalisation of the Sybil attack, when not the number of nodes controlled by a malicious entity becomes relevant, but rather the ratio of total computing power of the network versus that of the attacker. Even though it can be argued that this kind of attack targets the application layer, it cannot be dismissed as easily. In fact, this is directly related to an entity's ability of carrying out a Sybil attack [Douceur, 2002, pp. 255–257]. In general, the possibility of a single entity becoming too powerful has to be kept in mind whenever arguing security (even on application level) of a P2P-based system. Therefore, this threat is considered not only application-independent, but also implementation-independent and P2P-specific.
Bitcoin, for example, recognises this fact, but for some attacks, in essence, assumes that the network is large enough and therefore the computing power required to successfully execute attacks is simply improbable to be raised by a single entity [Nakamoto, 2008, p. 4].

Aside from the described threats, most well-known classes of attacks, such as eavesdropping, forgery, impersonation, or even implementation-specific timing-attacks apply. Due to the general nature of such attacks, meaning that these apply to virtually any information system, such attacks are not considered P2P-specific and therefore not discussed here in spite of their relevance. For the same reason, principles such as mutual authentication to counter Sybil attacks were not elaborated on.

3.3 Anonymity Networks

Considering the distributed nature of P2P networks, an application in the form of anonymity networks, essentially hiding users amongst users, comes to mind. In fact, many noteworthy proposals of such systems are distributed in nature. In order to discuss anonymity networks, the terms *unlinkability*, *anonymity* and *unobservability* need to be defined first.
Unlinkability, according to the *International Organization for Standardization*, "ensures that a user may make multiple uses of resources or services without others being able to link these uses together" [International Organization for Standardization, 2011, p. 74]. As for the other terms, the terminology defined by Pfitzmann and Köhntopp [Pfitzmann and Köhntopp, 2001] will be used:

> *Anonymity* is the state of being not identifiable within a set of subjects, the *anonymity set.* [Pfitzmann and Köhntopp, 2001, p. 2]
>
> *Unobservability* is the state of IOIs [items of interest, author's note] being indistinguishable from any IOI at all. This means that messages are not discernible from "random noise". [Pfitzmann and Köhntopp, 2001, p. 3]
>
> A weaker property than each of sender anonymity and recipient anonymity is *relationship anonymity*, i.e. it may be traceable who sends which messages and it may

> also be possible to trace who receives which messages, but it is untraceable who communicates to whom. [Pfitzmann and Köhntopp, 2001, p. 3]

In addition, the term *linkability* will be used to denote the opposite of unlinkability.

Anonymity and linkability are related to each other and this connection is obviously relevant in practice, as linkability can lead to de-anonymisation through, for example, correlation attacks. Furthermore, all of the above concepts do not only apply to a single communication layer:

> A distinction can be made between connection anonymity and data anonymity. Data anonymity is about filtering any identifying information out of the data that is exchanged in a particular application. Connection anonymity is about hiding the identities of source and destination during the actual data transfer. [Díaz et al., 2003, p. 54]

Obviously, the same statement can be made about unlinkability as well as every other properties related to privacy. Failure to provide a desired property on any layer is here considered a complete failure, if it results – in the case of anonymity – in identifying the user. This raises the question of measuring the quality of any system to provide such properties. Due to the connectedness of these concepts, the primary focus here lies on anonymity. Therefore (un)linkability as well as unobservability will be discussed in the context of anonymity.

The notion of measuring the degree of anonymity provided by a system, even when based on sound models, is considered a rough indicator of a system's quality on a very high level at best. The reason for this assessment is that when incorporating all the variables relevant in a real-world scenario, things easily become too complex to quantify reliably. This fact becomes clear when looking at proposals for anonymity networks such as *Tor* [Dingledine, Mathewson and Syverson, 2004] or *Tarzan* [Freedman and Morris, 2002]. Forgetting to consider one detail or a single wrong assumption can lead to gravely false conclusions. However, a certain minimum degree of anonymity provided by a system is considered an essential requirement; a system whose model does not even hold up to an information-theoretical evaluation of anonymity, (as proposed by Díaz et al. [Díaz et al., 2003], for example) is considered unfit. On the other hand, even a "perfect" model is of no use if it cannot withstand real-world threats. Therefore, the focus of this section lies on systems' goals, feature-sets, real-world impact, usability and market acceptance, while an unsatisfactory evaluation result of a system's model is simply considered a knock-out criterion. Naturally, some of these properties can only be quantified based on empirical data.
Knock-out criterion aside, real-world data is considered a superior benchmark to evaluating theoretical models. The fact that some systems have been operational for years, used by millions of users world-wide, makes these systems subject to regular peer reviews and repeated scrutinisation from various parties. In turn, these systems are improved continuously to deal with novel attacks and changes in the environment which makes real-world data all the more significant. In addition, the model of any serious proposal for an anonymity network must already have

withstood an evaluation to be taken seriously. Firstly, however, the theoretical foundation in the form of *mix networks* needs to be discussed, since most designs are closely related to– or based on this approach.

3.3.1 Mix Networks

Mixes (and *mix networks*) aim at providing anonymous communication: While originally not decentralised, key parts of the concept have been adapted for peer-to-peer networks, making networks of mixes the theoretical foundation of many practically relevant anonymity networks. In addition to the basic principles of the concept, the differences between mix cascades and mix networks need to be discussed, since mixes can be utilised in both ways.

3.3.1.1 Underlying Principles

The general idea is to mask the correspondence of two communicating parties, providing anonymous communications on public networks without the need for a trusted third party. This is achieved by introducing intermediaries, called *mixes*, into the communication channel. The basic principle can be illustrated by means of a simple example, based on the original proposal by Chaum [Chaum, 1981]: “A participant prepares a message M for delivery to a participant at address A by sealing it with the addressee’s public key K_a, appending the address A, and then sealing the result with the mix’s public key K_1” [Chaum, 1981, p. 85]. This single-mix setup is in practice extended to feature more than one mix to mitigate the effect of hostile mixes. The introduction of a chain of mixes leads to the following general formula:

$$M' = (R_n,\ enc_{e_N}(R_{n-1},\ enc_{e_{N-1}}(\ldots R_2,\ enc_{e_2}(R_1,\ enc_{e_1}(M),\ 1),\ 2\ldots),\ N-1),\ N)$$

where enc_{e_X} denotes the encryption under the public key of X, R_x is a random string and the numbers $1\ldots N$ represent the addresses of the mixes 1 to N. This definition also illustrates the fact that mixes heavily rely on asymmetric cryptography.

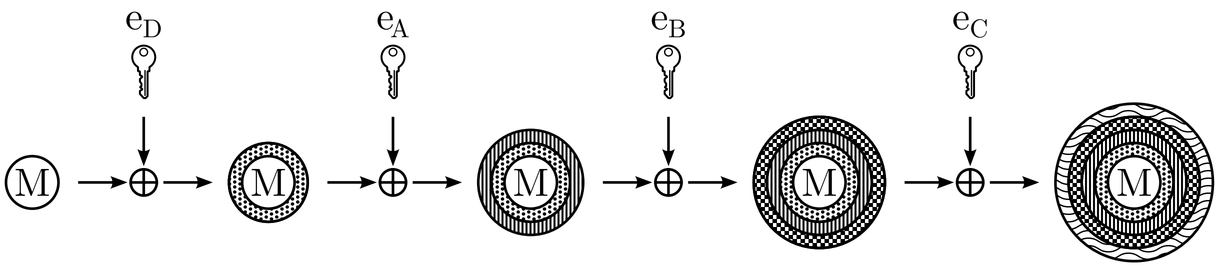


Figure 3.3: Encryption of a message prior to sending it through a chain of mixes

When transmitting a message, a layer of encryption is stripped-off at each mix and the resulting message is forwarded to the next mix. These layers of encryption ensure that each mix knows only about the next mix in line and learns nothing about the content of the message being delivered. This application of layered encryption is called *onion routing* [Goldschlag, Reed and Syverson, 1996, pp. 137, 140–142]. Figure 3.3 illustrates the encryption process and Figure 3.4 shows the transfer of a message through a chain of mixes.

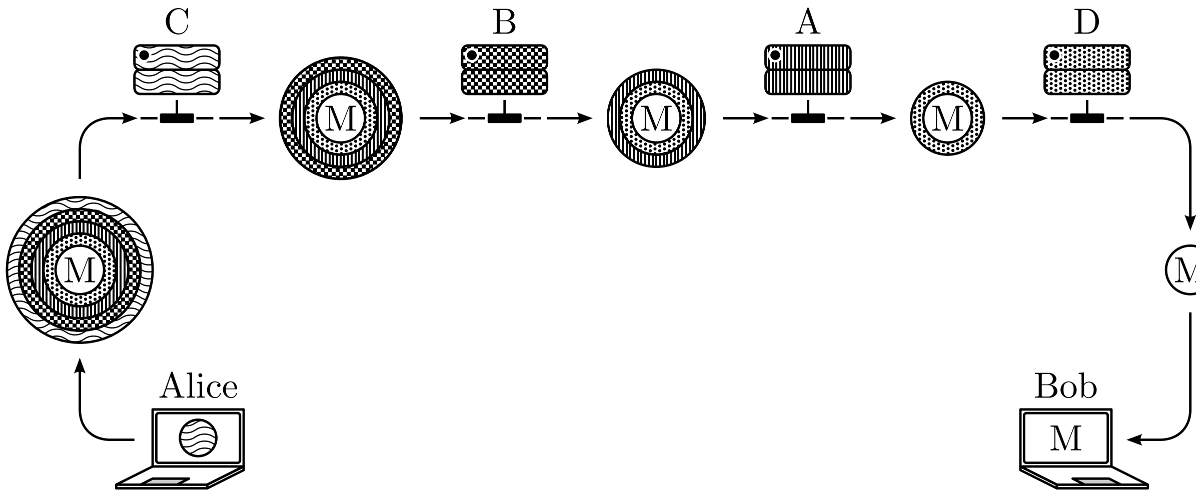


Figure 3.4: Transmission of a message through a chain of mixes

However, correlation attacks are still possible – even if all messages are of the same size – if two identical messages are processed by the same mix. In this case, the two identical messages leaving the mix can directly be related to the two identical messages entering the mix. Furthermore, if messages enter and leave a mix in the same order, a 1:1 correspondence between all outgoing and incoming messages can be established. For these reasons, messages are processed in batches and each mix is obliged to perform the following operations:

> The order of arrival is hidden by outputting the uniformly sized items in lexicographically ordered batches. [. . .] an important function of a mix is to ensure that no item is processed more than once. This function can be readily achieved by a mix for a particular batch by removing redundant copies before outputting the batch. If a single mix is used for multiple batches, then one way that repeats across batches can be detected is for the mix to maintain a record of items used in previous batches. (Records can be discarded once a mix changes its public key by, for example, announcing the new key in a statement signed with its old private key.) A mix need not retain previous batches if part of each random string R_1 contains something—such as a time-stamp—that is only valid for a particular batch. [Chaum, 1981, p. 85]

The second important feature of the mix concept (in addition to masking the correspondence of communicating parties) are untraceable return addresses. These keep the true identity of the sender secret from the recipient, while still making it possible for the receiver to reply to the sender. This is also considered an integral feature of onion routing [Goldschlag, Reed and Syverson, 1996, p. 139]. Such an untraceable return address can easily be created by the sender. A single-mix example based on the design by Chaum [Chaum, 1981] illustrates the communication of two parties using such a return address:

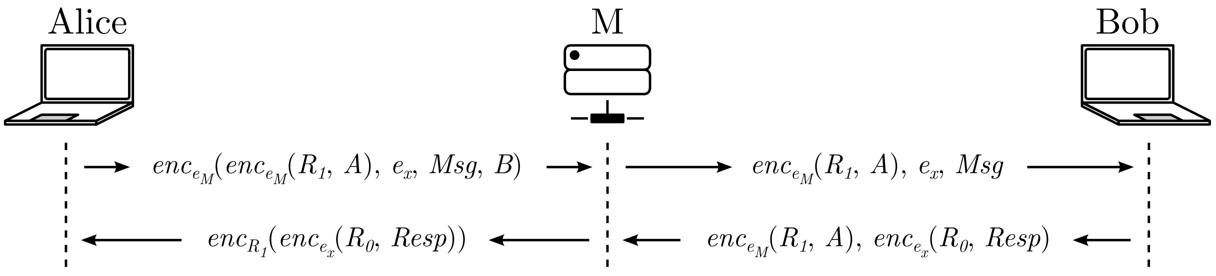

Alice
M
Bob
$enc_{e_M}(enc_{e_M}(R_1, A), e_x, Msg, B)$
$enc_{e_M}(R_1, A), e_x, Msg$
$enc_{R_1}(enc_{e_x}(R_0, Resp))$
$enc_{e_M}(R_1, A), enc_{e_x}(R_0, Resp)$

Figure 3.5: Sending and replying to a message using untraceable return addresses

A party Alice wanting another party Bob to reply to a message *Msg* creates an untraceable return address $enc_{e_M}(R_1, A),\ e_x$ and prepends it to the message prior to sending it to the mix *M*, which then delivers everything to Bob as usual. The recipient Bob can then issue a response *Resp* and encrypt it under e_x, prepend the encrypted part of the return address $enc_{e_M}(R_1, A)$ and send it to the mix *M*. The mix can decrypt this part to recover R_1 and add another layer of encryption to the already encrypted message by encrypting it under R_1. Finally, the original sender Alice is able to recover the response *Resp*, since she chose both e_x and R_1 [Chaum, 1981, pp. 85–86]. This whole procedure is illustrated in Figure 3.5. Naturally, this setup can easily be extended to incorporate multiple mixes. However, it has to be kept in mind, that it must not be allowed for “address parts to be repeated—for the same reason that items of regular mail must not be repeated” [Chaum, 1981, p. 86].

3.3.1.2 Mix Cascades vs. Mix Networks

The basic properties of mixes already provide relationship anonymity: Even if just a single mix is used, the transport network learns nothing about who communicates with whom. Furthermore, sender anonymity is provided from the recipient’s point-of-view (or more generally, at the output of the mix). At the same time, untraceable return addresses make it possible for recipients to reply to messages without learning anything about the original sender.
However, such a single-mix setup presents a single point-of-failure and requires all communicating parties to unconditionally trust the mix. Therefore, multiple mixes are used in practice. Ways of doing so fall into two basic categories, *mix cascades* and *mix networks*: In mix networks, senders choose a path of mixes they trust from a set of available mixes prior to sending a message, while the concept of a mix cascade defines a single valid path of mixes which is used by all communicating parties [Berthold, Pfitzmann and Standtke, 2001, p. 32].

Operating a mix cascade obviously requires high bandwidths, since all traffic is routed through the same path. This can be somewhat mitigated by offering a set of cascades to choose from. In any case, users still need to unconditionally trust the operators of mix cascades since the individual user has no say in the choice of mixes. On a global scale, this means that the operator of a mix cascade is in fact a trusted third party and therefore also constitutes a single point of failure. This is a direct contradiction to the assertion that no trusted third party is needed as it was made in the original proposal by Chaum [Chaum, 1981, p. 84]. In addition, "mix cascades have intrinsic single points of failure. The failure of any node in the cascade will bring the operation of the whole cascade to a halt. Bypassing the failed node will require manual intervention, and key redistribution" [Böhme et al., 2005, p. 250].

Availability– and trust-arguments alone might be considered an indication that mix networks are preferable to mix cascades. Decades after the original proposal of the concept and years after the discussion about which concept is superior, global-scale widely-accepted onion routing networks such as *Tor* [Dingledine, Mathewson and Syverson, 2004] seem to have proven the superiority of mix networks over mix cascades. However, at first glance low-latency onion routing concepts offer weaker security guarantees:

> While Chaum's Mixes could store messages for an indefinite amount of time waiting to receive an adequate number of messages to mix together, a Core Onion Router (COR) is designed to pass information in real time, which limits mixing and potentially weakens the protection. Large volumes of traffic (some of it perhaps synthetic) can improve the protection of real time mixes. [Syverson et al., 2001, p. 98]

The potentially high-latency characteristic of the original mix design, on the other hand, makes such a system unfit for many low-latency applications – even web browsing. However, applications where messages are easily deferrable without compromising the applications' utility (and usability) are still candidates for integration of the described mix scheme. Furthermore, mix cascades may be suitable for the use in scenarios where all communicating parties must inherently trust the operator of a mix cascade. These, however, are special use-cases and when seeking anonymous communication on a global scale, trust becomes a key issue: "This choice [about who to trust, author's note] cannot, and must not, be 'outsourced' to any other third party" [Böhme et al., 2005, p. 250]. Therefore, distributing trust amongst many distinct entities, as it is done within mix networks, is considered another argument in favour of mix networks.

3.3.2 Early Anonymity Networks by Example

General-purpose anonymity networks are typically based upon some form of onion routing due to the low-latency capabilities of the approach. This section will only deal with such networks and not focus on anonymous storage systems, anonymous remailers, or anonymising proxy services. The key characteristic of general-purpose anonymity networks is considered "not to provide anonymous communication. Parties are free to (and usually should) identify themselves within a message. But the use of a public network should not automatically give away the identities and locations of the communicating parties" [Goldschlag, Reed and Syverson, 1996, p. 138]. While an anonymity network which does not aim for anonymous communication may initially seem like an oxymoron, it actually perfectly describes a service to anonymise web traffic, for example:

> [. . .] imagine a researcher who uses the World Wide Web to collect data from a variety of sources. Although each piece of information that he retrieves is publicly known, it may be possible for an outside observer to determine his sensitive interests by studying the patterns in his requests. Onion Routing makes it very difficult to match his HTTP requests to his site. [Goldschlag, Reed and Syverson, 1996, pp. 138–139]

Of course, this approach can be generalised and anonymity networks can be considered services providing at least relationship anonymity from the network's point-of-view. If implemented accordingly, the use of an anonymity network also does not impact user experience negatively. This is all the more true, if such networks can be used by means of a proxy, as this approach transparently provides the desired anonymity properties to both users and service operators.

The original *Onion Routing*[8] proposal by Goldschlag, Reed and Syverson [Goldschlag, Reed and Syverson, 1996] describes the foundations for later low-latency anonymity networks such as Tor. The already discussed onion routing approach is used in conjunction with untraceable return addresses to construct virtual circuits on top of encrypted links between nodes of the anonymity network. Once established, all traffic is routed through these circuits [Goldschlag, Reed and Syverson, 1996, p. 142]. Depending on the point-of-view, this network can be classified anywhere ranging from decentralised to centralised: From the user's point-of-view, the network acts as a single entity operated by (potentially) many individual parties and the users do not (and cannot easily) partake in the network, giving it the characteristics of a centralised system.

More sophisticated implementations of the concept, such as *Tarzan*, aim for a decentralised design, where users can also easily join the network and contribute to the onion routing scheme [Freedman and Morris, 2002, p. 199]. Naturally, this raises issues, such as how to combat Sybil attacks and how to distribute keys and, more fundamentally, how to bootstrap the network. DHT designs such as Kademlia have already illustrated that bootstrapping and scalability are basically non-issues if discovery and routing mechanisms are implemented accordingly. Tarzan, however, does not follow the DHT approach and requires every node to connect to every other node in the network in order to validate other nodes' identities [Freedman and Morris, 2002, pp. 196–197]. *Onion Routing*, on the other hand, does not even touch the issues of key distribution and bootstrapping, but recognises these issues as open ones as it was never intended to be deployed

[8] In order to distinguish between the concept of onion routing and the original implementation of the mechanism, known under the same name, *Onion Routing* will be used when referring to the implementation.

in a real-world scenario, but rather to present the technical foundations for a low-latency mix network [Goldschlag, Reed and Syverson, 1996, p. 147].
Any design aimed at a real-world scenario, however, needs to deal with these issues and needs to present defence mechanisms against various attacks. As an example, Tarzan employs a heuristic to combat Sybil attacks in addition to basic validation mechanisms in order to realise a fully decentralised architecture [Freedman and Morris, 2002, p. 197]. The (very real) issue of malicious nodes is, in general, mitigated by the onion routing approach, since each node only has knowledge about traffic flowing from and to adjacent nodes. Message authentication codes can offer further tamper resistance and prohibit the tracking of messages through the onion routing network [Freedman and Morris, 2002, p. 202].

However, anonymity networks which do not employ onion routing have also been proposed and therefore the security properties provided by onion routing approaches do not hold. *Crowds*, for example, tries to hide the origin of a message by routing it through multiple nodes just as mix networks do. Yet, no layered encryption is employed and therefore every node has full access to all messages passing through the network, even if TLS is employed [Reiter and Rubin, 1998, pp. 76, 80]. The design of Crowds mandates this unrestricted access since (in contrast to onion routing) the route of any message is not defined by the sender: Each hop (randomly) chooses either to forward a received message to its final destination or to another intermediate hop [Reiter and Rubin, 1998, p. 73]. Obviously, this requires users to fully trust all nodes of the network.

As already outlined in the beginning of this section, the complex nature of network security in combination with anonymity properties provided by different designs makes it hard to judge the overall quality of such systems. The differences between *Onion Routing* and Tarzan alone perfectly illustrate this fact: Both are onion routing schemes, however, Tarzan even strives for a certain degree of unobservability from the network's point-of-view by introducing cover traffic [Freedman and Morris, 2002, p. 202]. Different schemes, such as Crowds, do not aim at hiding correlation between messages entering and leaving nodes and do not even try to counter DoS attacks [Reiter and Rubin, 1998, p. 71, 75]. These different system properties make it clear that, while all systems hold up to the information-theoretical evaluation proposed by Díaz et al. [Díaz et al., 2003], such results are of little relevance in practice (unless a system does not hold up to such an evaluation). However, the strengths and weaknesses of these designs are practically irrelevant, since none of the early anonymity networks were deployed on a global scale; only Tor, an evolution of *Onion Routing*, prevailed.

3.3.3 Current Anonymity Networks

The early anonymity networks, as already mentioned, have been superseded by more sophisticated designs specifically aimed at withstanding real-world threats even when deployed on a

global scale. Currently, only *Tor*, *I2P*, and *Freenet* are of relevance; any system having too few users by definition features a small anonymity set and is also much more likely to fall victim to a variety of attacks. Not only is it much easier to overpower a small network, but DoS attacks are also easier to mount, since smaller networks also have less resources at their disposal to withstand such attacks. This situation can, however, lead to a vicious circle, if the majority of users prefer larger networks, simply due to network size. Conversely, few new users may be willing to join smaller networks (also due to network size).
Finally, widely used systems are more likely to be subject to scrutinisations which can reveal various flaws. Ideally, this results in corrections, which in turn lead to an overall more correct and secure system. Open-source designs make it even possible for users to dissect their implementations and contribute improvements. Therefore, only systems with implementations being available as open-source software will be considered.

3.3.3.1 Tor

Tor is a low latency onion routing network aimed at anonymising TCP traffic in an application-independent manner, "so long as the application supports, or can be tunneled across, TCP" [Dingledine, Mathewson and Syverson, 2004, p. 4]. It is designed as a centralised P2P network, where a small set of trusted nodes, called *directory servers*, serve network state information, the *directory*, to other nodes [Dingledine, Mathewson and Syverson, 2004, p. 12]. This state information includes the cryptographic keys of the network's nodes; the directory servers are thus responsible for key distribution and are, by extension, also responsible for the authentication of nodes. In short, the directory servers assume the role of a central authority, a trusted third party as described in the network security model discussed in Section 3.2.1. This obviously leads to a single point-of-failure. However, the potential of directory server subversion is mitigated due to multiple directory servers being present: Tor clients are designed to require a certain threshold of directory servers to reach a consensus on the current network state before accepting an advertised directory [Dingledine, Mathewson and Syverson, 2004, p. 12]. Yet, concerns about the resilience of this design against targeted attacks remain.
On the other hand, Tor is designed to perform well against a multitude of threats and features a highly detailed attack-model, has well-defined goals and non-goals, and is (due to its popularity) continuously undergoing peer review. This, in effect, leads to an increasingly hardened system. In addition, users are urged to familiarise themselves with the key characteristics of Tor, the security services it provides, and to which extent these are provided. Specifically, the grave impact of information leakage through side channels is highlighted [The Tor Project, Inc., no date(d)].

Obviously, since Tor follows the P2P design, a well-defined system and educated users are not enough to guarantee security. Malicious nodes will always be a relevant factor. However, as with all onion routing designs, the impact of such nodes depends on their role within the network. Nodes joining the Tor network can assume three roles: *entry/guard node*, *relay*, and *exit node.*

Pure relays never act as final hop, but only forward incoming messages destined for other nodes (no traffic exits the Tor network), whereas exit nodes are relays configured to allow for traffic to leave the Tor network to connect to arbitrary hosts on the Internet [Dingledine, Mathewson and Syverson, 2004, p. 11]. Tor nodes can be configured to act either way. Entry nodes, as the name implies, are the first hop in the path through the onion routing network.
An attacker controlling the first or last hop can effectively de-anonymise traffic routed through these nodes to varying degrees [Syverson et al., 2001, p. 106]. Consequently, a malicious entity controlling both the first and the last hop can mount correlation attacks to further de-anonymise traffic. Tor tries to mitigate this threat by the introduction of guard nodes.

> To help improve this situation [...] the Tor client picks a few Tor nodes as its "guards", and uses one of them as the first hop for all circuits [...]. Essentially, the guard node approach recognises that some circuits are going to be compromised, but it's better to increase your probability of having *no* compromised circuits at the expense of also increasing the proportion of your circuits that will be compromised if any of them are. [Mathewson and Murdoch, no date]

Essentially, the status *guard node* is assigned by the directory servers, if a node has been reliably serving the network for at least eight days [Dingledine, no date]. This increases the cost of operating a malicious node at sensitive points on a routing path.

Tor also defines a special type of entry nodes, called *bridges*. Contrary to regular entry nodes, bridges are not published by directory servers in the hopes of circumventing censorship: Users need to manually configure their local Tor instance to use a bridge as first hop, as these are unlisted [McLachlan and Hopper, 2009, p. 32]. For the same reason, it is also impossible for a network operator or *internet service provider* (ISP) to automatically prevent users from connecting to the Tor network.
Furthermore, so-called *pluggable transports* are supported to counter *deep packet inspection* (DPI). "Pluggable transports transform the Tor traffic flow between the client and the bridge. This way, censors who monitor traffic between the client and the bridge will see innocent-looking transformed traffic instead of the actual Tor traffic" [The Tor Project, Inc., no date(c)], thus making Tor resilient even to DPI.

Additionally, Tor provides means to operate services, which do not require exit nodes, so called *(location-)hidden services* [Dingledine, Mathewson and Syverson, 2004, pp. 9–10]. This not only enables the anonymous operation of servers, but also rules out various attacks aimed at identifying clients and de-anonymising traffic, since exit nodes do not exist as an attack surface. Both operating hidden services and simply anonymising client traffic works through the same mechanism: The Tor software used to join the Tor network, called the *onion proxy*, acts (as the name implies) as a SOCKS proxy and is thus capable of anonymising traffic transparently to the application:

> Bob configures his onion proxy to know the local IP address and port of his service, a strategy for authorizing clients, and his public key. [...] Bob's web-server is unmodified, and doesn't even know that it's hidden behind the Tor network.
> Alice's applications also work unchanged—her client interface remains a SOCKS proxy. We encode all of the necessary information into the fully qualified domain name (FQDN) Alice uses when establishing her connection. Location-hidden services use a virtual top level domain called `.onion`: thus hostnames take the form `x.y.onion` where `x` is the authorization cookie and `y` encodes the hash of the public key. Alice's onion proxy examines addresses; if they're destined for a hidden server, it decodes the key and starts the rendezvous. [Dingledine, Mathewson and Syverson, 2004, p. 10]

The aforementioned rendezvous is a procedure intended to keep clients and servers from discovering each other's identity, including their IP address and location. In essence, client and server do not directly connect to each other, but build virtual circuits to intermediaries, which then extend the circuits to each other [Dingledine, Mathewson and Syverson, 2004, pp. 9-10]. This way, the client does not need to know the server's identity, but only the location of the intermediary (called *rendezvous point*) chosen by the service operator. Naturally, de-anonymisation of hidden services is always possible through information leakage, such as protocol headers, timezone disclosure, and so forth. However, the anonymisation of, for example, protocol headers and meta-data on the application layer is left up to external supporting applications [Dingledine, Mathewson and Syverson, 2004, p. 13].

Finally, in addition to being available for a multitude of operating systems (even on mobile platforms) and being easy to integrate with any TCP-based application, Tor provides extensive documentation and detailed, near-real-time metrics of the network. As of September 2015, Tor had roughly 2,000,000 simultaneously active, directly connected clients [The Tor Project, Inc., no date(a)]. While this is only a small fraction of all Internet users, in absolute terms it is obviously a large anonymity set. The number of relays, on the other hand, is multiple orders of magnitude smaller, with less than 7,000 simultaneously running relays as of September 2015 [The Tor Project, Inc., no date(b)].

3.3.3.2 Freenet

In contrast to general-purpose anonymity networks, *Freenet* primarily aims at providing distributed, anonymous data storage in a fully decentralised manner [Clarke et al., 2001, pp. 46-47]. While having DHT-like characteristics, with data essentially being identified by keys, its goals are somewhat different. Most importantly, security and privacy are integral properties of the design, such as "Anonymity for both producers and consumers of information [...], Deniability for storers of information [...], Resistance to attempts by third parties to deny access to information" [Clarke et al., 2001, p. 47].

The anonymity properties are achieved using the mechanism originally employed by Crowds, where requests are not submitted directly, but through a number of hops, where each hop decides independently whether or not to submit the request or forward it to another intermediary [Clarke et al., 2001, p. 49]. Thus, as with Crowds, a single malicious node along a path is not only able to eavesdrop, but also to manipulate any message passing through it. Freenet introduces a social approach to combat this issue: An alternate mode of operation, called *Darknet*, employs a routing algorithm where “nodes only connect to trusted contacts, which have to be added manually” [Roos et al., 2014, p. 265]. While this approach eliminates the need for a trusted third party, and thus serves the decentral spirit of Freenet, it inherently requires proactive, manual user action which raises the question of scalability.
The aforementioned deniability property is achieved by means of encryption: In the simplest case, a data’s key is the cryptographic hash of a descriptive string. This string, at the same time, serves as the (cryptographic) key used to encrypt the data prior to storage [Clarke et al., 2001, p. 49]. Assuming strong cryptographic functions are employed, the content and the description of some data cannot be recovered by the node it is stored on, thus enabling plausible deniability.

While originally designed as distributed storage system, any DHT-like structure can also be (mis)used for other purposes. This potential was recognised by the community surrounding Freenet and the original design was since extended and used as a foundation for providing (amongst others) an e-mail system and web hosting within Freenet [Roos et al., 2014, p. 264]. In general, Freenet is a closed network in the sense that it does not allow for anonymising web traffic or connecting to external networks[9]; it provides certain services with the aforementioned security properties. It is therefore not possible to use Freenet as a secure transport layer for arbitrary applications. Finally, with only some tens of thousands of estimated Freenet installations as of 2012, the anonymity set provided by Freenet is rather small [Roos et al., 2014, p. 275]. Current statistics do not account for the number of individual installations, but record the number of simultaneously active nodes, which amounts to an estimated anonymity set size of roughly 9,500 nodes as of September 2015 [Dougherty, no date].

3.3.3.3 I2P

I2P is designed as a general-purpose onion routing network, primarily aimed at connecting I2P-based applications “via a fully decentralized peer-to-peer network, which runs as an overlay on top of IP” [Egger et al., 2013, p. 434]. I2P thus strives for tight integration of services and applications. The design as a whole, however, is considered to be a hybrid P2P network: Network state information is distributed to nodes, called *routers*, through a Kademlia-based DHT, the *netDB*, which is formed by a small percentage of all nodes meeting specific criteria such as reliability– and bandwidth-constraints [Timpanaro, Chrisment and Festor, 2012, p. 138]. Every router, by default, contributes to the onion routing scheme. The peers forming the DHT

[9]The World Wide Web is here also considered an external network.

are called *floodfill routers*. This approach eliminates a single point-of-failure. On the other hand, it inherently relies on the cooperation of many individual nodes, which are implicitly trusted. By default, I2P routers are configured to use a small set of predefined seed nodes (and their public keys) to automate the bootstrapping process [Schimmer, 2015]. This property can be utilised to provide some degree of censorship-resistance: In case the default seed nodes are blocked (for example, by an ISP), simply using other seed nodes makes it possible to join the network.

Another way to look at I2P is from a software-architectural point-of-view. From this perspective, I2P is an application framework, as it allows for tight integration of arbitrary client– and server-software, using either "a TCP-like protocol called `NTCP` or a UDP-like protocol called `SSU`" [Egger et al., 2013, p. 434]. This enables connection-oriented applications (such as HTTP) as well as media streaming or file sharing (which would traditionally be done using UDP) to be implemented as I2P applications. With every hop being part of an onion routing network and thus not generating outbound traffic, attacks based on traffic entering and exiting the network can obviously not be mounted.

The following example based on the illustrations by Timpanaro, Chrisment and Festor [Timpanaro, Chrisment and Festor, 2012, pp. 136–137] demonstrates how both services and clients can be used anonymously according to the model defined by I2P:

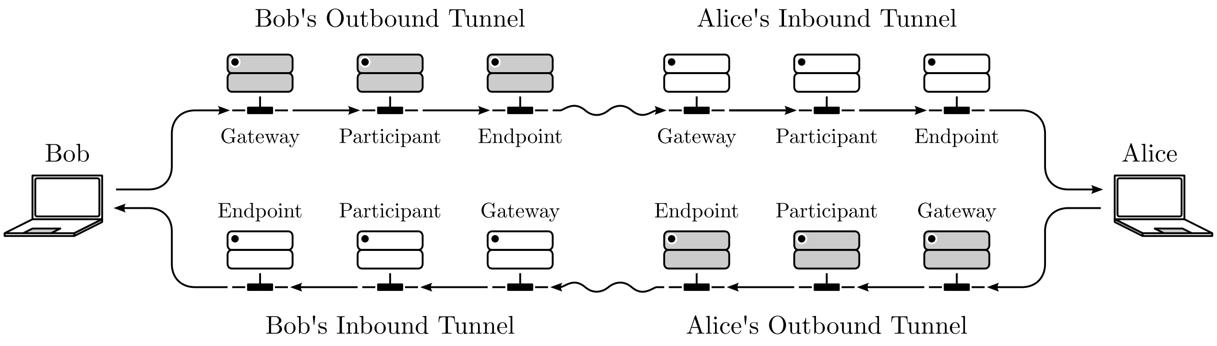


Figure 3.6: Tunnel-oriented communication in I2P based on Timpanaro, Chrisment and Festor [Timpanaro, Chrisment and Festor, 2012, p. 137]

Suppose Bob wants to run an I2P-based service to share information with Alice: Once Bob's service is up and running, he will instruct his router to set-up *inbound tunnels* and *outbound tunnels* connected to his service. As the name implies, inbound tunnels will be used to receive incoming client requests, and outbound tunnels will be used to serve data. Alice's router also creates outbound and inbound tunnels to be able to send and receive information. Due to onion routing being employed, each tunnel consists of a *gateway* through which information is sent into the tunnel, a number of intermediate hops, called *participants*, and an *endpoint*, which either connects to a service or to the gateway of an inbound tunnel. The tunnels connecting to Bob's service can be reached by means of a *destination* published to the netDB. The data containing information about the destination associated to Bob's service, such as gateways, endpoints, and public keys, is called a *leaseset* [Egger et al., 2013, p. 435]. Alice's tunnels connect to Bob's tunnels as illustrated in Figure 3.6. This way, the only information participants can learn about each other are which peers serve as gateways and endpoints, thus preserving anonymity.

I2P specifically advertises the tight integration of a number of services [The I2P Project, no date(b)]. While possible through so-called *outproxies*, connecting to outside services (which essentially means using I2P as an anonymisation service to connect to unmodified web services) is not actively pursued or even recommended; I2P actually relies on Tor to enable anonymous web browsing due to the scarcity of available outproxies [Schimmer, 2015].
As far as integrating services is concerned, I2P provides application-specific mechanisms (such as HTTP proxies and an *Internet Relay Chat* (IRC) server) to do so. These also implement protocol cleaning to prevent information leaks, such as users' IP addresses being listed in protocol headers [Schimmer, 2015]. Additionally, it is also possible to integrate any socket-based application into I2P either through a SOCKS proxy or directly by using the I2P *streaming library.*

Lastly, I2P is available on Android and on the most popular desktop platforms, however, the anonymity set provided by I2P is rather small: As of September 2015, the estimated number of routers is roughly 30,000 [The I2P Project, no date(c)]. The lack of manpower, funding, and outside review is also an issue. However, these facts are explicitly documented on the project's web site [The I2P Project, no date(a)].

3.3.3.4 Current Anonymity Networks by Comparison

Essentially all anonymity networks aim at supporting freedom of information, escaping global-scale surveillance as well as circumventing censorship. When comparing current anonymity networks, there is, however, an obvious similarity between Tor and I2P, while Freenet seems to strive for somewhat different goals. Yet, there are also technological similarities. Although Freenet was originally designed merely for anonymous information distribution, it has evolved to offer web hosting and an e-mail system, both of which can also be implemented on top of Tor or I2P. However, the latter two systems are designed to be generic and thus (almost) arbitrarily extensible. Therefore, even though Freenet is still comparable in size to I2P and its network size even exceeds the number of Tor relays[10], it will not be discussed further, since it is not (yet) a general-purpose anonymity network. The remainder of this section will therefore focus on the comparison between Tor and I2P.

Clearly, the anonymity sets of I2P and Tor differ by multiple orders of magnitude in favour of Tor. In addition, it is possible to almost completely masquerade the use of Tor to the network by employing pluggable transports while it can be considered public information whether someone is running an I2P router[11]. Tor also has to be considered more censorship-resilient due to the availability of pluggable transports, since the use of bridges in Tor essentially provides the same

[10]All of these numbers are just estimates, since it is technically impossible to provide perfectly accurate figures.
[11]Since router information is published to the netDB, anyone joining the I2P network can enquire this information.

benefits as using custom seed nodes in I2P. While censorship is not an issue in most parts of the world, it is still of high relevance: Censorship is commonly associated with repressive governments which is exactly the kind of situation where freedom of information matters most. The simple fact of I2P being far less popular makes it a less attractive target for directed censorship, while still providing an anonymity set large enough to make it impossible to observe the actions of individual users. This statement obviously only holds assuming that I2P works as advertised. This is hard to validate, since I2P openly admits the lack of outside review.

Tor's large user base and the large number of publications on Tor, on the other hand, point to a strong public interest in the system. According to Linus's Law[12], this results in an overall more correct and therefore more secure system [Raymond, 2001, p. 30]. The simple fact that both systems are still operational as of October 2015 is no indicator for the security and correctness of these systems: It is possible that weaknesses of any system are actively being exploited without anyone noticing. There is, however, a strong indicator that Tor does hold up to its advertised security properties: The case of *Silk Road*, "an online anonymous marketplace" [Christin, 2013, p. 214] was operated as a Tor hidden service. "Given the nature of the goods sold on Silk Road, it is quite clear that various law enforcement agencies have a strong interest in disrupting Silk Road operations. They appear, so far, to have been unsuccessful since the site is still up and has grown in size" [Christin, 2013, p. 221]. Even though now defunct, the fact that Silk Road remained operational for more than two years [Anderson and Farivar, 2013] serves as an empirical illustration of the security of Tor.

Given the sheer amount of research focusing on Tor and how difficult it apparently is to de-anonymise Tor hidden services in a real-world scenario, Tor does seem suitable for providing secure, anonymous communication. This can be utilised to evade mass surveillance, escape repressive Internet policies and support the exchange and discussion of dissident views without needing to fear prosecution. Unfortunately, less popular systems such as I2P generally lack regular scrutinisations, security assessments, and well-examined case studies. Therefore, more popular systems are preferable, as the advertised security properties are more probable to hold. Given similar portability and availability of Tor and I2P, this leads to the conclusion that Tor is preferable as a secure transport layer due to its higher popularity.

[12]"Given enough eyeballs, all bugs are shallow" [Raymond, 2001, p. 30].

4 CrySIL

The *Cryptographic Service Interoperability Layer* (CrySIL) is a framework encompassing a set of interfaces and a protocol designed to provide cryptographic operations on a variety of platforms. Furthermore, it enables off-device cryptography and off-device key storage if required [Reimair, Teufl and Zefferer, 2015, p. 42]. As the name implies, it is designed as a cryptographic interoperability layer, enabling arbitrary sets of cryptographic operations in web applications and on devices of any form factor. While cloud-based solutions exist to bring complex cryptographic capabilities to different devices, these solutions typically cater to specific needs and therefore lack CrySIL's flexibility.
The document management system *SigningHub* [Ascertia Limited, 2015a], and the Austrian mobile phone signature [Digital Austria, 2015], for example, offer off-device key storage, but are limited to providing digital signature services. CrySIL, aiming at unifying the interfaces of different services, can integrate such existing solutions to incorporate their features. Given an API is available to enable the integration of an existing service, CrySIL can provide a superset of this service's features through integration. This design enables CrySIL to offer any kind of cryptographic operation including digital signature services, encryption and authentication. SigningHub, for example, advertises its REST API as a way to integrate it into other solutions [Ascertia Limited, 2015b]. Proof-of-concept implementations relying on external services for authentication are provided with the CrySIL prototype. *OAuth*, "An open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications" [The OAuth Community, 2015], is one such service which can be used with CrySIL.

This ability to integrate external services is accomplished through a modular and extensible design, which enables the creation of both general as well as highly application-specific configurations. Its modular architecture and the availability of a prototype capable of a multitude of cryptographic operations makes CrySIL an ideal candidate for developing custom solutions and exploring novel approaches with respect to authentication and secure information exchange. Therefore, the CrySIL architecture and the structure of its key components shall be explained in more detail.

4.1 Architecture Overview and Terminology

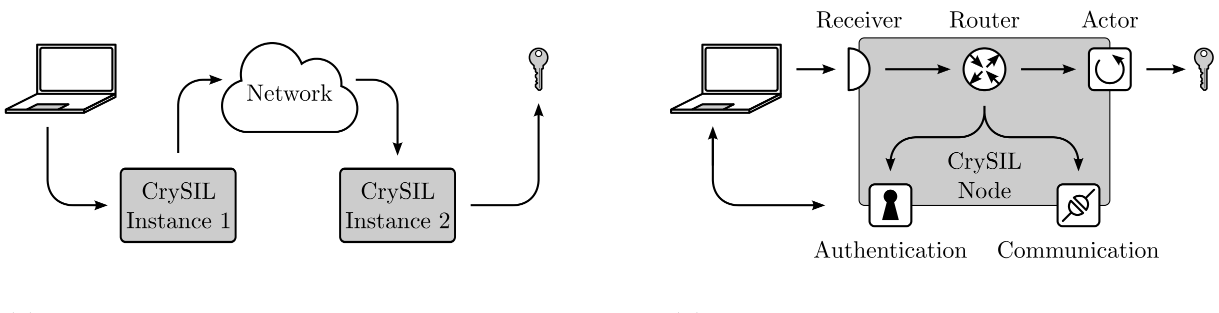
CrySIL is designed to be as modular as possible. Being a cryptographic interoperability layer, "CrySIL does not implement any cryptographic service itself. It solely integrates existing solutions and makes them accessible over various APIs even on other devices" [Reimair, Teufl and Zefferer, 2015, p. 42]. Individual modules can be combined and configured as the use case requires. To achieve a high degree of flexibility, several classes of modules exist, each having a well-defined API. This allows for configurations to span over multiple devices [Reimair, Teufl and Zefferer, 2015, p. 39]. Due to different features being explicitly modelled as modules, these shall be explained in more detail.

The following description of the different CrySIL modules is based mainly on the first publicly available architecture description by Reimair, Teufl and Zefferer [Reimair, Teufl and Zefferer, 2015] and review of the CrySIL source code. At first, however, the basic CrySIL-specific terminology needs to be defined.

A CrySIL instance on a single device composed of *modules* is referred to as a *node* and at the very least incorporates the following modules: A *receiver* exposing cryptographic functions, an *actor* performing cryptographic operations, and a *router*, responsible for relaying requests from the receiver to the actor. While not particularly useful in a single-receiver-single-actor configuration, in setups featuring multiple actors each exposing different functionality, the router is responsible for choosing the appropriate actor for each request.

More advanced actors exposing higher-level or application-specific features (called *actors+*) potentially relying on other actors may also be present. Furthermore *communication modules* (typically incorporated into the router) responsible for connecting multiple CrySIL nodes in a distributed setup are also defined. A typical scenario employing a distributed setup is off-device key storage which can be required by certain security policies. A high-level view of such a distributed setup is depicted in Figure 4.1a.
Finally, one or more *authentication modules* implementing different authentication methods are usually in place depending on the required security level (see Section 2.3.2 for more details on authentication).

For every type of module, ready-to-use prototypical versions exist and can be used for creating CrySIL nodes. Of course, custom modules can also be built meeting specific needs of a given setup. Commonly, the actors, the communication–, and authentication modules will be subject to such customisations, given the already provided cryptographic functions suffice for the application at hand. The receiver, which typically acts as an interface to the programmer utilising a CrySIL instance, is decoupled from the authentication process and therefore not usually subject to large changes.
The architecture of a CrySIL node based on Reimair, Teufl and Zefferer [Reimair, Teufl and Zefferer, 2015, p. 39] is illustrated in Figure 4.1b.



(a) High-level view of a distributed CrySIL setup for off-device key storage

(b) The architecture of a CrySIL node based on [Reimair, Teufl and Zefferer, 2015, p. 39]

Figure 4.1: The CrySIL architecture

In the following, the structure of the actors and the communication modules provided with the CrySIL prototype will be elaborated on. Furthermore, the authentication workflow will be explained in detail, since it is considered one of the most crucial parts of the system; as with any system dealing with sensitive data, access control policies need to be enforced to protect this data.

4.2 Actors and Communication Modules

Actors execute commands issued to the receiver by a CrySIL-based application. As already mentioned, router modules act as the central building block connecting receivers to actors. The interfaces between these modules are designed to be as generic as possible. The `Actor`-interface itself only defines a single method, `take`, which operates on a single parameter of the type `SRequest`, returning an `SResponse` upon completion of the requested operation. This design can be considered somewhat atypical given well-established OOP paradigms such as polymorphism, abstract classes, and interfaces are commonly employed in similar situations. However, it does ensure maximum flexibility (albeit at the cost of having to manually implement control-flow logic to execute different operations and error handling based on the received command type).

On the other hand, this generic interface makes it possible to easily implement proxy-actors, relaying any command to remote CrySIL nodes. As previously stated, the router is intended to handle delegating of commands to different (possibly remote) actors. However, setups are conceivable in which both local and remote nodes are involved and for the sake of simplicity (or cleaner, more consistent abstractions) a single actor may incorporate both local and remote cryptographic operations. Furthermore, situations can arise in which the final destination of a command can only be determined after performing a cryptographic operation. In such cases, the actor is an adequate location to implement remote calls.

The communication module is in general more of a conceptual construct than an actually defined interface or a discrete building block. Sometimes, it can simply be significantly easier and more

sensible to tightly integrate the communication logic into an actor rather than creating a sophisticated command-agnostic router implementation.
Furthermore, the CrySIL prototype already provides well-defined, platform-independent marshalling mechanisms to ease the implementation of multi-device setups. These are based on the *JavaScript Object Notation* (JSON) most recently defined by Bray [Bray, 2014] as a proposed standard. Although not particularly expressive as a transport format, its simple structure makes JSON an ideal choice whenever portability and easy integration of different technologies is a priority.

4.3 Authentication Workflow

The authentication interfaces present in CrySIL enable a complete separation of the authentication mechanisms from the cryptographic operations. This separation is achieved to such an extent that not even the communication modules, connecting multiple nodes, need to be adapted whenever authentication-related workflows change. This total abstraction does, however, come at a price: complexity. The authentication module consists of multiple interconnected components spanning over all involved nodes. Executing even the simplest of authentication protocols (or just a dummy authentication protocol allowing anyone to perform all operations) involves more than ten steps performed by seven sub-modules. These are:

Interceptor – Intercepts commands issued by the user

Authorisation process – Stateful representation of the authorisation/authentication process

Gatekeeper – Enforces the access control policies and executes the authorisation process

Authentication plug-ins – Define and provide authentication method definitions to the authorisation process

Configuration – Provides authentication plug-in configuration to the gatekeeper

Authentication dialogues – Define the user-facing interfaces for authentication (may not involve any graphical user interface)

Authentication selectors – Select and execute the appropriate authentication dialogue based on the received authentication challenge

An *interceptor* is set up as part of an authentication module when configuring a CrySIL node. Multiple interceptors can be registered with the router, typically at the node where sensitive data needs to be protected. For example, a node responsible for key storage will usually intercept any incoming request and issue authentication challenges instead of directly providing access to all key material.
The general nature of the CrySIL API, operating on any type of `SRequest` and returning the

result in form of an `SResponse`, makes it possible to implement request interception independently of cryptographic functions. This is achieved by defining authentication-related requests and responses as subtypes of `SRequest` and `SResponse`. This way, the protocol remains consistent even if authentication policies are added or removed. In short, the authentication workflow is fully decoupled from all actors. A requirement exists, however, to achieve strict separation of concerns: The actor must invoke the gatekeeper (given a gatekeeper is configured) prior to executing any command received. This makes it possible to integrate different authentication methods and access control policies into any given CrySIL configuration.

The actual execution of an authentication scheme is performed by the *gatekeeper*, based on the representation of authentication workflows defined by the *authorisation process*.
The authorisation process itself is an abstract, generic definition of a time-discrete one-way (although not necessarily acyclic) process composed of an arbitrary number of steps. Therefore, it does not define any authentication methods and –factors. The gatekeeper simply performs one step after the other. This simple model enables the implementation of a workflow applicable to all conceivable authentication protocols. The actual definition of an authentication method, including the number of steps it is composed of, is provided by an *authentication plug-in*. The *configuration* is then used to register the desired plug-ins with the gatekeeper and to forward authentication-related information to the appropriate plug-ins. This, again, ensures maximum flexibility by making it possible to provide arbitrary configurations to the gatekeeper. The gatekeeper and the authorisation process never operate on any information specific to an authentication method, as this is done by the authentication plug-ins.

Finally, the user-facing part of a CrySIL application needs to be able to respond to authentication challenges. This can be achieved by implementing an *authentication selector* which (possibly automatically) selects and executes the *authentication dialogues* corresponding to any received authentication challenges. An authentication dialogue must implement a set of predefined methods (such as presenting a challenge to the user and storing user input) as well as triggering callbacks when the user has finished responding to a challenge.

Although the name implies otherwise, none of the steps executed by an authentication dialogue necessarily involves a graphical user interface. A basic password-based authentication, checking user-provided password strings against a set of predefined passwords, can easily be implemented by simply relying on a command-line user interface. Other, more involved authentication methods based on inherence– or possession-factors, on the other hand, may mandate more elaborate user action and can incorporate special hardware-interfaces or remote procedure calls. In addition, the capabilities of different devices can vary greatly, often making it necessary to adapt the user interfaces depending on the device an application is deployed to.
Furthermore, not all authentication methods mandate the same degree of integration with an application's user interface, which also speaks to implementing the user-facing part of the authentication process separately from the definition of authentication methods. The required

security level is another key aspect which can mandate different authentication methods based on the environment an application is executed in.

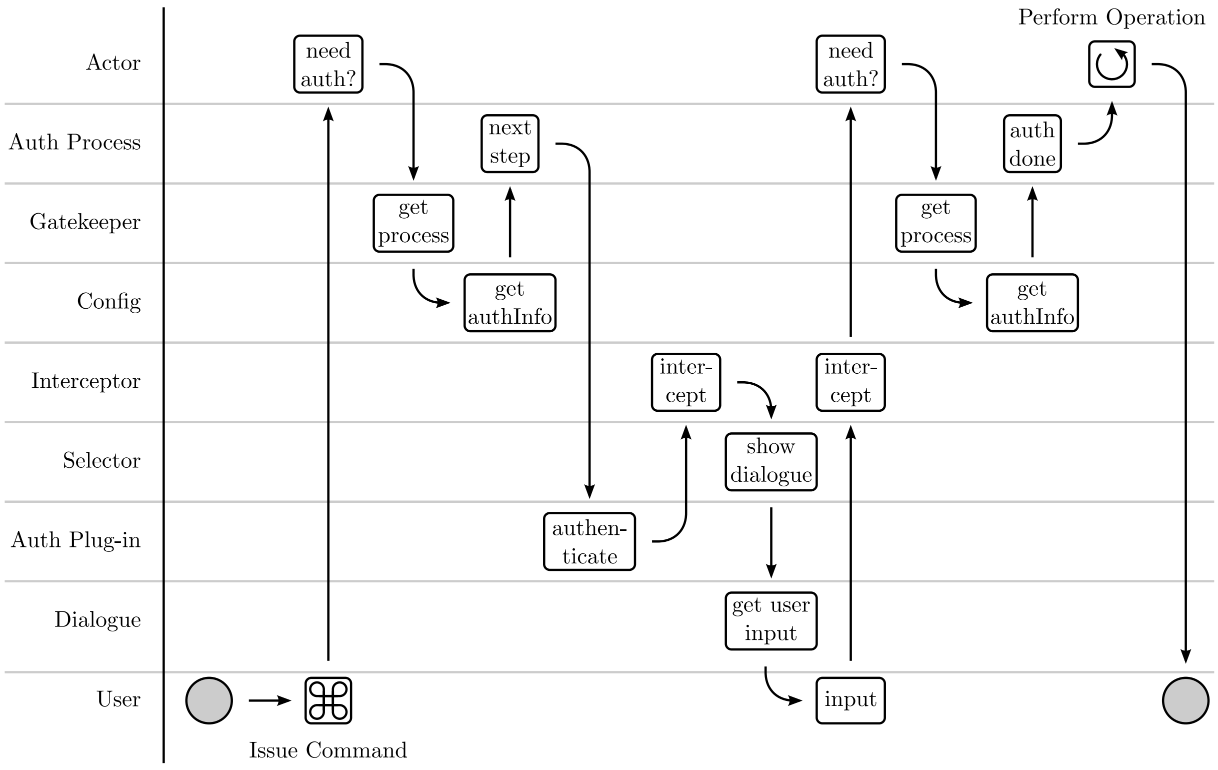


Figure 4.2: Workflow of a user issuing a request to a CrySIL instance and successfully standing the required authentication challenge. Even after successfully completing the first challenge, an additional check is performed by the gatekeeper, since multiple authentication steps could be required. The selection of appropriate authentication plugins is in this case performed automatically.

The interception of commands and the authentication process as a whole is admittedly complex. A basic example is therefore illustrated in Figure 4.2 which depicts the process of successfully performing a single cryptographic operation requiring authentication.

As previously mentioned, the authentication process features such a high level of complexity due to being designed with flexibility in mind. In general, CrySIL provides an overall flexible cryptographic framework applicable to a variety of scenarios. Developed as an interoperability layer, CrySIL enables the exploration of novel authentication approaches, encryption of data for specific recipients, and other key aspects of secure information exchange.

5 Grið

In times of constant global surveillance, protecting personal data and securely exchanging information becomes increasingly important, but also increasingly difficult. Anonymity networks such as the ones discussed in Section 3.3 offer confidentiality and integrity and to some extent enable users to use familiar services while remaining anonymous to the network. Cloud-based storage solutions ease the distribution of files to selected users or groups. However, without any user-controllable authentication mechanisms or end-to-end encryption in place, such services are only of limited use when it boils to exchanging highly sensitive information.

This chapter presents a novel approach to the trust-related issues discussed in Section 2.3, while leveraging existing cloud storage solutions to enable the secure exchange of data based on the CrySIL framework discussed in Chapter 4. Relying on Tor (see Section 3.3.3.1), but also employing additional communication channels to eliminate a single point-of-failure, this work is considered an improvement over existing solutions (see Chapter 7 for more information on related work). All exchanged data is encrypted prior to transmission performed in a way which allows for the sender to remain anonymous if desired. A prototype called *Grið* has been implemented. It enables users to encrypt arbitrary data and transfer the resulting cryptogram through whichever service the users see fit. The novelty of the approach boils down to an inverted trust model explained in the following section. This trust model in combination with sender anonymity is considered the main improvement over existing solutions.
The name *Grið* has been chosen for its distinctiveness (incorporating a somewhat exotic glyph), its similarity to the English word *grid*, as well as for its meaning: *Grið* is an Old Norse word whose plural interpretation roughly translates to peace, ceasefire, protection, or security [Baetke, 1978, p. 211].

5.1 Secure Information Exchange Based on Inverted Trust

Contrary to well-established solutions such as PKIX or PGP, Grið employs an inverted trust model. The general idea is based on using hybrid encryption schemes in a similar way to CMS (see Section 2.2.2). CMS encrypts a one-time key used for bulk encryption under the recipients' public keys, which enforces the trust model supported by the recipient on the creator of an encrypted container. This can easily lead to a problematic situation, given the sender does not trust the public keys provided by the recipient or possibly even distrusts the PKI used by the recipient as a whole.

In both hierarchical trust models and webs of trust, the sender has to trust the public keys (or some certificate) provided by recipients. If this is not the case, the sender cannot in good conscience encrypt any data for the recipient. Such a predicament can, in fact, arise easily even when a recipient's key is trusted: In case a public key does not fulfil some security requirements it cannot be used. One likely way for this to happen are long key life times: If a communication party does not keep up with recommendations regarding key length, for example, a key conforming to such recommendations at some point in time can easily be in violation of a later revision of the very same recommendation.
Furthermore, any trust model based on certificates or public keys by definition involves publicly available information. In general, such information can be used to identify at least one of the involved parties.

Grið inverts the just-mentioned, well-established, and widely-used trust approach and places the trust burden on the recipient using off-device key-storage concepts. An ephemeral key is generated for each encryption process to encrypt the given data. However, the encrypting party does not need to trust any certificate or public keys of the recipients. Instead, it defines a key-expiry-time and an authentication challenge the recipients must stand in order to gain access to the ephemeral key. The combination of an ephemeral key and its corresponding authentication information (as illustrated in Figure 5.1) is referred to as a *one-time key*. A newly-created one-time key is subsequently encrypted under the sender's *long-term key* resulting in a cryptogram called the *wrapped key*.

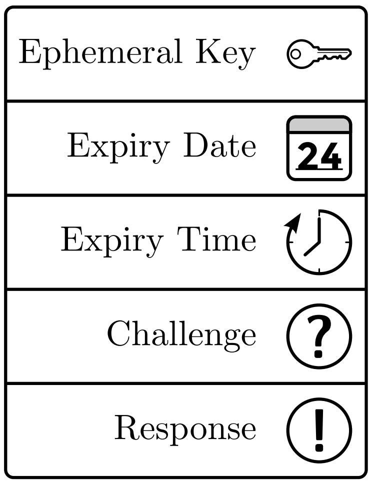


Figure 5.1: Grið's one-time key structure

Having created the wrapped key, it is sent to the recipients along with the data previously encrypted under the ephemeral key (similar to a CMS container). Since all authentication information is bound to the ephemeral key as part of a wrapped key structure, there is no need for the sender to keep track of key material. Only the long-term key which is completely unrelated to any recipients needs to be maintained secret. This approach does not require information identifying any involved party to be transmitted publicly.
However, the encrypting entity needs to be contacted by the recipients to gain access to the ephemeral key, since the sender's long-term key is required to unwrap it. Grið employs Tor hidden services (see Section 3.3.3.1) to create P2P connections between Grið instances on demand. This enables recipients to contact the encrypting entity while remaining mutually anonymous at the network layer.

Having asserted that Grið does not require parties to publicly disclose any identifying information may appear to be a contradiction to the above statement. However, since no direct correlation between some entity contacting a hidden service and some other entity disclosing a hidden service descriptor can be made without additional information, this assertion is still considered to hold. In the following section a detailed account of the data exchange workflows and more details about the employed data structures are provided.

5.1.1 Data Exchange Workflows

Due to the novel approach employed by Grið being its main contribution it shall be discussed in detail. Grið's encryption and decryption workflows differ significantly from each other and from well-established public-key– or certificate-based approaches. However, the creation of an encrypted container works similarly to existing hybrid schemes.

Encrypting data for a recipient works as follows: First the to-be-encrypted data is compressed, which is mainly done to reduce the size of the resulting encrypted container in order to ease its exchange in case bandwidth and/or storage capacity is limited. Next, an ephemeral key is created and the previously compressed data is encrypted using an efficient symmetric (authenticated) encryption algorithm employing this key. Afterwards, an authentication challenge, as well as an expected response and a key expiry date are defined and attached to the ephemeral key. The resulting one-time key is then wrapped using the encrypting entity's long-term key to produce a wrapped key as explained earlier. The encrypted payload and the wrapped key are subsequently packaged into a streaming-capable container-format. In order to ease the exchange of containers, Grið includes an interface to store these at arbitrary locations. Possible locations include cloud storage and local storage or directly sending a container to recipients through e-mail.

Regardless of the storage choice, after all data has been output, an identifier indicating the storage location of the container is presented to the encrypting entity. The main motivation for working with identifiers is to ease the exchange of containers: In case an online file hosting service (or a storage medium shared with the recipient) has been chosen, only a container's identifier needs to be exchanged. Figure 5.2 outlines the whole encryption procedure. The implemented prototype includes a local storage back-end and a direct upload back-end to an anonymous online file hosting service.

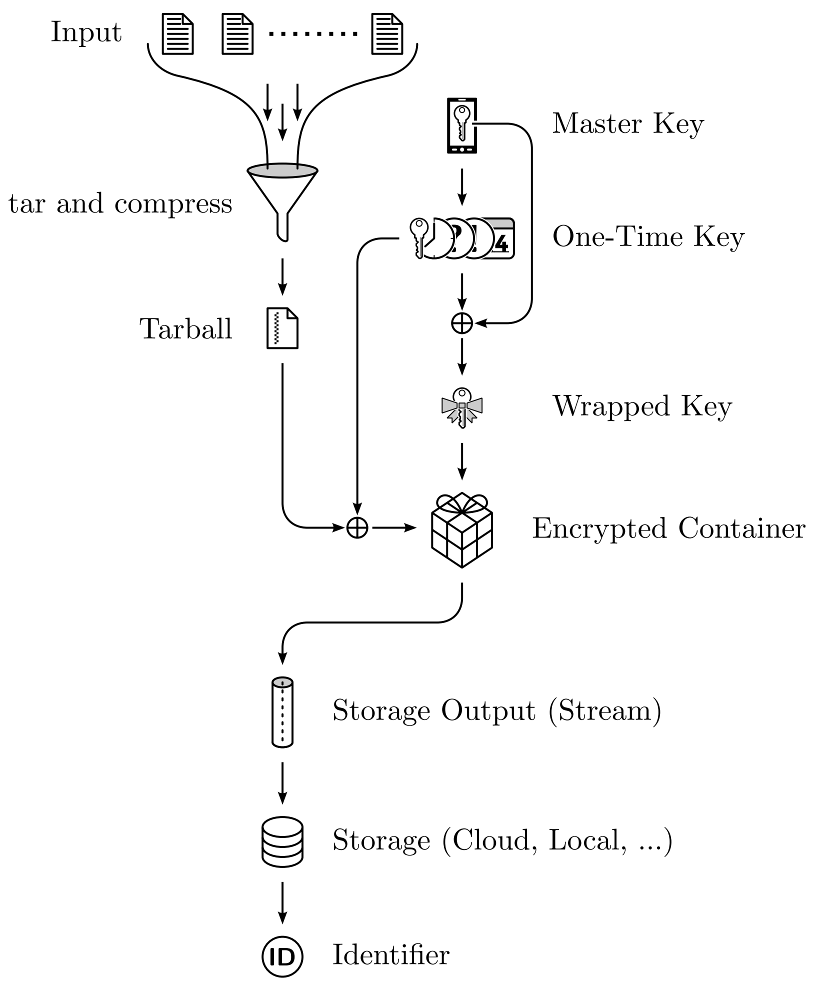


Figure 5.2: Encrypting data using Grið; the wrapped key includes all authentication information and can therefore be packaged alongside the encrypted data.

Having discussed the (essentially straightforward) encryption workflow, the decryption of received data involving multiple devices can be elaborated on.
Given an identifier, the recipient can fetch the corresponding encrypted container. The wrapped key is then extracted and sent to the original sender running the Grið instance having access to the long-term key. (Obviously, the recipient must also be given any information needed to be able to contact the sender.) Next, the sender unwraps the wrapped key, checks the key's expiry date and presents the recovered authentication challenge to the original recipient. Naturally, the recipient must successfully stand this challenge in order to gain access to the ephemeral key. Upon receiving the correct response corresponding to the recovered authentication challenge, the

sender transmits the ephemeral key to the recipient who can finally decrypt and decompress the payload and recover the original data. Figure 5.3 illustrates the decryption workflow (omitting the authentication process, since this is discussed separately).

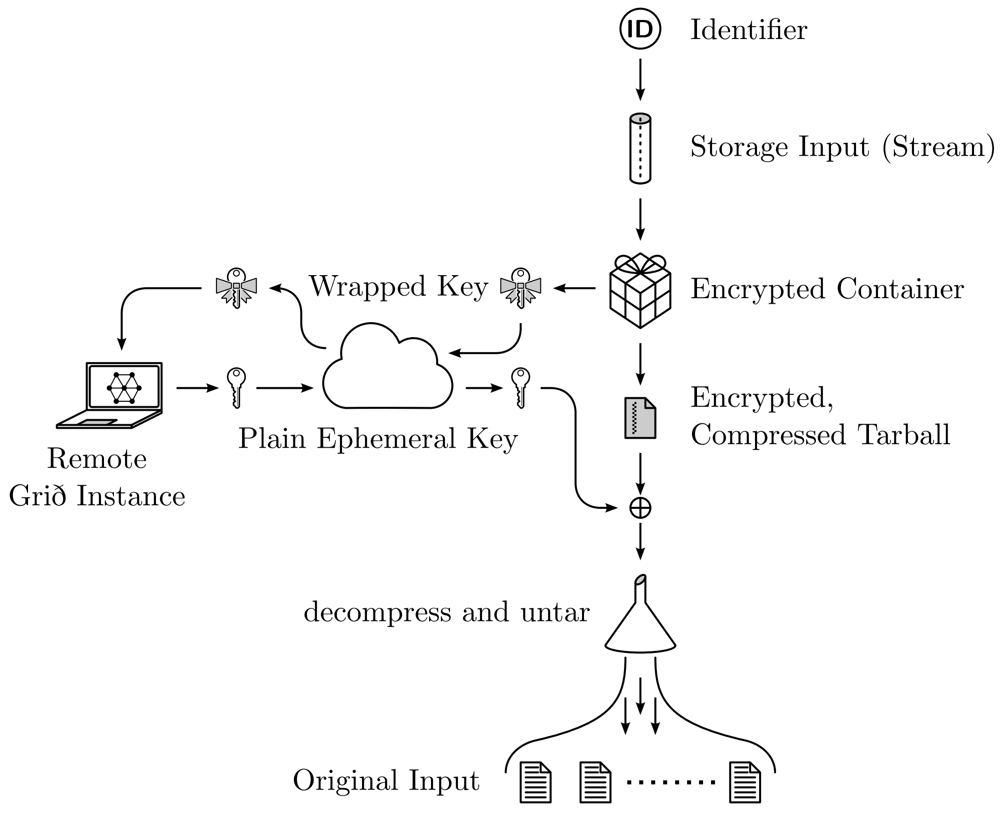


Figure 5.3: Decrypting data using Grið; the authentication procedure has been omitted for the sake of clarity.

Even without considering any potential authentication mechanisms, Grið already relies on multiple communication channels to implement secure information exchange. It is possible to use distinct channels for every step: Encrypted data can be stored in the cloud and the identifier pointing to it can be sent to the recipient using an instant messaging service. The key-exchange including the challenge-response protocol is carried out over another communication channel (in the case of the implemented prototype, Tor hidden services). Notifying the recipient about how to reach the sender can be accomplished using yet another communication channel. This ensures a high degree of flexibility and, again, empowers the encrypting entity, since it can freely choose communication channels it considers trustworthy for the given tasks. Depending on the employed authentication mechanism, even more distinct channels can be employed if desired (which adds further flexibility and introduces new possibilities).

Building upon CrySIL, Grið includes the prerequisites to implement arbitrary authentication mechanisms. In general, CrySIL already provides a foundation to support all of Grið's functionality due to its design as an interoperability layer. Especially the extensible, highly versatile authentication workflow described in Section 4.3 perfectly fits with Grið's goals.

The Grið prototype currently relies on a simple challenge-response protocol, enabling the encrypting party to enter arbitrary text for the challenge and a reference value for the expected response (as well as an expiry date). Due to the general nature of this approach, it can be utilised in various ways with the most obvious application simply relying on pre-shared secrets. However, more elaborate schemes are conceivable, with the challenge containing instructions about a task which needs to be fulfilled by the receiving party in order to obtain the correct response value. This enables the encrypting entity to define the security level it deems adequate depending on the data by specifying appropriate challenges. A natural next step would be to integrate other authentication mechanisms such as OAuth or concrete implementations of some common challenge-response schemes involving more than one factor. However, the implemented challenge-response authentication plug-in already serves to illustrate what can be done.
Having elaborated on the data exchange workflows, the data exchange format created for Grið can be discussed next.

5.1.2 Streaming-Capable Container Format

The container format used to store and exchange data, simply called *encrypted container*, was specifically created for Grið. This was done mainly to ensure streaming capabilities and interoperability. Due to the simple structure of the exchanged data (essentially consisting of a wrapped key and the actual payload), no further constraints were imposed on the container format. Therefore, the structure illustrated in Listing 5.1 was chosen.

Listing 5.1: Hexdump of an *encrypted container*

```
00000000 d0 6e f0 0d 00 00 3c 7f  30 82 3c 7b 06 09 2a 86  |.n....<.0.<{..*.| header, start of wrapped key
00000010 48 86 f7 0d 01 07 02 a0  82 3c 6c 30 82 3c 68 02  |H........<l0.<h.|
00000020 01 01 31 0f 30 0d 06 09  60 86 48 01 65 03 04 02  |..1.0...‘.H.e...|
00000030 01 05 00 30 82 34 1f 06  09 2a 86 48 86 f7 0d 01  |...0.4...*.H....|
00000040 07 01 a0 82 34 10 04 82  34 0c 7b 0a 20 20 22 65  |....4...4.{.  "e|
00000050 6e 63 6f 64 65 64 4b 65  79 22 20 3a 20 22 4d 49  |ncodedKey" : "MI|
00000060 49 59 46 51 59 4c 4b 6f  5a 49 68 76 63 4e 41 51  |IYFQYLKoZIhvcNAQ|
         -- -- -- -- -- -- -- --  -- -- -- -- -- -- -- --
00003c80 33 33 5b 9f 75 ed a7 30  80 06 09 2a 86 48 86 f7  |33[.u..0...*.H..| start of data at byte 0x3087
00003c90 0d 01 07 03 a0 80 30 80  02 01 00 31 82 01 9a 30  |......0....1...0|
         -- -- -- -- -- -- -- --  -- -- -- -- -- -- -- --
00a52990 83 33 59 43 a2 9a 24 04  10 60 a7 d7 9c 9b 46 2a  |.3YC..$..‘....F*|
00a529a0 75 d7 97 b5 f1 d4 d8 c2  69 00 00 00 00 00 00 00  |u.......i.......|
```

The header takes up the first eight bytes of a container. It contains the magic number `0xD06EF00D` followed by a little-endian representation of a 32 bit signed integer denoting the length (the number of bytes) of the wrapped key. The magic number is used to perform a very basic check when decoding a container in order to prevent Grið from trying to process arbitrary files. The actual wrapped key data starts at the ninth byte and is stored as CMS *enveloped-data* containing a JSON representation of the cryptographic key material and authentication information.
The length of the wrapped key is the only information essential to decoding a container: Given this information, the wrapped key can be read as a whole, while all remaining data is treated as streamable CMS enveloped-data. As public keys of the recipients are irrelevant due to the inverted trust model, the encrypting entity's public keys (which can be randomly generated, disposable ones) are being written to CMS containers. This approach was chosen, since CMS eases key management: Due to the public keys being part of a CMS container as defined by the CMS specification (described in Section 2.2.2), recipients can easily verify that a recovered ephemeral key actually belongs to a container's encrypted payload.
Furthermore, due to authenticated encryption being used, not knowing the total amount of encrypted data does not pose a problem. At the same time, this simple format enables both the encryption and decryption of data without the need to know the total amount of input data beforehand, thus resulting in a fully streaming-capable format. This is particularly useful since all input data is being compressed prior to encryption.

Considering that all other operations in the encryption and decryption pipeline are also streaming capable, the container format enables the creation and transfer of even large amounts of data with minimal time overhead. In essence, every step in the pipelines illustrated in Figures 5.2 and 5.3 operates on a chunk of data as soon as it becomes available, leading to an overall highly efficient process. This is especially noticeable when processing large payloads, since it is not necessary for any step of the pipeline to wait for the previous one to finish.

Following this overview of the employed concepts, a more detailed architecture description on how these have been implemented shall be provided.

5.2 Architecture

Grið's design is heavily influenced by the modular structure of the underlying CrySIL framework. Although it makes extensive use of the prototypical implementations of various modules shipped with CrySIL, most of these were customised to some extent. Furthermore, new modules have been implemented to provide the required additional functionality, while others have been split-up and some have been rewritten from scratch.

5.2.1 Overview

Apart from the components defined by the CrySIL framework a communication layer has been implemented, enabling the secure information exchange between Grið instances. A back-end for the communication layer based on Tor hidden services is another integral part of Grið, since it enables entities to contact each other while remaining anonymous. In general, a secure communication channel is an essential requirement for being able to exchange secret key material and for performing authentication procedures involving sensitive data.
The authentication modules provided by the CrySIL prototype have also been substantially re-engineered to incorporate the novel wrapped key structure. These changes also introduced slight modifications to CrySIL's API and the provided actors to enable the correct interpretation and processing of the additional authentication information. Figure 5.4 depicts a high-level view of the components Grið consists of as well as their relationships.

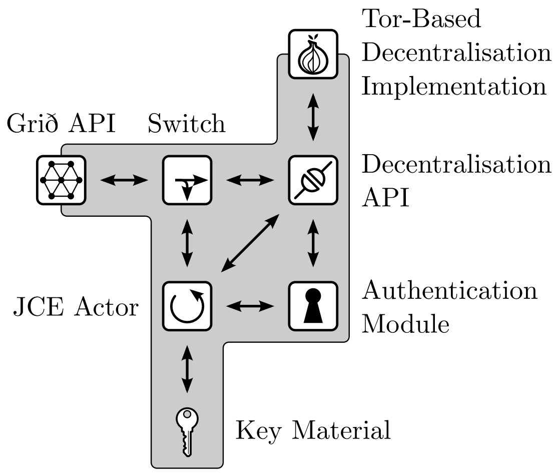


Figure 5.4: High-level view of Grið's architecture

In essence, Grið is one large *actor+* whose API (the *Grið API*) allows for executing only two commands: locally creating an encrypted container and decrypting such a container involving remote decryption of its wrapped key. Due to the fact that no remote parties are involved when creating a cryptogram, no trust issues can arise even in this distributed setup.
A simple *switch* decides whether a command is to be executed locally (which is the case for generating a wrapped key and for creating an encrypted container) or on a remote Grið instance (which is necessary to decrypt a wrapped key). The communication between instances is not directly carried out through Tor, but by means of an additional abstraction layer, the *decentralisation API*, which supports arbitrary back-ends. However, due to Tor supporting all required security features (by providing an authenticated communication channel, ensuring confidentiality, and guaranteeing integrity) and being the most popular and best-evaluated anonymity network (see Section 3.3.3.1), other back-ends are not maintained anymore.

The decentralisation API also receives incoming requests from instances wishing to decrypt a wrapped key. Apart from authentication-related commands, only wrapped key decrypt requests are being processed. Any other received requests are rejected which serves as a firewall to ensure that no remote parties are ever able to access local key material.
Naturally, an *authentication module* needs to be present to only grant authenticated parties the ability to decrypt a wrapped key. As can be seen in the architecture overview depicted in Figure 5.4, the *JCE actor*, which is the only component to ever access the local key material, is itself never directly accessible both from the local Grið API as well as the network layer. The decentralisation API includes a rudimentary firewall based on whitelisting of individual commands and the authentication module intercepts requests and enforces access control, while the Grið API hides the complexity of the internals.
The following section dealing with the implementation details highlights some of this complexity.

5.2.2 Implementation

The implementation of Grið follows CrySIL's modular structure. However, the modules defined in the system's architecture do not directly map to the implemented module structure. This is mainly caused by some components being tightly connected on a conceptual level, while others encompass multiple features, even though some of these are typically grouped into different modules from an architectural point-of-view. All modules used by Grið define interfaces to make it possible to exchange the actual implementation of any component if desired. Java was used to implement the prototype for an easy integration of the CrySIL prototype (which is also Java-based). Furthermore, Java's cross-platform capabilities and the possibility to easily extend Grið to Android also factored into this decision.
The following list provides an overview of all modules a Grið instance is composed of:

`lib-gridh` – definition and implementation of Grið's core functionality

`lib-decentral` – interface definition of the communication framework

`lib-gridh-tor` – implementation of the Grið core functionality utilising Tor as secure communication layer

`actor-decentral` – implementation of an *actor+* which redirects requests based on their type utilising the above modules

`lib-tor` – adapter module to use Tor with `lib-decentral`

`jtorproxy` – Java wrapper around the official Tor distribution, used with `lib-tor`

`lib-tor-silvertunnel` – integration of a pure Java implementation of Tor (used as fallback) `actor-jce` – CrySIL's *Java Cryptography Extension* (JCE) actor, adapted to handle wrapped keys including authentication information

`receiver-jce` – a customised variant of CrySIL's receiver for commands to be executed by a JCE actor

`gatekeeper` – CrySIL's gatekeeper module

`interceptor-authentication` – heavily customised version of CrySIL's interceptor

`auth-basic` – refactored and reduced version of CrySIL's authentication plug-ins with additional plug-ins being moved to a separate module

The following description of Grið's implementation is based on the above module structure, starting with the core functionality. Additionally, technical details about the user interface implementation are provided.

5.2.2.1 Grið Core Functioniality

Modules: `lib-gridh`

Grið's core module is naturally the most complex building block of the system, tying together all low-level components to provide an easy-to-use high-level API. Apart from an asynchronous, callback-based interface, the core mainly consists of IO-related functionality. It defines the *encrypted container* format as well as storage IO interfaces including Java streams providing read– and write-access to local files as well as *Dropfile.to*, an anonymous file hosting service [Dropfile.to, no date]. Additionally, streams to write to and read from encrypted containers are provided. Furthermore, ways to incorporate *Lempel–Ziv–Markov chain algorithm* (LZMA) compression into the encryption/decryption pipeline are present.
Obviously, the pipeline modelling the encryption and decryption workflows is also defined in the core module. Due to all components of this pipeline adhering to Java's stream API and the authentication workflow being performed asynchronously, this pipeline definition essentially consists of a connection of multiple streams. This entails that changes to the pipeline can easily be made by rearranging and/or exchanging (some of) the streams the pipeline consists of. Consequently, the principal functionality is implemented in the aforementioned streams. Therefore, these shall be discussed in more detail.

Starting from the bottom, storage IO streams including storage identifiers shall be elaborated on. Grið defines storage identifiers on two levels: once simply referencing the location some information (an encrypted container) is stored at and on another level as a more elaborate custom *iniform resource identifier* (URI) also including the information required to contact the creator of an encrypted container.
Arbitrary storage locations including their unique identifier types can be added to a registry, given corresponding implementations of the custom storage stream interfaces are provided. This

makes it possible to easily add support for additional storage locations. Although adhering to Java's stream API, these interfaces have been extended to also define ways to register progress listeners to provide users with feedback about the amount of data already processed at any point in time.

The next step in the pipeline deals with the streaming-capable encrypted container format. These streams actually do not include any encryption logic, but simply provide convenience methods for reading from and writing to encrypted containers. Apart from creating and processing container headers (see Section 5.1.2) the input stream enables the extraction of a container's wrapped key and advances the underlying stream to the first byte of the encrypted payload.
The encrypted payload is in turn processed by CMS streams, which contain the actual cryptography-related logic and are thus capable of using wrapped keys. These streams, too, provide callbacks for listeners to notify the user about the progress of cryptographic operations.
The final step of the pipeline implements the creation of solid, LZMA-compressed archives based on the *tar* format.

Grið exposes an asynchronous, request-based interface to the cryptographic pipelines. The pipelines themselves are parallelised as much as possible, which is directly supported due to every pipeline step being implemented through streams. In general, callbacks, listeners, and multithreading are employed wherever possible to offload any long-running tasks from the application's main thread.

Grið's other major components apart from the custom IO pipeline concern the incorporation of the CrySIL framework and the decentralisation API to implement the creation of wrapped keys and the already described remote authentication. These are, however, delegated to the sub-modules described in the following sections.

5.2.2.2 Decentralisation API, CrySIL Integration

Modules: `{lib,actor}-decentral`

In order to implement a distributed setup, a decentralisation API was defined, enabling remote processing of decryption requests. This API identifies a Grið instance as a `DecentralNode` implementing a `CommunicationBehavior`. Communication behaviours can be both synchronous and asynchronous, processing arbitrary types of messages. Currently, only communication behaviours based on Tor transmitting JSON representations of `SRequest` and `SResponse` are implemented.
Each node is uniquely defined by its identifier. In the case of Tor hidden services being used, identifiers correspond to Tor hidden service descriptors. The identifiers themselves are simply

strings, whose interpretation is up to the concrete implementation of a node and its corresponding communication behaviour. This way, the interface definitions remain as generic as possible.

The CrySIL functionality is incorporated into this concept through a specialised implementation of the `Actor`-interface. This actor relays requests either to the local JCE actor or to a specified remote instance by utilising a decentral node and its communication behaviour. Although violating the semantics, implementing a custom actor was chosen over extending the router provided by CrySIL, since it resulted in a cleaner, easier-to-maintain code-base. Furthermore, none of these details are exposed through the Grið API which hides this semantic inconsistency.

The switch depicted in Figure 5.4 is, however, not part of this actor implementation, but part of the Grið API on a higher level. The reason for this is extensibility: In case the CrySIL protocol is adapted, the actors can remain unchanged. In such cases, only the whitelist and the internals of the Grið API will be affected.

Having discussed the decentralisation API, its implementation based on Tor needs to be elaborated on, since it is one of the key components ensuring security properties of the implemented prototype.

5.2.2.3 Tor Integration

Modules: `lib-tor`, `lib-tor-silvertunnel`, `lib-gridh-tor`, `jtorproxy`

Grið implements inter-node communication based on Tor. Even at this level, an abstraction layer has been introduced, allowing for different ways to integrate Tor. One of the main motivations for this design choice was the fact that no established way of integrating Tor with Java applications is directly supported by the Tor developers. Currently, a Java-implementation of Tor, *SilverTunnel-NG* [Böse, 2015], as well as a wrapper around the native Tor implementation, called *JTorProxy*, is supported. While the former was readily available, the latter has been implemented in coordination with the *Bitsquare* project [Karrer, 2015] based on an abandoned Tor wrapper originally implemented for the *Thali* project [Goland and Poon, 2015]. JTorProxy provides callback-based interfaces to host hidden services and allows for integrating a Tor-based network layer through the standard Java socket API. In essence, it hides the process of installing and configuring Tor and setting up hidden services. Due to Tor itself being available as self-contained, pre-compiled binary distributions for a multitude of platforms, incorporating native code does not rise any (portability) issues. JTorProxy currently supports Android starting from version 4.1 on all architectures also supported by Tor, Apple OS X, versions of all Linux distributions supported by Tor (albeit limited to 32 bit and 64 bit Intel architectures), as well as all recent versions of Windows supported on Intel hardware.
By integrating the official Tor binaries provided by the Tor project, all security properties of these are also applicable to Grið's network layer. Relying on Tor also solves most network

security issues discussed in Section 3.2.1 and virtually all issues related to the P2P approach (see Section 3.2) the network layer is based on. In addition, security fixes to Tor and its protocol can easily be incorporated into Grið by simply updating to the latest versions of Tor.
JTorProxy has been made available on GitHub for anyone to use [Karrer and Prünster, 2015].

The final functional component of Grið, discussed in the following section, deals with authentication of remote entities wishing to decrypt wrapped keys.

5.2.2.4 Authentication Module

Modules: `auth-basic`, `gatekeeper`, `interceptor-authentication`

The CrySIL prototype already includes all components needed to implement even the most complex authentication schemes (see Section 4.3). However, the packaging and implementation of the provided authentication module were insufficient for Grið's use-cases. Most importantly, even without implementing any graphical authentication front-ends, it would still depend on *JavaFX*. During Grið's development it became clear that this made it impossible to provide authentication modules on platforms not supporting JavaFX, such as Android or even on desktop operating systems relying on OpenJDK prior to version 8. Furthermore, the development process itself was severely hindered because of this situation.
Therefore, the provided authentication module was completely rewritten, while still adhering to the API defined by CrySIL. Interfaces were defined for all involved components such as the *authentication dialogues* and the authentication selector. The user-facing part of the authentication-related workflow was modelled similarly to the *authentication process* described in Section 4.3. By introducing this separation of implementation and interface definition, it became possible to provide a multitude of different authentication dialogues for any authentication method, each suited to different environments and devices. Furthermore, any authentication information processed by the authentication module is not limited to any specific data-type anymore.
In addition to restructuring and modularising the authentication module, the *interceptor* (see Section 4.3) has been extended to support the newly introduced authentication information incorporated into the wrapped key structure.

Finally, the user interface developed for Grið shall be discussed from a technological point-of-view, highlighting the widget-toolkit-choice and UI-related aspects of the workflow from a user's point-of-view.

5.2.2.5 User Interface

Grið's user interfaces relies on *Apache Pivot* as its widget toolkit. While primarily aimed at creating *rich internet applications* (RIAs), Pivot also supports implementing traditional desktop applications. The main reasons for choosing Pivot were the possibility to design user interfaces descriptively using *BXML*, an XML dialect conceptually similar to JavaFX's *FXML* [Apache Software Foundation, 2014], but also its aesthetics. In addition, it is considered superior to other alternatives: Java's default toolkit, *Swing*, almost always leads to inconsistencies when striving for a native look-and-feel, but also provides fewer features. In addition, Swing is in the process of being deprecated in favour of JavaFX [Oracle Corporation, no date]. JavaFX itself, on the other hand, is poorly supported on non-Windows platforms prior to Java version 8. Furthermore, JavaFX employs native code which can also result in portability issues. Relying on Pivot and the availability of ready-to-use and freely-distributable versions of the Java runtime environment (in form of the *unofficial JDK builds* [Kashchenko, 2015]) makes it possible to ship self-contained versions of Grið, while also providing a feature-rich user interface which looks and feels the same on all platforms. Additionally, Pivot enforces concurrency to prevent long-running tasks from blocking the rendering pipeline, which fits well with Grið's asynchronous API.

In general, the implemented user interface is kept as simple as possible. Upon start, users have to enter a master key to unlock (or, if Grið is started for the first time, create) a key-store and are then presented with the main user interface.
Since the Grið API strives for hiding the underlying complexity of the cryptographic pipelines, the implemented prototype tries to reflect this goal in its user interface. As already mentioned, in case a cloud-based storage solution is used to exchange cryptographic containers, only their identifiers and information about the encrypting entity need to be exchanged. This can conveniently be achieved by means of the URI described in Section 5.2.2.1 which is presented to the encrypting entity after a successful encryption process.

The recipient seeking to decrypt data can simply enter such a URI into the corresponding input field of the graphical user interface. All underlying complexity is hidden from the user: After transparently contacting the encrypting entity's Grið instance, the user is presented with a form displaying the authentication challenge, prompting the user for the corresponding response. This does not look or feel any differently from well-known login forms, which means users only have to deal with workflows and user interface paradigms they are already familiar with.

Creating an encrypted container is slightly more involved in comparison, even though it is conceptually simpler. First, the users need to add files and choose a destination (currently, either Dropfile.to or storing the encrypted container locally). The input form used to specify a key expiry date and an authentication challenge is presented to the user only after having specified all files and the storage location.

A comprehensive user manual can be found in Appendix A. Therefore, no more user interface details shall be discussed here. Instead, a security evaluation follows, based on the process defined by *ISO/IEC 15408-1:2008* to assess the security properties of Grið.

6 Security Evaluation

This chapter encompasses the security evaluation of Grið based on *ISO/IEC 15408-1:2008 Information technology — Security techniques — Evaluation criteria for IT security — Part 1: Introduction and general model* [International Organization for Standardization, 2014]. This industry standard defining a methodology for evaluating a system's security properties is commonly known as *Common Criteria for Information Technology Security Evaluation* (*Common Criteria* (CC), for short).
The purpose of this evaluation is not to check for implementation flaws or cryptographic weaknesses, but to reach a verdict about the fitness of the procedures defined by Grið with respect to securely exchanging information. Most importantly, the authentication workflow is examined in different scenarios, only some of which are concerned with anonymity. A basic scenario with few constraints is defined first in order to establish a baseline. Next, more complex ones are discussed including one which incorporates third parties in the hopes of averting MITM attacks. By defining these multiple scenarios and evaluating Grið's ability to protect its assets, its versatility can be demonstrated. First, however, the evaluation methodology needs to be defined.

6.1 Methodology

Common Criteria provides a standardised approach to evaluating the security properties of a system with respect to the assets this system is intended to protect [International Organization for Standardization, 2014, p. 23]. However, the formal nature of the CC methodology makes it not directly applicable to the security evaluation of a prototype such as Grið – considering that a prototype may not be deployed in the configuration it has been evaluated, each time adjustments are made, the whole evaluation process would have to be repeated. Another reason for not fully conforming to CC is the simple fact that a thorough description of all relevant components has already been provided in the previous chapters. CC, on the other hand, demands system descriptions on different abstraction levels [International Organization for Standardization, 2014, p. 40] as part of a security evaluation. Other (mostly formal) requirements cannot be fulfilled either due to the implementation of Grið being prototypical, as opposed to a concrete product or discrete versions of a software system. Organisational policies which are usually part of such

an evaluation [International Organization for Standardization, 2014, p. 43] are also non-existent for the same reason.
To better be able to judge the security properties of the novel approach introduced by Grið, different scenarios leading to different prerequisites, assumptions, threats, and even assets shall be evaluated. This is another deviation from the standardised CC evaluation methodology. However, since the overall process defined by CC is applied, the standardised evaluation structure shall be elaborated on, followed by a description of the derived methodology used for the security evaluation of Grið.

6.1.1 Common Criteria Evaluation Process

The standardised *Common Criteria for Information Technology Security Evaluation* defines a framework and processes for evaluating the security properties of IT systems. The result of such an evaluation explicitly states whether the evaluated system fulfils all *Security Functional Requirements* (SFRs) [International Organization for Standardization, 2014, p. 35]. However, the general nature of such an evaluation also incorporates a system's environment including a set of assumptions the evaluation is based on. These assumptions are simply stated and the environment is not part of the evaluation [International Organization for Standardization, 2014, p. 25]. Obviously, the verdict of the evaluation may not be correct anymore in case assumptions do not hold and, as a consequence, the environment differs. Keeping this in mind, evaluation results can be interpreted accordingly.

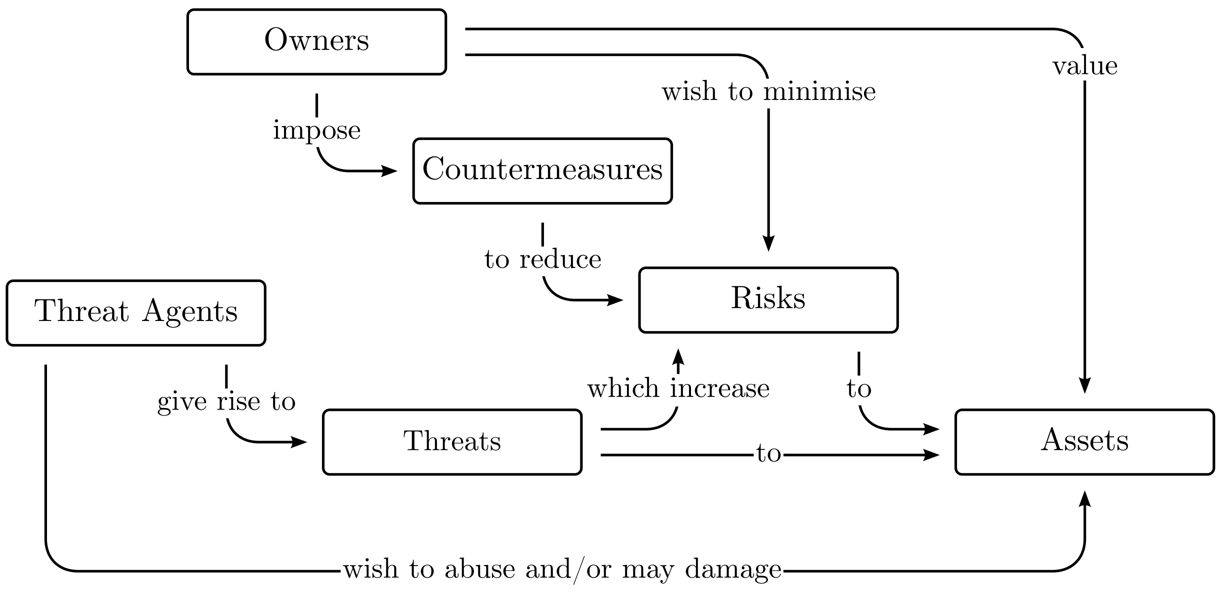


Figure 6.1: Security concepts and relationships based on ISO/IEC 15408-1:2008 [International Organization for Standardization, 2014, p. 24]

As already mentioned, any security-related system is designed to protect assets by countering threats (as depicted in Figure 6.1 based on ISO/IEC 15408-1:2008 [International Organization for Standardization, 2014, p. 24]). The fitness of an evaluated system (referred to as the *Target of Evaluation* (TOE)) with respect to protecting these assets is evaluated either based on so-called *Protection Profiles* (PPs) or *Security Targets* (STs). In case of Grið's evaluation, STs are more relevant, since a concrete system and not a specification or a technical standard is being evaluated [International Organization for Standardization, 2014, p. 56]. Figure 6.2 based on ISO/IEC 15408-1:2008 [International Organization for Standardization, 2014, p. 39] outlines the contents of an ST-based evaluation conforming to the CC evaluation methodology. These are:

> a) *an ST introduction* containing three narrative descriptions of the TOE on different levels of abstraction;
>
> b) *a conformance claim*, showing whether the ST claims conformance to any PPs and/or packages, and if so, to which PPs and/or packages;
>
> c) *a security problem definition*, showing threats, OSPs [Organisational Security Policies, author's note] and assumptions;
>
> d) *security objectives*, showing how the solution to the security problem is divided between security objectives for the TOE and security objectives for the operational environment of the TOE;
>
> e) *extended components definition* (optional), where new components [. . .] may be defined. These new components are needed to define extended functional and extended assurance requirements;
>
> f) *security requirements*, where a translation of the security objectives for the TOE into a standardised language is provided. This standardised language is in the form of SFRs. Additionally this subclause defines the SARs [Security Assurance Requirements, author's note];
>
> g) *a TOE summary specification*, showing how the SFRs are implemented in the TOE. [International Organization for Standardization, 2014, p. 38]

However, due to the formalities of the approach (some of which cannot be fulfilled) and its intentions not reflecting the goal of this security evaluation, a customised methodology was derived. While a reduced version of an ST-based evaluation exists [International Organization for Standardization, 2014, pp. 51–52], it is not well-suited for this (also somewhat reduced) security evaluation: Not only would it suffer from the same problems as a regular ST-based evaluation, but the result would also be incomplete compared to a customised methodology. The individual steps of the conducted evaluation (consisting of a reduced variant of an ST-based evaluation) and its components are therefore elaborated on in the following section.

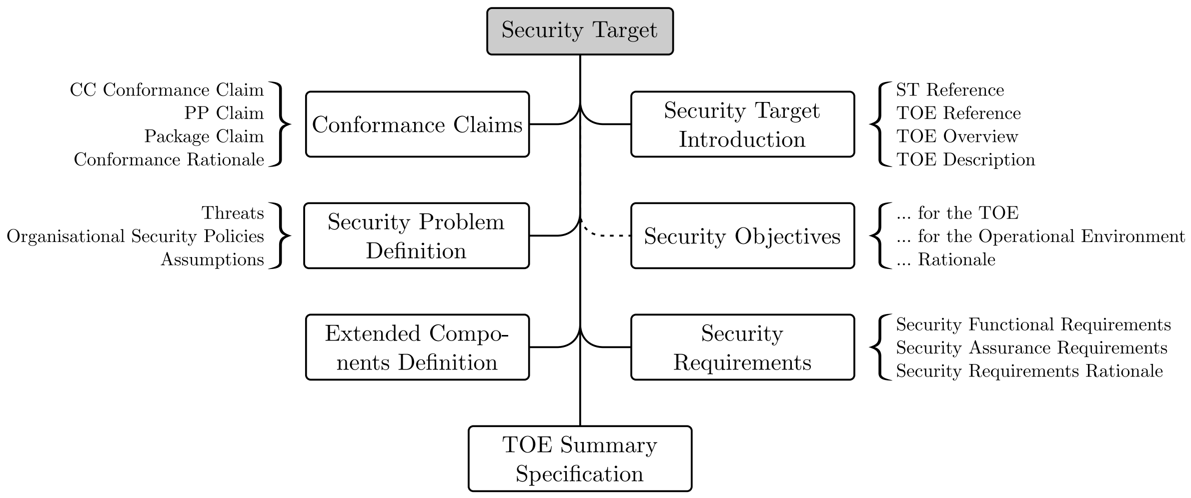

Security Target
CC Conformance Claim
PP Claim
Package Claim
Conformance Rationale
Conformance Claims
Security Target Introduction
ST Reference
TOE Reference
TOE Overview
TOE Description
Threats
Organisational Security Policies
Assumptions
Security Problem Definition
Security Objectives
... for the TOE
... for the Operational Environment
... Rationale
Extended Components Definition
Security Requirements
Security Functional Requirements
Security Assurance Requirements
Security Requirements Rationale
TOE Summary Specification

Figure 6.2: ST contents based on ISO/IEC 15408-1:2008 [International Organization for Standardization, 2014, p. 39]

6.1.2 Derived Evaluation Methodology

This section presents the customised methodology used to evaluate the security properties of Grið. Wherever applicable the terminology defined by the International Organization for Standardization [International Organization for Standardization, 2014, pp. 2–9] is used. Therefore, these definitions are not reproduced here, but the used acronyms are written out on their first occurrence as it has been done throughout this work.

The derived evaluation process is structured as follows: Firstly, the TOE is defined in terms of which components of Grið are subject to the evaluation, since Grið utilises existing solutions where possible. Conformance claims are omitted, since these are not relevant to reach a verdict. Secondly, the general security problem definition is provided.
Having defined the overarching security problem, different scenarios are evaluated. These include definitions of the TOE's operational environment, all assumptions, assets, threat agents, and all threats to the previously defined assets.
Next, the security objectives are elaborated on. Most importantly, this evaluation step includes a mapping of threats to assets and a mapping of countermeasures to threats. The strict formalisation of the security objectives in terms of security requirements as specified in ISO/IEC 15408-1:2008 [International Organization for Standardization, 2014, pp. 47–49] is omitted, since it is not required to reach a verdict. Each scenario concludes with a discussion including a verdict regarding the fitness of the TOE with respect to countering all threats and thus protecting all assets in the scenario's operational environment.
Finally, a concluding comparison of all scenarios is provided as well as a discussion about the TOE's overall fitness with respect to protecting all assets. Being based on the CC evaluation process, this derived methodology is also based on the concepts illustrated in Figure 6.1.

6.2 Target of Evaluation

Due to the complexity of Grið and the incorporation of pre-existing technologies, the system will not be evaluated as a whole. Some components (such as the network layer, for example) are assumed to be secure (see Section 6.5.2).
The TOE consists of the following:

The core functionality implemented by `lib-gridh` as discussed in Section 5.2.2.1

The container format containing both the wrapped key and the encrypted payload as explained in Section 5.1.2

The authentication workflow based on the wrapped key data structure as illustrated in Section 5.1.1

Not subject to an evaluation are the CrySIL framework, the network layer based on Tor, the pre-existing authentication framework provided by CrySIL, and the *JCE Actor* performing all cryptographic operations, as well as the cryptographic algorithms themselves. These components are simply assumed to operate securely and correctly as expected. In addition, the communication channel used to transmit the storage location and the identifier of the encrypting entity's Grið instance as well as the storage provider used for the actual exchange of encrypted containers will not be evaluated. However, requirements for these infrastructures will be defined for each evaluated scenario.

6.3 Security Problem Definition

Grið seeks to enable secure information exchange based on an inverted trust model while remaining anonymous to the network. The following use-case serves to recapitulate this process.

Alice wishes to deliver some secret information to Bob. She therefore informs Bob about her Grið instance's identifier. This can be done prior to the actual information exchange or as part of it. The implemented prototype employs Tor hidden services to enable communication parties to connect to each other's Grið instances. Hence, an instance's identifier corresponds to the identifier of the Tor hidden service operated by this instance. Since Tor is used, it is reasonable to assume that eavesdropping is not possible and even active attacks altering the information transmitted through Tor cannot be mounted. In general, all implemented functionality, including the cryptographic operations, are assumed to work as intended. Resilience against local attackers having access to the users' devices are also neglected, since the novel approach introduced by Grið is to be evaluated and not a device's or a user's capability to avert traditional attacks, none of which are specific to the TOE and its environment.
As soon as Alice is in possession of the information she wants to share with Bob, she creates

an *encrypted container* (including an authentication challenge, as explained in Section 5.1.1). Alice uploads this container to some storage medium Bob also has access to (such as a cloud storage) and tells Bob how to access this container. Alternatively, Alice could directly transmit the container to Bob. In any case, Bob retrieves the encrypted container and initiates the challenge-response process to authenticate himself. Upon fulfilling the authentication challenge, Alice transmits the container's ephemeral key to Bob, enabling him to decrypt the payload and recover the secret information Alice wanted to share with him.
Obviously, the sender (Alice) knows of a way to contact the recipient (Bob). However, Alice and Bob do not necessarily know each other; Alice may publicly (possibly anonymously) announce that she is willing to share some information with anyone who holds some specified (secret) knowledge (such as knowing a password). Whichever party becomes aware of this announcement and is in possession of the aforementioned knowledge can contact Alice and authenticate itself to her by presenting the secret knowledge and thus receive the information from Alice. This concludes the security problem definition.
Grið's fitness to protect all assets is evaluated based on the following scenarios which specify the exact circumstances of the just-explained information exchange.

6.4 Baseline Scenario

Alice uses an insecure channel to transmit her Grið instance's identifier to Bob. Alice and Bob do not know each other and no additional parties are deliberately involved. This scenario is used to establish a baseline to compare other, more elaborate scenarios to. Alice's and Bob's anonymity are irrelevant. However, since Alice and Bob do not know each other, neither of them can easily verify the other's identity.

6.4.1 Operational Environment

The general security problem definition and the current, basic scenario description already illustrate an operational environment consisting of multiple interconnected parts: Obviously, the devices running a Grið instance are part of this environment. Secondly, the communication channel used to convey the identifier of the encrypting entity's Grið instance also needs to be considered. In case the encrypted container is exchanged using a different communication channel, it also becomes part of the operational environment. This distinction is, however, irrelevant for the evaluation of the current scenario as it has no impact on the result. The Tor network used to carry out the authentication protocol and the subsequent exchange of the ephemeral key (as explained in Section 5.2.2.3) is the final component of Grið's operational environment.
Naturally, this environment itself needs to operate correctly and securely in order to enable a

secure and correct operation of the TOE. Therefore, the assumptions stated in the following section must hold.

6.4.2 Assumptions

These following assumptions must be upheld in order to guarantee a correct operation of the TOE:

AS1 Every device running a Grið instance is free from malicious soft– and hardware and works as designed and intended by the user and is thus trustworthy.

AS2 Only the intended recipients are able to stand the authentication challenge defined by the sender.

AS3 Users' long-term keys are not disclosed.

The above assumptions are the user's responsibility and can therefore be influenced and should be confirmed to hold by the user every time Grið is used.

6.4.3 Assets

From Grið's description provided in Chapter 5 and this scenario's definition, the following assets can be derived:

A1 The *data* transmitted by means of an encrypted container

A2 The *one-time key* including the ephemeral key and the authentication information encrypted under the sender's long-term key in form of a wrapped key

A3 The sender's *long-term key* used to wrap the one-time key, stored on the sender's device

A4 The corresponding *response* to a recovered *authentication challenge*

Obviously, the TOE is designed to protect these assets from all threats. In order to evaluate to which extent the system succeeds in doing so, all possible threats and threat agents need to be considered.

6.4.4 Threat Agents and Threats

To assess the impact of possible threats to the previously defined assets, threat agents need to be identified first. The actual set of threats to these assets can be derived from the agents. The identified threat agents for the baseline scenario are:

TA1: Observer – Can observe the announcement and retrieval of an encrypted container as well as the announcement of an instance's identifier

TA2: Interceptor – Has full access to the channels used to transfer an encrypted container and an instance's identifier

TA3: Impersonator – Is fully decoupled from the actual workflow, but may want to advertise himself as someone else to Bob during the initial contact

These agents pose the following threats:

T1: Denial-of-service – An attacker could replace the encrypted container created by Alice with some other container or simply keep Bob from receiving the original container. If Bob does not receive the original container created by Alice, he cannot access the information intended for him.
Affected assets: A1, A2, A4
Involved threat agents: TA2

T2: Man-in-the-middle – An attacker could intercept and replace the transmission of Alice's Grið identifier, mount a MITM attack, and subsequently authenticate himself to Alice as Bob.
Affected assets: A1, A2, A4
Involved threat agents: TA2

T3: Decryption – An Attacker could try to decrypt an encrypted container, possibly by guessing/brute-forcing the authentication challenge and/or the ephemeral key.
Affected assets: A1, A2, A4
Involved threat agents: TA1 or TA2

T4: Spoofing – An attacker could try to modify the encrypted payload by directly operating on the ciphertext part of an encrypted container which holds the payload. The attacker could thus alter the plain text which is recovered upon decrypting a container.
Affected assets: A1
Involved threat agents: TA2

T5: Impersonation – An attacker may contact Bob under false pretences and convince him of a false identity. He could deliberately supply Bob with disinformation. The actual impact of such an attack varies greatly.

Affected assets: A1
Involved threat agents: TA3

Threats which are not applicable due to assumptions have been omitted for obvious reasons. However, the most prominent non-threat shall still be stated, as it helps establishing the baseline for all scenarios. This non-threat can be specified as follows:

N1: Long-term key extraction – An attacker gaining access to a device running a Grið instance could extract the long-term key and thus decrypt any encrypted container whose one-time key has been encrypted using the long-term keys stored on the device during the time-frame the attacker has access to it.
Affected assets: A1, A2, A3, A4
Not applicable due to assumptions: AS1, AS4

Table 6.1 graphically illustrates which assets are threatened by which threats (and non-threats).

Threats	Assets			
	A1	A2	A3	A4
T1	⊗	⊗	○	⊗
T2	⊗	⊗	○	⊗
T3	⊗	⊗	○	⊗
T4	⊗	○	○	○
T5	⊗	○	○	○
N1	⊗	⊗	⊗	⊗

Table 6.1: Baseline scenario mapping of threats to assets

The following security objectives illustrate how these threats can (and cannot) be countered with respect to this scenario's assumptions.

6.4.5 Security Objectives

Any system aimed at secure information exchange must be able to protect this information from being exposed to third parties. The users' long-term keys must only be known to them and cryptographic primitives must be employed which enable the system to protect all assets. This leads to the following security objectives:

O1 Users must ensure that the devices used running a Grið instance are free from malicious soft– and hardware.

O2 Alice defines an authentication challenge which only the intended recipient, Bob, can stand.

O3 Grið stores a user's long-term key locally, at a location only the user has access to.

O4 Grið relies on well-established ciphers and appropriate key-lengths.

O5 Grið employs authenticated encryption.

As depicted in Table 6.2, the baseline scenario leaves much to be desired.

Objectives	Threats						Assumptions		
	T1	T2	T3	T4	T5	N1	AS1	AS2	AS3
O1	○	○	✓	✓	○	✓	✓	○	○
O2	○	○	✓	○	○	○	○	✓	○
O3	○	○	✓	○	○	○	○	○	✓
O4	○	○	✓	○	○	○	○	○	○
O5	○	○	○	✓	○	○	○	○	○

Table 6.2: Baseline scenario mapping of objectives to threats and assumptions

Upon precise examination of this scenario, it becomes apparent, that the information exchange is vulnerable to two distinct kinds of MITM attacks whose impacts differ significantly:

Disinformation – The DoS and MITM attacks can be used as a vehicle to feed false information to Bob. In this case no sensitive information is disclosed to the attacker. An attacker can simply create an additional container containing disinformation and transmit it to Bob. Since Bob cannot authenticate the information he received, he can be lead to falsely believe in the authenticity of the counterfeit information he received.

Exposure – The MITM attack itself can also be utilised in the classical sense to eavesdrop on the authentication workflow and thus recover secret information: An attacker can act as a transparent proxy by intercepting the transmission of Alice's Grið instance's identifier and feeding his instance's identifier to Bob. Bob, not being able to authenticate the identifier he received, will then connect to the attacker's instance who can simply forward all messages exchanged as part of the authentication workflow and thus gain access to the one-time key used to encrypt a container's payload.

Considering the impact of these attacks, impersonation can effectively be considered a generalisation of both the DoS attack and the MITM attack with respect to supplying Bob with disinformation. The next scenario presents an improvement over this situation.

6.5 Scenario 1: Authenticated Channel

An Alice unknown to Bob – a whistle-blower who wants to remain anonymous, for example – uses a *sufficiently authenticated channel* to inform Bob about her Grið instance's identifier. In case Alice is in fact a whistle-blower wishing to disclose secret information it is reasonable to assume that she will carefully select a journalist to share her information with. Therefore, it can be assumed she knows Bob to some extent and knows about a secure way to contact him.

Sufficiently authenticated in this context means that Alice trusts the channel she selected to contact Bob enough to be reasonably certain that the transmitted identifier is not intercepted and replaced by an adversary. Obviously, this is highly dependent on Alice's judgement. Examples of a sufficiently authenticated channel could be private messages on social networks. To focus on the essentials, Alice's judgement about the trustworthiness of the selected channel is considered to be correct. Therefore, the channel selected by Alice is considered to be equivalent to an authenticated channel in the traditional sense. There are no requirements on the channel used to publish the encrypted container. Hence, this is defined to be done through a different channel. It is assumed that Alice can use both channels without the need to disclose her identity and location.

The fact that Alice would want to use two distinct channels, only one of which is authenticated, may seem nonsensical. After all, she could just use the one authenticated channel she trusts to publish both the encrypted container as well as her Grið instance's identifier. It is, however, easily conceivable that the one authenticated channel used to convey an identifier is not suited for transmitting large amounts of (binary) data. Therefore, this scenario assumes two distinct channels with different requirements.

6.5.1 Operational Environment

Compared to the baseline scenario, this scenario's environment is subject to more constraints. It must allow for Alice's identity to remain secret and enable her to publish containers and transmit identifiers while remaining anonymous. Yet, the operational environment is still composed of largely the same components (which must obviously still operate as intended). An explicit distinction between the channel used to convey an instance's identifier and the channel used to share an encrypted container is, however, made. Therefore, the assumptions stated in the following section must hold.

6.5.2 Assumptions

Compared to the baseline scenario two additional assumptions must hold in order to protect Alice's identity from being exposed. For the sake of completeness, the full set of assumptions is restated:

AS1 Every device running a Grið instance is free from malicious soft– and hardware and works as designed and intended by the user and is thus trustworthy.

AS2 The transmission of a Grið instance's identifier is carried-out over an authenticated channel.

AS3 Only the intended recipients are able to stand the authentication challenge defined by the sender.

AS4 Users' long-term keys are not disclosed.

AS5 Encrypted containers and identifiers can be published without disclosing the publisher's identity and location.

Again, the user is responsible for upholding the above assumptions.

6.5.3 Assets

In the whistle-blower scenario, Alice's anonymity is an important asset. Therefore, this scenario's assets are:

A1 The *data* transmitted by means of an encrypted container

A2 The *one-time key* including the ephemeral key and the authentication information encrypted under the sender's long-term key in form of a wrapped key

A3 The sender's *long-term key* used to wrap the one-time key, stored on the sender's device

A4 The corresponding *response* to a recovered authentication challenge

A5 *Alice's anonymity* from the used channels' points-of-view

Next, the set of threat agents and the threats to the above assets need to be identified.

6.5.4 Threat Agents and Threats

The threat agents and the threats applicable to the current scenario differ significantly from the previous baseline scenario; the following threat agents can be identified:

TA1: Observer – Can observe the announcement and retrieval of an encrypted container as well as the announcement of an instance's identifier

TA2: Interceptor – Has full access to the storage medium/transmission used to transfer an encrypted container

TA3: Impersonator – Is fully decoupled from the actual workflow, but may want to advertise himself as someone else to Bob during the initial contact

The threat agents pose the following threats:

T1: Denial-of-service – An attacker could replace the encrypted container created by Alice with some other container or simply keep Bob from receiving the original container. If Bob does not receive the original container created by Alice, he cannot access the information intended for him.
Affected assets: A1, A2, A4
Involved threat agents: TA2

T2: Deanonymisation – An attacker observing the announcement of an encrypted container and/or identifier could try to find out who published it.
Affected assets: A5
Involved threat agents: TA1

T3: Decryption – An Attacker could try to decrypt an encrypted container, possibly by guessing/brute-forcing the authentication challenge and/or the ephemeral key.
Affected assets: A1, A2, A4
Involved threat agents: TA1 or TA2

T4: Spoofing – An attacker could try to modify the encrypted payload by directly operating on the ciphertext part of an encrypted container which holds the payload. The attacker could thus alter the plain text which is recovered upon decrypting a container.
Affected assets: A1
Involved threat agents: TA2

T5: Impersonation – An attacker may contact Bob under false pretences and convince him of a false identity. He could deliberately supply Bob with disinformation. The actual impact of such an attack varies greatly.
Affected assets: A1
Involved threat agents: TA3

Threats not applicable due to the current scenario's assumptions have again been omitted. However, some non-threats still need to be considered to be able to compare different scenarios, especially threats transformed into non-threats due to this scenario's properties. The most prominent non-threats are:

N1: Long-term key extraction – An attacker gaining access to a device running a Grið instance could extract the long-term key and thus decrypt any encrypted container whose one-time key has been encrypted using the long-term keys stored on the device during the time-frame the attacker has access to it.
Affected assets: A1, A2, A3, A4
Not applicable due to assumptions: AS1, AS4

N2: Man-in-the-middle – An attacker could intercept and replace the transmission of Alice's Grið identifier, mount a MITM attack, and subsequently authenticate himself to Alice as Bob.
Affected assets: A1, A2, A4
Not applicable due to assumptions: AS2

These threats and non-threats highlight the impact of a modified set of assumptions and assets on a scenario's vulnerabilities. Table 6.3 graphically illustrates the mapping of threats to assets.

Threats	Assets				
	A1	A2	A3	A4	A5
T1	⊗	⊗	○	⊗	○
T2	○	○	○	○	⊗
T3	⊗	⊗	○	⊗	○
T4	⊗	○	○	○	○
T5	⊗	○	○	○	○
N1	⊗	⊗	⊗	⊗	○
N2	⊗	⊗	○	⊗	○

Table 6.3: Scenario 1 mapping of threats to assets

The next section explains how the assets defined for the current scenario can be protected given the specified security objectives are achieved.

6.5.5 Security Objectives

In addition to the previous scenario's objectives, Alice must also remain anonymous. This leads to an extended set of security objectives:

O1 Users must ensure that the devices used running a Grið instance are free from malicious soft– and hardware.

O2 Alice must ensure (by all means at her disposal) that the transmission of her instance's identifier to Bob is carried out over an authenticated channel.

O3 Alice defines an authentication challenge which only the intended recipient, Bob, can stand.

O4 Grið stores a user's long-term key locally, at a location only the user has access to.

O5 Alice contacts Bob and publishes encrypted containers in a way which allows for remaining anonymous to the network.

O6 Grið relies on well-established ciphers and appropriate key-lengths.

O7 Grið employs authenticated encryption.

As can be seen in Table 6.4, which maps this scenario's security objectives to threats and assets, not all threats can be countered. DoS attacks can still be mounted. Realistically, however, this is a non-issue; in case the communication channel employed to transmit the encrypted container from Alice to Bob is unavailable, no deliberate attack is required to cause a denial of service. What differentiates Grið from simpler systems (and this scenario from the baseline scenario) is the fact that even an active attacker, replacing or manipulating a container, can only mount a DoS attack. Container manipulation will still not cause Bob to receive modified information. This guarantee obviously results from relying on an authenticated channel to transmit an instance's identifier as shown in Table 6.4. Establishing such a channel can be difficult in reality which is why this scenario may not easily be applicable to real-world situations.

Objectives	Threats							Assumptions				
	T1	T2	T3	T4	T5	N1	N2	AS1	AS2	AS3	AS4	AS5
O1	○	○	✓	○	○	✓	○	✓	○	✓	✓	○
O2	○	○	○	○	○	○	✓	○	✓	○	○	○
O3	○	○	✓	○	○	○	○	○	○	✓	○	○
O4	○	○	✓	○	○	✓	○	○	○	○	✓	○
O5	○	✓	○	○	○	○	○	○	○	○	○	✓
O6	○	○	✓	○	○	○	○	○	○	○	○	○
O7	○	○	○	✓	○	○	○	○	○	○	○	○

Table 6.4: Scenario 1 mapping of objectives to threats and assumptions

In addition, it is important to keep in mind that the previous conclusions only hold in case Bob was actually contacted by Alice in the first place. In a whistle-blower scenario, Alice will want to remain anonymous under all circumstances. Therefore, Bob can not easily verify Alice's identity claim; the data Alice disclosed to him must essentially provide credible evidence about its authenticity. This problem, however, is out of scope and cannot be solved purely by technological means. The disinformation attack described in the baseline scenario therefore still applies to some extent. Bob can, however, be sure that subsequent information received through the same (authenticated) channel used to establish the initial contact originates from the same entity. Obviously, this requires a persistent authenticated channel which might be hard to achieve in practice, while still upholding all other constraints stated in the form of this scenario's assumptions.

6.6 Scenario 2: Bob Knowing Alice's Public Key

Bob may know Alice, but definitely knows her public key even if he does not really know who Alice is. In a whistle-blower scenario, where Alice wants to convey secret information to Bob, Alice will at least want to remain anonymous/pseudonymous to the network. Given the fact that Bob knows Alice' public key, she can sign the Grið instance's identifier she transmits to Bob[13]. There are no security requirements regarding the communication channel Alice used to contact Bob. It is assumed, that she uses a distinct channel to transmit encrypted containers to Bob. There are also no constraints on this channel.

6.6.1 Operational Environment

Obviously, Alice cannot remain truly anonymous, but only pseudonymous since Bob knows Alice's public key and she needs to sign her identifier prior to transmission. However, Alice's anonymity is still considered an asset in the following sense: Her true identity shall not be disclosed, but only the pseudonym associated with the public key known to Bob. Consequently, this scenario's environment is subject to mostly the same constraints as the previous scenario with respect to Alice's anonymity. Therefore, the environment must allow for Alice to publish encrypted containers and contact Bob while remaining pseudonymous. Again, an explicit distinction between the channel used to publish containers and the one used to transmit an identifier to Bob is made. The assumptions stated in the following section are expected to hold during the evaluation of this scenario to establish the just defined operational environment.

[13]She can obviously also sign the encrypted container she wants Bob to have. This is, however, largely irrelevant.

6.6.2 Assumptions

Compared to the previous scenarios, additional assumptions must hold in order to protect Alice's identity from being exposed. Therefore, all of the following assumptions must hold:

AS1 Every device running a Grið instance is free from malicious soft– and hardware and works as designed and intended by the user and is thus trustworthy.

AS2 Bob knows Alice's public key.

AS3 Only the intended recipients are able to stand the authentication challenge defined by the sender.

AS4 Users' long-term keys are not disclosed.

AS5 Encrypted containers and identifiers can be published without disclosing the publisher's identity and location.

AS6 Identifiers published by the sender, Alice, are signed with her public key.

AS7 Alice's private key is not disclosed.

In this scenario, users are again responsible for establishing an environment in which all of these assumptions hold.

6.6.3 Assets

Alice's anonymity/pseudonymity is an important asset in this setting. In the whistle-blower scenario, Alice will want to keep her true identity secret at all costs. This scenario's assets are therefore largely the same as previously:

A1 The *data* transmitted by means of an encrypted container

A2 The *one-time key* including the ephemeral key and the authentication information encrypted under the sender's long-term key in form of a wrapped key

A3 The sender's *long-term key* used to wrap the one-time key, stored on the sender's device

A4 The corresponding *response* to a recovered authentication challenge

A5 *Alice's anonymity/pseudonymity* from the used channels' points-of-view

A6 *Alice's private key*

As before, threat agents and threats need to be identified.

6.6.4 Threat Agents and Threats

Given the fact that this scenario can be considered an alteration of the previous one, a subset of the previous threat agents is relevant:

TA1: Observer – Can observe the announcement and retrieval of an encrypted container as well as the announcement of an instance's identifier

TA2: Interceptor – Has full access to the channels used to transfer an encrypted container and an instance's identifier

Since not all threat agents are present compared to the previous scenario, the resulting set of threats also differs:

T1: Denial-of-service – An attacker could replace the encrypted container created by Alice with some other container or simply keep Bob from receiving the original container. If Bob does not receive the original container created by Alice, he cannot access the information intended for him.
Affected assets: A1, A2, A4
Involved threat agents: TA2

T2: Deanonymisation – An attacker observing the announcement of an encrypted container and/or identifier could try to find out who published it.
Affected assets: A5
Involved threat agents: TA1

T3: Decryption – An Attacker could try to decrypt an encrypted container, possibly by guessing/brute-forcing the authentication challenge and/or the ephemeral key.
Affected assets: A1, A2, A4
Involved threat agents: TA1 or TA2

T4: Spoofing – An attacker could try to modify the encrypted payload by directly operating on the ciphertext part of an encrypted container which holds the payload. The attacker could thus alter the plain text which is recovered upon decrypting a container.
Affected assets: A1
Involved threat agents: TA2

Consequently, the non-threats also differ from the previous scenario's ones:

N1: Impersonation – An attacker may contact Bob under false pretences and convince him of a false identity. He could deliberately supply Bob with disinformation.
Affected assets: A1
Not applicable due to assumptions: AS2

N2: Long-term key extraction – An attacker gaining access to a device running a Grið instance could extract the long-term key and thus decrypt any encrypted container whose one-time key has been encrypted using the long-term keys stored on the device during the time-frame the attacker has access to it.
Affected assets: A1, A2, A3, A4
Not applicable due to assumptions: AS1, AS4

N3: Man-in-the-middle – An attacker could intercept and replace the transmission of Alice's Grið identifier, mount a MITM attack, and subsequently authenticate himself to Alice as Bob.
Affected assets: A1, A2, A4
Not applicable due to assumptions: AS2, AS6, AS7

N4: Private key extraction – An attacker gaining access to the sender's device could extract the private key and thus sign counterfeit identifiers and subsequently mount MITM attacks.
Affected assets: A6
Not applicable due to assumptions: AS1, AS7

The above enumerations make it obvious how even a single alteration to the assumptions affects the number of relevant threats and how some of these are effectively prevented as a consequence of different assumptions. Table 6.5 graphically illustrates the mapping of threats to assets.

Threats	Assets					
	A1	A2	A3	A4	A5	A6
T1	⊗	⊗	○	⊗	○	○
T2	○	○	○	○	⊗	○
T3	⊗	⊗	○	⊗	○	○
T4	⊗	○	○	○	○	○
N1	⊗	○	○	○	○	○
N2	⊗	⊗	⊗	⊗	○	○
N3	⊗	⊗	○	⊗	○	○
N4	○	○	○	○	○	⊗

Table 6.5: Scenario 2 mapping of threats to assets

Having discussed assets, threats, and threat agents, security objectives can be elaborated on.

6.6.5 Security Objectives

This section essentially reflects the differences to the previous scenario, resulting on a longer list of security objectives:

O1 Users must ensure that the devices used running a Grið instance are free from malicious soft– and hardware.

O2 Alice must inform Bob about her public key out-of-band.

O3 Alice defines an authentication challenge which only the intended recipient, Bob, can stand.

O4 Grið stores a user's long-term key locally, at a location only the user has access to.

O5 Alice contacts Bob and publishes encrypted containers in a way which allows her to remain anonymous to the network.

O6 Grið relies on well-established ciphers and appropriate key-lengths.

O7 Grið employs authenticated encryption.

O8 Alice must sign every published identifier with her private key.

O9 Alice must not disclose her private key.

O10 Bob must verify the signatures on all received identifiers.

Being able to verify the origin of a received identifier obviously averts impersonation and MITM attacks, as can be seen in Table 6.6.

The assumption that Alice can convey her public key to Bob out-of-band is crucial, especially if Bob only knows Alice's pseudonymous identity. In case Alice and Bob met in person to exchange key material and planned ahead for a future information exchange, all stated assumptions hold and third parties cannot impersonate Alice during any further information exchange. Depending on the point-of-view, this can either be considered no improvement over the previous scenario in case a third party conveys its public key out-of-band and assumes Alice's pseudonym, or in diametrical opposition to all other scenarios, in case the initial contact is in fact established between Alice and Bob. The latter situation can easily arise if Bob actually knows Alice. However, the simple fact that Alice is known to Bob does not necessarily prevent her from remaining pseudonymous to the network during the actual information exchange. In this case, only DoS attacks remain. However, this is considered a non-issue, since any connected system can be subject to outages.

Objectives	Threats								Assumptions						
	T1	T2	T3	T4	N1	N2	N3	N4	AS1	AS2	AS3	AS4	AS5	AS6	AS7
O1	○	○	✓	○	○	✓	○	✓	✓	○	✓	✓	○	○	✓
O2	○	○	○	○	✓	○	○	○	○	✓	○	○	○	○	○
O3	○	○	✓	○	○	○	○	○	○	✓	○	○	○	○	○
O4	○	○	✓	○	○	○	○	○	○	○	○	✓	○	○	○
O5	○	✓	○	○	○	○	○	○	○	○	○	○	✓	○	○
O6	○	○	✓	○	○	○	○	○	○	○	○	○	○	○	○
O7	○	○	○	✓	○	○	○	○	○	○	○	○	○	○	○
O8	○	○	○	○	✓	○	✓	○	○	○	○	○	○	✓	○
O9	○	○	○	○	✓	○	✓	○	○	○	○	○	○	○	✓
O10	○	○	○	○	✓	○	✓	○	○	○	○	○	○	○	○

Table 6.6: Scenario 2 mapping of objectives to threats and assumptions

6.7 Scenario 3: Relying on External identity providers

This scenario, albeit not directly[14] supported by the implemented prototype, involves a trusted third party in the form of an *identity provider* (IDP).
Alice relies on an IDP to authenticate Bob instead of defining an authentication challenge based on mutual knowledge. Alice must therefore know some personal details of Bob (such as his e-mail address) and must also be aware of some trustworthy IDP Bob is registered with.

Alice registers her Grið instance at an IDP of her choice, encrypts some data for Bob, and transmits the resulting encrypted container and her Grið instance's identifier to Bob. The channel(s) used to transmit this information are not subject to any security constraints. It can be assumed, however, that different channels are used.
When Bob contacts Alice's Grið instance with the wrapped key he extracted, Alice recovers the information pointing to the IDP. She subsequently redirects Bob to the IDP where he must identify himself. Upon successful identification the IDP redirects Bob to Alice's Grið instance. More specifically, Alice previously registered a redirection URI pointing to the hidden service operated by her Grið instance with the IDP. GitHub, Inc., for example, very clearly state that this workflow can be achieved by relying on *GitHub* as an IDP through OAuth [GitHub, Inc., 2015]. When Bob connects to Alice through the URI he received from the IDP after identifying himself, she can be sure that she is actually receiving an incoming connection from Bob[15]. Obviously, Bob can also be certain that he is indeed connecting to Alice, since an honest IDP

[14]Depending on the exact workflow of the chosen IDP, this scenario could be emulated using the provided prototype.

[15]Either the IDP only presents the URI to Bob or the URI is structured in such a way that Alice knows who identified himself to the IDP.

will redirect him to the URI Alice registered. Bob then retransmits the wrapped key to Alice through the connection established to the redirection URI he received from the IDP. Alice subsequently unwraps the one-time key and finally presents the ephemeral key to Bob. Figure 6.3 illustrates this workflow.
The specifics of this scenario's operational environment, assumptions, threat agents, and threats are discussed in the following sections.

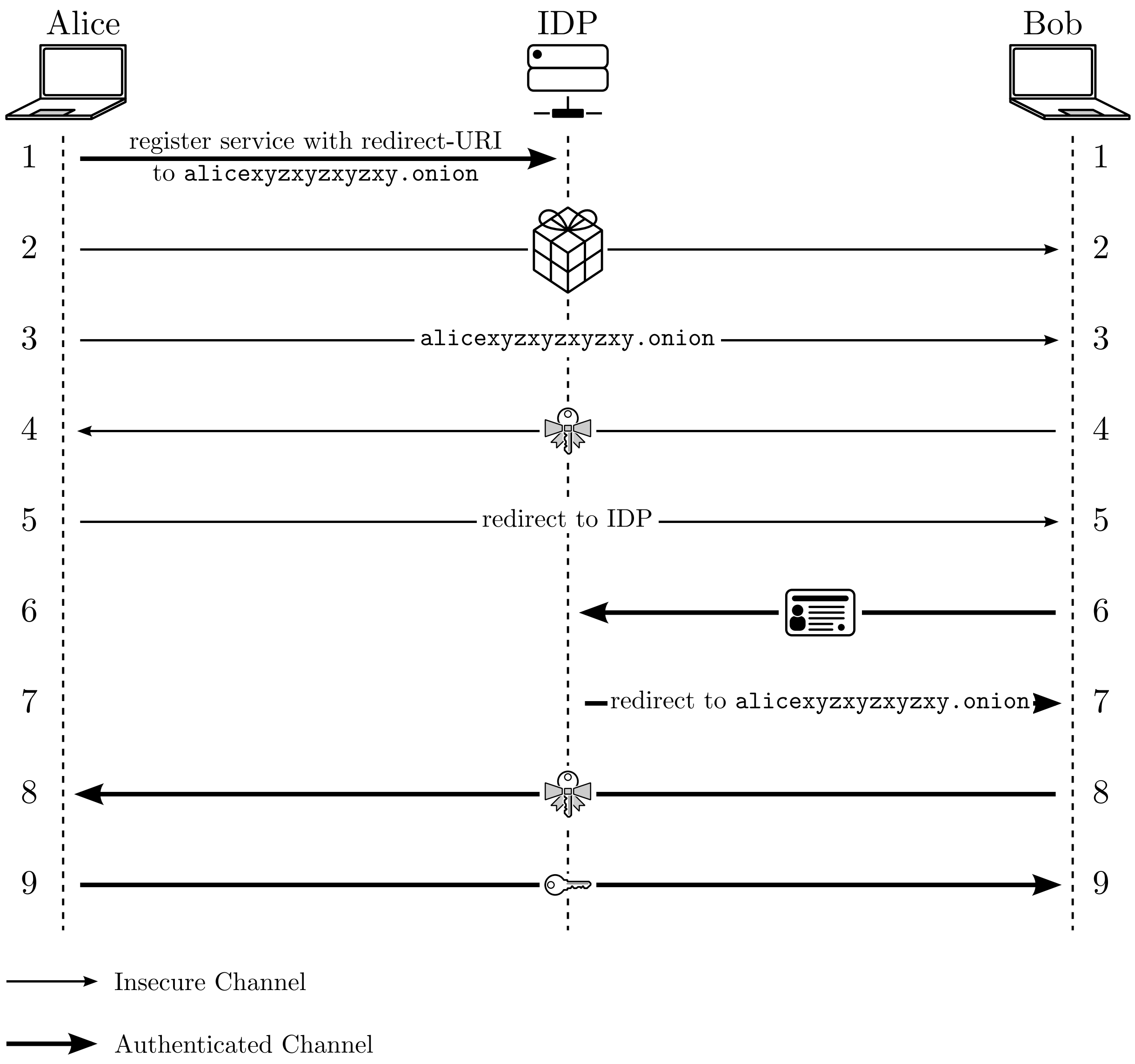

Alice
IDP
Bob
1
register service with redirect-URI to alicexyzxyzxyzxy.onion
2
3
alicexyzxyzxyzxy.onion
4
5
redirect to IDP
6
7
redirect to alicexyzxyzxyzxy.onion
8
9
Insecure Channel
Authenticated Channel

Figure 6.3: Evaluation scenario 3 workflow

6.7.1 Operational Environment

This scenario's environment is, perhaps, the most complex compared to the previously discussed ones. Alice's anonymity is essentially irrelevant to the workflow, but it is implicitly assumed that Bob does not know her and has no way of truly authenticating her. Consequently, the same holds for any data she sends Bob. All relevant assumptions are discussed in the following section.

6.7.2 Assumptions

In contrast to the previously evaluated scenarios, further assumptions must hold since a third party is involved. The full set of assumptions relevant to this scenario is as follows:

AS1 Every device running a Grið instance is free from malicious soft– and hardware and works as designed and intended by the user and is thus trustworthy.

AS2 The IDP selected by Alice is trustworthy and so are the communication channels used to communicate with it.

AS3 Consequently, only the intended recipients are able to identify themselves as such to the IDP.

AS4 Users' long-term keys are not disclosed.

Compared to all previous scenarios, not all assumptions can be asserted to hold by the user. While the IDP's trustworthiness can be established to some extent, such judgements can easily become obsolete without anyone noticing in case the IDP becomes compromised. However, such events and their impact are considered to be out of scope for this evaluation. The current evaluation scenario considers only the threat agents and threats discussed in the corresponding section.

6.7.3 Assets

Since Alice's anonymity is not considered relevant to this scenario (but rather implied) and the trustworthiness of the IDP relied on is simply assumed, the following assets are considered:

A1 The *data* transmitted by means of an encrypted container

A2 The *one-time key* including the ephemeral key and the authentication information (in this case, the URL pointing to the IDP) encrypted under the sender's long-term key in form of a wrapped key

A3 The sender's *long-term key* used to wrap the one-time key, stored on the sender's device

A4 The *secret IDP information* such as the *redirect URL* and the *information indicating who is allowed to access a container's payload* upon successful identification to the IDP.

Asset A4 is apparently closely related to Asset A1. However, the IDP is in part responsible for keeping this information secret. Therefore it is treated as a distinct asset.

As before, the final step before establishing a verdict about Grið's resilience to threats is to define all relevant threat agents and threats.

6.7.4 Threat Agents and Threats

Due to the fact that the IDP is assumed to be trustworthy and attacks on the IDP are not considered, the following threat agents are relevant to the current scenario:

TA1: Observer – Can observe the announcement and retrieval of an encrypted container as well as the announcement of an instance's identifier.

TA2: Interceptor – Has full access to the channels used to transfer an encrypted container and an instance's identifier.

TA3: Impersonator – Is fully decoupled from the actual workflow, but may want to advertise himself as someone else to Bob during the initial contact.

The identified threat agents pose the following threats:

T1: Denial-of-service – An attacker could replace the encrypted container created by Alice with some other container or simply keep Bob from receiving the original container. If Bob does not receive the original container created by Alice, he cannot access the information intended for him.
Affected assets: A1, A2, A4
Involved threat agents: TA2

T2: Decryption – An Attacker could try to decrypt an encrypted container, possibly by brute-forcing the ephemeral key.
Affected assets: A1, A2, A4
Involved threat agents: TA1 or TA2

T3: Spoofing – An attacker could try to modify the encrypted payload by directly operating on the ciphertext part of an encrypted container which holds the payload. The attacker could thus alter the plain text which is recovered upon decrypting a container.
Affected assets: A1
Involved threat agents: TA2

T4: Impersonation – An attacker may contact Bob under false pretences and convince him of a false identity. He could deliberately supply Bob with disinformation. The actual impact of such an attack varies greatly.
Affected assets: A1
Involved threat agents: TA3

Assuming the selected IDP is in fact trustworthy and Bob does not share his credentials used to identify himself to the IDP with anyone, the following are considered non-threats:

N1: Long-term key extraction – An attacker gaining access to a device running a Grið instance could extract the long-term key and thus decrypt any encrypted container whose one-time key has been encrypted using the long-term keys stored on the device during the time-frame the attacker has access to it.
Affected assets: A1, A2, A3, A4
Not applicable due to assumptions: AS1, AS4

N2: Man-in-the-middle – An attacker could intercept and replace the transmission of Alice's Grið identifier, mount a MITM attack, and subsequently authenticate himself to Alice as Bob.
Affected assets: A1, A2, A4
Not applicable due to assumptions: AS2

A graphical mapping of threats to assets is provided in Table 6.3 .

Threats	Assets			
	A1	A2	A3	A4
T1	⊗	⊗	○	⊗
T2	⊗	⊗	○	⊗
T3	⊗	○	○	○
T4	⊗	○	○	○
N1	⊗	⊗	⊗	⊗
N2	⊗	⊗	○	⊗

Table 6.7: Scenario 3 mapping of threats to assets

Lastly, the security objectives of this scenario shall be discussed to reach an evaluation verdict.

6.7.5 Security Objectives

This scenario's security objectives are more dependent on external factors than previously due to the IDP being an integral part of the authentication workflow:

O1 Users must ensure that the devices used running a Grið instance are free from malicious soft– and hardware.

O2 Alice must select a trustworthy IDP.

O3 Bob must not share the credentials he uses to identify himself to the IDP with anyone.

O4 Grið stores a user's long-term key locally, at a location only the user has access to.

O5 Grið relies on well-established ciphers and appropriate key-lengths.

O6 Grið employs authenticated encryption.

Table 6.8 illustrates how relying on an external IDP can enable secure information exchange through unauthenticated channels without falling victim to MITM attacks.

Objectives	Threats						Assumptions			
	T1	T2	T3	T4	N1	N2	AS1	AS2	AS3	AS4
O1	○	○	○	○	○	○	✓	○	○	✓
O2	○	○	○	○	○	✓	○	✓	✓	○
O3	○	○	○	○	○	○	○	○	✓	○
O4	○	✓	○	○	✓	○	○	○	○	✓
O5	○	✓	○	○	○	○	○	○	○	○
O6	○	○	✓	○	○	○	○	○	○	○

Table 6.8: Scenario 3 mapping of objectives to threats and assumptions

Upon examination of this scenario's properties, it becomes apparent that no MITM attacks are possible. Even if some malicious third party intercepts the encrypted container created by Alice and/or Alice's Grið instance's identifier and acts as a man in the middle, all assets are still safe: In case an attacker tries to transparently act as a relay by transmitting his Grið instance's identifier to Bob instead of Alice's, the IDP will redirect Bob to Alice's Grið instance upon identification. At this point, Bob can be sure that he can directly connect to Alice without anyone eavesdropping on the one-time key exchange. In case an attacker modifies an encrypted container, Bob will never receive the original redirection URI and will thus not be able to contact Alice. Therefore, MITM attacks can effectively be downgraded to DoS attacks. However, the disinformation attack discussed as part of the baseline scenario still applies.

6.8 Comparison and Discussion

When evaluating different scenarios, it could be concluded that Grið does not provide any benefits compared to existing solutions such as PGP or PKIX: After all, either an authenticated channel needs to be employed, public keys need to be known, or third parties need to be involved as summarised in Table 6.9. This, however, is a superficial point-of-view neglecting the details which substantially differentiate Grið from well-established approaches.

	Requirements				Properties				
Scenarios	Authenticated channel	Alice's public key known	Bob's public key known	Trusted third party	Alice knows Bob	Bob knows Alice	Immune to MITM attacks	Immune to impersonation	Immune to DoS attacks
Baseline	○	○	○	○	○	○	○	○	○
Scenario 1	✓	○	○	○	✓	○	✓	○	○
Scenario 2	○	✓	○	○	○	✓	✓	✓	○
Scenario 3	○	○	○	✓	○	○	✓	○	○

Table 6.9: Comparison of evaluated scenarios

The simple fact of being able to easily evaluate a system in multiple different environments speaks to its flexibility. In addition, the encrypting entity is in total control, since it does not need to rely on the integrity of the receivers' public keys. This enables senders to freely choose an authentication mechanism and communication channels they feel comfortable with. In essence, the encrypting entities define the security properties of the environment and can thus deliberately avert or take certain risks depending on the desired security level. If secret information is disclosed to unauthorised third parties during transmission, the encrypting entity can only blame itself for it. Consequently, the burden of assessing the situation with respect to its security properties is where it belongs: Someone wishing to share secret information should be responsible to keep it from unauthorised third parties, since it is also this person's responsibility to choose with whom to share this information in the first place.

7 Related Work

Any system's utility can be judged to some extent by comparing it to existing ones. Consequently, Grið is compared to existing solutions aimed at secure information exchange in order to evaluate its utility, but also to differentiate it from other approaches. Grið's unique selling point is the inverted trust model in conjunction with a highly flexible authentication mechanism. Additionally, due to relying on Tor, it is possible for both senders and receivers to remain anonymous to the network. Due to cryptograms being transferred in the form of the CMS-inspired encrypted container format, the channel/medium used for exchanging data can be chosen freely. The Grið prototype already integrates an anonymous file hosting service to demonstrate this. This feature-set is considered a major improvement over the existing solutions discussed in the following sections. Due to Grið's flexibility and the incorporation and combination of various concepts, the approaches discussed in this chapter differ significantly. Yet, all of these are related to Grið in some aspects, which already speaks to Grið's utility and versatility. Each of the systems Grið is compared to serves to illustrate different aspects of Grið's novel approach to secure information exchange.

Starting with widely-adopted systems (such as PGP) featuring well-defined security properties, more specialised ones will also be discussed, as these relate more closely to Grið.

7.1 Pretty Good Privacy and PKIX-Based Systems

Public key infrastructures based on PGP and PKIX have already been discussed as part of Chapter 2. With both certificate-based PKIXs and PGP being highly popular, Grið's properties need to be evaluated against these established systems. Obviously, these two approaches already differ, since PKIXs inherently rely on third parties, while PGP places the burden of establishing trust on the individual users.
However, these systems essentially provide the same service and, from a high-level point-of-view, their workflow is the same: Alice wanting to transmit some data to Bob without someone else being able to access it encrypts it under Bob's public key prior to transmission. Obviously, only Bob is able to recover the original data, since only he has access to his private key which is

needed to perform the decryption process. How Alice got to know Bob's public key does not matter at this point[16].

What differentiates Grið from both PGP and PKIXs is its inverted trust model as thoroughly explained in Section 5.1. Using Grið, the encrypting entity is not at the receiving party's mercy anymore: The sender defines an authentication mechanism he or she considers to provide an appropriate security level. Therefore, someone wishing to convey some secret information to another party does not need to rely on whichever (potentially compromised, possibly too short, or maybe even unsupported) public key is associated with the receiving party. On the contrary: The sender is always in full control and can freely define how recipients must authenticate themselves in order to gain access to the decryption keys needed to recover the transmitted data.

Although PGP and PKIXs have so far been considered equivalent, there is one major difference between these two which also relates to Grið: The decentralised approach of PGP does not force communicating parties to reveal their identity to anyone. Both parties can even remain pseudonymous to each other in some cases. The same is also true for Grið, whereas a PKIX also deals with liability issues (see Section 2.3.1.2). Due to the identity validation typically performed by a certificate authority in the PKIX context, additional effort is required to anonymously obtain a digital certificate – the system is simply not designed for such a use-case.

Due to its unique features such as its inverted trust model which puts the sender in total control Grið is considered an improvement over both PGP and PKIX-based systems.
The next approach discussed in this chapter is specifically designed to enable secret, authenticated, and disreputable communication. In some cases it can therefore be a significant improvement over PGP and PKIX-based systems.

7.2 Off-the-Record Messaging

The *Off-the-Record Messaging* (OTR) protocol enables truly private online conversations by trying to transfer the properties of a personal *off-the-record* conversation to instant messaging conversations [Borisov, Goldberg and Brewer, 2004, p. 78]. Through clever application of well-known cryptographic primitives, it enables confidential and authenticated messaging while at the same time providing plausible deniability. The key properties of the protocol are outlined as follows:

- Perfect forward secrecy will be used to ensure our past messages cannot be recovered retroactively.

[16]Naturally, in case a PKIX is involved, it must not be compromised for the exchanged data to remain confidential.

- Digital signatures will be used so that Bob knows with whom he's communicating.
- Message authentication codes will be used to prove Alice's authorship of a message to Bob, while at the same time *preventing* such a proof to third parties.
- Malleable encryption will be used to provide for forgeability of transcripts, repudiation of contents, and plausible deniability. [Borisov, Goldberg and Brewer, 2004, p. 78]

Even though the protocol has evolved significantly since its original publication, all of the above statements still hold. However, in order to perform an accurate comparison between Grið and OTR, the latest protocol specification published by Goldberg and the OTR Development Team [Goldberg and the OTR Development Team, no date] is referred to wherever applicable.

While additional technical details are not relevant for being able to compare OTR-secured instant messaging to Grið, the process of key-agreement and mutual authentication is. OTR employs a combination of *authenticated key exchange* (AKE) based on the Diffie-Hellman key exchange in conjunction with the *Socialist Millionaires' Protocol* (SMP) [Goldberg and the OTR Development Team, no date].
In essence, the secure channel used to convey off-the-record messages is established first and the SMP allows for detection of MITM attacks. This is achieved by employing *oblivious transfer* in the following way: Two communicating parties agree on a shared secret and initiate an OTR session. Once a secure channel is established, the shared secret is compared by means of oblivious transfer. MITM attacks can effectively be averted due to the fact that the shared secret is bound to the cryptographic keys used to establish the channel [Goldberg and the OTR Development Team, no date].

Obviously, OTR seems to be a considerable improvement over PGP and PKIX, as well as Grið. However, due to the protocol's design catering to real-time communication, it cannot easily be applied to high-latency applications media such as e-mail [Borisov, Goldberg and Brewer, 2004, p. 82]. This is not the case with Grið; its authentication workflow (although typically completed within few minutes) does not impose any timing constraints. On the other hand, Grið's flexible design could make it feasible to integrate OTR-based authentication in order to bring some of the protocol's properties to Grið. Until then, OTR-secured instant messaging could be used as an authenticated channel to initiate the exchange of large files through Grið. This is potentially relevant in practice since OTR operates on top of existing instant messaging protocols [Borisov, Goldberg and Brewer, 2004, p. 81], some of which do not support file transfer.

In case communicating parties value their anonymity, Grið could theoretically be considered a mere complement to OTR for exchanging large files. However, due to OTR typically relying on infrastructures operated by third parties (since it operates on top of existing instant messaging protocols), it is generally difficult to reach a reliable conclusion about a communicating party's

anonymity.
In the following section, another approach aimed at securing existing systems for information exchange shall be compared to Grið.

7.3 Cloud Storage Encryption Solutions

Cloud storage encryption solutions provide a service for encrypting files stored at cloud storage providers. From this point-of-view such services follow an approach similar to OTR which also operates as a security layer on top of existing protocols. In this section *BoxCryptor* [Secomba GmbH, no date(a)] will be used to discuss this approach in order to compare Grið to a real-world product aimed at cloud storage encryption.

Secomba GmbH provide a thorough overview about BoxCryptor's technical aspects covering the employed cryptographic primitives as well as access control and the file-sharing workflow [Secomba GmbH, no date(b)][17].
BoxCryptor is advertised as being *zero-knowledge* in the sense that the infrastructure used by the system never gains access to users' key material. At the same time, the system enables users to share files with others by relying on hybrid encryption similar to CMS: A file is encrypted using AES in *cipher block chaining* (CBC) mode under a randomly-generated key. This key is then wrapped using the recipient's public key and subsequently transferred. The recipient being in possession of the corresponding private key can then recover the AES key and decrypt the file contents. The BoxCryptor infrastructure is in such cases used as a public key directory (as described in Section 2.2). Therefore, the workflow advocated by BoxCryptor only differs in its execution from PKIX-based approaches.

Obviously, users may not easily be able to remain anonymous when using such a service, since accounts at the service provider are required. Moreover, the reliance on a centralised infrastructure to enable convenient sharing of encrypted files is a drawback compared to certificate-based approaches (as explained in Section 2.2.1). At the same time, trust in the recipient's public keys is required much in the same way as mandated by PGP and typical PKIX-based solutions. Due to the dependency on the public key directory operated by a single party, cloud storage encryption solutions such as BoxCryptor are inherently inflexible and prone to service outages.

Finally, due to cloud storage encryption approaches typically being offered as a product by some company seeking to monetise it, the implementations may remain closed-source. In such cases it is difficult to (dis)prove the claims of a product's vendor. More importantly, the correctness of the implemented functionality cannot easily be verified. BoxCryptor's technical documentation

[17]The conclusions reached in this section will be based on the technical overview published by Secomba GmbH [Secomba GmbH, no date(b)], since it is the only official documentation available.

(upon which this section is based) mentions relying on a custom cryptographic library developed in-house in order to offer the service for one of the supported platforms.

In summary, Grið can also be utilised to offer secure cloud-based file sharing, however, without needing to trust a third party or the public keys of the receiving parties.
The last system discussed in this chapter is more closely related to Grið as it relies on Tor hidden services to enable P2P-based file exchange and also does not mandate trust in either recipients' public keys or a third party.

7.4 OnionShare

OnionShare is a tool to "securely and anonymously share a file of any size" [Lee, no date]. The same could also be said about Grið's intentions. Furthermore, OnionShare's incorporation of Tor hidden services results in a workflow similar to Grið's (as described in Section 5.1.1). According to its *security design document*, OnionShare works as follows:

> First, the sender chooses files and folders they wish to share with the recipient. OnionShare then starts a web server at `127.0.0.1` on a random port. It generates a random string called a slug, and makes the files available for download at `http://127.0.0.1:[port]/[slug]/`. It then makes the web server accessible as Tor hidden service, and displays the URL `http://[hiddenservice].onion/[slug]` to the sender to share. [Lee, 2015]

OnionShare does not encrypt files, nor does it include any authentication mechanism. Its security model effectively boils down the sender's ability to communicate a URL in secret. *HTTP basic authentication*, for example, could improve this situation. However, as explored in Chapter 6, attacks could still be mounted without an authenticated channel in place. Grið's flexible authentication framework based on CrySIL presents a considerable improvement over OnionShare's workflow and can be utilised to counter various threats. OnionShare, on the other hand, lacks even the most basic authentication mechanisms and requires confidential transmission of a URL to achieve even a basic level of security, whereas Grið does not require communication channels offering confidentiality. OnionShare's purely peer-to-peer nature does, however, constitute an advantage. Due to this design aspect, OnionShare does not depend on any external infrastructure (except Tor) which allows for sharing files without involving any third parties.

In summary, OnionShare enables secure P2P file sharing, given senders are able to convey a URL in secret to all recipients. Its literally non-existent authentication mechanism is, however, considered a major drawback compared to Grið's flexible authentication framework.

Of course, other systems aimed at secure information exchange exist. Due to the fact that these are either too distinct from Grið's approach or operate similarly to the already discussed approaches, exploring these is not expected to yield any new insights. Therefore, Grið is not compared to any other systems given the fact that none of the already examined ones are on feature-parity with Grið.

8 Conclusions and Outlook

This work presented Grið, a new approach to secure information exchange based on an inverted trust model. Building upon the CrySIL framework and on the Tor anonymity network, Grið enables anonymous, secure information exchange without the need to express trust in entities' public keys.
Contrary to existing solutions such as PGP, where data is encrypted using the recipients' public keys, Grið employs randomly generated encryption keys created by the sender. In order to gain access to this key material, recipients must authenticate themselves to the sender, thus inverting the traditional trust relationship. This design empowers the encrypting entity (who decided to share sensitive information in the first place) and does not leave it at the recipient's mercy. By creating the key material and choosing an appropriate authentication method, the security level for a particular information exchange can be defined by the sender.

This novel approach, however, also introduces new challenges: Obviously, recipients must be able to contact the sender through some secure communication channel. Since one of Grið's goals is to enable both communication parties to remain anonymous to the network, this channel must also provide anonymity in addition to confidentiality and integrity. At first glance it may seem as if these issues are unrelated. However, after exploring the theoretical foundation of peer-to-peer networks (which enable communicating parties to establish direct communication channels) it becomes apparent that strong anonymity properties can only be achieved within a large network of peers. The decision to rely on the world's most popular anonymity network, Tor, as transport layer enables Grið to provide secure, anonymous communication channels to carry out the previously mentioned authentication procedures.
In addition, a transport format was developed which eliminates potential security flaws introduced by key management policies: By packaging encryption keys alongside encrypted data, key management itself becomes a non-issue. Inspired by the Cryptographic Message Syntax, Grið's transport format utilises key wrapping using long-term secret keys to be able to distribute encryption keys as well as all authentication information in conjunction with the encrypted data. During the authentication procedure, the recipients transmit this enciphered bundle of key material and authentication information to the original sender who can recover it and present the previously defined authentication challenge. Consequently, the sender does not have to maintain

a record of distributed encryption keys or implement stateful access control protocols; the recipients are responsible for providing the required authentication information. In case recipients fail to provide this information, it is impossible to recover the key material required to decrypt the payload due to the fact that the original encryption keys are part of this wrapped data structure.

Furthermore, due to Grið's versatility, properties such as sender anonymity and data authenticity can be selectively achieved or deliberately neglected. Different scenarios have been evaluated with respect to their security properties and aspects such as the aforementioned sender anonymity. Grið can employ traditional public-key approaches or even rely on trusted third parties to provide additional security features if the circumstances demand it.

Apart from evaluating the security properties of Grið, the design was also compared to various existing systems aimed at secure information exchange. On the one hand, this evaluation highlighted the novelty and advantages of Grið's workflow compared to existing approaches. On the other hand, some room for improvement has also been discovered in doing so.
Most prominently, the combination of OTR and the Socialist Millionaires' Protocol would provide a welcome addition to Grið's authentication scheme, as it could be used to effectively avert MITM attacks during the authentication process itself. For instance, a successful password-based authentication would also guarantee that a direct connection between two communicating parties has been established without the need to rely on a third party or to express trust in an entity's public key. This property would considerably secure Grið's overall workflow: Most importantly, it would enable users to utilise insecure channels to initiate an information change without the risk of falling victim to MITM attacks. In general, implementing additional authentication mechanisms is desirable in order to put more of Grið's potential at the user's disposal.

Additionally, the ability to directly transmit files to recipients without being forced to rely on external storage media is also considered a prime candidate for improvement over the current prototype. As demonstrated by OnionShare, being able to utilise a single channel for information exchange can make it significantly easier for users to remain anonymous to the network; users would not have to worry about the properties of a variety of different channels.

In situations where anonymity is irrelevant, but the authenticity of a transmission is of high importance, the direct incorporation of public-key cryptography into Grið's workflow can lead to considerable improvements. While this can already be done, public-key-based approaches are not integrated into the current prototype. When reviewing the scenarios evaluated in Chapter 6, it becomes apparent how relying on well-defined electronic signature approaches can eliminate some attacks. The scenario discussed in Section 6.6 illustrates how public-key cryptography can be combined with Grið's inverted trust approach without shifting control to recipients while at the same time averting various classes of attacks. This observation not only presents room for improvement of the prototype, but also highlights some potential for further research.

Lastly, some outstanding issues with the current implementation need to be touched: Most importantly, licensing issues with the CrySIL code-base arose during the development of Grið. Consequently, the currently implemented prototype cannot be open-sourced or even distributed. This is considered a major issue. Thus, providing Grið as open-source software is deemed a high priority task for future work. However, this endeavour has to be coordinated with the CrySIL developers and will therefore remain an open issue for the time being.
Secondly, an implementation of Grið for Android mobile devices is not only desirable but also feasible. Both CrySIL's and Grið's modular architecture allow for implementing the current prototype's feature-set on the Android platform. Due to lack of resources, however, such an application has yet to be implemented and therefore also remains a proposal for future work. Still, the implemented prototype successfully demonstrates Grið's potential and readily enables secure information exchange based on inverted trust.

Appendix A

User Manual

Grið is a framework for secure information exchange based on an inverted trust model. Its concepts, its workflow, and the terminology used throughout this user manual are described in detail in Chapter 5. Therefore, the contents of Chapter 5 will not be reproduced here.

System Requirements

Grið runs on Microsoft Windows (tested only on Windows 7), recent Linux distributions, and recent versions of Apple OS X. Since Grið is a graphical application based on OpenJDK, it requires X11 to be installed on OS X and Linux.
The required Java runtime environment is provided with Grið. Therefore, it is unnecessary to install Java beforehand.

A.1 First Run

Grið is provided as a self-contained archive for all platforms and does therefore not require installation; extracting the archive is enough.
Grið can be launched by executing *gridh.sh* on Linux and Apple OS X and *gridh.bat* on Microsoft Windows. It is important not to close any terminal / command prompt windows as this would terminate the Grið process, since Grið does not fork. Upon launch the user is presented with the application's main window as shown in Figure A.1.

Note: It can take up to half a minute before the application is fully loaded, depending on the system's performance.

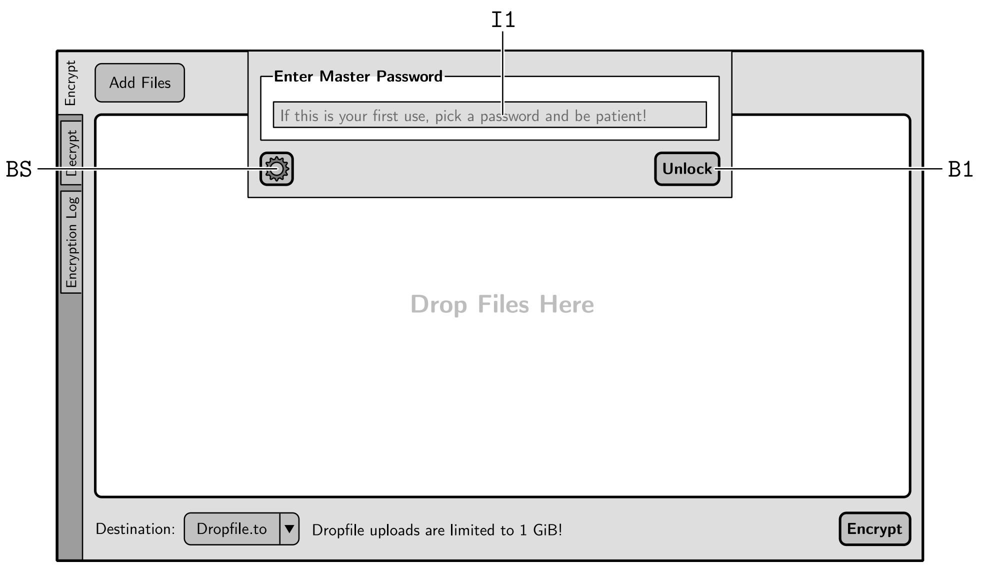


Figure A.1: Main application window

Before being able to encrypt and decrypt any data, a master password needs to be set to protect the long-term key used to wrap ephemeral keys and authentication information for subsequent cryptographic operations. Users can freely choose any password they see fit (there are no restrictions on the characters the password may be composed of) by entering it into Input Field `I1` and subsequently pressing Button `B1` labelled *Unlock*.
Afterwards, Grið will generate a new long-term key and protect it using the previously entered master password; this process can take up to a minute. When done, the main user interface depicted in Figure A.2 is presented to the user.

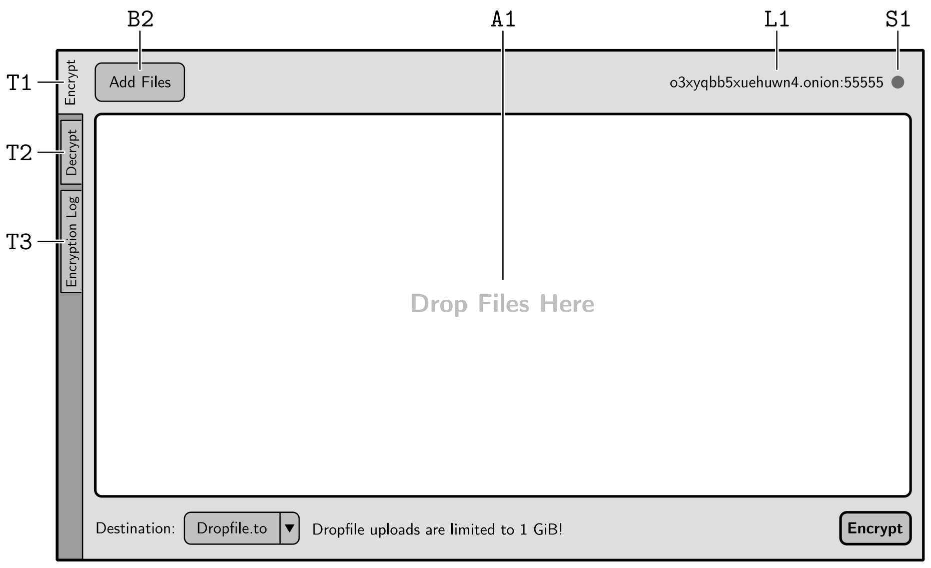


Figure A.2: Fully loaded application/encryption tab

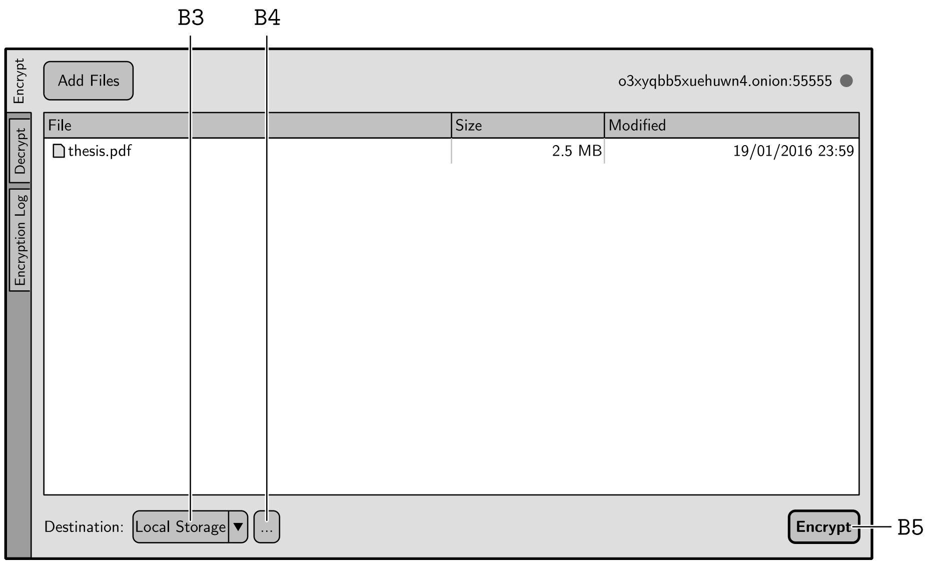
Note: The master password needs to be entered on every launch of Grið in the same way to unlock the long-term key.

A.2 Encryption

Upon successfully unlocking the master key, the main user interface is shown as illustrated in Figure A.2 and Tab `T1`, labelled *Encrypt*, should be active. Label `L1` displays this Grið instance's identifier; Status Indicator `S1` turns from red to green as soon as the hidden service associated with the instance's identifier has been successfully published.

To encrypt files, these need to be added first: Files can be added either by dragging them into Area `A1` (displaying the text *Drop Files Here*) or by pressing Button `B2`, labelled *Add Files*, which opens a file chooser dialogue to select files.

As soon as the first file is added Area `A1` will be replaced by a table containing a list of all added files as depicted in Figure A.3.



File	Size	Modified
thesis.pdf	2.5 MB	19/01/2016 23:59

Figure A.3: Encryption tab with local file storage

Note: It is possible to drag files onto this table to add them in the same manner as files can be dragged onto Area `A1`.

When all files have been added, users can select where the resulting encrypted container shall be stored. This can be done by pressing Dropbutton `B3`. Currently, only local storage and *Dropfile.to*, an anonymous online file hosting service, are supported. In case local storage has been chosen, Button `B4` becomes visible, which can be used to choose a destination file.

Next, the encryption challenge and the corresponding response need to be set, by pressing Button `B5`, labelled *Encrypt*. This brings up the dialogue depicted in Figure A.4 where all authentication information can be set. Currently users can choose between question-type and task-type challenges (both of which work the same way) by pressing Dropbutton `B6`. The challenge text presented to the recipient of a container can be entered into Input Field `I2`. The corresponding response the recipient needs to enter to receive the one-time key used to encrypt the resulting container's payload needs to be entered into Input Field `I3`. The container's expiry date and time can be set using Date Picker `D1` and Time Pickers `D2` and `D3`.

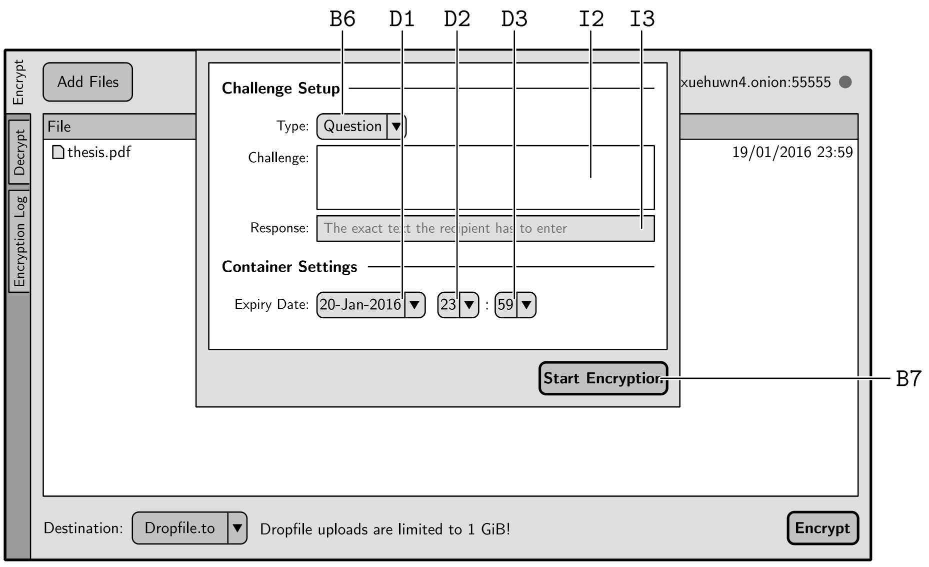


Figure A.4: Authentication information and expiry date details

When all values are set as desired, pressing Button `B7` starts the encryption and storage process which is indicated by progress indicators as illustrated in Figure A.5.

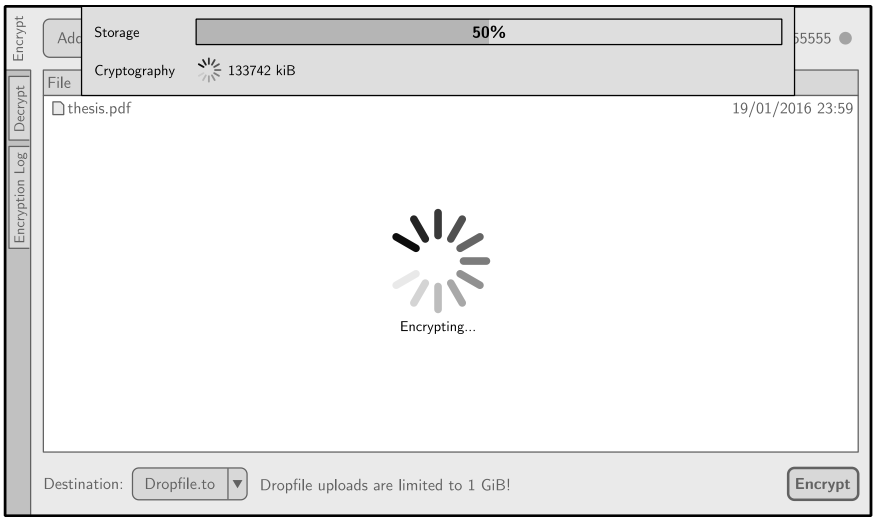
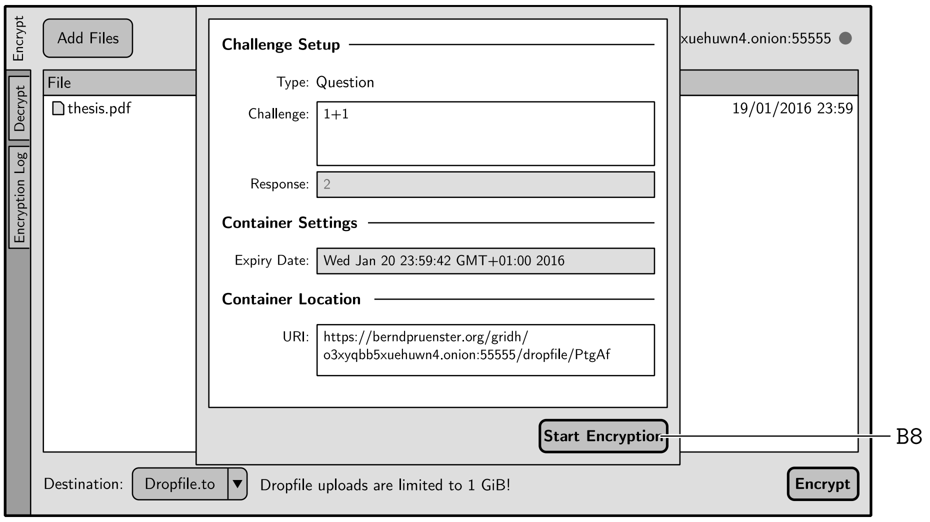


Figure A.5: Encryption in progress

Upon successful encryption, a summary is displayed as shown in Figure A.6. Pressing Button `B8` copies the URI pointing to the just-created container to the clipboard and dismisses the summary.



Challenge Setup

Type: Question

Challenge: 1+1

Response: 2

Container Settings

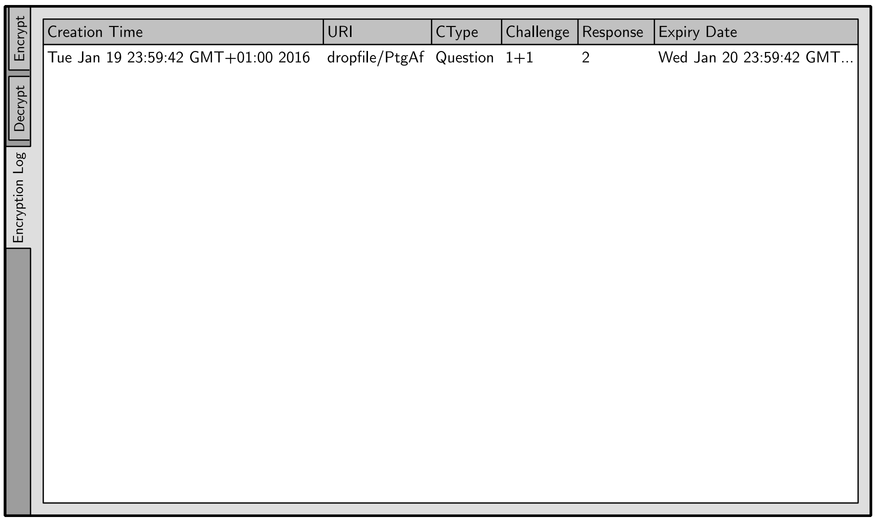
Expiry Date: Wed Jan 20 23:59:42 GMT+01:00 2016

Container Location

URI: https://berndpruenster.org/gridh/o3xyqbb5xuehuwn4.onion:55555/dropfile/PtgAf

Figure A.6: Encryption summary

Note: An overview of all successful encryptions since the last start of Grið is provided by activating Tab `T3`, labelled *Encryption Log* (see Figure A.7). Double-clicking on an item in the table shown in Tab `T3` brings up its corresponding summary as described previously; none of this information is stored persistently.



Creation Time	URI	CType	Challenge	Response	Expiry Date
Tue Jan 19 23:59:42 GMT+01:00 2016	dropfile/PtgAf	Question	1+1	2	Wed Jan 20 23:59:42 GMT...

Figure A.7: Encryption log

In case *Dropfile.to* has been selected as storage location, only the URI needs to be sent to the intended recipients to share a container (see Section A.3 for more details).

Note: *Dropfile.to* only stores files for 24 hours.

A.3 Decryption

To decrypt files, Tab `T2`, labelled *Decrypt*, needs to be active (see Figure A.8). In case a container stored at *Dropfile.to* shall be decrypted, a URI retrieved from the encryption summary of this container (see Section A.2) needs to be entered into Input Field `I4` depicted in Figure A.8. If a container stored locally is to be decrypted, it can be selected by first clicking on Header `H2` and subsequently pressing Button `B10` depicted in Figure A.9 to invoke a file chooser dialogue. Additionally, the sender's Grið instance's identifier needs to be entered into Input Field `I5`.

Note: Clicking on Header `H1` and `H2` allows for switching between decryption based on a URI or from a container file and its creators Grið instance's identifier.

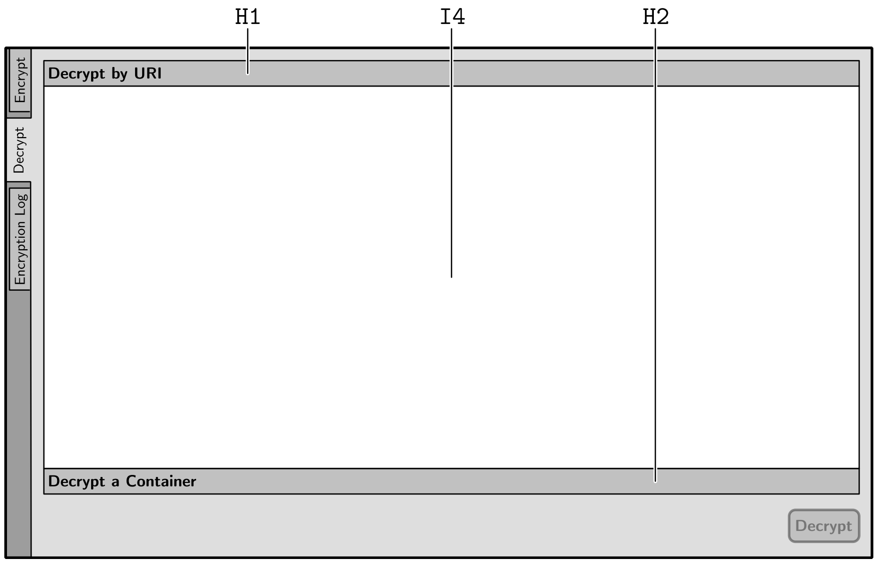


Figure A.8: Decryption by URI

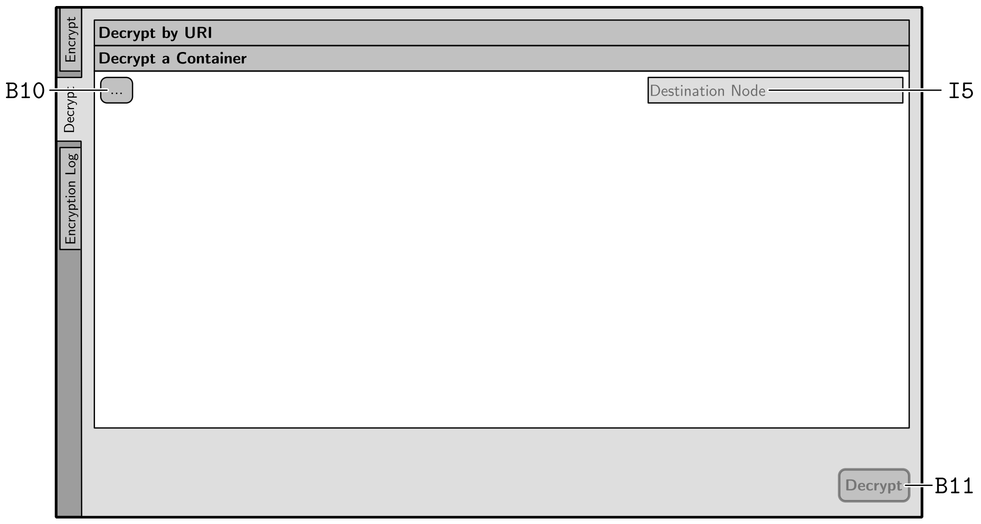


Figure A.9: Decryption of a locally stored container

Button B11 becomes active as soon as all information needed to start the decryption process has been entered. Pressing it starts the decryption process illustrated in Section 5.1.1 and brings up a progress indicator as shown in Figure A.10.

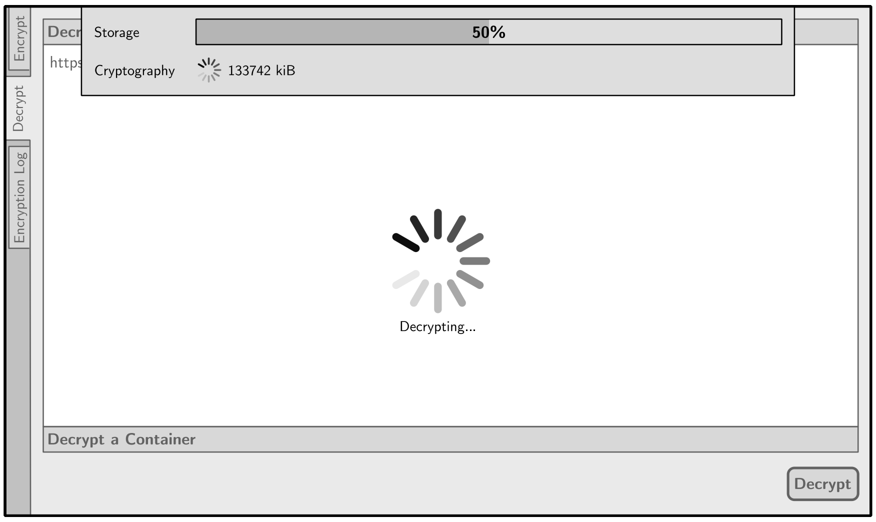


Figure A.10: Decryption in progress

Note: The original sender's Grið instance needs to be running and its hidden service successfully published to decrypt a container.

Once a connection to the original sender has been established, the challenge associated with the to-be-decrypted container is presented to the user as depicted in Figure A.11. The response to this challenge can be entered into Input Field `I6`. When done, pressing Button `B12` transmits this response to the original sender.

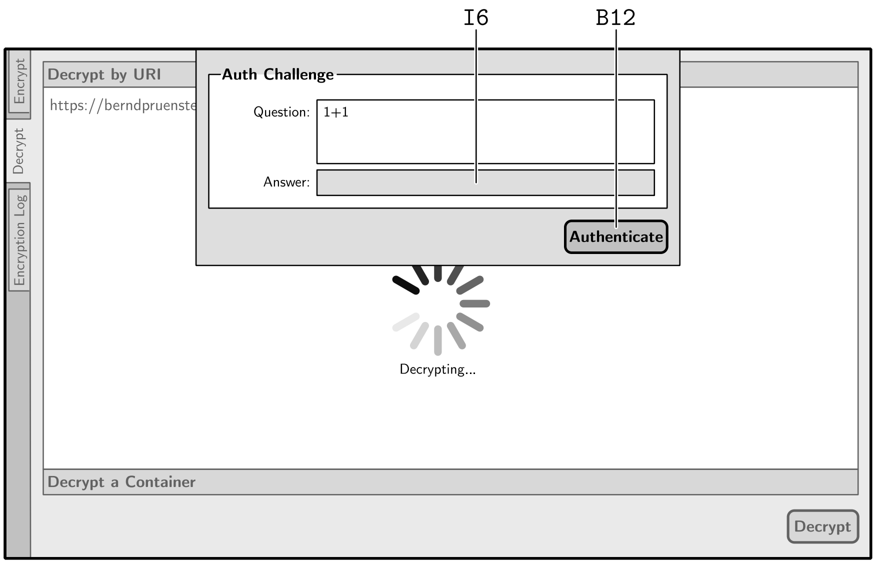
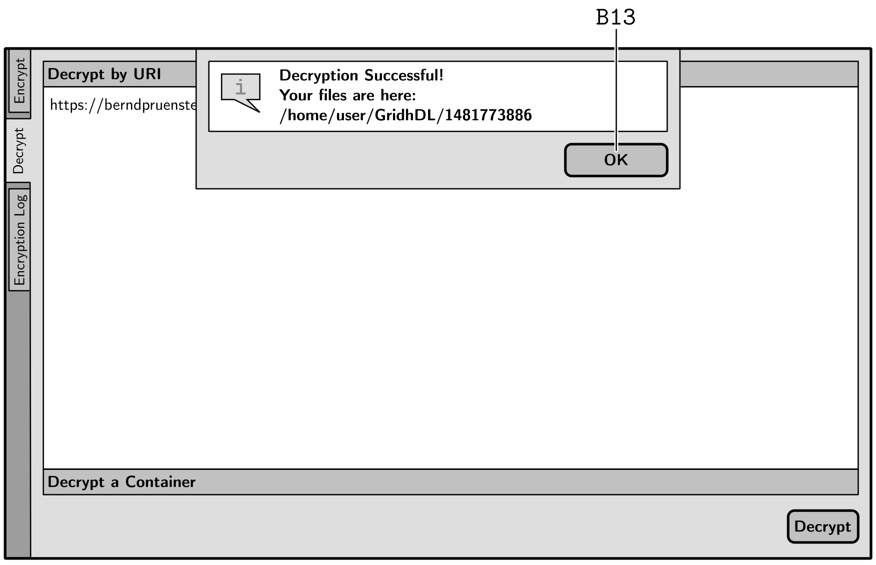


Figure A.11: Authentication challenge dialogue during a decryption process

A correct response will trigger the transmission of the one-time key used to encrypt the container's payload and subsequently recover this payload and inform the user about its location (see Figure A.12). Pressing Button `B13` closes this information dialogue and (on some systems) opens the directory containing the recovered payload. An incorrect response yields an error message.

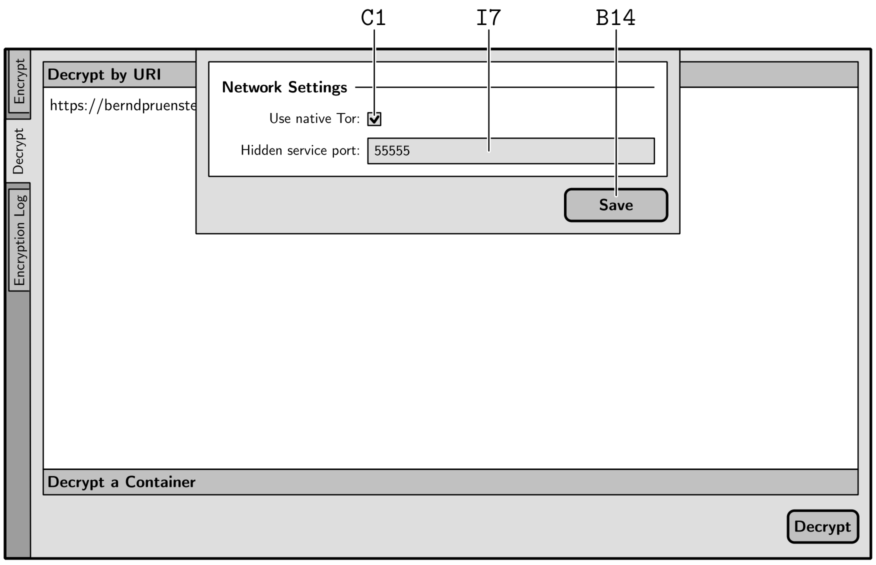


Decryption Successful!
Your files are here:
/home/user/GridhDL/1481773886

Figure A.12: Successful decryption

A.4 Troubleshooting

Due to the prototypical nature of Grið, feedback about errors is limited. In case unlocking the master key does not work (and Grið refuses to start) switching to a Java-based Tor implementation or changing the Tor hidden service port may help. Pressing Button `BS` depicted in Figure A.1 brings up a dialogue where these settings can be adjusted (see Figure A.13). Checkbox `C1` allows for switching between Tor implementations, while Input Field `I7` can be used to change the port for the hidden service. Pressing Button `B14` acknowledges the current settings.



Use native Tor: ✓
Hidden service port: 55555

Figure A.13: Grið settings

Bibliography

Anderson, Nate and Cyrus Farivar [2013]. *How the feds took down the Dread Pirate Roberts.* 3rd Oct 2013. `http://arstechnica.com/tech-policy/2013/10/how-the-feds-took-down-the-dread-pirate-roberts/` (Retrieved: 18/10/2015).

Apache Software Foundation [2014]. *BXML Primer.* 2014. `http://pivot.apache.org/tutorials/bxml-primer.html` (Retrieved: 07/12/2015).

Ascertia Limited [2015a]. *Digital Signatures, online document signing | SigningHub.com^{TM}.* 2015. `https://www.signinghub.com/` (Retrieved: 18/11/2015).

Ascertia Limited [2015b]. *Web Services Developers Guide.* 2015. `http://info.signinghub.com/web-services-developers-guide` (Retrieved: 18/11/2015).

Atkinson, R. [1995]. *Security Architecture for the Internet Protocol.* RFC 1825. Aug 1995. `http://www.rfc-editor.org/rfc/rfc1825.txt` (Retrieved: 15/09/2015).

Baetke, Walter [1978]. *Wörterbuch zur altnordischen Prosaliteratur.* Zweite, durchgesehene Auflage. Berlin, German Democratic Republic: Akademie-Verlag, 1978.

Berthold, Oliver, Andreas Pfitzmann and Ronny Standtke [2001]. 'The Disadvantages of Free MIX Routes'. In: *Designing Privacy Enhancing Technologies.* Edited by Hannes Federrath. Volume 2009. Lecture Notes in Computer Science. Berlin, Heidelberg, Germany: Springer-Verlag, 2001, pages 30–45.

Böhme, Rainer et al. [2005]. 'On the PET Workshop Panel "Mix Cascades Versus Peer-to-Peer: Is One Concept Superior?"' In: *Privacy Enhancing Technologies.* Edited by David Martin and Andrei Serjantov. Volume 3424. Lecture Notes in Computer Science. Berlin, Heidelberg, Germany: Springer-Verlag, 2005, pages 243–255.

Borisov, Nikita, Ian Goldberg and Eric Brewer [2004]. 'Off-the-record Communication, or, Why Not to Use PGP'. In: *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society.* WPES '04. Washington DC, USA: ACM, 2004, pages 77–84.

Böse, Tobias [2015]. *SilverTunnel-NG.* 21st Sep 2015. `http://silvertunnel-ng.sourceforge.net` (Retrieved: 07/12/2015).

Bray, T. [2014]. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 7159. Mar 2014. `http://www.rfc-editor.org/rfc/rfc7159.txt` (Retrieved: 14/11/2015).

Buchmann, Johannes A., Evangelos Karatsiolis and Alexander Wiesmaier [2013]. *Introduction to Public Key Infrastructures*. Berlin, Heidelberg, Germany: Springer-Verlag, 2013.

Buford, John F. and Heather Yu [2010]. 'Peer-to-Peer Networking and Applications: Synopsis and Research Directions'. In: *Handbook of Peer-to-Peer Networking*. Edited by Xuemin Shen et al. New York, NY, USA: Springer Science+Business Media, LLC, 2010, pages 3–45.

Buford, John F., Heather Yu and Eng Keong Lua [2009]. *P2P Networking and Applications*. Amsterdam, Netherlands: Elsevier B.V., 2009.

Castro, Miguel et al. [2002]. 'Secure Routing for Structured Peer-to-peer Overlay Networks'. In: *Proceedings of the 5th symposium on Operating systems design and implementation*. Volume 36. SIGOPS Operating Systems Review SI. New York, NY, USA: ACM, Dec 2002, pages 299–314.

Chaum, David L. [1981]. 'Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms'. In: *Communications of the ACM*. Volume 24. 2. New York, NY, USA: ACM, Feb 1981, pages 84–90.

Christin, Nicolas [2013]. 'Traveling the Silk Road: A Measurement Analysis of a Large Anonymous Online Marketplace'. In: *Proceedings of the 22nd International Conference on World Wide Web*. WWW '13. Rio de Janeiro, Brazil: International World Wide Web Conferences Steering Committee, 2013, pages 213–224.

Cisco Systems, Inc. [2015]. *The Zettabyte Era—Trends and Analysis*. May 2015. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/VNI_Hyperconnectivity_WP.html` (Retrieved: 11/01/2016).

Clarke, Ian et al. [2001]. 'Freenet: A Distributed Anonymous Information Storage and Retrieval System'. In: *Designing Privacy Enhancing Technologies*. Edited by Hannes Federrath. Volume 2009. Lecture Notes in Computer Science. Berlin, Heidelberg, Germany: Springer-Verlag, 2001.

Díaz, Claudia et al. [2003]. 'Towards Measuring Anonymity'. In: *Proceedings of the 2nd International Conference on Privacy Enhancing Technologies*. San Francisco, CA, USA: Springer-Verlag, 2003, pages 54–68.

Dierks, T. and C. Allen [1999]. *The TLS Protocol Version 1.0*. RFC 2246. Jan 1999. `http://www.rfc-editor.org/rfc/rfc2246.txt` (Retrieved: 15/09/2015).

Diffie, Whitfield [1988]. 'The First Ten Years of Public-Key Cryptography'. In: *Proceedings of the IEEE*. Volume 76. 5. May 1988, pages 560–577.

Diffie, Whitfield and Martin E. Hellman [1976]. ‘New directions in cryptography’. In: *IEEE Transactions on Information Theory.* Volume 22. 6. Nov 1976, pages 644–654.

Digital Austria, Federal Chancellery of Austria [2015]. *Mobile phone signature.* 2015. `https://www.bka.gv.at/site/6791/default.aspx` (Retrieved: 18/11/2015).

Dinger, Jochen and Hannes Hartenstein [2006]. ‘Defending the Sybil Attack in P2P Networks: Taxonomy, Challenges, and a Proposal for Self-Registration’. In: *Proceedings of the First International Conference on Availability, Reliability and Security.* ARES ’06. Washington, DC, USA: IEEE Computer Society, 2006, pages 756–763.

Dingledine, Roger [no date]. *The lifecycle of a new relay.* `https://blog.torproject.org/blog/lifecycle-of-a-new-relay` (Retrieved: 17/07/2015).

Dingledine, Roger, Nick Mathewson and Paul Syverson [2004]. ‘Tor: The Second-Generation Onion Router’. In: *Proceedings of the 13th USENIX Security Symposium.* San Diego, CA, USA, Aug 2004, no pages.

Douceur, John R. [2002]. ‘The Sybil Attack’. In: *Peer-to-Peer Systems.* Edited by Peter Druschel, Frans Kaashoek and Antony Rowstron. Volume 2429. Lecture Notes in Computer Science. Berlin, Heidelberg, Germany: Springer-Verlag, 2002, pages 251–260.

Dougherty, Steve [no date]. *Freenet Statistics.* Mirror of Freesite `freenet:USK@pxtehd-TmfJwyNUA W2Clk4pwv7Nshyg21NNfXcqzFv4,LTjcTWqvsq3ju6pMGe9Cqb3scvQgECG81hRdgj5WO4s,AQACAAE/statistics/998/`. `http://asksteved.com/stats/` (Retrieved: 15/10/2015).

Dropfile.to [no date]. *About.* `https://dropfile.to/about` (Retrieved: 27/11/2015).

Egger, Christoph et al. [2013]. ‘Practical Attacks against the I2P Network’. In: *Research in Attacks, Intrusions, and Defenses.* Edited by Salvatore J. Stolfo, Angelos Stavrou and Charles V. Wright. Volume 8145. Lecture Notes in Computer Science. Berlin, Heidelberg, Germany: Springer-Verlag, 2013, pages 432–451.

Fletcher, George H.L., Hardik A. Sheth and Katy Börner [2005]. ‘Unstructured Peer-to-Peer Networks: Topological Properties and Search Performance’. In: *Agents and Peer-to-Peer Computing.* Berlin, Heidelberg, Germany: Springer-Verlag, 2005, pages 14–27.

Ford, Bryan, Pyda Srisuresh and Dan Kegel [2005]. ‘Peer-to-peer Communication Across Network Address Translators’. In: *Proceedings of the Annual Conference on USENIX Annual Technical Conference.* USENIX Annual Technical Conference, General Track ’05. Berkeley, CA, USA: USENIX Association, 2005, pages 179–192.

Freedman, Michael J. and Robert Morris [2002]. ‘Tarzan: A Peer-to-peer Anonymizing Network Layer’. In: *Proceedings of the 9th ACM Conference on Computer and Communications Security.* CCS ’02. Washington, DC, USA: ACM, 2002, pages 193–206.

Garfinkel, Simson [1995]. *PGP: Pretty Good Privacy.* Sebastopol, CA, USA: O'Reilly & Associates, 1995.

GitHub, Inc. [2015]. *OAuth.* 2015. `https://developer.github.com/v3/oauth/` (Retrieved: 22/12/2015).

Goland, Yaron Y. and Jason Poon [2015]. *Tor_Onion_Proxy_Library.* 14th Feb 2015. `https://github.com/thaliproject/Tor_Onion_Proxy_Library` (Retrieved: 07/12/2015).

Goldberg, Ian and the OTR Development Team [no date]. *Off-the-Record Messaging Protocol version 3.* `https://otr.cypherpunks.ca/Protocol-v3-4.0.0.html` (Retrieved: 06/01/2016).

Goldschlag, David M., Michael G. Reed and Paul F. Syverson [1996]. 'Hiding Routing information'. In: *Information Hiding.* Edited by Ross Anderson. Volume 1174. Lecture Notes in Computer Science. Berlin, Heidelberg, Germany: Springer-Verlag, 1996, pages 137–150.

Housley, R. [1999]. *Cryptographic Message Syntax.* RFC 2630. Jan 1999. `http://www.rfc-editor.org/rfc/rfc2630.txt` (Retrieved: 31/10/2015).

Housley, R. et al. [1999]. *Internet X.509 Public Key Infrastructure Certificate and CRL Profile.* RFC 2459. Jan 1999. `http://www.rfc-editor.org/rfc/rfc2459.txt` (Retrieved: 31/10/2015).

International Organization for Standardization [2011]. *ISO/IEC 15408-2:2008 Information technology — Security techniques — Evaluation criteria for IT security — Part 2: Security functional components.* Technical report. Geneva, Switzerland, 1st Jun 2011.

International Organization for Standardization [2014]. *ISO/IEC 15408-1:2008 Information technology — Security techniques — Evaluation criteria for IT security — Part 1: Introduction and general model.* Technical report. Geneva, Switzerland, 15th Jan 2014.

International Telecommunication Union [2012]. *Recommendation ITU-T X.509 Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks.* Technical report. Geneva, Switzerland: ITU-T – Telecommunication Standardization Sector of ITU, 2012.

Karrer, Manfred [2015]. *Bitsquare – The decentralized bitcoin exchange.* 2015. `https://bitsquare.io` (Retrieved: 07/12/2015).

Karrer, Manfred and Bernd Prünster [2015]. *JTorProxy.* 17th Nov 2015. `https://github.com/ManfredKarrer/jtorproxy` (Retrieved: 07/12/2015).

Kashchenko, Alex [2015]. *OpenJDK unofficial installers for Windows, Linux and Mac OS X.* 7th Jul 2015. `https://github.com/alexkasko/openjdk-unofficial-builds` (Retrieved: 07/12/2015).

Kizza, Joseph Migga [2015]. ‘Authentication’. In: *Guide to Computer Network Security.* Computer Communications and Networks. London, GB: Springer-Verlag, 2015, pages 205–223.

Korzun, Dmitry and Andrei Gurtov [2013]. *Structured Peer-to-Peer Systems.* 1st edition. New York, NY, USA: Springer Science+Business Media, 2013.

Lee, Micah [no date]. *OnionShare.* `https://onionshare.org/` (Retrieved: 06/01/2016).

Lee, Micah [2015]. *Security Design Document.* 1st Aug 2015. `https://github.com/micahflee/onionshare/blob/master/SECURITY.md` (Retrieved: 06/01/2016).

Liu, Lu and Nick Antonopoulosu [2010]. ‘From Client-Server to P2P Networking’. In: *Handbook of Peer-to-Peer Networking.* Edited by Xuemin Shen et al. New York, NY, USA: Springer Science+Business Media, LLC, 2010, pages 71–89.

Mathewson, Nick and Steven Murdoch [no date]. *Top changes in Tor since the 2004 design paper (Part 2).* `https://blog.torproject.org/blog/top-changes-tor-2004-design-paper-part-2` (Retrieved: 17/07/2015).

Maymounkov, Petar and David Mazières [2002]. ‘Kademlia: A Peer-to-Peer Information System Based on the XOR Metric’. In: *Revised Papers from the First International Workshop on Peer-to-Peer Systems.* IPTPS ’01. London, UK: Springer-Verlag, 2002, pages 53–65.

McLachlan, Jon and Nicholas Hopper [2009]. ‘On the Risks of Serving Whenever You Surf: Vulnerabilities in Tor’s Blocking Resistance Design’. In: *Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society.* WPES ’09. Chicago, Illinois, USA: ACM, 2009, pages 31–40.

Modadugu, Nagendra and Eric Rescorla [2004]. ‘The Design and Implementation of Datagram TLS’. In: *Proceedings of ISOC NDSS.* 2004, no pages.

Nakamoto, Satoshi [2008]. *Bitcoin: A Peer-to-Peer Electronic Cash System.* 2008. `https://bitcoin.org/bitcoin.pdf` (Retrieved: 25/04/2015).

Oracle Corporation [no date]. *JavaFX FAQ.* `http://www.oracle.com/technetwork/java/javafx/overview/faq-1446554.html%5C#6` (Retrieved: 07/12/2015).

Orthacker, Clemens, Martin Centner and Christian Kittl [2010]. ‘Qualified Mobile Server Signature’. In: *Security and Privacy – Silver Linings in the Cloud.* Edited by Kai Rannenberg, Vijay Varadharajan and Christian Weber. Volume 330. IFIP Advances in Information and Communication Technology. Berlin, Heidelberg, Germany: Springer-Verlag, 2010, pages 103–111.

Pfitzmann, Andreas and Marit Köhntopp [2001]. ‘Anonymity, Unobservability, and Pseudonymity – A Proposal for Terminology’. In: *Designing Privacy Enhancing Technologies.* Edited by Hannes Federrath. Volume 2009. Lecture Notes in Computer Science. Berlin, Heidelberg, Germany: Springer-Verlag, 2001, pages 1–9.

Postel, Jon [1981]. *Internet Protocol.* RFC 791. Sep 1981. http://www.rfc-editor.org/rfc/rfc791.txt (Retrieved: 15/09/2015).

Ratnasamy, Sylvia et al. [2001]. 'A Scalable Content-addressable Network'. In: *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications.* SIGCOMM '01. San Diego, California, USA: ACM, 2001, pages 161–172.

Raymond, Eric S. [2001]. *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary.* Sebastopol, CA, USA: O'Reilly Media, Inc., Jan 2001.

Reimair, Florian, Peter Teufl and Thomas Zefferer [2015]. 'WebCrySIL – Web Cryptographic Service Interoperability Layer'. In: *11th International Conference on Web Information Systems and Technologies.* 2015, pages 35–44.

Reiter, Michael K. and Aviel D. Rubin [1998]. 'Crowds: Anonymity for Web Transactions'. In: *ACM Transactions on Information and System Security.* Volume 1. New York, NY, USA: ACM, Nov 1998, pages 66–92.

Rescorla, E. and N. Modadugu [2006]. *Datagram Transport Layer Security.* RFC 4347. Apr 2006. http://www.rfc-editor.org/rfc/rfc4347.txt (Retrieved: 15/09/2015).

Risson, John and Tim Moors [2006]. 'Survey of research towards robust peer-to-peer networks: Search methods'. In: *Computer Networks.* Volume 50. Amsterdam, Netherlands: Elsevier B.V., Dec 2006, pages 3485–3521.

Roos, Stefanie et al. [2014]. 'Measuring Freenet in the Wild: Censorship-Resilience under Observation'. In: *Privacy Enhancing Technologies.* Edited by Emiliano De Cristofaro and Steven J. Murdoch. Volume 8555. Lecture Notes in Computer Science. Cham, ZG, Switzerland: Springer International Publishing, 2014.

Schimmer, Lars, Treasurer of I2P [2015]. *On I2P.* Personal Interview. Institute of Computer Graphics and Knowledge Visualization, Graz University of Technology, 16th Oct 2015.

Secomba GmbH [no date(a)]. *Highest Security for your Files in the Cloud – Boxcryptor – Secure your Cloud.* https://www.boxcryptor.com/en (Retrieved: 06/01/2016).

Secomba GmbH [no date(b)]. *Technical Overview.* https://www.boxcryptor.com/en/technical-overview (Retrieved: 06/01/2016).

Shirey, R. [2000]. *Datagram Transport Layer Security.* RFC 2828. May 2000. http://www.rfc-editor.org/rfc/rfc2828.txt (Retrieved: 30/10/2015).

Stallings, William [2013]. *Cryptography and Network Security: Principles and Practice.* 6th edition. Upper Saddle River, NJ, USA: Pearson, 2013.

Stoica, Ion et al. [2001]. 'Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications'. In: *Proceedings of the 2001 Conference on Applications, Technologies, Architectures,* *and Protocols for Computer Communications.* SIGCOMM '01. San Diego, California, USA: ACM, 2001, pages 149–160.

Syverson, Paul et al. [2001]. 'Towards an Analysis of Onion Routing Security'. In: *Designing Privacy Enhancing Technologies.* Edited by Hannes Federrath. Volume 2009. Lecture Notes in Computer Science. Berlin, Heidelberg, Germany: Springer-Verlag, 2001, pages 96–114.

The I2P Project [no date(a)]. *I2P Compared to Tor.* `https://geti2p.net/en/comparison/tor` (Retrieved: 17/10/2015).

The I2P Project [no date(b)]. *Intro - I2P.* `https://geti2p.net/en/about/intro` (Retrieved: 16/10/2015).

The I2P Project [no date(c)]. *Three Month View for Total Routers.* Eepsite only reachable through I2P. `http://stats.i2p/cgi-bin/total_routers_3month.cgi` (Retrieved: 17/10/2015).

The OAuth Community [2015]. *OAuth Community Site.* 2015. `http://oauth.net/` (Retrieved: 18/11/2015).

The Tor Project, Inc. [no date(a)]. *Tor Metrics — Direct users by country.* `https://metrics.torproject.org/userstats-relay-country.html?graph=userstats-relay-country&start=2015-09-01&end=2015-09-30&country=all&events=off` (Retrieved: 01/10/2015).

The Tor Project, Inc. [no date(b)]. *Tor Metrics — Relays and bridges in the network.* `https://metrics.torproject.org/networksize.html?graph=networksize&start=2015-09-01&end=2015-09-30` (Retrieved: 01/10/2015).

The Tor Project, Inc. [no date(c)]. *Tor: Pluggable Transports.* `https://www.torproject.org/docs/pluggable-transports.html.en` (Retrieved: 09/10/2015).

The Tor Project, Inc. [no date(d)]. *Want Tor to Really Work?* `https://www.torproject.org/download/download-easy.html.en#warning` (Retrieved: 09/10/2015).

Timpanaro, Juan Pablo, Isabelle Chrisment and Olivier Festor [2012]. 'A Bird's Eye View on the I2P Anonymous File-Sharing Environment'. In: *Network and System Security.* Edited by Li Xu, Elisa Bertino and Yi Mu. Volume 7645. Lecture Notes in Computer Science. Berlin, Heidelberg, Germany: Springer-Verlag, 2012, pages 135–148.

Yu, Haifeng et al. [2006]. 'SybilGuard: Defending Against Sybil Attacks via Social Networks'. In: *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications.* SIGCOMM '06. Pisa, Italy: ACM, 2006, pages 267–278.