Anonymous Communication Networks

Minimizing key requirements of DC-Networks in presence of colluders

Project supervisor
Dr. Luke J. O'Connor

Academic supervisor
Prof. Refik Molva

Jean Lehmann

PROJECT REPORT SUBMITTED FOR THE PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE TITLE OF

Enterprise Communications Engineer

Institut Eurécom
2229, route des Crêtes
B.P. 193
06904 Sophia Antipolis Cedex

IBM Research Division
Zurich Research Laboratory
Säumerstrasse 4
CH-8803 Rüschlikon
Switzerland

December 1998

Contents

Chapter 1

Abstract

This report presents the existing proposals for anonymous communication systems that are designed to keep the recipient and sender or at least their relationship unobservable. Concepts necessary to the understanding of anonymous communication systems will be defined.

A special concern will be made in the study of the DC-Network. We explain how such a network is modeled in a graph theoretic framework, and how its security is related to the connectivity of the corresponding graph. Typically, a set of colluders could partition the graph into connected components and it is shown that unconditional sender anonymity is preserved among a given connected component.

We will look for algorithms to minimize the number of edges of the graph while maintaining full sender anonymity in presence of colluders. Two approaches to build such graphs will be proposed. The first is deterministic and is suitable in a LAN environment, whereas the second is statistic and might be required if the DC-Network is constituted by a dynamic collection of participants distributed across the Internet.

Chapter 2

Introduction

In present-day communication networks, the identity of senders and receivers can be revealed by the message's content as well as by the internal protocols used to transmit messages over the network. Indeed in practically all proposed or realized public communication networks, user stations can easily be identified at the physical, data link or network layer [11]. Therefore an attacker such as the network operator, or an intruder, could easily observe who sends how many messages to whom and at what time (traffic analysis), even if the content of a message can be sufficiently hidden by end-to-end encryption. With the increasing use of ISDN and the maturing of the Internet for private communications, loss of privacy becomes a severe threat and major parts of any user's life might easily be observed by the network operator or by an intruder. For example, while encrypting communication to and from web servers (e.g., using SSL [23]) can hide the content of the transaction from an eavesdropper (e.g., an Internet service provider or a local system administrator), the eavesdropper can still learn the IP address of the client and server machines, the length of the data being exchanged, and the time and frequency of exchanges [7].

There are some well-known measures allowing users acting alone to decrease their observability. To hide the information provided by traffic analysis, they can use for example in the case of services like telephony, public network stations (e.g. telephone boxes) instead of private ones. This will prevent observation but it may be inconvenient for the users. If they use private network stations, they can only try to hide their behavior by maintaining a constant traffic exiting from their station in order to simulate activity. This is an easy but expensive measure and not suitable for services where the bandwidth is expensive.

Thus, it appears that the only way to decrease user observability and foiling traffic analysis, seems to be to design a network for anonymity and not to try to realize anonymity afterwards [12].

Much work has been done in recent years in the field of untraceable communications. The MIX-Network [1] and the DC-Network [3] constitute the two main approaches to the untraceable communication problems.

A MIX is a logical component that acts as a remailer at the application layer, attempting to foil traffic analysis. Its main purpose is to hide the relationship between incoming and outgoing message traffic, by decorrelating in time and in content inputs and outputs. The MIX-network is relatively efficient but leaves open a certain number of loopholes: it is not completely secure against a powerful and determined active attacker who can master the line at its entrance, namely the isolate-and-identify attack. However, traffic analysis can be made more difficult by using a chain of MIXes instead of a single MIX. The kind of anonymity provided by a MIX is called "Unlikability of Sender and Recipient": senders and receivers can be identified as acting in some communication but cannot be identified as communicating with each other The MIX-Network also introduces the notion of untraceable return addresses that allow a recipient to reply to an anonymous message without knowing the identity of the original sender.

On the other hand, the DC-Network offers to its members unconditional sender anonymity and recipient untraceability. This means that no matter how much computational power may have an observer, the sender of a message cannot be identified. Sender anonymity is provided through the use of the superposed sending technique, and recipient anonymity, because all messages are broadcast. Indeed, receiving a message can be made anonymous to the the network, by delivering the message to all stations. Although theoretically attractive, the DC-Network presents certain weaknesses. First, the anonymity of a sender can be compromised by other participants of the networks forming collusions, second participants can be prevented from sending (disruption attack), and third, sender and receiver anonymity are guaranteed, but on the assumption of a reliable broadcast network. Solutions for the two latter are investigated in [3, 18, 20, 19], and a solution to the collusion problem will be proposed in this report. We also mention that the DC-Network is quite wasteful in its use of bandwidth.

Most of the existing variants to the untraceable communication problems that will be presented are based on MIXes.

ISDN MIXes: In [16], mixes are used to provide untraceable communication in an ISDN network. In an ISDN environment, each subscriber line is assigned to a particular local exchange, and local exchanges are interconnected by a long distance network. Anonymous calls in ISDN rely on an anonymous connection between members of two local exchanges which is obtained by routing calls through a predefined series of mixes, and matching two connections established one by the sender and the other by the recipient. Before data being exchanged sender and recipient establishes a connection each to a series of MIX by distributing keys that will be used for data exchange in order to prevent content correlation. Each connection is assigned a label, and during the establishment phase, two connections with the same label are linked to provide a connection from sender to recipient. The protocol uses a time-slice technique to maintain a constant traffic, dummy or not, in the network. This approach relies on the fact that in an ISDN, there is no real cost associated with keeping an ISDN line active permanently, either by making calls to itself, or to other users. It thus complicates traffic analysis by maintaining a constant traffic and foiling tracking of coincidences. In anonymous ISDN, the mixes hide communications within the local exchange, but communications between local exchanges is not hidden.

Onion Routing: In [8], the notion of anonymous socket connection is introduced, and is somewhat similar to an ISDN anonymous connection. It has the advantage to propose a connection which is application independent, whereas MIXes act in general at the application layer. The application at the client side, instead of making a connection to a responding server, will make a connection to an onion routing proxy that will set an anonymous connection through several onion routers by building a layered data structure called an onion and sending it through the onion routing network. Each onion router will establish a connection in a similar way as ATM networks do, by assigning Anonymous Connection Identifiers. It also knows some cryptographic information that will allow him to encrypt or decrypt the data flow, depending on its direction, in order to prevent content correlation. Thus data passing along the anonymous connection will appear different at and to each onion router. As in MIXes, each onion router only knows the adjacent routers, but only the first onion router in the path (the onion routing proxy knows the entire path).

Due to the volume attack, whereby an onion router counts the packets it delivers per ACI, any two collaborating onion routers can correlate all the traffic between them. It is also the case that for low traffic situations an eavesdropper can easily correlate incoming an outgoing traffic through an onion router. Moreover, the initiator's proxy must be explicitly trusted. In many cases, the initiator's proxy, which is located in the enterprise firewall is the entity which is the least trusted by the users seeking anonymity.

Crowds: In [7], a very simple scheme to provide untraceability to participants willing to browse a web server is presented. The model is constituted of a set of n participants $\{P_1, ..., P_n\}$ called the crowd, and a web server S. The original sender sends his request to a random member of the crowd. The latter flips a biased coin and either forwards the request to a random member of the crowd, or forwards the request to the end server. Choices are made independently. This mechanism can be seen as a sender anonymity based scheme since, with respect to a local observer or to the server, every member of the crowd appears as likely as any other member to be the originator. However, crowds ignores the problem of correlation in time between incoming and outgoing messages. Hence it is quite vulnerable to eavesdropping. It follows that in order to make eavesdropping hard, participants must be far apart or at least in different domains. This has the adverse effect of increasing latency, regardless of the load of participants.

This report attempts to make an overview of the existing proposals for anonymous networks that are designed to keep the recipient and sender, or at least their relationship, unobservable. Concepts necessary to the understanding of anonymous communications will be defined. A special concern will be made in the study of the DC-Network. We will see that such a network can be modeled as a key graph whose security in presence of a given number of attackers, namely colluders, directly depends on the connectivity of the corresponding graph. The key requirements of the DC-network in presence of colluders will be optimized. Two approaches to build such networks are proposed. The first is deterministic and is suitable in the context of a LAN environment, whereas the second is probabilistic and might be required if the DC-Network is constituted with a dynamic collection of participants distributed across the Internet.

Chapter 3 presents the basic concepts and definitions of anonymity that we use throughout this report. It aims to provide formal definitions of anonymity. Chapter 4 to 7 present the principal existing proposals for anonymous networks. Chapter 8 explains the basic concepts of graph theory needed to understand the model of DC-Networks and corresponding security. The remaining chapters are devoted to the study of the optimization of such networks in presence of colluders.

Chapter 3

Basic concepts for anonymous networks

Most of the definitions of anonymity appear to be vague. As in [12], three types of anonymity are derived: sender anonymity, recipient anonymity and unlinkability of sender and recipient. Sender and recipient anonymity refer to the strong property to keep the sender and the recipient of a message secret with respect to a given attacker model. A weaker possibility resides in the unlinkability of sender and recipient: one keeps their relationship secret, i.e. sending and receiving of physical messages is observable (sender and receiver can each be identified as participating in some communication), but it is infeasible for an attacker to link the physical message sent by the sender and the physical message received by the recipient. We will use the same terms while trying to define them in the context of a distributed system.

For the sake of understanding, it might be useful to imagine an anonymous communication network as a layered structure in which primitives (or events) such as Send() and Receive() may occur at the user interface. Such primitives may trigger the exchange of physical messages at a lower interface that can be intuitively called the physical interface, since we assume those messages to be visible by any attacker able to tap the lines at that level. In this context, several definitions will be derived. We have to be aware of the fact that those definitions are relative to some attacker's model. We define two attackers model: the first one is an eavesdropper able to tap one or more points in the network. He can record, remove, alter or inject packets at the physical interface. The second one is a collusion of participants who gather some distributed information in order to compromise the anonymity of an other participant. According to these possible attackers, we derive the following definitions of anonymity.

Definition 3.0.1 (Unobservability) An event in a system, e.g. the sending of a message, is called unobservable by a given attacker, if the attacker does not even learn that the event occurred. More precisely, the party should not gain any additional information or knowledge, e.g., it should not be able to guess better than with a priori probability, for instance this party should not be able to guess better who sent a message by observing the physical messages exchanged.

Definition 3.0.2 (Sender anonymity) Sender anonymity means that the Send() primitive corresponding to the sending of a message, occuring at the user interface is unobservable by a given attacker, i.e. an eavesdropper or a collusion. The latter does not even learn that the event occured. We thus keep the sender of a message secret. This allows us to derive a formal definition of sender anonymity. We consider a collection of n participants, $\{S_1, ..., S_n\}$, whose sending of a message is a priori not visible at the user interface. Such a group forms an anonymity set. This is described by a stochastic variable $S \in (\mathbb{Z}/2\mathbb{Z})^n$, where $S_i = 0$ if participant S_i is not sending an actual message and $S_i = 1$ otherwise. On the other hand, we assume that all participants output a message at the physical interface. This is described by a stochastic variable $O \in \mathbb{F}^n$ where $\mathbb{F}$ is a finite Abelian group. Then sender anonymity means:

$$P(S|O) = P(S) \tag{3.1}$$

Knowing the messages that are sent at the physical interface doesn't provide any information about the identity of the actual senders. Every member of the anonymity set appears as likely as any other member to be an actual sender.

Definition 3.0.3 (Recipient anonymity) Recipient anonymity means that the Receive() primitive corresponding to the receiving of a message, occuring at the user interface is unobservable by a given attacker. The latter does not even learn that the event occured. We thus keep the recipient of a message secret. This allows us to derive a formal definition of recipient anonymity. We consider a collection of m participants, $\{R_1, ..., R_m\}$, whose receiving of a message is not visible at the user interface. Such a group forms an anonymity set. This is described by a stochastic variable $R \in (\mathbb{Z}/2\mathbb{Z})^n$, where $R_i = 0$ if participant R_i is not receiving an actual message and $S_i = 1$ otherwise. On the other hand, we assume that all participants receive a message at the physical interface. This is described by a stochastic variable $O \in \mathbb{F}^m$, where $\mathbb{F}$ is a finite Abelian group. Then receiver anonymity means:

$$P(R|O) = P(R) \tag{3.2}$$

Knowing the messages that are received at the physical interface doesn't provide any information about the identity of the actual receivers. Every member of the anonymity set appears as likely as any other member to be an actual recipient or not.

Definition 3.0.4 (Unlinkability of sender and recipient) Unlinkability of sender and receiver means that though the sender and receiver can each be identified as participating in some communication (Send() and Receive() of a same message might be observable by an eavesdropper), they cannot be identified as communicating with each other, i.e. the attacker cannot link the primitive Receive() that resulted from its corresponding Send() primitive.

In [15], the following definition of unlinkability is given: two roles, say R_1 and R_2, in two events, E_1 and E_2 are called unlinkable for a certain party if that party does not learn whether the same user played the two roles. For instance, buyers are called unlikable in payments if other parties do not learn whether two payments were made by the same person. Unlinkability between different classes of events may lead to a classification of pseudonyms.

Chapter 4

Recipient anonymity

Recipient anonymity can be provided by sending a message to a group of users containing the intended recipient. In this case, sending of a message will be observable, but with respect to an observer, any member of the group of users will be as likely as any other member to be the intended recipient. However, this anonymity is not unconditional or information theoretic-, since the message has to contain an attribute by which the intended recipient and nobody else can recognize the message as addressed to him. This attribute is called an implicit address. Such an address might be a pseudonym of the receiver, or a code prepend to the message, on which sender and receivers could have previously agreed on. If the same pseudonym or code is used for different communications with different groups of receivers (or anonymity sets), each containing the intended recipient, an observer could finally have good chances to guess the identity of the recipient by intersecting the anonymity sets. This attribute can also be the public key of the intended recipient. The message would be encrypted with the corresponding key, which would give the possibility for a powerful attacker to determine which public key is used for encryption and thus finding the identity of the receiver. The public key can be seen as an invisible implicit address, whereas a pseudonym is a visible implicit address. This will firstly lead us to define a classification of addresses. Secondly, a switched broadcast network structure will be presented as an illustration of this technique. Moreover, it will introduce the notion of untraceable return addresses that will allow the receiver of an anonymous message to reply to the sender who will remain anonymous.

4.1 Broadcasting

As mentioned earlier, receiving a message can be made anonymous to the network by delivering the message to all stations (broadcast). In the context of the Internet we may consider multicasting a message to a group of users. Since the message is delivered to N members of the network, every member appears as likely as any other member to be the intended recipient. If the public key is used to encrypt the message, it can be seen as a special address which is used by the sender to convey messages to their intended recipient through the network. This will firstly lead us to define a classification of addresses.

Using broadcast to provide recipient untraceability was first introduced by A. Pfitzmann in [11]. This switching broadcast network structure will be presented in the next section. It allows two participants belonging to two local two-way broadcast networks interconnected by a switching center, to communicate anonymously. Recipient anonymity is provided because the original message is broadcast, and the use of untraceable return addresses allows the receiver of a message to reply to the sender who remains anonymous. The second technique using broadcasting to provide recipient untraceability is the DC Network [3]. However, the main purpose of the DC Network is to ensure sender anonymity through superposed sending (cf. §10.3).

If a message has an intended recipient, it has to contain an attribute by which he and nobody else can recognize the message as addressed to him [12, 13]. This attribute is called an implicit address. In contrast, an explicit address describes a place in the network. We can distinguish two types of implicit addresses according to their visibility. An implicit address is called invisible, if it is only visible to its recipient and is called visible otherwise [14].

Invisible implicit addresses can be realized with a public key cryptosystem. A message is addressed by encrypting it (or a part of it) with the public key of the recipient and then broadcast. Each station decrypts the messages it receives with its private key and uses the messages redundancy to decide which messages are addressed to it.

Visible implicit addresses can be realized much easier: Users choose arbitrary names for themselves, which can then be prefixed to messages. This can be referred to as pseudonymity. An example of the use of pseudonyms for communicating anonymously will be presented in the next section.

Another criterion to distinguish implicit addresses is their distribution. An implicit address is called public, if it is known to every user (like telephone number today) and private if the sender received it secretly from the recipient either outside the network or as a return address. We will see in the next section examples of the use of untraceable return addresses.

4.2 A switched broadcast network to decrease user observability

4.2.1 Presentation

A switching/broadcast network structure (SBNS) is presented, enabling users to decrease their observability. It gives an example of the use of broadcasting to provide recipient anonymity as well as the use untraceable return addresses. The following concepts are derived from [11].

The SBNS proposal is physically based on cheap and powerful microelectronics (e.g. personal computer) and on the enormous bandwidth and inherent broadcast facility of local networks and logically based on the generation of random numbers and keys of a public key cryptosystem. The backbone of the SBNS proposal is a conventional circuit or packet switched ISDN. The terminals of the switched ISDN are gateways. Each gateway routes to the switching center, the messages from the local two-way broadcast network which connects the users stations of a user group. Implementations of local two-way broadcast networks are discussed and protocols derived, which together can partially hide the sender and receiver of a message and enable the generation of untraceable return addresses.

We use the following notation:

1. k_e denotes a public key used for encipherment (and for checking signatures) in a public key cryptosystem [21].
2. k_d denotes a private key used for decipherment (and signature) in some public key cryptosystem. k_d is sometimes called secret key instead of private key.
3. $k_e(M)$ denotes a message M encrypted with the key k_e. M is augmented with a random bit string of appropriate length to reduce the probability to guess M and verify the guess by enciphering M with the publicly known public key k_e. The notation will hide the random bit string.
4. $k_d(k_e(M)) = M$ is the property needed, and for some public key cryptosystems we may also have the property $k_e(k_d(M)) = M$.

4.2.2 The physical structure

As mentioned earlier, the backbone of the SBNS proposal is a conventional circuit or packet switched ISDN. The terminals of the switched ISDN are gateways. Each gateway masters a local two-way broadcast network. Each two-way broadcast network connects the user stations of a user group. See figure 4.1.

We assume that an attacker can only tap one single point in the network. Taking this assumption into account, it should be impossible to identify the sender or receiver of a message in the local two-way broadcast networks by access to the local network at any single point. If the local two-way broadcast network is implemented as a bus, the user stations at the ends can be identified as senders by tapping the bus between them and the rest of the user stations and monitoring the direction of signal flow. If the local two-way broadcast network is implemented as a loop and every sender removes the messages he sent after one trip around the loop and a random access protocol is used, there are no possibilities to determine the sender or receiver of a message, assuming that an attacker can only access the network at a single point [11].

4.2.3 Organization and Generation of addresses

The addresses of user stations in a SBNS are composed of two parts, which together are enciphered with a fixed public key of the switching center. The first part is a logical address of a gateway. A gateway may have several different but fixed logical addresses. The logical addresses of a gateway are only known to the switching-center and to the user stations connected with the gateway by the two-way broadcast network. The second part is a pseudonym of a user station. Each user station may have several different pseudonyms.

For every address in a SBNS, there exists a corresponding public key k_e. All messages to an address will be enciphered with the corresponding public key k_e. The receiver holding the address knows the private key k_d corresponding to k_e and deciphers all messages with the appropriate private key k_d. One address with the corresponding public key of every user is published in a roster of public addresses of user stations.

Instead of a public address a user station may create private addresses with corresponding public keys, which the user may pass to the desired communication partner. Private

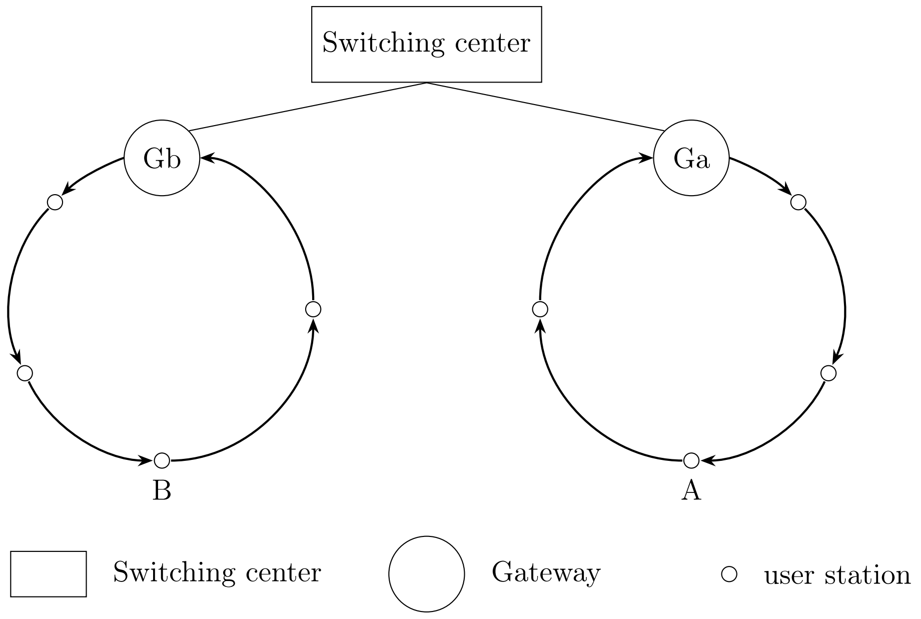

Figure 4.1: The physical structure of the SBNS (Figure 4.1)

addresses with corresponding public key that are used only once, are called untraceable return addresses [11].

4.2.4 The logical operation

The user A wants to send a request Q_q to user B without revealing his identity to this user but still giving him the possibility to reply to A. For the sake of understanding, we will describe the mechanism with only one switching center and with two local two-way broadcast networks that communicate.

A knows a logical address g_a of its gateway G_A and the public key k_s of the switching center. A also knows a public address b of user B and the corresponding public key B_{k_e}. A generates a random number Q_n that is used as a pseudonym to compute his private address $Q_p = k_s(g_a, Q_n)$. A also generates a public key Q_{k_e} and a corresponding private key Q_{k_d}. (Q_p, Q_{k_d}) forms an untraceable return address which will be used by the recipient B to reply to user A without A's identity being revealed.

The logical operations follow the following steps:

1. A sends to the gateway G_A, $(b, B_{k_e}(Q_P, Q_{k_e}, Q_q))$ where (Q_P, Q_{k_e}) is the untraceable return address and Q_q is the request. Inside the local two-way broadcast network, as it is mentioned earlier, one shouldn't be able by tapping at a single point of the network who is sending to whom. Indeed such an attack couldn't inform the eavesdropper or any other member of the local two-way broadcast network that someone from this local network is sending a message to B, since the address b of B is encrypted with the switching center's public key. Also, the attacker could make an exhaustive search on the public roster in order to compromise B's identity. This is defeated by augmenting the encryption of the address with a random bit string of appropriate length.

2. The gateway G_A sends to the switching center the message $(b, B_{k_e}(Q_p, Q_{k_e}, Q_q))$.

Similarly the address b of B is encrypted by the switching center's public key, and thus an eavesdropper taping the line at this single point would be prevented from knowing who is the intended recipient of the message.

3. On receiving the message $(b, B_{k_e}(Q_p, Q_{k_e}, Q_q))$, the switching center deciphers the address b of B and splits it into the address g_b of the gateway G_B of B and a pseudonym of B. The switching center uses g_b to send the message to the gateway G_B of B. It also knows that someone belonging to the broadcast network whose gateway is G_A is sending to B.

4. The gateway G_B receives the message and broadcasts it. It doesn't know who is the intended receiver of the message. B recognizes that by decrypting the message and checking its message redundancy that he is the intended recipient. Receiver anonymity is preserved since the message is broadcast. B computes Q_p, Q_k Q_q and retrieves the request Q_q. B generates then his response Q_a and enciphers it with Q_k, so that nobody but A will see the content of the response. Also, since Q_P is encrypted with the switching center's public k_s, B doesn't see the address of A and doesn't know to whom he answers.

5. B sends to the gateway G_B his response encrypted with the public key generated by A together with the untraceable return address: Q_p Q_k (a).

6. The gateway G_B relays the message to the switching center that can decrypt the untraceable return address $Q_P = k_e(g_a, Q_n)$ to retrieve (g_a, Q_n). The switching center doesn't know who is the intended recipient of the message and can only make the relation between G_A and G_B. The switching center uses g_a to send the message to the gateway G_A of A.

7. The gateway G_A of A broadcasts the received message. A recognizes its private address Q_P and deciphers the answer with the corresponding private key Q_{k_d}.

If it seems appropriate, B may also have generated an untraceable return address to allow A to reply to his message. Hence an anonymous communication where both parties don't know each others identities can take place.

4.2.5 Privacy provided

Within a local network, we assume that sender anonymity is provided with respect to an attacker able to tap only one point in the network. Indeed, if an anonymous medium access protocol, such as ALOHA [22], is used, the sending of a user station is only observable if its two neighbor stations collude or the lines are tapped [10]. Also sender and recipient anonymity are provided with respect to the gateways. We notice that the switching center is maybe the more sensitive entity in the network since he is able to make the link between the group G_A (i.e. a user belonging to the local network whose gateway is G_A) and the user B since he receives $b = k_e(g_b, pseudo(B))$, unless the pseudonym is only recognizable by B (visible implicit address). In this case, the switching center is only able to link a communication between G_A and G_B.

The interesting idea is here that B can respond to A but doesn't need to know A's identity. It relies on kind of a third party which is the switching center. We have an anonymous communication between A and B passing by an intermediary that only knows the relation between groups. As mentioned in chapter 3, such a group is an anonymity set, i.e. a group of potential receivers or senders.

We finally notice that this mechanism doesn't provide protection against an eavesdropper who can monitor every line of the network: content correlation is possible and one can know that a user of group G_A is communicating with a user of group G_B without knowing one's identity, by tapping inputs and outputs at the switching center. Since we assume that the two-way local networks are physically protected from outside, the switching center will only see the relation between groups of users but not between individuals. This mechanism is suitable if the attacker can only tap one single point in the network.

Chapter 5

Unlinkability of sender and recipient

This form of anonymity can be realized by a MIX, which is an entity that collects a batch of messages from many distinct senders, discards repeats, changes the encodings of messages and forwards them to the recipients in a different order [1]. The MIX can be seen as a logical component that acts at the application layer as a remailer. It forwards e-mail messages and-in the process- decorrelates the relationship between incoming and outgoing message traffic. It also gives the possibility to the receiver of a message to reply to the sender, who remains anonymous. We will describe the basic concepts of MIXes as well as the possible attacks that can be performed. Those are adapted from [1, 6].

5.1 MIX basic principles

The following notation will be used:

- M: message sent by the sender S
- $E_X(M)$: encryption of M using X's public key
- $D(M)$: decryption of M using X's private key
- $K(M)$: conventional encryption of M with key K
- (M_1, M_2): concatenation of M_1 and M_2
- $\mathcal{A}_X$: X's network address
- $\lceil M \rceil^{\Omega}$: padding string M to length Ω (by appending random bits)

As mentioned earlier, the MIX is an entity that gathers a batch of messages from distinct senders and forwards them to their intended recipients while attempting to hide the relationship between incoming and outgoing message traffic, see figure 5.1. We assume the existence of a powerful eavesdropper able to record, remove or alter messages entering or leaving the MIX. The eavesdropper is also able to generate spurious messages entering the MIX.

There are two ways for the eavesdropper to correlate incoming and outgoing messages: first by content, i.e. message data or message length, second by time, indeed if the messages leave the MIX in the same order they entered it, it will be easy for the eavesdropper to correlate inputs and outputs even if contents don't give any information between incoming and outgoing messages. In general, content correlation can be addressed by using standard cryptographic techniques along with padding. Time correlation can be countered if the incoming traffic volume is sufficiently high. We describe in the next section the behavior of a MIX according to the passive attacks described above. Active attacks will also be investigated.

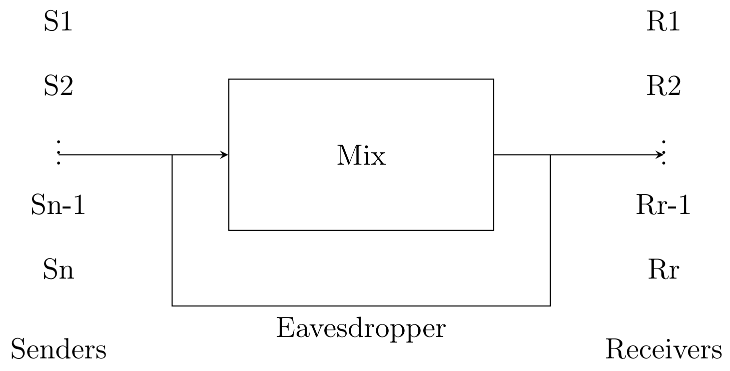

Figure 5.1: a MIX

5.2 Passive attacks

These attacks (content correlation and time correlation) can be carried out by merely observing the incoming and outgoing traffic.

5.2.1 Content correlation

Two elements can help in content correlation: actual content and length. For prevention of correlating incoming and outgoing messages by comparing the contents, it suffices to encrypt inputs so that outputs will always be different from inputs. It is also important to prevent the eavesdropper from being able to derive an input message from an output. For prevention of correlating incoming and outgoing messages by comparing their lengths, it suffices that all messages to/from a MIX be of uniform length. We denote this length by Ω.

Let's assume that a sender S wants to send a message M to a receiver R by passing through a MIX. He encrypts his message M and the destination address $\mathcal{A}_R$ with the MIX's public key, thus sending the input $I = E_{mix}(\mathcal{A}_R, M)$ to the MIX, where $\mathcal{A}_R$ is the receiver's network address. Upon reception and successful decryption of I by the MIX, the string $(\mathcal{A}_R, M)$ is revealed and the output message O consisting of message M in clear text is forwarded to its intended recipient R. The eavesdropper may attempt to correlate O and I by encrypting the string $(\mathcal{A}_R, M)$ with MIX's public key and comparing the result to I to check if it matches. For prevention, a random bit string must factored into the encryption to ensure that successive encryptions of the same message yield different results. In hybrid systems that combine the use of public key and conventional key encryption, the random bit string may be useless. Such systems use a random session key to encrypt user data with a symmetric key algorithm and a public key encryption algorithm to encrypt the random session key. Each encryption with a public key uses a different session key, which is revealed only to the owner of the private key. In this case, the sender S sends to the MIX: $(E_{mix}(K), K(\mathcal{A}_R, M))$. Thus, the eavesdropper is unable to correlate I and O even though he is able to re-encrypt $(\mathcal{A}_R, M)$. The reader should refer to [17] for cryptographic attacks on straight-RSA implementation of MIXes.

In order to avoid size correlation, message sizes must be constant throughout the entire MIX network. Message size uniformity can be achieved by padding to a constant length (Ω) with random data. The reader should refer to [6] for a detailed discussion on padding.

We can notice that the security of the system relies on the integrity of the MIX which acts as kind of a third party. Indeed, in a single-MIX architecture, if the MIX is somehow forced to reveal its private key, the unlinkability of users will be compromised. Instead of a single MIX, a chain of MIXes can be used to increase the security of the whole system. This will be discussed below.

5.2.2 Time correlation

If the messages processed by the MIX leave the MIX in the same order they entered it, it will be obvious for the eavesdropper to correlate incoming and outgoing message traffic and thus compromising the unlikability of senders and recipients. One simple solution is to output messages in batches, as explained in [1]. In this scheme, at least N input messages are accumulated during a time T before being forwarded in random order. N is called the size of the batch and we assume N and T to be public. We refer to this scheme as regular batching.

Under low incoming traffic, a batch of size N couldn't be formed within a reasonable time T. One possibility could be to decrease the size of the batch, but it would increase for an eavesdropper the chances of guessing the good permutation between incoming and outgoing messages. An other possibility could be to increase the parameter T, but as a direct consequence, the delay of messages would dramatically increase.

Sending out random-looking fake messages to random destinations may be a reasonable solution. Fake messages are indistinguishable from normal messages except that they are immediately discarded by their recipients after decryption. In this scheme, after each period of time T, the MIX performs the following procedure: it collects n messages during period T, if $n \geq N$ then regular batching is performed, otherwise, the MIX creates $N - n$ fake messages and forwards the batch of size N. This approach is similar to broadcasting a message to a collection of participants that contains only one intended recipient. Indeed, if for instance only one message is collected by the MIX during period T, with respect to the eavesdropper, every receiver (fake or not) will be as likely as any other receiver to be the intended recipient of the original message.

5.3 Active attacks

In this section we discuss active attacks, i.e. those involving direct modifications to message flow, by altering or inserting messages.

5.3.1 Isolate and Identify

We assume that the eavesdropper controls the line at the entrance of the MIX, thus he is able to prevent all messages but 1 to be delivered to the MIX during period T. In the same time, the eavesdropper may submit $N-1$ messages to the MIX. Upon arrival of a genuine message, the entire batch is forwarded and the eavesdropper can simply pick out the message she did not generate [12]. Although the genuine message may be encrypted, the eavesdropper is able to correlate the genuine message with its corresponding output, thus unlinkability of sender and recipient is compromised. This attack is difficult to prevent. However, it assumes that the eavesdropper completely controls the input line and may discard incoming messages, which is a strong assumption.

5.3.2 Message Replay

The eavesdropper can try to defeat a MIX by recording a genuine message and reinserting it later into the message stream. As an incoming message I results in the same output O when replayed, associating the two is trivial [6]. It is possible to prevent replays by keeping track of incoming messages and discarding replays [1]. However it might be undesirable to keep track of all the messages that have been submitted so far. A simple solution is to time-stamp messages and ignore message entries after some fixed system-wide time interval. Note that a similar mechanism for time-stamping coins in the e-cash payment system [4, 5] is used.

5.4 Forward path

We now assume that messages are sent through a chain of MIXes from senders to receivers. As mentioned earlier, if only one single MIX is used, that MIX acts as kind of trusted party that knows who is communicating with whom. To distribute this information, a sender may use a chain of MIXes to forward his messages [1], see figure. The system thus becomes more secure, since a MIX will only know the relationship between its two adjacent MIXes and the task of the eavesdropper will become much more difficult. Indeed, in order to link sent and received messages, the eavesdropper has to defeat all MIXes of the chain.

We describe the process of generating anonymous messages and the way they are processed by MIXes. Most of the material presented below is adapted from [1, 6, 3, 2]. We assume for the sake of clarity, that cryptographic operations (encrypting, decrypting) results in no effect on message length. The reader should refer to [6] for the description on how the message length can be kept constant.

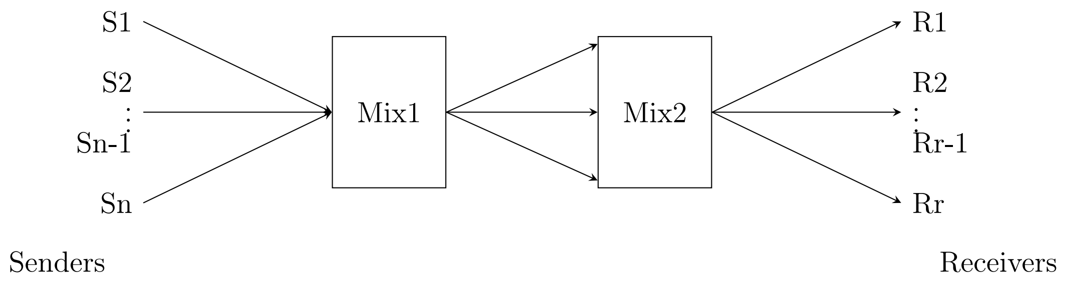

Figure 5.2: Chaining MIXes

5.4.1 Composition by sender

We assume that a sender S_i wants to send a message M to a receiver R_j through k MIXes, $M_1, ..., M_k$. This set of MIXes is referred to as the forward path as in [6]. The message that will be sent to the first MIX on the path is composed according to the following mechanism:

1. We assume that the message is padded to the length Ω and that encryption and decryption results in no effect on the size of messages. Let us denote $\lceil M \rceil^\Omega$ the padded message.
2. The padded message is then encrypted for every MIX, starting with the last MIX M_k according to the following recursion:

$$x_1 = E_{M_k}(\mathcal{A}_R, [M]^\Omega) \tag{5.1}$$

$$x_i = E_{M_{k-i+1}}(\mathcal{A}_{M_{k-i+2}}, x_{i-1}), 1 < i \leq k \tag{5.2}$$

$$x_k = E_{M_1}(\mathcal{A}_{M_2}, E_{M_2}(...E_{M_{k-1}}(\mathcal{A}_{M_k}, E_{M_k}(\mathcal{A}_R, [M]^\Omega))...)) \tag{5.3}$$

   E_{M_i} denotes the public key encryption with MIX M_i public key. The result is analogous to an onion where each layer of encryption represents a layer of skin. To access inner layers, outer layers must be stripped off first.

Once the onion is computed, it is sent to the first MIX on the forward path. We may notice that all of these operations are performed by the sender which will be the only entity to know the exact path of the message he will send.

5.4.2 Processing by MIXes

The first MIX M_1 receives from the sender the onion:

$$x_k = E_{M_1}(\mathcal{A}_{M_2}, E_{M_2}(...E_{M_{k-1}}(\mathcal{A}_{M_k}, E_{M_k}(\mathcal{A}_R, [M]^\Omega)), ...)) \tag{5.4}$$

Upon reception of this message, the MIX strips off the first layer of encryption using its private key D_{M_1} and discovers the address $\mathcal{A}_{M_2}$ of the next hop to which he will send the result of the decryption. Similarly, each MIX on the forward path removes one layer of encryption and forwards the result to the next hop. The last MIX on the forward path discovers the padded message (that might be encrypted to ensure confidentiality) as well as the address of the receiver and forwards the message to the receiver. Every MIX on the path only knows its predecessor and its successor, but doesn't know the entire path since each address is protected by an encryption layer of the previous hop. Similarly the receiver knows that the actual message was delivered by the MIX M_k but doesn't know the identity of the previous MIXes nor that of the sender. We may notice that the MIXes could form collusion to compromise the unlikability of sender and recipient, but it is sufficient to have only one honest MIX on the path to prevent other MIXes from knowing the relation between sender and receiver. We may also notice that one could prevent a MIX from knowing the identity of the next hop on the path. An alternative would be that instead of sending the onion to a particular MIX, the onion would be encrypted with the next hop's public key and broadcast to all MIXes.

5.5 Return path

It might be sometimes desirable for the recipient of an anonymous message to be able to reply to that message, without the identity of the original sender being revealed. This can be achieved by giving the possibility to include a Return Path Information (RPI) [6] in the anonymous message. This Return Path Information can be seen as an Untraceable Return Address [1, 2, 11].

5.5.1 Creating the RPI

The RPI is composed by the original sender according to the following procedure:

1. The original sender chooses MIXes $N_1, N_2, ..., N_l$ for the return path and MIXes $M_1, M_2, ..., M_k$ for the forward path. Those two chains are completely independent.
2. The sender randomly chooses a key seed KS, and computes l keys $K_1, K_2, ..., K_l$. For instance we can have, $K_i = E(KS, i)$ with $1 \leq i \leq l$. These keys will be used by the MIXes on the return path to encrypt the reply.
3. The key seed together with the number of hops l is first encrypted with the original sender's public key to form $y_0 = E_S(KS, l)$.
4. Then, once for every MIX on the return path, starting with the last, N_l, the following encryption is performed:

$$y_i = (\mathcal{A}_{N_{l-i+1}}, E_{N_{l-i+1}}(K_{l-i+1}, y_{i-1})) \tag{5.5}$$

   for $1 \leq i \leq l$. The final outcome is the RPI:

$$y_r = \mathcal{A}_{N_1}, E_{N_1}(K_1, \mathcal{A}_{N_2}, E_{N_2}(K_2, ..., E_{N_l}(K_r, \mathcal{A}_S, E_S(KS, l))...)) \tag{5.6}$$

5. The original sender inserts the RPI into the beginning of the clear text message he wishes to send. Then the message together with the RPI is sent through the forward path according to the procedure we described. The last MIX on the forward path will detect the RPI and forwards it to the receiver. This later will not be able by observing the RPI to derive the identity of the original sender to whom he will reply nor the path the reply will follow, since it is hidden by multiple layers of encryption.

The original receiver only knows the identity of the first MIX N_1 on the return path.

The original receiver will compose his reply and simply prepends the RPI he received from the original sender. He then sends this message to the first MIX on the return path,

R_1. Upon receiving the reply, R_1 detects the included RPI and extracts it. Let us denote this original RPI by RPI_0 whose size is assumed to be ω. R_1 then performs the following steps:

1. Pad the message M' that constitutes the reply, to the size $\Omega - \omega$
2. Decrypt RPI_0 to reveal the random key K_1 and $\mathcal{A}_{N_2}$, the address of N_2. Let RPI_1 denote the new RPI, which has one fewer layer of encryption. We assume that decrypting results in no effect on the size of the RPI which is still ω.
3. Encrypt $[M']^{\Omega-\omega}$ with K_1 to form Y_1
4. Send (RPI_1, Y_1) to N_2. We notice that the size of this message is Ω.

The next $l-1$ MIXes on the return path will perform a similar operation. At MIX N_i:

1. Upon reception of (RPI_{i-1}, Y_{i-1}), the MIX decrypts RPI_{i-1} to reveal $\mathcal{A}_{i+1}$ and K_i. The resultant value is denoted RPI_i
2. Encrypt Y_{i-1} by K_i to form $Y_i = K_i(Y_{i-1})$
3. Send (RPI_i, Y_i) to the next hop to the address $\mathcal{A}_{i+1}$

Finally, the original sender will receive the string (RPI_l, Y_l), with:

$$RPI_l = E_s(KS, l) \tag{5.7}$$

$$Y_l = K_l(K_{l-1}(...K_1(\lceil M' \rceil)...)) \tag{5.8}$$

Decrypting the RPI with his private key allows the original sender to reveal KS and l and to regenerate $K_1, ..., K_l$. Successive decryptions of Y with these keys yield M'. This mechanism is proposed in [6]. In this scheme, keys are embedded in the reply, whereas in Chaum's proposal [1], the original sender has to remember the keys $K_1, ..., K_l$.

Chapter 6

Sender anonymity

A very limited possibility of achieving sender anonymity is that each user station in the network generates constant traffic, dummy or not. In this case, an eavesdropper cannot notice when a user station really has something to send and how much. A problem arises if the attacker is the recipient of the message, then he knows the identity of the sender.

The MIX-Network [1] as well as Onion Routing [8] networks can be regarded as sender anonymity schemes if every participant acts as a MIX (resp. an onion router), since the message a participant sends is hidden among the messages of a complete batch.

The scheme proposed in [7] can also be seen as a sender anonymity based mechanism, since with respect to the web server, every member of the crowd appears as likely as any other member to be the actual sender of the request.

However, the only scheme that provides unconditional sender anonymity is the DC-Network through superposed sending [3]. This technique will be described in chapter 10.

One can also design networks for preventing attackers from physically observing all lines connecting a user with the rest of the world. This issue won't be investigated in this report, since we are interested in anonymity from an information-theoretic point of view. However, we can mention that a simple and efficient way to do so is to connect the user stations by RINGs. If an anonymous medium access protocol is used, such as ALOHA [22], the sending of a user station is only observable if its two neighbor stations collude or the lines are tapped. The reader should refer to [10, 12] for more information on this topic.

Chapter 7

Other related techniques

7.1 ISDN MIXes

In [16], mixes are used to provide untraceable communication in an ISDN network. ISDN-Mixes are a combination of a variant of the basic MIX, dummy traffic on the subscriber line, and broadcast of incoming-call messages in the subscriber area. They provide untraceability in a network with narrow-bandwidth limitations. The notion of anonymous communication channel will be introduced. We can already mention that the principal problems with such MIX-channels will come from the delay and the traceability in releasing connections.

7.1.1 A variant of MIXes

A variant of the MIX basic principles are used in ISDN-Mixes. Assume A wants to send a message N to B through the sequence of MIXes, $MIX_1, ..., MIX_m$ Each MIX_i with $i \geq 2$ has initially chosen a key pair (c_i, d_i) of a public key cryptosystem (c is the corresponding public key). And A shares a key k_{A1} with MIX_1. A will successively encrypt the message, providing that A additionally wants to pass data D_i to each MIX_i Thus A will recursively form the following encrypted messages, where N_i is the message which MIX_i will receive:

$$N_{m+1} = N \tag{7.1}$$

$$N_i = c_i(D_i, N_{i+1}),\ \text{for } i = (m, m-1, ..., 2) \tag{7.2}$$

$$N_1 = K_{A1}(D_1, N_2) \tag{7.3}$$

N1 is sent to MIX_1 and is called a MIX-input-message. One purpose of D_i is to include a time-stamp to avoid replays, and to distribute some cryptographic information that will be used to encrypt or decrypt the flow of data in order to prevent content correlation.

7.1.2 MIX-Channels

MIX-Channels are adapted from MIXes so that they can handle a continuous stream of data almost without delay. A MIX-Channel will consist of two parts: a MIX-sending-channel from the sender to MIX_m, and a MIX-receiving-channel from MIX_m to the recipient B. Then each part will be interconnected, see figure. For several practical reasons of delay, it is sent a sending-channel establishment message (SendEstab-message) before the user data start, into the signaling channel.

Upon receiving the N_i part of a SendEstab-message, each MIX:

1. reserves an outgoing 64-kbit/s channel C_i to MIX_{i+1} for the following data.
2. tells the position C_i to MIX_{i+1} together with the decrypted message N_{i+1}
3. It stores the correspondence between C_i and the incoming channel C_{i-1}
4. It stores the private key k_i which it has found in N_i that will be used to decrypt data in the forward direction. This is done to prevent correlation in content of the data traveling on a channel.

Now each MIX_i can immediately decrypt each bit of data arriving on C_{i-1} with k_i and send it out on C_i This is called a MIX-sending-channel.

So far, only A is untraceable by B. Therefore we need the second half of the MIX-channel, a MIX-receiving-channel, to protect B from A. Indeed, we will establish a communication between two parties who don't know each other identity, we thus need to protect the recipient from the sender. To establish such a channel, the recipient B sends a receiving-channel-establishment message through the MIX-cascade. This message travels across the signaling channel of the ISDN network and delivers a key k_i to each MIX_i The channel is established and used in the reverse way: channels C'_i from MIX_{i+1} to MIX_i are reserved. When user data arrives on C'_i MIX_i encrypts the data and forwards it to the previous mix on C'_{i-1} Thus B receives multiple-encrypted data and decrypts it with all his keys k_i.

Each half of a MIX-channel protects the participant who has established it. Now, the two halves have to be connected. The resulting channel is called a MIX-channel. Hence, MIX_m must know which channels to connect. Hence it must receive a common information of both the SendEstab and RecEstab messages. This attribute is called a label l_{AB} which must be known to both A and B.

To complicate traffic analysis, a sufficiently large number of channels must be established with the same batches of SendEstab and RecEstab messages; and the user data on them must start at the same time and end at the same time. Otherwise, an observer who sees the user data on A's MIX-sending-channel, and a moment later the data arriving on B's MIX-receiving-channel, could guess that these channels are connected. It might be difficult to obtain enough connections starting at the same time. More generally, this problem can be seen as the ability to maintain a constant traffic in the network. A solution will be proposed in the complete technique of ISDN-MIXes.

7.1.3 ISDN-MIXes

Figure 7.2, describes the structure of the ISDN-MIXes network infrastructure. Functionally, local exchanges have two halves: the first half administers the communication between the subscribers and MIX_1 the second half the communication between MIX_m and the long distance network. Untraceability will be achieved within the set of all subscribers at one local exchange. This set can be seen as an anonymity set.

To maintain constant traffic, a connection between A and B is divided into a sequence

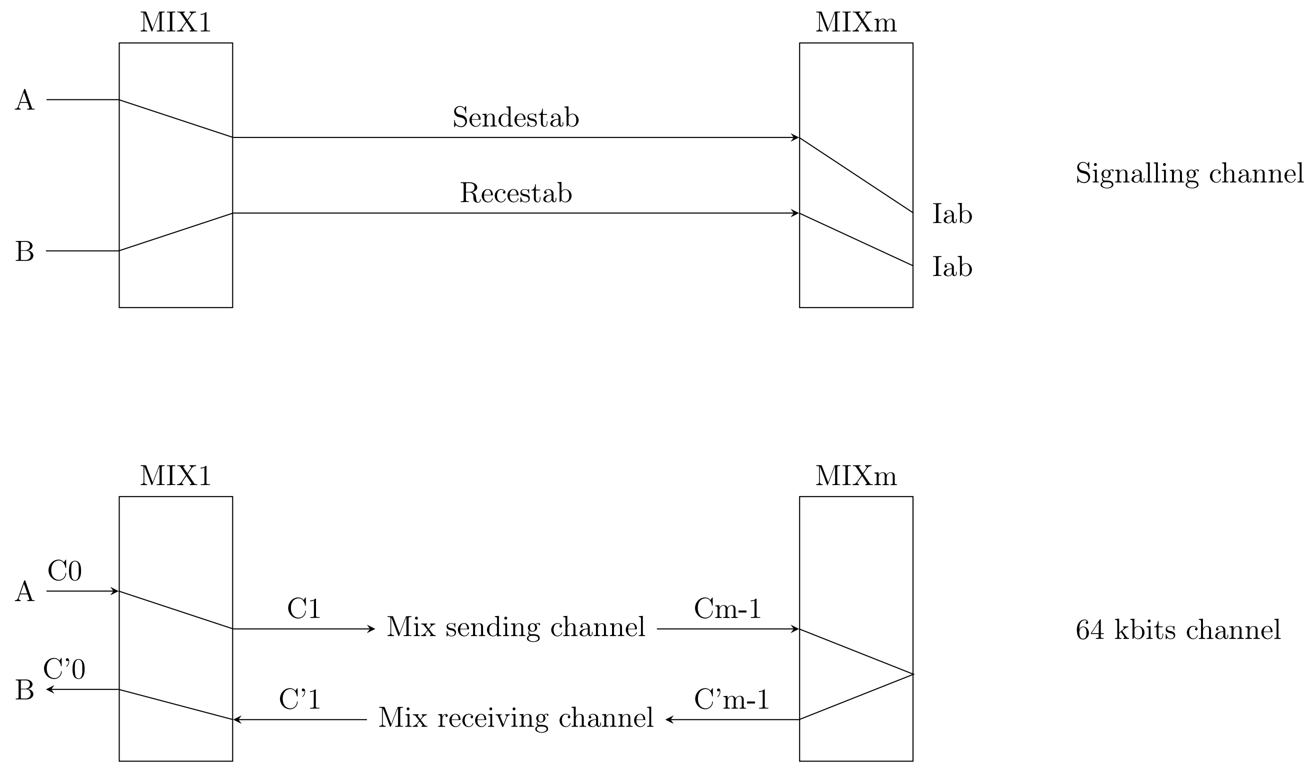

Figure 7.1: MIX channels

of time-slice-channels. With each new time-slice, participants can release connections and/or establish new ones. Also, each participant who does not use a channel during a time-slice establishes a dummy time-slice channel.

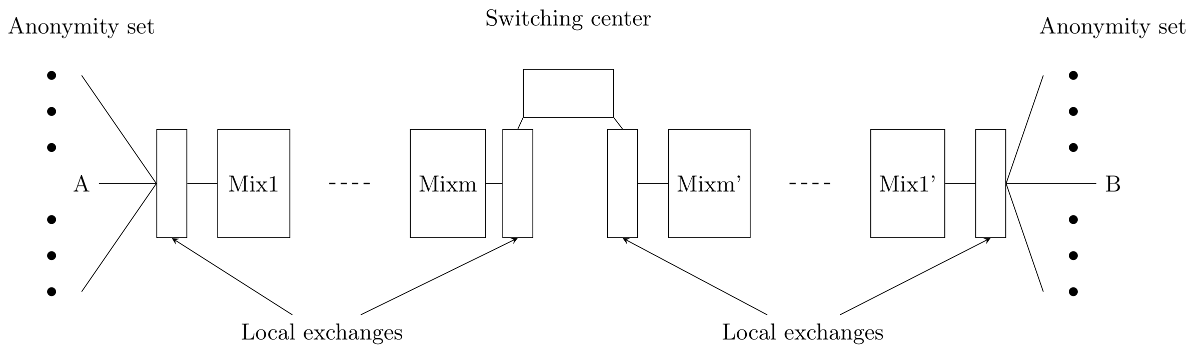

Figure 7.2: ISDN MIXes

Time-slice channels During each time-slice, each subscriber maintains two MIX-sending channels and two MIX-receiving-channels. They are called time-slice sending-channels and time-slice receiving channels. If A has a real connection with B, A's SendEstab message contains the address of B's local exchange and the corresponding label that will allow to connect to the connection established by B. If A has no real connection to establish, the corresponding Sen Estab and RecEstab message carry the same label, thus A establishes a connection with himself.

Establishing calls There is a need in coordination between A and B, in particular, if A wants to communicate with B, he needs a way to tell him to stop setting up dummy channels and set channels to meet A's channel instead. For that purpose, incoming-call messages are broadcast in B's anonymity set. It contains an implicit address of B as well as the time slice t_0 when the call is to start. Also A and B must use the same labels in the corresponding SendEstab and RecEstab messages. This can be achieved if A includes a seed s for a pseudo-random number generator in the incoming-call message. From s, all the labels are derived. A sends this incoming-call message as N in a MIX-input-message, called call-establishment message.

7.1.4 Privacy provided

As mentioned earlier, ISDN-MIXes rely on the cheapness of dummy traffic on the subscriber lines. It is a solution to provide unlinkability between caller and callee in an ISDN environment by creating an anonymous communication channel between those two parties. Contrary to the MIXes, content decorrelation is provided by successive encryptions/decryptions with symmetric keys that were previously distributed during the establishment of the channel. This allows to experience lower delay at each MIX on the path. However, even if unlikability between subscribers of two different local exchanges is provided, an eavesdropper or the network could easily observe the relation between two local exchanges. The reader should refer to [16] for more specific attacks on the system.

7.2 Onion routing

In [8, 9], is presented the notion of anonymous socket connections that are bi-directional and real-time anonymous communication channels for any protocol such as HTTP, SMTP, TELNET, and that can be adapted to use a proxy service. The technique is called onion routing and aims to protect against traffic analysis from both active and passive eavesdroppers the same way a MIX does, but on the basis of an anonymous connection. It works in the following way: the initiating application, instead of making a connection directly to a responding server, makes a connection to the appropriate onion routing proxy on some remote machine basically located on the firewall of the organization the initiator belongs to. That onion routing proxy builds an anonymous connection through several other onion routers to the destination. Each onion router can only identify adjacent onion routers along the route. Data passed along the anonymous connection appears different at and to each onion router to prevent content correlation to track the data. Also, even if adjacent onion routers can collude, the system is secure if at least one onion router is honest.

The onion routing proxy defines a route through the onion routing network by constructing a layered data structure called an onion and sending that onion through the onion routing network. This structure is somehow similar as the one of the messages sent to a MIX Network. Each layer of the onion defines the next hop in a route as well as a cryptographic information that will be used by the onion router to encrypt data passing through it, either forward or backward. An onion router that receives an onion peels off its layer, reads from that layer the name of the next hop and the keys (one will be used to decrypt data in the forward direction and the other will be used to encrypt data in the backward direction) associated with its hop in the anonymous connection, pads the embedded onion to some constant size, and sends the padded onion to the next onion router. Sending the onion across the onion routing network distributes some cryptographic information that will be used by each hop to encrypt/decrypt the flow of data for prevention of correlation in content.

Before sending data over an anonymous connection, the initiator's onion routing proxy adds a layer of encryption for each onion router in the path. As data moves through the private connection, each onion router removes one layer of encryption, so it finally arrives as plaintext to the destination. The last onion router forwards data to another type of proxy, called the responder's proxy, whose job is to pass data between the onion network and the responding server or intended recipient. This layering occurs in the reverse order for data moving back to the initiator. So data that has passed backward through the anonymous connection must be repeatedly decrypted to obtain the plaintext.

7.2.1 Overview

In onion routing, instead of making socket connections directly to a responding machine, initiating applications make connections through a sequence of intermediate hops called onion router. The onion routing network allows the connection between the initiator and responder to remain anonymous. These anonymous connections aim to hide from external eavesdroppers who is communicating with whom. The onion routing network topology is shown in figure 7.3.

In this configuration, an onion router sits on the firewall of the sensitive site. This onion router serves as an interface between machines behind the firewall and the external network. To complicate tracking of traffic originating or terminating within the sensitive site, this onion router should also route data between other onion routers. Onion routers in the network are connected by longstanding socket connections. Anonymous connections through the network are multiplexed over the longstanding connections. For any anonymous connection, the sequence of onion routers in a route is strictly defined by the first node, which is also a proxy for the service being requested (e.g. HTTP requests). Therefore, this Proxy/Routing node is the most sensitive one. However, each onion router will only be able to identify the previous and the next hop along a route.

The onion routing network is accessed via proxies. An initiating application makes a socket connection to an application specific proxy on some onion router. That proxy defines a route through the onion routing network by constructing a layered data structure called an onion and sending that onion through the network. Each layer of the onion defines the next hop in a route. An onion router that receives an onion peels off its layer, identifies the next hop, and sends the embedded onion to that onion router. After sending the onion, the initiator's proxy sends data through the anonymous connection. The last onion router forwards data to another type of proxy called the responder's proxy, whose job is to pass data between the onion network and the responder.

7.2.2 Onions

The transmission protocol has two phases: connection setup, and data transmission. The first is related to the sending of the onion across the onion routing network and the setup of the anonymous communication channel, whereas the second is related to the actual transmission of data. To begin a session between an initiator and a responder, the initiator proxy identifies a series of routing nodes forming a route through the network and constructs an onion which encapsulates that route. The onion data structure is

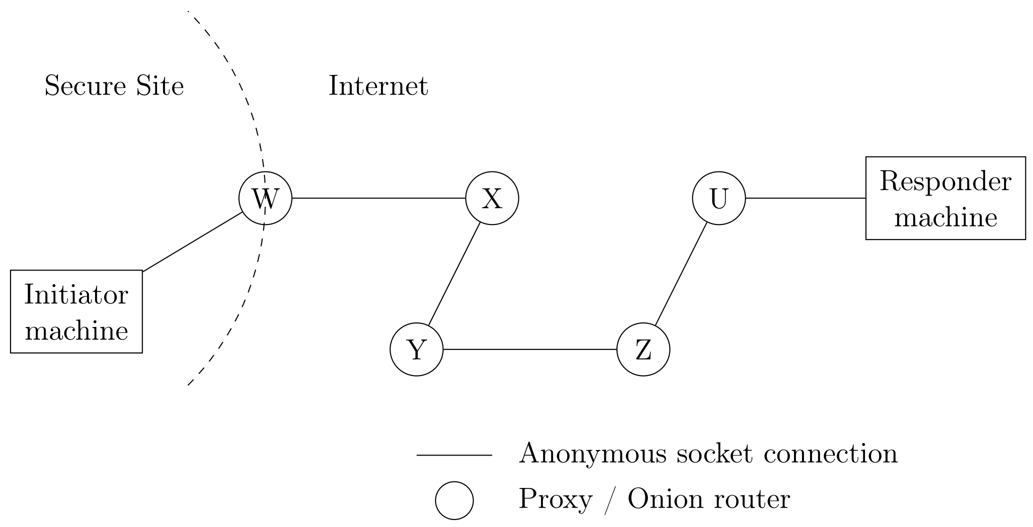

Figure 7.3: Onion routing infrastructure

composed of layer upon layer of encryption wrapped around a payload. The initiator proxy encrypts first for the responder proxy, then for the preceding node on the route, and so on back to the first routing node to whom he will send the onion. When the onion is received each intermediate node only knows the adjacent onion routers but not the complete path. An intermediate node P_x receives the onion:

$$\{\text{exptime}, \text{nexthop}, F_f, K_f, F_b, K_b, \text{payload}\} \tag{7.4}$$

Here $PK_\bullet$ denotes the public key encryption for routing node P_x The decrypted message contains an expiration time for the onion, the next routing node to which the payload is to be sent, the payload, and two function/key pairs specifying the cryptographic operations and keys to be applied to data that will be sent along the connection, either forward or backward. The forward pair (F_f, K_f) is applied to data moving in the forward direction (defined as data moving in the same direction that the onion traveled). The backward pair (F_b, K_b) is applied to data moving in the backward direction (defined as data moving in the opposite direction that the onion traveled). For any intermediate routing node, the payload will be another onion. The expiration time is used to detect replays.

Once the onion has traveled across each intermediate onion router toward the responder proxy, and that a connection is established between each adjacent onion router, the sending phase can take place. Note that when the connection between two intermediate nodes is setup, two symmetric keys can be generated at the nodes to perform link-by-link encryption [8].

7.2.3 Creating the circuit

Communication between onion routers is packaged into fixed sized cells, which will allow for the multiplexing of anonymous connections and control information over the anonymous communication channel. Four types of cells are created in the onion routing system [9]: PADDING, CREATE, DATA, and DESTROY. Cells have the following structure:

$$\{\text{ACI, Command, Length, Payload}\} \tag{7.5}$$

The ACI (anonymous connection identifier) and Command fields are encrypted using the link encryption between neighboring nodes. Additionally, the length and Payload fields are encrypted using the link encryption between neighboring nodes if the command is either PADDING or DESTROY. For CREATE commands, the length is link encrypted but the payload is already encrypted because it carries the onion. For DATA commands, the length and entire payload are encrypted using the anonymous connection's forward or backward cryptographic operations.

Each anonymous connection is assigned an ACI at each onion router, which labels an anonymous connection that will be multiplexed over the longstanding connection to the next onion router. To move an onion through the system, an onion router peels off the outermost layer, identifying the next hop. It checks the freshness (not expired and not replayed) of the onion, computes the necessary cryptographic keys, seeds the forward and backward cryptographic information, chooses a new ACI for the next hop in the new connection, and maps incoming to outgoing ACIs. The rest of the onion is padded randomly to its original length, placed into CREATE cells, and then sent out in order to the appropriate neighbor, in order to prevent intermediate nodes from inferring route information from the monotonically diminishing size of the onion. The payload of the last cell is padded with random bits to fill the cell if necessary (so that each cell be of same length).

Data moves through an anonymous connection in DATA cells. At each onion router, both the length and payload fields are encrypted using the appropriate keys. The new cell is sent out to the appropriate neighbor. The initiator's onion router must repeatedly encrypt (or decrypt depending on the direction of the data) data to either add the appropriate layers of encryption on outgoing data, or remove layers of encryption from incoming data.

If a connection is broken, a DESTROY command is sent to clean up state information. The ACI field of the DESTROY command carries the ACI of the broken connection. The PADDING command is used to inject data into a longstanding connection to further confuse traffic analysis. Each onion router also reorder cells moving through it, in order to prevent time correlation.

7.2.4 Privacy provided

Anonymous socket connections provide a substantial protection against eavesdropping and traffic analysis. However it is not invulnerable to traffic analysis attacks. Due to the volume attack, whereby an onion router counts the packets it delivers per ACI, any two collaborating onion routers can correlate all the traffic between them. It is also the case that for low traffic situations an eavesdropper can easily correlate incoming and outgoing traffic through an onion router. Moreover, the initiator's proxy must be explicitly trusted which might not be desired since it resides on the firewall of the enterprise.

7.3 Crowds

In [7], a simple mechanism to provide untraceability to a set of participants willing to initiate requests to a web server is presented. It works in the following way: a user is represented in a crowd by a process called a jondo. This jondo will establish a random path of jondos via which users transactions to the web server will be carried. Upon receiving the first user request from a browser, the jondo picks an other jondo from the crowd at random. When this jondo receives the request, it flips a biased coin to determine whether or not to forward the request to another jondo. If the result is to forward, then the jondo picks another jondo at random and sends it the request, otherwise it sends the request directly to the end server. Subsequent requests initiated at the same jondo will follow the same path, and server replies will traverse the same path as the requests, in the reverse order. Indeed, on receiving a request for the first time, the jondo will create a corresponding path id that will be used to forward requests initiated by a given jondo, and to deliver replies of the end server. We might notice that the jondo will hold at each hop in the path a different identifier for the path, in order to avoid infinite loop if the jondo occupies multiple positions on a path. We finally mention that all communications on a path are encrypted with a path key, which all jondos on a path possesses. Distribution of such keys is discussed in [7]. This key is used to prevent a local eavesdropper from learning the intended receiver (the web server) of a request. However, if the local eavesdropper is located at the output of the last jondo on the path, he will obviously detect the intended recipient.

The security provided by this scheme is relatively weak. Neither protections against content correlation nor against time correlation are provided. It is thus very easy for an eavesdropper to track the request en route, and moreover to track it back to its originator since at that time, an input will not result in a corresponding output. Similarly, a local eavesdropper might observe that a request output by the initiator's machine did not result from a corresponding input, thus compromising sender anonymity. The only interest of crowds is to provide sender anonymity with respect to the end server. Indeed, according to this mechanism, every member of the crowd will appear as likely as any other member to be the original sender of the request. Collaborating jondos may not be very efficient. It is shown in [7], that if the number of members in the crowd is relatively close to the number of colluders, the first collaborator's immediate predecessor appears more likely to be the initiator of the request than another non-collaborating jondo, which is intuitively apparent.

Chapter 8

Basic concepts on graph theory

8.1 Graphs

We will present in this chapter the basic concepts of graph theory that we use throughout this report. They are mostly adapted from [24, 26].

Intuitively speaking, a graph is a set of points, and a set of lines, with each line joining one point to another. The points are called the vertices of the graph, and the lines are called the edges of the graph.

The set of vertices of a graph is generally denoted by V, and the set of edges of a graph is generally denoted by E. For example, in the graph in figure 8.1,

$$V = \{a, b, c, d\}, \text{ and } E = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

A graph is completely determined by its vertices and by the family of its edges.

Formally, a graph G is defined to be a pair (V, E), where

1. V is a set $\{x_1, ..., x_n\}$ of elements called vertices, and
2. E is a family $(e_1, ...e_m)$ of elements of the Cartesian product $V \times V$, called edges. The direction needs not to be specified.

An element [x, y] of $V \times V$ can appear more than once in this family. A graph in which no element of $V \times V$ appears more than p times is called a p-graph. One can also talk about multigraphs, without specifying the maximum number of edges that might join any two vertices.

The number of vertices in a graph is called the order of the graph.

An edge of G of the form [x, x] is called a loop.

For an edge $u = [x, y]$, x and y are both endpoints of the corresponding edge.

For an edge $u = [x, y]$, vertex y is called a neighbor of z.

The set of all neighbors of x is denoted by

$$\Gamma_G(x) = \{y \in V; [x, y] \in E\} \tag{8.1}$$

If $\Gamma(x) = \emptyset$, x is called an isolated vertex. For $A \subset V$, let

$$\Gamma_G(A) = \bigcup_{a \in A} \Gamma_G(a). \tag{8.2}$$

If $x \in \Gamma_G(A), x \notin A$ then z is said to be adjacent to set A.

The family $(e_1, ..., e_m)$ of the edges of G is often denoted by its set of indices $E = \{1, ..., m\}$.

A graph is called a simple graph if:

1. it has no loops,
2. no more than one edge joins any two vertices.

We will only consider simple graphs throughout this report, since DC networks are modeled by simple graphs. We can notice that for simple graphs, we have the relation: $0 \leq m \leq \binom{n}{2}$ where m is the number of edges of the simple graph.

Graphs and multigraphs often appear under other names: sociograms (psychology), simplexes (topology), electrical networks, organizational charts, communication networks, family trees, etc. It is often surprising to learn that these diverse disciplines use the same theorems. The primary purpose of graph theory was to provide a mathematical tool that can be used in all disciplines.

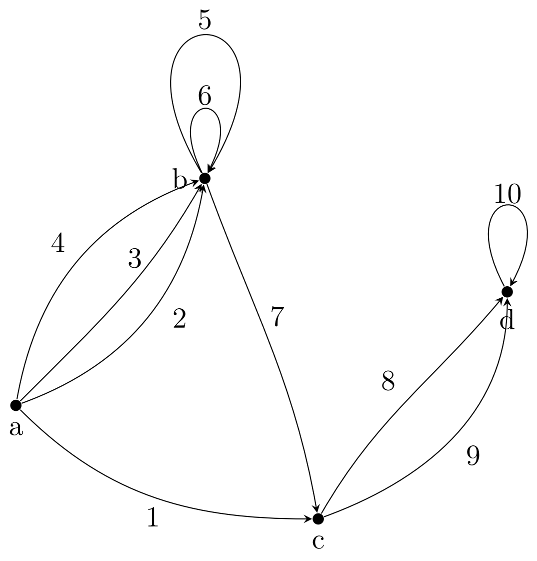

Figure 8.1: A 3-graph of order 4

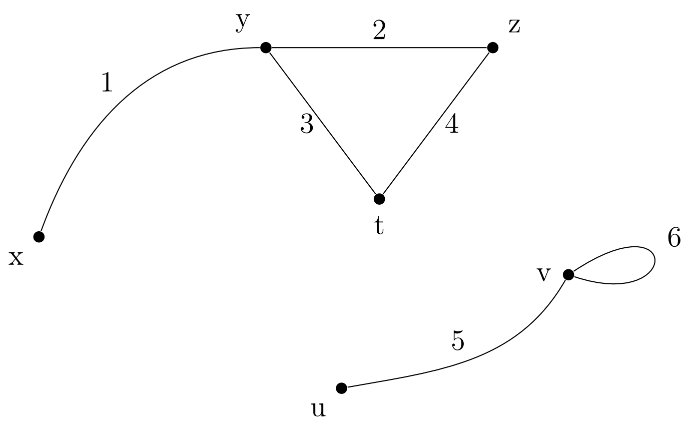

Figure 8.2: A 1-graph of order 6

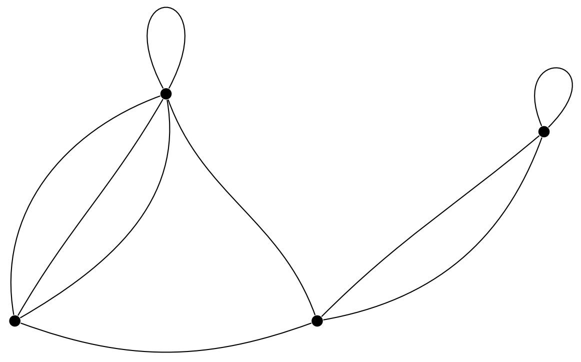

Figure 8.3: Multigraph

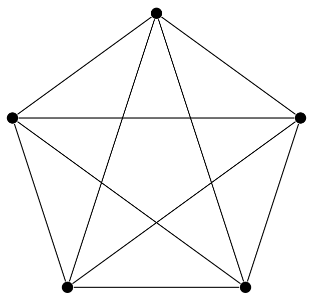

Figure 8.4: Complete graph K_5

8.2 Basic definitions

Definition 8.2.1 (Multiplicity) The multiplicity of a pair z, y is defined to be the number of edges with z and y as corresponding endpoints. Denote this number by $m_G(x, y)$. If $x \neq y$ then $m_G(x, y)$ denotes the number of edges with both x and y as endpoints. If G is a simple graph, then for all z, y $\in X$ $m_G(x, y) = 1$ if z and y are adjacent, or $m_G(x, y) = 0$ elsewhere.

Definition 8.2.2 (Degree) The degree of vertex x is the number of edges with x as an endpoint each loop being counted twice. The degree of x is denoted by $d_G(x)$. For a simple graph, the degree of vertex x is the number of edges incident to this vertex. If in a graph each vertex has the same degree k, this graph is said to be k-regular.

Definition 8.2.3 (edge incident to a set $A \subset V$) If an endpoint of an edge e belongs to A, and if the other endpoint of edge e does not belong to A, then e is said to be incident to A, and we write $e \in \omega(A)$.

Definition 8.2.4 (Complete graph) A graph G is said to be complete if

$$m_G(x, y) \geq 1 \tag{8.3}$$

for all z, $y \in V$, such that $x \neq y$. A simple, complete graph on n vertices is called and n-clique, and is often denoted by K_n. It has $|E| = \binom{n}{2}$. See figure 8.4.

Definition 8.2.5 (Subgraph of G generated by $A \subset V$) The subgraph of G generated by A is the graph with A as its vertex set and with all the arcs in G that have both their endpoints in A.

Definition 8.2.6 (Partial graph of G generated by $X \subset E$) This is the graph (V, X) whose vertex set is V and whose edge set is X. In other words, it is graph G without the edges $E - X$.

Definition 8.2.7 (Partial subgraph of G) A partial subgraph of G is the subgraph of a partial graph of G.

Definition 8.2.8 (path of length $q > 0$) A path is a sequence

$$\mu = (e_{i_1}, e_{i_2}, ..., e_{i_q}) \tag{8.4}$$

of arcs of G such that each edge in the sequence has one endpoint in common with its predecessor in the sequence and its other endpoint in common with its successor in the sequence. The number of edges in the sequence is the length of of the path μ. A path that does not encounter the same vertex twice is called elementary. A path that does not use the same arc twice is called simple. For a 1-graph, a path is completely determined by the sequence of vertices $x_1, ...$ that it encounters. Hence we often write

$$\mu = ((x_1, x_2), (x_2, x_3), ...) = [x_1, ..., x_k, x_{k+1}] = \mu[x_1, x_{k+1}] \tag{8.5}$$

Vertex x_1 is called the initial endpoint and vertex x_{k+1} is called the terminal endpoint of path μ. Similarly, for a simple graph, a chain μ with endpoints x and y is determined by the sequence of its vertices, and we may write

$$\mu = \mu[x, y] = [x, x_1, x_2, ..., y] \tag{8.6}$$

Definition 8.2.9 (Vertex-disjoint paths) Two paths $((x_1, x_2), (x_2, x_3), ..., (x_{k-1}, x_k))$ and $((y_1, y_2), (y_2, y_3), ..., (y_{l-1}, y_l))$ are called vertex-disjoint paths if $\forall 1 \leq i \leq n$, $1 \leq j \leq l$, $x_i \neq y_j$.

Definition 8.2.10 (Connected graph) A connected graph is a graph that contains a path $\mu[x, y]$ for each pair z, y of distinct vertices.

Definition 8.2.11 (Connected component of a graph) Clearly, the relation $\{x = y$ or $x \neq y$ and there exists a chain in G connecting and y$\}$ denoted by $x \equiv y$ is an equivalence relation because

$$x \equiv x \text{ (reflexivity)} \tag{8.7}$$

$$x \equiv y \Rightarrow y \equiv x \text{ (symmetry)} \tag{8.8}$$

$$x \equiv y, y \equiv z \Rightarrow x \equiv z \text{ (transitivity)} \tag{8.9}$$

The classes of this equivalence relation partition V into connected subgraphs of G called the connected components. For example, the graph in figure 8.2, possesses two connected components.

Definition 8.2.12 (Articulations set) For a connected graph, a set A of vertices is called an articulation set (or a cutset) if the subgraph of G generated by $V - A$ is not connected. For example, {a,c} and {c} are two articulation sets of the graph in figure 8.1; vertex c is also called an articulation vertex (or a cut-vertex).

Definition 8.2.13 (Incidence matrix) If G has vertices $V = \{x_1, ..., x_n\}$, and edges $E = \{e_1, ..., e_m\}$, the matrix $[m_{ij}]$ with the vertices labeling the rows and the edges labeling the columns and with $m_{i,j} = 1$ if the edge e_j is incident to the vertex x_i and $m_{i,j} = 0$ otherwise, is called the incidence matrix. For example, in the graph of figure 8.2, the incidence matrix is:

$$[m] = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \tag{8.10}$$

8.3 List of symbols

$\mathbb{R}$	Set of all real number.
$\mathbb{N}$	Set of all non-negative integers.
$\mathbb{Z}$	Set of all integers.
$\emptyset$	Empty set.
$\|A\|$	Cardinality of A. (i.e. number of elements)
$\{x/...\}$	Set of all x such that....
$a \in A$	a is an an element of set A.
$a \notin A$	a is not an element of set A.
$A \cup B$	Union of sets A and B.
$A \cap B$	Intersection of sets A and B.
$A - B$	A less B (the elements of A that are not in B).

$A \subset B$	Set A is contained in set B.
$A \not\subset B$	Set A is not contained in set B.
$A \times B$	Cartesian product of A and B (the set of all pairs (a, b) where $a \in A$ and $b \in B$.
$\Gamma(\alpha)$	Image of element α in correspondence Γ.
$\Gamma(A)$	Image of set A in correspondence Γ, or $\bigcup_{a\in A} \Gamma(a)$
$\mathcal{P}(A)$	Set of all subsets of set A.
$(1) \Rightarrow (2)$	Property (1) implies property (2).
$(1) \Leftrightarrow (2)$	Property (1) is equivalent to property (2).
$\binom{p}{q} = \frac{p!}{q!(p-q)!}$	Binomial coefficient.
$\log p$	Logarithm of p.
$\lfloor p/q \rfloor$	Integer part of p/q
$\lceil p/q \rceil$	Ceiling of p/q. Smallest integer greater than or equal to p/q.
$p \equiv q(mod.k)$	Integer p is equal to q modulo k.
$[m_{i,j}]$	matrix whose entry in the $i-th$ rows and $j-th$ column is $m_{i,j}$
$d_G(x)$	Degree of vertex x in graph G.

8.4 Degrees

For a graph $G = (V, E)$, the degree $d_G(x)$ of a vertex x is defined to be the number of edges having x as an endpoint, each loop being counted twice. We thus have a result, due to Euler [25] that will be useful throughout this report.

Theorem 8.4.1 The sum of the degrees of the vertices of a multigraph $G = (X, E)$ with $V = (x_1, ..., x_n)$ and $E = (e_1, e_2, ..., e_m)$ is twice the number of edges,

$$\sum_{i=1}^{i=n} d_G(x_i) = 2m \tag{8.11}$$

Definition 8.4.2 (Degree sequence) The degree sequence of a graph $G = (V, E)$ is the sequence of the degrees of its vertices arranged in decreasing order. $d = (d_1, ..., d_n)$ with $d_1 \geq d_2 \geq, ..., \geq d_n$ is the degree sequence.

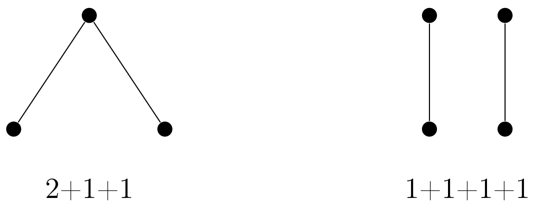

Figure 8.5: The graphical partitions of 4

8.5 Partitions

The degrees $d_1, ..., d_n$ of the vertices of a simple graph $G = (V, E)$ with $|V| = n$, $|E| = m \leq \binom{n}{2}$, form a sequence of nonnegative integers, whose sum is 2m where m is the number of edges of the graph, according to theorem 8.4.1. In number theory it is usual to define a partition of a positive integer n as a list or unordered sequence of positive integers whose sum is n. Under this definition, 4 has five partitions:

$$4, 3+1, 2+2, 2+1+1, 1+1+1 \tag{8.12}$$

The degrees of a graph with no isolated points determine such a partition of 2m, but because of the importance of having a general definition holding for all graphs, it is convenient to use an extended definition, changing positive to nonnegative. We define a partition of a nonnegative integer q to be a finite list of nonnegative integers with sum q. We define the partition of a graph to be the partition of 2m as the sum of the degrees of the vertices, $2m = \sum_i d_i$ as in Theorem 8.4.1. We notice that only two of the five partitions of 4 into positive summands belong to a graph, according to figure 8.5. We now present the relation between partitions of positive numbers and simple graphs.

Definition 8.5.1 A partition $(d_1, ..., d_n)$ of an even number 2m into n parts is graphical if there exists a simple graph G whose points have degree d_i. One also says that the sequence $(d_1, ..., d_n)$ is graphic.

If such a partition is graphical, then certainly $\forall i \in \{1, ..., n\}$, $k_i \leq n-1$ since a vertex is at most adjacent to all others vertices. It is easy to see that this condition is not sufficient for a partition of an even number to be graphical. Two related questions arise. First, how can one tell whether a given partition is graphical? Second, how can one construct a graph for a given graphical partition? The existence question was answered by Erdös and Gallai [25] who have derived a necessary and sufficient condition for a partition to be graphic. The theorem is given and explained in §11.1 The second will be illustrated in section 8.1

8.6 Connectivity

The connectivity $k(G)$ of a graph G is defined to be the minimum number of vertices whose removal disconnects G or reduces G to a single vertex. If G is not a clique, there exist two non-adjacent vertices a and b; thus $X - \{a, b\}$ is a set whose removal disconnects G, and, consequently,

$$k(G) \leq |X - \{a, b\}| = n - 2 \tag{8.13}$$

On the other hand, if G is the n-clique K_n, we have

$$k(K_n) = n - 1 \tag{8.14}$$

A graph G is said to be k-connected if its connectivity $k(G) \geq k$. An articulation set of G is any set of vertices whose removal disconnects G. Let $\mathcal{A}_G$ denote the family of all articulation sets of G. A graph $G \neq K_n$ is k-connected if, and only if,

$$k(G) = min_{A \in \mathcal{A}_G} |A| \geq k \tag{8.15}$$

Thus G is k-connected if, and only if,

1. $k \leq n - 1$, and
2. there is no articulation set A of G with $|A| = k - 1$

We present some results that can be applied to simple graphs.

Theorem 8.6.1 For a simple connected graph G,

$$k(G) \leq min_{x \in X} d_G(x) \tag{8.16}$$

Theorem 8.6.2 (Menger's theorem) A necessary and sufficient condition for a simple graph to be k-connected is that any two distinct vertices a and b can be joined by k vertex-disjoint paths.

Chapter 9

Motivations for minimizing key requirements for DC Networks in presence of colluders

In 1988, David Chaum [3] described a beautiful technique, the DC Network, that allows a group of participants to send and receive messages anonymously. The DC Network is a virtual network consisting in a set $\mathcal{P} = \{P_1, P_2, ..., P_n\}$ of participants belonging to a same LAN or distributed across the Internet. Every pair of participant shares a common secret key. Thus the DC Network is a virtual network initially modeled as a complete key graph. The vertices represent the participants and an edge k_{ij} a shared secret key between participant P_i and participant P_j, with $k_{ij} = k_{ji}$. To send their messages anonymously, this group uses the superposed sending technique that basically consists of each participant in outputting his message hidden by a one-time-pad of the keys he shares with the other participants. The sum of each individual output will then be computed and broadcast. Since every key appears twice in the sum, they all cancel out, revealing the sum of each individual input. This represents the unconditionally untraceable message. Also, since the latter is the sum of all individual inputs, all participants but one must keep silent. Thus, they have to derive a transmission rule allowing them to send, without their messages being interfered with others. Basically, every participant will send in a first round, a reservation message indicating when he is willing to send.

The DC Network provides sender anonymity through superposed sending, and recipient anonymity because the final message is broadcast. It is important to notice that throughout this chapter, as in [3], attackers are assumed to be unable to manipulate the consistency of broadcast. We assume that all the attackers can do is pooling keys, as it is explained below.

It is shown that the security of the system depends on both the degree sequence of the graph and its connectivity. Let $\mathcal{C} = \{C_1, C_2, ..., C_c\}$ be the set of colluders, such that $\mathcal{C} \subseteq \mathcal{P}$. Colluding means that the members of $\mathcal{C}$ pool their key information in order to compromise a set of honest participants, a participant being compromised if he shares keys with all colluders. In this case, the colluders knowing the keys would deduce by adding with xor, the message sent by the participant. To preserve the anonymity of participant P_i, it will be sufficient for him to share at least one key with an honest participant. Therefore if $|\mathcal{C}| = c$ colluders are present, we must ensure that each vertex representing a participant has degree larger than $c+1$. However, since the degree sequence of the graph doesn't necessarily imply anything about its connectivity, the subset $\mathcal{C}$ could still be a cut vertex set. In this case, the anonymity wouldn't necessarily be compromised but the collusion would partition the graph such that the colluders could know from which connected component originates the message.

Indeed, the set of colluders may partition the key graph into connected components. As in chapter 3, such a connected component is called an anonymity set seen by the set of colluders. We will prove that the system as a whole is unconditionally secure against an external eavesdropper, but also that anonymity is preserved with respect to the colluders among a given connected component. The only thing the collusion will learn about the members of a connected component will be the overall sum of their individual inputs. However according to the transmission rules, only one participant at a time is allowed to send during superposed sending, thus all inputs but one will be set to zero. In this case, the colluders won't be able to compromise the anonymity of a participant inside a connected component, but they will know from which connected component originates the message. Therefore for a higher level of security we want the connectivity of the graph to be at least $c+1$, such that c colluders couldn't be a cut vertex set.

In the complete key graph each participant exchanges a key with each other participant. One thus has to compute $\frac{n(n-1)}{2}$ keys. It appears that having a fully connected graph seems to be a heavy requirement if the number of colluders verifies $c \ll (n-1)$, and we would like to find a way to minimize the number of edges in the graph while maintaining fully sender anonymity in presence of c colluders. The question arises: how to build a graph with a given connectivity while minimizing the number of edges. We show that the minimal number of edges, i.e. the minimal number of keys required to ensure security, is $m = \lceil \frac{n(c+1)}{2} \rceil$. Building a graph with a given degree sequence will also be investigated.

First of all, we will describe the behavior of the DC Network i.e. the superposed sending technique. Proof of its unconditional security will be given in the case of a field of order 2. Secondly, we will show how the security of the system modeled using graph theory is related to the degree sequence and the connectivity. Finally we will describe the algorithms that compute graphs with this requirement on the degree sequence (each degree sequence larger than $c+1$) and the connectivity ($(c+1)$ connectivity) while minimizing the number of edges. We can note that the latter condition infers the former. Concerning the problem of the degree sequence, we can already say that the degree of each vertex will have to be as close as possible than $c+1$ if we want to minimize the number of edges. Depending on the relative values of n and c, we won't always be able to find a graph for which we'll have for each i, $d_G(P_i) = c+1$ where $d_G(P_i)$ is the degree of vertex P_i in graph G, but we will still try to compute the cheapest topology in terms of number of edges. We will also check if the graph is connected.

Chapter 10

Modeling the DC Network using graph theory

10.1 First example

To explain how the DC network works, we will first take an example of four participants willing to send and receive messages according to figure 10.1.

- k_i: shared secret key between P_i and P_{i+1} where addition is taken modulo 4.
- M_i: Individual input of participant P_i.
- P_i: Participant willing to send and receive messages.
- B: Outsider. He may read all individual outputs.

We note that k_i and M_i are vector bits of the same length. The following operations are considered to be xors. We will see in the third section a variant of the superposed sending technique.

Each participant P_i outputs his message hidden by a one-time-pad of the keys he shares respectively with participant P_{i-1} and participant P_{i+1} where addition is taken modulo 4. By outputting we mean that the quantity $M_i \oplus k_{i-1} \oplus k_i$ is broadcast. The only way for an attacker to retrieve the input M_i from the output $M_i \oplus k_{i-1} \oplus k_i$ is to know both keys, or the sum of the keys, which is equivalent since we assume an attacker to be an insider, i.e. a member of the system. For instance if P_2 wants to know the message sent by P_1, he will have to collude with P_0 so that the attacking entity $\{P_0, P_2\}$ know the set of keys $\{k_1, k_0\}$. In this system a participant acting alone cannot learn evidence of a message sent by another participant, whatever may be his computational power. In this sense the system is unconditionally secure since the only way to attack it is to form collusions.

If B is the intended receiver of an original message, i.e. an initial input M_{i_1} he has to have a way to retrieve a given input from the different outputs of the system. This is done by computing the sum of every individual output he receives (remember that those are broadcast, thus public). Since every k_i appears twice in the sum, B will deduce $\Sigma_{i=0}^{3} M_i$ without knowing the relation between M_i and P_i. We can note that, according to this protocol, if P_i wants effectively to send a message anonymously, all participants but P_i must keep silent, i.e. set M_i to zero.

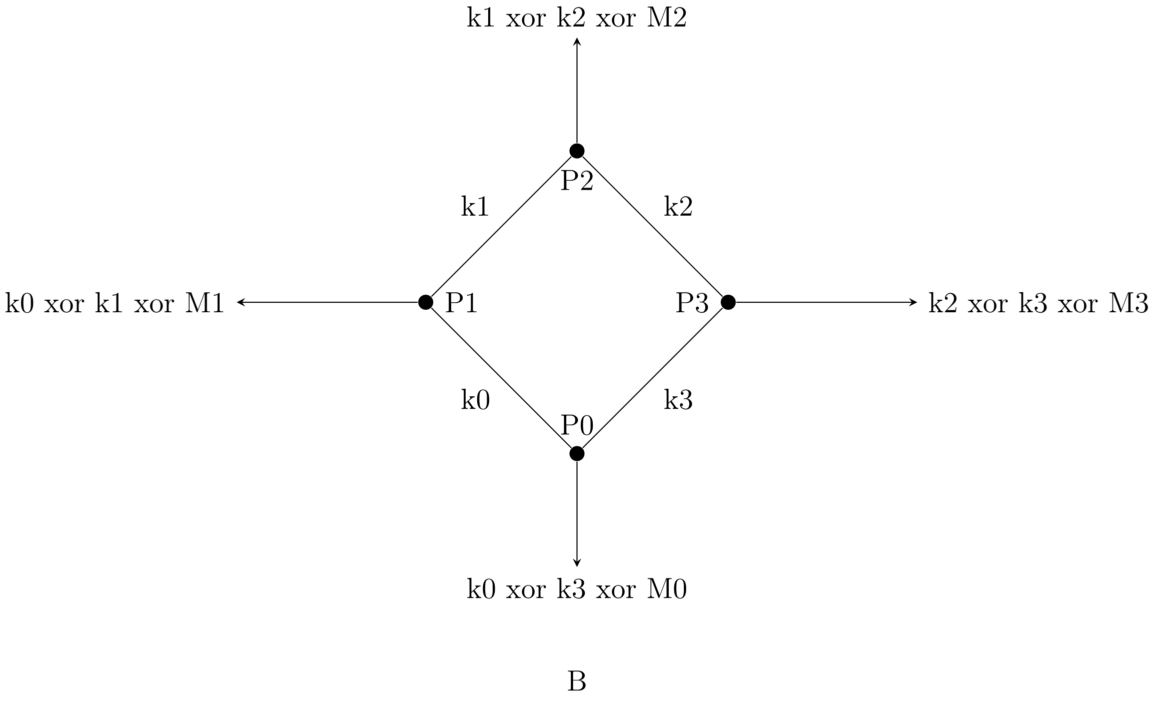

Figure 10.1: the DC Network with 4 participants

10.2 Superposed sending

We rely on superposed sending without expansion to enable each participant to compute a sum of inputs $\sum_{i=1}^{n} M_i$, where M_i is the input of participant P_i, without any participant (acting alone) gaining knowledge about the individual inputs. In this case the keys and the messages are vector bits of the same length.

Let $\mathcal{P} = \{P_1, P_2, ..., P_n\}$ be the set of participants. Let k_{ij} for $i \neq j$ be a shared secret key vector bit between P_i and P_j. We have $k_{ij} = k_{ji}$. All additions are considered to be xors of the vector bits. The DC Network is modeled by a complete graph where the vertices correspond to the participants and an edge to a shared secret key between two participants. In the complete graph every participant shares a key with every other participants. Another topology could be deduced by setting some keys to zero.

Let M_i be P_i contribution to superposed sending, i.e. his individual input. Each P_i, $i \in \{1, ..., n\}$ outputs:

$$o_i = M_i \oplus \sum_{j \neq i} k_{ij} \tag{10.1}$$

with $M_i = M$ if P_i is the one allowed to send and $M_i = 0$ otherwise. We will see at the end of this section the transmission rules for the senders. Each participant as well as an outsider can collect the individual outputs of all the others and compute:

$$\sum_i \sum_{j \neq i} k_{ij} \oplus M_i = \sum_i M_i \tag{10.2}$$

because every key appears twice in the sum, i.e. $\sum_i \sum_{j \neq i} k_{ij} = 0$. Note that no information regarding the individual inputs M_i is leaked, i.e. the relation between M_i and

P_i is hidden during the computation. Superposed sending thus guarantees unconditional untraceability. Proof of this unconditional security will be given.

We explained the superposed sending technique that is used for sender untraceability. We noticed that it enables each participant as well as an outsider to compute a sum of inputs $\sum_{i=1}^{n} M_i$, where M_i is the input of participant P_i without being able to know which participant input which message. Since the result is the sum of each individual input, all inputs but one must be set to zero if a participant wants to effectively send a message. Thus during the sending phase, only one participant will be allowed to send an actual message. The others will only output the sum of the keys they are sharing. We will call "Disruption" the sending of messages that disturb other messages. We can make a slight difference between disruption and collision where collision is a number of messages that disturb each other, but are sent following to the transmission rule. We won't discuss in this report the way to resolve disruptions and collisions. The reader will refer to [19, 18, 20] for more details on this topic. We just mention that the only way to detect disrupters in the DC Network will be to make public all information that passed during the sending phase. We will just present the transmission rules in the following. Recall that only one participant is allowed to send at a time, we must derive a scheme that will allow a participant to indicate that he is willing to send.

The protocol is called the slot reservation technique and is originally proposed by Chaum [3]. The idea is to fix in advance the time somebody is going to use. It works like this:

1. There is a "reservation phase" that consists of a long stream of group elements. Everybody is obliged to send exactly one 1 value in this phase.
2. If this phase does not contain as many 1's as there are participants, there was a collision and the whole process is repeated again.
3. The order of participants is the order in which they sent 1's.

We can note that up to now the field used to describe the keys was $\mathbb{Z}/2\mathbb{Z}$ or $(\mathbb{Z}/2\mathbb{Z})^k$, depending on the length of the keys and messages. Therefore no expansion in the sums were to be considered. We may use another field to describe the keys where expansion in the sums would be considered.

We rely on superposed sending with expansion to enable each participant to compute a sum of inputs $p = \sum_n M_n$, where M_n is the input of participant P_n without any participant gaining knowledge about the individual inputs. The definition presented here is adapted from [18].

Let $\mathcal{P} = \{P_1, ..., P_n\}$ represent the set of participants. Let $\mathbb{F}_q$ be a finite field of prime order q such as $\mathbb{Z}/q\mathbb{Z}$ and G an undirected simple graph with nodes in P. Without loss of generality it will be assumed that the graph is connected since each connected component could be considered as a separate untraceable-sender system.

Each pair of participants P_i and P_j who are connected by an edge of G, generate a shared secret key, $k_{ij} \in \mathbb{F}_q$ with $k_{ij} = k_{ji}$. Let M_i denote P_i contribution to superposed sending. Each P_i, $i \in \{1, ..., N\}$ outputs:

$$o_i = M_i + \sum_{j|(i,j)\in G} \delta_{ij} k_{ij} \tag{10.3}$$

Each participant collects the outputs of all the others. It proceeds to compute:

$$p = \sum_{i=1}^{n} o_i = \sum_{i=1}^{n} M_i + \sum_{(ij)\in\emptyset} \delta_{ij} k_{ij} = \sum_{i=1}^{n} M_i \tag{10.4}$$

All keys k_{ij} cancel out as they are added once and subtracted once. Note that no information regarding the individual inputs is leaked. The reader should refer to [18] for a rigorous proof in the case the keys are described using a finite Abelian group.

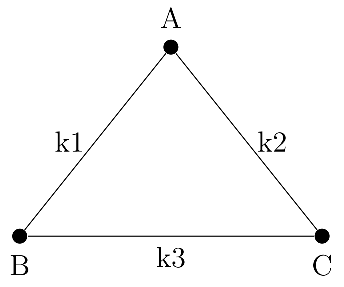

Figure 10.2: DC network with three participants

10.3 Proof of security

We present the possible attacks that can be performed on the DC network. The first one is a passive attacker able to control all of the outputs of the network. The second one is a subset of P called colluders that pool their keys in order to compromise the anonymity of a participant. The protocol is "unconditionally untraceable", which means that there is no mathematical way to determine the sender of a message, given the outputs of all participants. We won't prove only the unconditional untraceability of the network as a whole but we will also prove that the protocol is unconditionally untraceable among a connected component of the graph. This means that the colluders may partition the graph into connected components and that inside a given connected component, sender anonymity is preserved with respect to the colluders.

The first attacker is an eavesdropper able to tap and observe all of the lines of the network, such as a system administrator. He might see the individual outputs of every participant as well as the message sent. It is unconditionally infeasible for the eavesdropper to know by observing an individual output $\alpha_i = M_i \oplus \sum_{j\neq i} k_{ij}$ if M_i is equal to zero or not.

The second possible attackers are colluders. Colluding means that a subset $\mathcal{C} = \{C_1, ..., C_c\}$ of participants pool their key information in order to compromise a set of honest participants. A participant is compromised if and only if all of the keys he shares are known by $\mathcal{C}$ which is equivalent to say that all of his edges end on colluders. To preserve the identity of P_{i_1} it will be sufficient for him to share at least one key with an honest participant.

We give now the proof of the unconditional sender untraceability among a given connected component of the DC Network in presence of a given number of colluders. This proof is adapted from [3]. We assume the keys and inputs described in $\mathbb{Z}/2\mathbb{Z}$. The proof can be extended to any finite field. The reader will refer to [18] for the corresponding proof.

Theorem 10.3.1 Let $\mathcal{C}$ be the set of colluders knowing some set of keys. We remove the edges corresponding to these keys and consider any particular connected component G_L

of the remaining graph. The vertices of G_L form an anonymity set seen by the set of the keys known by the collusion. Then the only information that the collusion learns about the members of $\mathcal{C}$ is the sum of their individual inputs $\sum_{i \in G_L} M_i$.

The connected component is comprised of p vertices and q edges. The incidence matrix of G_L is denoted A with the vertices labeling the rows and the edges labeling the columns. Let K, I, and O be stochastic variables defined on $(\mathbb{Z}/2\mathbb{Z})^q$, $(\mathbb{Z}/2\mathbb{Z})^p$, $(\mathbb{Z}/2\mathbb{Z})^p$, respectively, such that K is uniformly distributed over $(\mathbb{Z}/2\mathbb{Z})^q$. K and I are mutually independent, and $O = (AK) \oplus I$. In terms of the protocol, K comprises the keys corresponding to the edges, I consists of the inversions corresponding to the vertices, and O is formed by the outputs of the vertices. Notice that the parity of O (i.e., the modulo two sum of its components) is always equal to the parity of I, since the columns of A each have zero parity. We give an example of the different parameters involved for a DC Network of three participants according to figure 10.2.

$$O = AK \oplus I = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} k_1 \\ k_2 \\ k_3 \end{pmatrix} \oplus \begin{pmatrix} M_1 \\ M_2 \\ M_3 \end{pmatrix} = \begin{pmatrix} k_1 \oplus k_2 \oplus M_1 \\ k_1 \oplus k_3 \oplus M_2 \\ k_2 \oplus k_3 \oplus M_3 \end{pmatrix}$$

The desired result is essentially that O reveals no more information about I than the parity of I. More formally:

Lemma 10.3.2 Let α be in $(\mathbb{Z}/2\mathbb{Z})^p$. For each i in $(\mathbb{Z}/2\mathbb{Z})^p$, which is assumed by I with non-zero probability and which has the same parity as α, $P\{O = \alpha | I = i\} = 2^{1-m}$. Hence $P\{I = i | O = \alpha\} = P\{I = i\}$. The knowledge of the output reveals no more information about the input than their overall parity.

Proof 10.3.3 Let $i \in (\mathbb{Z}/2\mathbb{Z})^p$ have the same parity as α. Consider the system of linear equations $(AK) \oplus i = \alpha$, in $k \in (\mathbb{Z}/2\mathbb{Z})^q$. Since the columns of A each have even parity, as mentioned above, its rows are linearly dependent over $(\mathbb{Z}/2\mathbb{Z})^p$. But as a consequence of the connectedness of the graph, every proper subset of rows of A is linearly independent. Thus, the rank of A is $p-1$, and so each vector with zero parity can be written as a linear combination of the columns of A. This implies that the system is solvable because $i \oplus \alpha$ has even parity. Since the set of n column vectors of A has rank $p-1$ the system has exactly 2^{q-p+1} solutions. Together with the fact that K and I are mutually independent and that K is uniformly distributed, the theorem follows easily.

Chapter 11

The algorithms

In this part, we firstly explain the notion of anonymity sets seen by a set of keys and show how the security of the DC-Network modeled as a key graph, is related to its degree sequence and connectivity. Secondly an algorithm to build a graph that has a given degree sequence if this sequence is graphic, is described. Finally we present a family of graphs that have a given connectivity with as few edges as possible.

11.1 Anonymity sets seen by a set of keys

As it has been explained in the previous chapter, the DC Network is modeled by a simple connected graph whose vertices represent the participants and an edge k_{ij} a shared secret key between P_i and P_j. There are two types of information injected in the network: the secret information which from the point of view of a participant P_i corresponds to the secret keys $k_{i,j}$ shared with other participants and to the individual input M_i that represents the message. And the public information, which corresponds to the topology of the graph (i.e. who shares keys with whom) and to the individual output of each participant i.e. $\alpha_i = M_i \oplus \sum_{j \neq i} k_{i,j}$. We will show in this part how the security of the system is related to the degree sequence and the connectivity. For that purpose, we will firstly define an anonymity set seen by a set of keys.

Let us consider a DC Network modeled according to the previous notations. We assume the graph to be connected since each connected component could be considered as a separate untraceable system. Let $\mathcal{C}$ be the set of colluders that cooperate by pooling their keys in order to trace the messages sent by some honest participants belonging to the of participants $\mathcal{P} - \mathcal{C}$. If c is the number of colluders, we assume have $1 \leq c \leq n - 1$. We consider the keys incident on the vertices of $\mathcal{C}$. By removing these keys or removing the set $\mathcal{C}$, one obtains a graph with a certain number of connected components. Such a connected component is called an anonymity set seen by the set of keys incident on the colluders. The main result which has been previously proved is that the only thing the colluders can learn about the members of an anonymity set is the sum of their individual inputs i.e. $\Sigma_{i \in G_i} M_i$ where G_i is the corresponding connected component, and doesn't know which participant in the connected component generated which message. This is intuitively apparent, since the messages of the participants in a connected component are each in effect hidden from the collusion by one or more unknown key bits, and only the sum of the individual inputs is known. Thus the messages are hidden by a one-time-pad, and only their sum is revealed.

In example of figure 11.1, we assume that $\mathcal{C} = \{P_1, P_2, P_3\}$ is the set of colluders. The anonymity sets seen by the set of keys incident to the set of colluders are thus $\{P_0\}$ and $\{P_4\}$. $\{P_0\}$ is a singleton anonymity set which means that he shares keys with all colluders. Therefore his anonymity is fully compromised. Indeed, By adding with xor, their keys with the output of $\{P_0\}$, the colluders can retrieve his initial input M_0 thus compromising his anonymity. To have a secure system if c colluders are present, we must ensure that no anonymity set seen by the set of keys incident on the colluders be a singleton set. A sufficient condition is that,

$$\forall i \in \{1, ..., n\}, d_G(P_i) \geq (c+1) \tag{11.1}$$

where $d_G(P_i)$ is the degree of P_i in graph G. This is not a necessary condition since there might exist in graph G a vertex P_i with $d_G(P_i) \leq c$ that could share at least one key with a honest participant. If $d_G(P_i)$ with $d_G(P_i) \leq c$ is the degree of a participant and c the number of colluders, we can comp

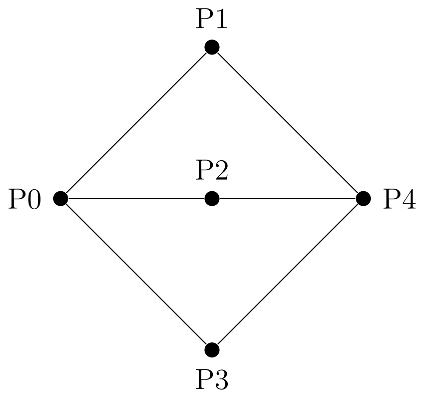

Figure 11.1: An example of anonymity sets

In example of figure 11.2, we assume that P_0 is an attacker. P_0 is a cut vertex of the graph and the anonymity sets seen by the set of keys incident on P_0 are respectively $\{P_1, P_2, P_3\}$ and $\{P_4, ..., P_7\}$. It is easy to see that by adding with xor, P_0 will deduce the sum of the messages of each connected component. According to the protocol, to send a message, only one participant at a time is allowed to speak. Therefore P_0 will know from which connected component originates the message. More generally, if $\mathcal{C}$ is the set of colluders knowing some set of keys, they will know from which anonymity set originates the message. Therefore to improve the security, one would like to minimize the number of anonymity sets in presence of a given number of colluders. To make the number of anonymity sets equal to one in presence of c colluders, a sufficient condition is that the graph be $c+1$ connected, since there won't exist a cut vertex set of size c. In this case a collusion would be prevented from partitioning the graph. We can note that, according to theorem, since $c+1$ connectivity implies that there exists between every pair of vertices, $c+1$ vertex disjoint paths, the previous condition on the degree sequence of the graph is ensured.

Those two examples show how the security is related to both the degree sequence of the graph and its connectivity. One would like to build such graphs while minimizing the number of edges, since for large systems it may be desirable to use fewer than the $n(n-1)/2$ keys required by the complete graph. The next section describes the algorithms that compute the graph with the condition on the degree sequence (each degree sequence larger than $c+1$) and the connectivity ($(c+1)$ connectivity) while minimizing the number of edges.

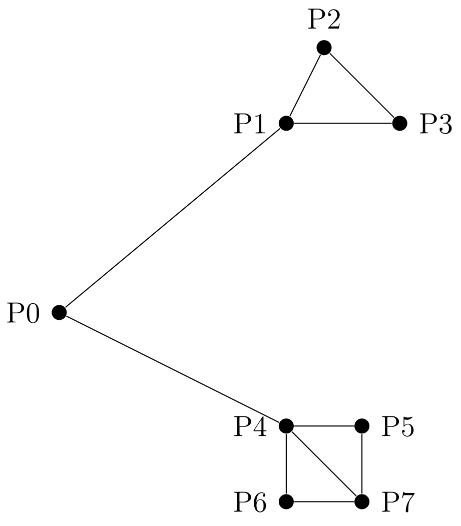

Figure 11.2: An example of anonymity sets

11.2 The degree sequence

Let $\mathcal{C} = \{C_1, ..., C_c\}$ be the set of colluders and we assume that $1 \leq c \leq n-2$. One wants to build a simple graph G with n vertices that would verify $\forall i \in \{1, ..., n\}$, $d_G(P_i) = c+1$. In this case fully sender anonymity would be preserved since every participant would share a key with at least one honest participant. Also the number of edges would be minimized since $\sum_{i=1}^{n} d_G(P_i) = 2m$ thus $m = \frac{n(c+1)}{2}$.

We thus have to verify if the sequence $d = (c+1, ..., c+1)$ is graphic. We will firstly prove the existence of such graphs using the theorem of Erdös and Gallai [26]. Secondly we will propose an algorithm that builds a graph with a given degree sequence [25]. Depending on the relative parities of n and c, the given sequence will not necessarily be graphic. In this case we will slightly modify the sequence to still have a cheap topology in terms of number of edges.

Let us present the theorem of Erdös and Gallai that gives a necessary and sufficient condition for a given degree sequence to be graphic.

Theorem 11.2.1 (Erdös and Gallai) Let $\Pi = (d_1, d_2, ..., d_n)$, be a decreasing sequence of n integers, with $d_1 \geq d_2, ..., \geq d_n$. Then there exists a simple graph whose vertices verify $\forall i \in \{1, ..., n\}$ $d_G(P_i) = d_i$ if and only if $\Sigma_{i=1}^{n} d_i$ is even and for each integer k, $1 \leq k \leq n$ we have

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} min\{k, d_j\}. \tag{11.2}$$

In the case we are interested in, we have $\sum_{i=1}^{n} d_i = nc' = n(c+1)$. Several cases are to be considered.

First case: n even or $c+1$ even:

Let $k \in \{1, ..., n-1\}$ then the left part of the theorem is

$$\textstyle\sum_{i=1}^{k} d_i = kc'$$

and the right part is

$$\begin{aligned} k(k-1) + \sum_{j=k+1}^{j=n} \min(k, d_j) &= k(k-1) + \sum_{j=k+1}^{j=m} \min(k, c') \\ &= k(k-1) + (n-k)\min(k, c') \end{aligned}$$

First subcase: $c' > k$

The right term in the inequality of Erdös and Gallai is:

$$k(k-1) + (n-k)k = k^2 - k + nk - k^2 = k(n-1)$$

We have $c \leq n-2$ thus $c+1 \leq n-1$ thus $kc' \leq k(n-1)$.

The inequality is verified.

Second subcase: $c' \leq k$

The right term in the inequality of Erdös and Gallai is:

$$k(k-1) + (n-k)min(k, c') = k(k-1) + (n-k)c'$$

Thus

$$kc' \leq (k-1) + (n-k)c' \Leftrightarrow c'(2k-n) \leq (k-1)$$

If $(2k-n) \leq 0$ then the inequality is obviously verified.

If $(2k-n) > 0$ then:

$$c'(2k-n) \leq (k-1) \Leftrightarrow c' \leq \tfrac{(k-1)}{(2k-n)}$$

Also

$$k \leq n-1$$

Thus

$$2k - n \leq k-1$$

Thus

$$k \leq \tfrac{k(k-1)}{(2k-n)}$$

And also

$$c' \leq k$$

Thus

$$c' \leq \frac{k(k-1)}{(2k-n)}$$

The inequality is verified.

We have for all $k \in \{1, ..., n-1\}$

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} min\{k, d_j\},$$
$$\text{and } \sum_{i=1}^{n} d_i \text{ is even}$$

Therefore the sequence $\Pi = (c+1, ..., c+1)$ is graphic and the number of edges of the graph is equal to $\frac{n(c+1)}{2}$.

Second case: n odd and $c+1$ are odd:

We cannot directly apply the theorem of Erdös and Gallai since Σ_{d_i} is odd. The trick consists in adding an edge to a vertex so that the condition on the degree sequence is still verified while minimizing the number of edges. We show in this case that the sequence $d = (c+2, c+1, ..., c+1)$ is graphic, making Σ_{d_i} even.

Let $k \in \{1, ..., n-1\}$ then the left part of the theorem is

$$\sum_{i=1}^{k} d_i = kc' + 1$$

and the right part is

$$\begin{aligned} k(k-1) + \sum_{j=k+1}^{j=n} min(k, d_j) &= k(k-1) + \sum_{j=k+1}^{j=m} min(k, c') \\ &= k(k-1) + (n-k)min(k, c') \end{aligned}$$

First subcase: $k \leq c'$

The right term in the inequality of Erdös and Gallai is:

$$k(k-1) + (n-k)min(k, c') = k(k-1) + (n-k)k = k(n-1).$$

We have

$$kc' + 1 \leq k(n-1) \Leftrightarrow c' \leq n - 1 - \tfrac{1}{k} \Leftrightarrow c' \leq n-2 \Leftrightarrow c \leq n-3$$

This is not verified if $c = n-2$ but in this case $c+1 = n-1$ is even because n is odd, which is by hypothesis excluded. Thus the inequality is verified.

Second subcase: $c' \leq k$

The right term in the inequality of Erdös and Gallai is:

$$k(k-1) + (n-k)min(k, c') = k(k-1) + (n-k)c'$$

Also

$$kc' + 1 \leq k(k-1) + (n-k)c' \Leftrightarrow c'(2k-n) + 1 \leq k(k-1)$$

If $2k - n \leq 0$, the inequality is verified because $2 \leq c+1 \leq k$

If $2k - n > 0$,

$$kc' + 1 \leq k(k-1) + (n-k)c' \Leftrightarrow c' \leq \tfrac{k(k-1)}{(2k-n)} - \tfrac{1}{(2k-n)}$$

Since $c' \leq k$, we have to see under which condition $k \leq \frac{k(k-1)}{(2k-n)} - \frac{1}{(2k-n)}$

If $c+1 < k$, since $k \leq n-1$ we have

$$k \leq \tfrac{k(k-1)}{(2k-n)}$$

Thus

$$\tfrac{k-1}{(2k-n)} \leq \tfrac{k(k-1)}{(2k-n)} - \tfrac{1}{(2k-n)}$$

Also

$$2k - n > 0$$

Thus

$$1 \leq 2k - n$$

Thus

$$c + 1 \leq k - 1 \leq k - \tfrac{1}{(2k-n)}$$

Remember that

$$c + 1 < k$$

Thus

$$c + 1 \leq \tfrac{k(k-1)}{(2k-n)} - \tfrac{1}{(2k-n)}$$

The inequality is verified.

If $k = c + 1$

$$\begin{aligned} & k \leq \frac{k(k-1)}{(2k-n)} - \frac{1}{(2k-n)} \\ \Leftrightarrow \quad & (2k-n)k \leq k(k-1) - 1 \\ \Leftrightarrow \quad & 2k^2 - nk \leq k^2 - k - 1 \\ \Leftrightarrow \quad & k^2 - nk + k + 1 \leq 0 \\ \Leftrightarrow \quad & k(k-n+1) + 1 \leq 0 \end{aligned}$$

We have

$$k \leq n - 1$$

Thus

$$k - n + 1 \leq 0$$

If $k - n + 1 < 0$ the inequality is verified.

If $k - n + 1 = 0$, $k = n - 1$ the inequality is not verified but we have in this case $c + 1 = n - 1$, thus $c + 1$ is even, which is excluded.

The inequality is always verified.

We now describe an algorithm that builds a simple graph with a given degree sequence if this graph exists. Recall that a partition of a number q into n parts $(d_1 + ... + d_n)$ is graphical if there exists a simple graph G whose points have degree d_i. If such a partition of $q = d_1 + ... + d_n$ is graphical, then q is even since in $d_1 = 2m$ according to theorem 8.4.1, and $\forall i \in \{1, ..., n\}$ $d_i \leq n-1$ since n is the number of vertices of G. Indeed a vertex is at most adjacent to all of the other vertices. But this condition is not a necessary and sufficient condition for a partition to be graphical. We now present a theorem giving such a condition [25].

Theorem 11.2.2 Let $\Pi = (d_1, ..., d_n)$ be a partition of an even number with $n - 1 \geq d_1, ..., \geq d_n$. Then Π is graphical if and only if the modified partition $\Pi_1 = (d_2 - 1, d_3 - 1, ..., d_{d_1+1} - 1, d_{d_1+2}, ..., d_n)$ is graphical.

This theorem gives an effective algorithm for constructing a graph with a given partition Π_0 if one exists.

Algorithm:

1. $\Pi = \Pi_0$.
2. Determine the modified partition Π^1 of Π as in the statement of the theorem, and reorder the terms of Π' so that they are non-increasing.
3. $\Pi = \Pi'$.
4. if $\Pi = (0, ..., 0)$, then halt. Otherwise, go to 2.

To illustrate this algorithm we test the partition $\Pi_0 = (5, 5, 3, 3, 2, 2, 2)$.

$$\begin{aligned}
\Pi &= \Pi_0 \\
\Pi' &= (4, 2, 2, 2, 1, 1) \\
\Pi &= \Pi' \\
\Pi' &= (1, 1, 1, 1, 0) \\
\Pi &= \Pi' \\
\Pi' &= (1, 1, 0, 0) \\
\Pi &= \Pi' \\
\Pi' &= (0, 0, 0) \\
\Pi &= \Pi'
\end{aligned}$$

The graph so constructed is shown in figure 11.3.

The algorithm can be applied to build a graph with a given degree sequence such as $(c + 1, ..., c + 1)$.

We can have an idea of the connectivity of such graphs, constructed using this algorithm. Since the graphs constructed using this algorithms are not unique, we can look for the ones that have the higher connectivity.

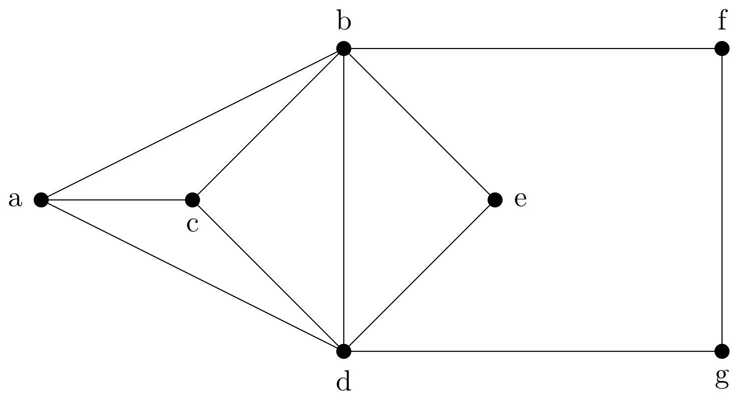

Figure 11.3: An example of the algorithm for graphical partitions

11.3 The connectivity

We now describe a family of simple k-connected graphs that have the minimum possible number of edges. According to Theorem 6.6.2, a necessary and sufficient condition for a simple graph to be k-connected is that any two distinct vertices P_i and P_j can be joined by k vertex-disjoint paths.

Thus $\forall i \in \{1, ..., n\}$

$$d_G(P_i) \geq k$$

and

$$2m \geq nk$$

where m is the number of edges of the graph, because

$$\sum_{i=1}^{i=n} d_G(P_i) = 2m_i$$

therefore

$$m \geq \lceil \tfrac{nk}{2} \rceil$$

where $\lceil \frac{nk}{2} \rceil$ is the ceiling of $\frac{nk}{2}$.

This number represents the lower bound on the number of edges for all k-connected graphs. We now present a family of k-connected graphs that reach this lower bound, i.e. that have exactly $\lceil \frac{nk}{2} \rceil$ edges. We shall show that this equality holds by constructing a k-connected graph $H_{k,n}$ on n vertices that has exactly $\lceil \frac{nk}{2} \rceil$ edges. The structure of $H_{k,n}$ depends on the parities of k and n.

Three cases are to be considered:

First case: k is even:

Then $H_{2r,n}$ is constructed as follows. It has vertices $x_0, ..., x_{n-1}$ and join each vertex x_i to the vertices of

$$\{x_j / j \equiv i \pm r'(mod.n); 1 \leq r' \leq r\}$$

Since each vertex has degree $2r$, the number of edges in the graph $H_{2r,n}$ is

$$m = \frac{1}{2}\sum_i d_{H_{2r,n}}(x_i) = \frac{nk}{2}.$$

It can be easily shown that this graph is $2r$ connected. The reader should refer to [26] for a rigorous proof. $H_{4,8}$ is shown in figure 11.4.

Second case: k is odd and n is even:

Then $H_{2r+1,n}$ is constructed by first drawing $H_{2r,n}$ and joining vertex x_i to vertex x_j if

$$j \equiv i + \frac{n}{2}(modn)\ j = 0, 1, ..., \frac{n}{2} - 1.$$

Thus, the number of edges of the graph is

$$m = \frac{1}{2}\sum_i d_H(x_i) = \frac{1}{2}(2r+1)n = \frac{nk}{2}.$$

It can be easily shown that this graph is $(2r+1)$ connected. The reader should refer to [26] for a rigorous proof. $H_{5,8}$ is shown in figure 11.5.

Third case: k and n are odd.

Then $H_{2r+1,n}$ is constructed by first drawing $H_{2r,n}$ and then adding edges joining vertex 0 to vertices $(n-1)/2$ and $(n+1)/2$ and vertex i to vertex $i+(n+1)/2$ for $1 \leq i \leq (n-1)/2$.

Then the number of edges of the graph is

$$m = \frac{1}{2}\sum_i d_H(x_i) = \frac{n(2r+1)+1}{2} = \lceil \frac{nk}{2} \rceil$$

Similarly, it can be shown that this graph is k-connected.

$H_{5,9}$ is shown in figure 11.6.

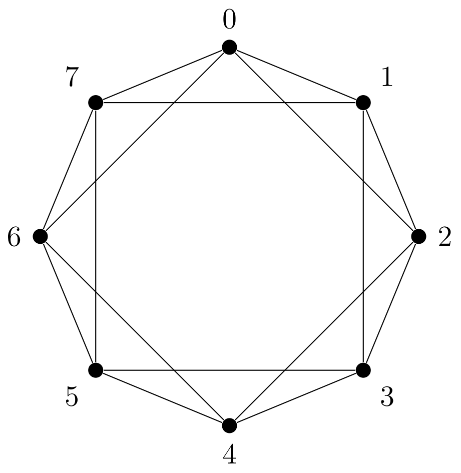

Figure 11.4: $H_{4,8}$

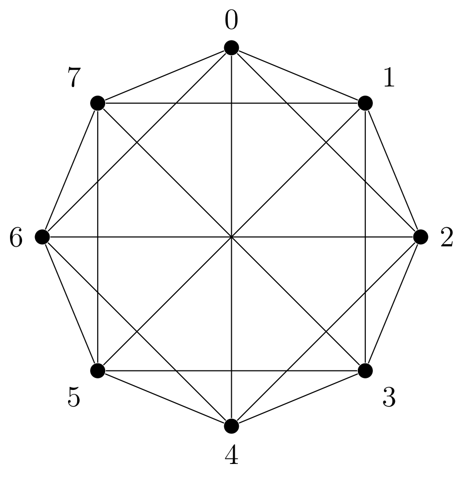

Figure 11.5: $H_{5,8}$

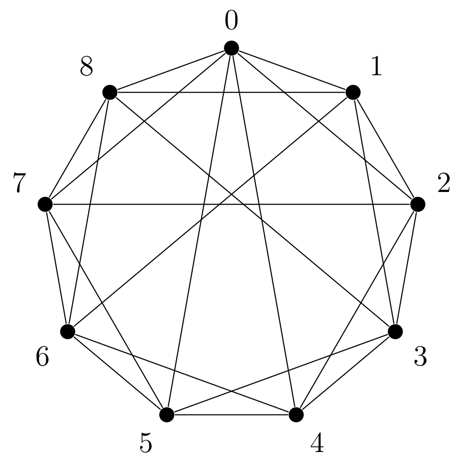

Figure 11.6: $H_{5,9}$

Chapter 12

Transposing the problem on to random graph theory

When deriving a protocol to build such key graphs, a problem arises if the set of participants is dynamic, that is if new participants want to join the network in order to send messages anonymously. One can easily notice that in the specific construction that has been presented, one has to recompute the whole topology of the graph if a new participant wants to join. An alternative to this problem relies on the use of random graphs. We will see that a threshold function on the number of edges to ensure k-connectivity of random graphs with high probability can be derived. In this case, a new participant willing to be a member of the DC-Network would join participants at random with a certain lower bound on his degree sequence. The new graph would still be random and ensure the condition on the connectivity. This prevents from reconstructing the whole topology. Therefore we present the basic concepts of random graph theory to make these considerations clearer, as well as the principal results on the k-connectedness of random graphs.

Chapter 13

Basic concepts on random graph theory

The theory of random graphs was founded by Erdös and Rényi [26]. The simplest model for random graph theory is the probability space consisting of all graphs with a given set of n labeled vertices and M edges, and each such graph is assigned the same probability. Usually we shall write G_M for a random element of this probability space. Then, if H is any graph with the given vertex set and M edges, then

$$P(G_M = H) = \binom{N}{M}^{-1} \tag{13.1}$$

where $N = \binom{n}{2}$.

In most cases we shall have a sequence of probability spaces. For each natural number n there will be a probability space consisting of graphs with exactly n vertices. We shall be interested in the properties of this space as $n \to \infty$. In this situation we shall say that a typical element of our space has a property Q when the probability that a random graph on n vertices has Q tends to 1 as $n \to \infty$. We also say that almost every (a.e) graph has property Q. Thus almost every G_M has property Q if the proportion of graphs with this property tends to 1 as $n \to \infty$. Once we are given such probability spaces of graphs, numerous natural questions arise. Is a typical graph connected? Is it k-connected? Those are the main properties we will be interested in.

The greatest discovery of Erdős and Rényi was that many important properties of graphs appear quite suddenly. If we pick a function $M = M(n)$ then, in many cases, either almost every graph G_M has property Q or else almost every graph fails to have property Q. In this vague sense, we have a $0 - 1$ law. The transition from a property being very unlikely to it being very likely is usually very swift. To make this more precise, consider a monotone (increasing) property Q, i.e. one for which a graph has Q whenever one of its subgraphs has Q. For many such properties there is a threshold function $M_0(n)$. If $M(n)$ grows somewhat slower than $M_0(n)$, then almost every G_M fails to have Q. If $M(n)$ grows somewhat faster than $M_0(n)$ then almost every G_M has the property Q. For example, $M_0(n) = \frac{1}{2}n \log n$ is a threshold function for connectedness in the following sense: if $\omega(n) \to \infty$ no matter how slowly, then almost every G is disconnected for $M(n) = \frac{1}{2}n(\log n - \omega(n))$ and almost every G is connected for $M(n) = \frac{1}{2}n(\log n + \omega(n))$.

We will present in the following the basic concepts of random graph theory, as well as a similar result for k-connected random graphs. We will show that one can build k-connected graphs in a random fashion that could model DC Networks, still maintaining an asymptotic improvement in terms of number of edges over the $\frac{n(n-1)}{2}$ edges required by the complete graph.

13.1 Models of random graphs

We will consider graphs with n vertices and take $V = \{1, 2, ..., n\}$ to be the vertex set. The set of all such graphs will be denoted by $\mathcal{G}^n$. Random graphs are considered to be simple graphs.

13.1.1 The first model: $\mathcal{G}(n, M)$

This first model consists of all graphs with vertex set $V = \{1, 2, ..., n\}$ having M edges, in which the graphs have the same probability. Thus with notation $N = \binom{n}{2}$, and $0 \leq M \leq N$, $\mathcal{G}(n, M)$ has $\binom{N}{M}$ elements and every element occurs with probability $\binom{N}{M}^{-1}$. Indeed $\mathcal{G}(n, M)$ has cardinality $\binom{N}{M}$, which corresponds to the possible number of choices of M edges among a set of $N = \binom{n}{2}$ edges. We notice that almost always M is a function of n: $M = M(n)$.

13.1.2 The second model: $\mathcal{G}(n, P(edge) = p)$

In the model $\mathcal{G}(n, P(edge) = p)$ we have $0 < p < 1$, and the model consists of all graphs with vertex set $V = \{1, 2, ..., n\}$ in which the edges are chosen independently and with probability p. In other words, if G_0 is a graph with vertex set V and it has m edges, then

$$P(\{G_0\}) = P(G = G_0) = p^m q^{N-m}. \tag{13.2}$$

where q stands for $1 - p$. We can notice that $\mathcal{G}(n, P(edges) = p) = \bigcup_{k=0}^{N} B_{k,n}$ where $B_{k,n}$ is the set of all graphs having n vertices and k edges. We can check that

$$P(\mathcal{G}(n, P(edges) = p)) = \sum_{k=0}^{N} \binom{N}{k} p^k (1-p)^{N-k} = 1 \tag{13.3}$$

Almost always p is a function of n: $p = p(n)$.

13.1.3 Basic definitions and results

We call Q a property of graphs of order n if Q is simply a subset of $\mathcal{G}^n$. The statement "G has Q" is then equivalent to $G \in Q$. A property Q is said to be monotone increasing if whenever $G \in Q$ and $G \subset H$ then also $G \in Q$. For instance, the property of containing a certain subgraph is monotone increasing.

If Q is any property, then $P_M(Q)$ has the natural meaning: it is the probability that a graph of $\mathcal{G}(n, M)$ belongs to Q. Of course $P_p(Q)$ is defined analogously. Let Ω_n be a model of random graphs of order n, we usually have $\Omega_n = \mathcal{G}(n, M)$ or $\Omega_n = \mathcal{G}(n, p)$. We shall say that almost every graph in Ω_n has a certain property Q if $P(Q) \longrightarrow 1$ as $n \to \infty$.

In most investigations, the models $\mathcal{G}(n, M)$ and $\mathcal{G}(n, p)$ are practically interchangeable provided M is close to pN. Erdős and Rényi [27] discovered the important fact that most monotone properties appear rather suddenly: for some $M = M(n)$ almost no G_M has Q while for 'slightly' larger M almost every G_M has Q. Given a monotone increasing property, a function $M^*(n)$ is said to be a threshold function for Q if

1. $M(n)/M^*(n) \longrightarrow 0$ implies that almost no G_M has Q, and,
2. $M(n)/M^*(n) \longrightarrow \infty$ implies that almost every G_M has Q.

For many property Q, it is possible to determine not just a threshold function but an exact probability distribution. We will be interested in the threshold functions for connectedness and k-connectedness.

13.2 The connectedness of random graphs

We present the following theorem that gives the threshold function for the connectedness of random graphs.

Theorem 13.2.1 Let $c \in \mathbb{R}$ be fixed and let $M = (n/2)\{\log n + c + o(1)\}$ and $p = \{\log n + c + o(1)\}/n$. Then

$$P(G_M \text{ is connected}), P(G_p \text{ is connected}) \longrightarrow e^{-e^{-c}} \tag{13.4}$$

$M_0(n) = \frac{n}{2}\log n$ is a sharp threshold function for the connectedness of a graph: if $M(n) = \frac{n}{2}\{\log n + \omega(n)\}$ then almost every G_M is connected if $\omega(n) \to \infty$ and almost no G_M is connected if $\omega(n) \to -\infty$.

We will present in the next section the corresponding result for the k-connectedness of random graphs.

13.3 The k-connectedness of random graphs

The following theorem gives the threshold function for the k-connectedness of random graphs.

Theorem 13.3.1 If $k \in \mathbb{N}$ and $x \in \mathbb{R}$ are fixed and $M(n) = (n/2)\{\log n + k \log\log n + x + o(1)\} \in \mathbb{N}$, then

$$P\{k(G_M) = k\} \longrightarrow 1 - e^{-e^{-x}/k!} \tag{13.5}$$

$$P\{k(G_M) = k+1\} \longrightarrow e^{-e^{-x}/k!} \tag{13.6}$$

$M(n) = (n/2)\{\log n + k \log\log n\}$ is a threshold function for $k+1$-connectedness of a graph: if $M(n) = \frac{n}{2}\{\log n + k \log\log n + \omega(n)\}$, then almost every G_M is $k+1$-connected if $\omega(n) \to \infty$, and almost no G_M is $k+1$-connected if $\omega(n) \to -\infty$.

It is intriguing to see that, concerning the property on the degree sequence, which is $Q = \{G \in \mathcal{G}; \delta(G) \geq k+1\}$ where $\delta(G)$ is the minimum degree of the graph G, then the threshold function is the same.

Chapter 14

Conclusion

We presented the existing proposals of anonymous communication systems that aim to foil traffic analysis. We saw that recipient anonymity can partially be provided by the use of broadcasting, but implicit addresses can give some substantial information to attackers. We presented the MIX-Network that provides a good computationally solution to the unlinkability of sender and recipient, but it is still sensible to powerful and determined active attackers. We described the DC-Network that is the only proposal for unconditional sender anonymity. However it is based on the assumption of a reliable broadcast network and needs a tremendous amount of key exchanges. An improvement to the latter has been proposed, whereas the former has been investigated in [18]. Onion routing and ISDN-Mixes introduced the notion of anonymous communication channels by distributing keys across the network in order to successively encrypt/decrypt the data stream, allowing for bi-directional and real-time anonymous communication (whereas MIXes work in a Store-and-Forward way). However, time correlation is difficult to prevent. None of these techniques are ideal, and realizing perfect sender-recipient anonymity over an open network in an efficient way is still an opened problem. The DC-Network constitutes a first step towards.

Bibliography

[1] D. Chaum, "Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms," *Communications of the ACM*, 24(2), Feb 1981, pp. 84-88.

[2] D. Chaum, "Security without Identification: Transaction Systems to make Big Brother Obsolete," *Communications of the ACM*, 28(10), 1985, pp. 1030-1044.

[3] D. Chaum, "The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability," *Journal of Cryptology*, 1(1), 1988, pp. 65-75.

[4] D. Chaum, "Blind Signatures for untraceable payments," *Advances in Cryptology, Proc. of Crypto 82*, Plenum Press, New York, 1983, pp. 199-203.

[5] D. Chaum, A. Fiat, M. Naor, "Untraceable Electronic Cash," *Advances in Cryptology-CRYPTO '88 Proceedings*, Springer-Verlag, 1990, pp. 319-327.

[6] C. Gülcü and G. Tsudik, "Mixing E-mail with Babel," *In 1996 Symposium on Network and Distributed System Security*, February 1996.

[7] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for Web Transactions," DIMACS Technical Report 97-15, AT&T Labs-Research, Murray Hill, NJ.

[8] P. Syverson, D. Goldschlag, and M. Reed, "Anonymous Connections and Onion Routing," *Proceedings of the Symposium on Security and Privacy*, Oakland, CA, May 1997.

[9] D. Goldschlag, M. Reed, P. Syverson, "Hiding Routing Information," in *Information Hiding*, R. Anderson, ed., LNCS vol. 1174, Springer-Verlag, 1996, pp. 137-150.

[10] A. Pfitzmann, "How to implement ISDNs without user observability-Some remarks," Institut für Informatik, University of Karlsruhe, Interner Bericht 14/85, 1985.

[11] A. Pfitzmann, "A switched broadcast ISDN to decrease user observability," *1984 International Zürich Seminar on Digital Communications*, IEEE, 1984, pp. 183-190.

[12] A. Pfitzmann, M. Waidner, "Networks without user observability-design options," *EUROCRYPT '85*, LNCS 219, Springer-Verlag, Berlin 1986, pp. 245-253; revised version: *Computers & Security* 6(2) (1987) pp. 158-166.

[13] D.J. Farber, K.C. Larson: Network Security Via Dynamic Process Renaming; *Fourth Data Communications Symp.*, Oct. 1975, Quebec City, Canada, pp.8-113..8-18

[14] M. Waidner: Datenschutz und Betrugssicherheit garantierende Kommunikationsnetze. Systematisierung der Datenschutzmaßnahmen und Ansätze zur Verifikation

der Betrugssicherheit; Diplomarbeit, Fak.f.Inform., Univ. Karlsruhe, Interner Bericht 19/85, Aug. 1985.

[15] B. Pfitzmann, M. Waidner, "Properties of Payment Systems - General Definition Sketch and Classification", IBM Research Report RZ 2823 (#90126) 05/06/96.

[16] A. Pfitzmann, B. Pfitzmann and M. Waidner, "ISDN-Mixes: Untraceable Communication with Very Small Bandwidth Overhead," *GI/ITG Conference: Communication in Distributed Systems*, Mannheim Feb. 20-22 1991, Informatik-Fachberichte 267, Springer-Verlag, Heidelberg 1991, pp. 451-463.

[17] A. Pfitzmann and B. Pfitzmann, "How to break the direct RSA-implementation of MIXes", *Advances in Cryptology-EUROCRYPT '89 Proceedings*, Berlin: Springer-Verlag, 1990, pp. 373-381.

[18] M. Waidner, "Unconditional Sender and Recipient Untraceability in Spite of Active Attacks," *Advances in Cryptology-EUROCRYPT '89 Proceedings*, LNCS, Springer-Verlag, Berlin 1990.

[19] J. Bos and B. den Boer, "Detection of disrupters in the DC protocol", *Advances in Cryptology-EUROCRYPT '89*, Springer-Verlag, 1990, pp. 320-328.

[20] M. Waidner, B. Pfitzmann, "The Dining Cryptographers in the Disco: Unconditional Sender and Recipient Untraceability with Computationally Secure Serviceability", *Advances in Cryptology-EUROCRYPT '89 Proceedings*, LNCS, Springer-Verlag, Heidelberg 1990, p. 690.

[21] R.L. Rivest, A. Shamir, L. Adleman, "A method for Obtaining Digital Signatures and Public-Key Cryptosystems", *Communications ACM* Vol. 21, No. 2, February 1978, pp. 120-126.

[22] A. Tanenbaum, "Computer Networks", third edition, 1996 by Prentice Hall, Inc.

[23] Kipp E.B. Hickman and Taher Elgamal, "The SSL Protocol". Internet draft draft-hickman-netscape-ssl-01.txt. 1995.

[24] J.A. Bondy, U.S.R. Murty, "Graph theory and applications", Elsevier Science Publishing Co., Inc. 1976.

[25] Frank Harary, "Graph theory", Addison-Wesley Publishing Company, Inc. 1972.

[26] Claude Berge, "Graphs and Hypergraphs", North-Holland Publishing Company, 1973.

[27] Béla Bollobás, "Random graphs", Academic Press, Inc. 1985.