APES

Anonymity and Privacy in Electronic Services

Deliverable 3 - Technologies overview

FINAL VERSION
12/11/2001

Bart De Win
Vincent Naessens
Claudia Diaz
Stefaan Seys
Caroline Goemans
Joris Claessens
prof. Bart De Decker
prof. Jos Dumortier
prof. Bart Preneel
Executive Summary

Anonymity and privacy have become important issues in the digital world. Various techniques that augment the level of anonymity are available, but their motivation and implementation is often based on an ad-hoc rationale, which makes it is hard to compare them. Moreover, reflecting upon and implementing improvements to the anonymity properties of a system is considerably complicated in this way.

In order to improve this situation, we present the first step towards a more solid foundation for the analysis, design and implementation of anonymity technologies. Anonymity techniques are often composed of several subcomponents that are each responsible for a particular anonymity aspect. In this deliverable, we focus on these basic building blocks. In this way, we will increase the understanding in the exact execution of existing anonymity techniques and enable a more uniform evaluation process.

In order to structure the description of basic building blocks, we first present a block taxonomy, which is mainly based on the distinction between connection vs. application-level blocks. For each block, we then describe its functionality and various other properties, such as requirements, anonymity type and performance. We also evaluate its correctness and security in an informal way. Hereby, we will only focus on building blocks that provide unconditional anonymity.

In the remainder of the document, we present the composition of basic building blocks, which is the key to build more powerful anonymity services. As an advantage, block composition often results in additional anonymity properties. We will describe composition requirements, dependencies and how it can be achieved using different composition strategies. Several examples are included to illustrate this process.
Contents

1 Introduction 1

2 Setting 3
  2.1 An overview of anonymity .......................... 3
  2.1.1 Model ............................................ 3
  2.1.2 Applications ...................................... 4
  2.1.3 Legal study ...................................... 5
  2.2 Attacker model ..................................... 5

3 Basic building blocks 7

4 Connection-level basic building blocks 10
  4.1 Overview ........................................... 10
  4.1.1 Changing appearance ................................ 10
  4.1.2 Changing flow .................................... 11
  4.1.3 Others ............................................ 12
  4.2 Properties .......................................... 13
  4.2.1 Performance ...................................... 13
  4.2.2 Attacks .......................................... 13
  4.2.3 Strength ......................................... 13
  4.2.4 Boundary behavior ................................ 14
  4.2.5 Requirements/Dependencies ....................... 14
  4.2.6 Intelligence/Sensitivity .......................... 14
  4.3 Description of blocks ............................... 14
  4.3.1 Encryption ....................................... 15
  4.3.2 Padding .......................................... 17
  4.3.3 Information substitution .......................... 19
  4.3.4 Compression ...................................... 21
  4.3.5 Reordering ....................................... 22
  4.3.6 Inserting latency .................................. 25
  4.3.7 Dummy traffic .................................... 27
4.3.8 No replay .................................................. 30
4.3.9 Filter ..................................................... 32
4.3.10 Caching .................................................. 33
4.3.11 Broadcast ............................................... 34
4.3.12 Multiplexing ............................................ 35
4.3.13 Bulletin board ......................................... 37
4.4 Summary .................................................... 38

5 Application-level basic building blocks ............................ 40
5.1 Introduction ................................................ 40
5.2 Properties .................................................. 40
  5.2.1 Assumptions and Requirements ....................... 41
  5.2.2 Unconditional Anonymity ......................... 41
  5.2.3 Trust ................................................. 41
  5.2.4 Efficiency ........................................... 41
  5.2.5 Security ............................................. 41
  5.2.6 Verifiability ....................................... 42
  5.2.7 Attacks .............................................. 42
5.3 Blind Signatures ......................................... 42
  5.3.1 Properties ........................................... 43
  5.3.2 Examples ............................................. 44
5.4 Fair Blind Signatures ..................................... 45
  5.4.1 Properties ........................................... 46
  5.4.2 Examples ............................................. 47
5.5 Group Signatures ......................................... 47
  5.5.1 Properties ........................................... 48
  5.5.2 Examples ............................................. 49
5.6 Threshold cryptosystems .................................. 49
  5.6.1 Properties ........................................... 50
  5.6.2 Examples ............................................. 51
5.7 Multi-Party Computation ................................ 51
  5.7.1 Properties ........................................... 51
  5.7.2 Examples ............................................. 52
5.8 Homomorphic Encryption .................................. 52
  5.8.1 Properties ........................................... 53
  5.8.2 Examples ............................................. 54
5.9 Secret Sharing Schemes .................................. 54
  5.9.1 Properties ........................................... 55
  5.9.2 Examples ............................................. 56
5.10 Untraceable Broadcast .................................... 56
  5.10.1 Properties ........................................... 56
CONTENTS

5.10.2 Examples .................................................. 57
5.11 Bulletin Board ........................................... 58
  5.11.1 Properties ........................................... 58
  5.11.2 Examples ........................................... 59
5.12 Interactive Proofs and Zero-knowledge .................. 59
  5.12.1 Properties ........................................... 61
  5.12.2 Examples ........................................... 62
5.13 Pseudonyms ............................................... 63
  5.13.1 Properties ........................................... 64
  5.13.2 Examples ........................................... 65
5.14 Deniable encryption .................................... 65
  5.14.1 Properties ........................................... 66
  5.14.2 Examples ........................................... 67
5.15 Trusted Third Parties .................................. 67
  5.15.1 Properties ........................................... 68
  5.15.2 Examples ........................................... 68
6 Composition of building blocks ......................... 69
  6.1 Connection-level composition .......................... 69
    6.1.1 Goal ................................................. 69
    6.1.2 Composition strategy .............................. 70
    6.1.3 Composition setup .................................. 71
    6.1.4 Case study: Onion Routing ....................... 74
  6.2 Application-level composition ......................... 78
    6.2.1 Multi-party Computation from Threshold Homomorphic Encryption ........................................ 78
    6.2.2 Fair Blind Threshold Signatures ................... 80

7 Conclusion and Future Work ............................... 82
Chapter 1

Introduction

As described in deliverable 2 of this project [38], various techniques exist to augment the level of anonymity in an application. Based on the type of application and its implementation, these techniques have different anonymity properties such as type, degree, strength etc. The motivation for and the implementation of these techniques is often based on concrete cases of exploiting privacy problems. This ad-hoc rationale thwarts the construction of a common and solid foundation, which makes it hard to compare different techniques. Moreover, apart from the lack of flexibility and extensibility of an ad hoc approach, reflecting upon and implementing improvements to the anonymity properties of a system is complicated considerably in this way.

In order to improve this situation, we take the first step towards a more solid foundation for the analysis, design and implementation of anonymity technologies. As will become clear during the rest of this document, anonymity techniques are often composed of several subcomponents, each responsible for a particular anonymity aspect. In this deliverable, we define and describe the basic building blocks distilled from existing anonymity techniques. A thorough insight in this matter will increase the understanding in the exact execution of existing anonymity techniques. Moreover, it will enable a more uniform evaluation process.

In order to structure the description of basic building blocks, we present a categorization of building blocks, mainly based on the difference in deployment location, i.e. the distinction between connection vs. application-level blocks. For each block, we analyze the existential requirements and various other properties, such as anonymity type, performance etc. For now, the correctness and security are evaluated in an informal way. Note that given the purpose of this deliverable, we will only focus on building blocks that provide unconditional anonymity.

Factorization of basic building blocks is not the silver bullet to the prob-
CHAPTER 1. INTRODUCTION

lems described in the previous paragraphs. The functionality of single blocks is very limited and they often depend on other blocks to ensure secure anonymity. Composition of basic building blocks is the key to build more powerful anonymity services. The advantage of block composition are the additional properties. We will analyze composition requirements and dependencies and strategies from a more theoretical point of view. Several examples are included to illustrate this technique.

The structure of this deliverable is as follows. In the following chapter, we first describe some general information that is necessary to situate this deliverable. Then, we describe the general rationale for the use of basic building blocks in anonymity services. A detailed discussion of the functionality and properties of the building blocks is separated into two parts, each described in a separate chapter. Afterwards, block composition is described in Chapter 6. And finally, a short recapitulation and conclusion complete this deliverable.
Chapter 2

Setting

2.1 An overview of anonymity

In this section we will summarize or restate a number of selected topics of the previous deliverable D2: Requirement Study [38]. This section is intended to familiarize the reader with a number of concepts that are explained in the previous deliverable.

Readers who are already familiar with deliverable D2 can skip the rest of this section.

2.1.1 Model

In Chapter 2 “Model used to describe anonymity properties” we presented clear definitions for several anonymity-related issues. These definitions provide the common terminology that is needed to compare different applications.

Concepts like , , and are described in detail, and the differences between these concepts are pointed out.

A second part of this chapter describes a model that can be used to categorize certain anonymity-related applications and building blocks. This model can be summarized as follows:

\[
\text{role} \quad \text{is} \quad \text{towards role}
\]

**roles** A number of roles can be identified in the applications we studied. From a functional point of view, the following roles can be present during the execution of an application:

**initiator** Initiates the application, usually with a request to some server.
**recipient or responder** The addressee of the request. A recipient is passive, while a responder react on the request by sending a reply.

**sender** Any entity that is used to send some message during the course of the application.

**receiver** Any entity that receives some message related to the application.

Other roles that can occur are:
this provider works as part of the anonymity service and thereby acquires information that could compromise the identity of the user, etc.

**Anonymity types** The types of anonymity we define are based on five properties we define:

1. The last two properties are unrelated to the first three and can be added if needed.

Depending on the selected properties chosen out of the first three, we get five useful types of anonymity:

<table>
<thead>
<tr>
<th>Properties</th>
<th>Type of anonymity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 2 and 3</td>
</tr>
<tr>
<td>2</td>
<td>1 and 2</td>
</tr>
<tr>
<td>3</td>
<td>2 and 3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

No anonymity whatsoever.

No anonymity, however, communication is not traceable.

Semi-anonymity this is a weak form of anonymity. The attacker needs some additional information not provided by the application to reveal any identifying information. This extra information can be shared keys, routing tables, etc.

Persistent anonymity or Pseudonymity the user hides his real identity by a pseudonym.

One-time anonymity the user hides his real identity behind a continuously changing pseudonym.

### 2.1.2 0

After the description of the model we are using to characterize different anonymity related properties, we have applied our model to a number of existing Internet applications.

We followed the same structural approach for every application we studied:
1. Description of the application at hand.

2. List of all the entities in the system and their .

3. Different anonymity requirements and/or properties this application has.

4. Short overview of existing solutions implementations of this application.

We have studied the following list of applications:

1. Anonymous Connections
2. Anonymous E-mail
3. Anonymous Publishing
4. Anonymous Browsing
5. Electronic Payments
6. Electronic Voting
7. Electronic Auctions

2.1. e l d

Finally the second deliverable also contained a short introduction to the possible legal issues these anonymous applications might put forward.

2.2 Attacker model

In the context of this project, it is important to set up an attack model in order to discuss the vulnerabilities and security properties of the described systems.

Using some of the properties described in [33] we can distinguish between the following attacker properties:

\[ \text{Ex} \quad : \text{An internal attacker controls one or several entities which are part of the system; an external attacker can only compromise communication channels.} \]
: A passive attacker can only listen to communication in a local point or globally; an active attacker is able to add, remove and modify messages.

\[ L \quad G \quad S \]

: A global attacker has access to the whole communication system, while a local attacker can only control part of the resources.

: Static adversaries choose the resources they compromise before the protocol starts and cannot change them once the protocol has started. Adaptative adversaries are allowed to change the resources they control while the protocol is being executed.

Different combinations of the previous properties are possible, for instance a global passive external attacker will be able to listen to all the channels, while a local internal active attacker can have full control of some of the resources and be unable to get any other information.

If a system with distributed secrets uses proactive security [24], then the shares of the secret are renewed periodically, and the attacker has to compromise enough resources within a period of time to break the system. If the attacker is not able to compromise enough shareholders during this period of time, all the information he has got is useless. Such an attacker has an additional time constraint.
Chapter 3

Basic building blocks

Various anonymity properties have been identified for modern distributed applications. In some applications e.g., e-mail, the initiator of a request wants to remain anonymous; in other applications e.g., web publishing, the responder wants to remain anonymous, etc. In order to meet these anonymity requirements, extra functionality must be implemented. This can be achieved either by adding it to the application, or through a separate dedicated anonymity provider.

Typically, the implementation of these techniques is performed in an ad-hoc manner. Code is added to the application in an incremental manner, not based on any solid foundation. However, anonymity properties often involve some similar functionality that can be reused. In this project, we try to identify this similar functionality by decomposing anonymity systems into smaller functional units. We then define anonymity building blocks, that can be reused for different systems. Hereby, an important question is the granularity of decomposition. In our opinion, the benefit of decomposition is maximal when striving for minimal, yet useful functionality for building blocks. Hence, we will focus on building blocks.

This decomposition strategy has several advantages. First, because of their simpler functionality, similar building blocks can be compared more easily than the global systems they originated from. Second, given a list of building blocks with their properties, deficiencies in existing systems can be discovered more easily. And third, implementing certain anonymity properties for a system will as such be reduced to composing an appropriate set of building blocks.

In this deliverable, we present an exhaustive list of building blocks. Their origin is twofold: either they are identified as being part of existing anonymizing services or they are proposed by this project as new, useful blocks. Besides a detailed description of their goal and functionality, we discuss useful
properties of each of these building blocks and illustrate them with some examples.

For the description of building blocks, we make a distinction between connection-level basic building blocks and application-level basic building blocks:

**Connection-level basic building blocks** These blocks are built in at the connection-level. They are used to set up an anonymous connection between two parties and to exchange messages along this anonymous connection. These building blocks are split up in blocks that change the appearance of messages and blocks that change the message flow. Connection level basic building blocks are discussed in Chapter 4.

**Application-level basic building blocks** These blocks are application specific. Although an application can use some of these building blocks, an application may probably need an anonymous connection in addition too. Application level basic building blocks are discussed in Chapter 5.

The following table lists the building blocks we will discuss in the rest of this text.
<table>
<thead>
<tr>
<th>name</th>
<th>application</th>
<th>connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>g</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>g</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$g$</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$fj$</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$g$</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$g$</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$x \cdot g$</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$g$</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$g$</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$g$</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>$w$ $g$</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Basic building blocks typically provide limited anonymity functionality. A reordering block for instance is not really useful if communication on a network can be eavesdropped. In order to construct more powerful anonymity services, building blocks should be composed. We will study and discuss composition of the basic building blocks in Chapter 6.
Chapter 4

Connection-level basic building blocks

The primary goal of basic building blocks that operate on the connection or network level is to hide or remove identifying information that is available at this level. Roughly speaking, the source of this information is twofold. First, network packets contain certain information. This explicit information is for instance an IP-address contained in the header of a network packet or a person’s name in the body of the packet. Second, normal network operation might result in more implicit information. Examples of the latter are the size or the flow of network packets, which can both help to trace the recipient of a packet.

In this chapter, we will first categorize connection-level building blocks into a taxonomy, which is mainly based on the functionality and the mode of operation of the different blocks. Then we will define several properties that are used to structure the discussion of the building blocks. Afterwards, we will describe in detail each of the building blocks and finally we will conclude this chapter with an overviewing summary of their properties.

4.1 Overview

Based on the functionality of the building block, we distinguish between the following categories:

4.1.1 Here

A potential privacy problem regards the information contained in packets that are sent over a connection. As explained before, their contents might
reveal specific information. But also their format and size might help to identify and trace certain packets.

To anticipate attacks that try to use this kind of information, the form of the packets must be changed. This can be achieved by changing the actual contents of the message contained in a packet, but this is not required. For instance, one could include dummy information in a packet and change this information regularly. The following blocks modify the format of a packet:

- encryption
- padding
- information substitution
- compression

4.1.2 Ch flow

Without knowing the exact contents of a network packet, an adversary might still be able to trace a packet using the characteristics of existing network infrastructure. For instance, a network router will typically handle packets in a FIFO First In First Out order. Based on this knowledge, eavesdropping the input and output of routers will make a connection traceable. Changing only the form of packets is not sufficient to disallow these kind of attacks. Therefore, the order of packets in the network should be changed.

Changing the packet flow can be achieved by changing the order of packets or by changing the communication model used to forward the packets. We distinguish between push based and pull based models.

4.1.2.1 Push based

In push based models, a building block receiving a message takes the initiative to change the flow and to forward the messages to the next hop in a connection. As such, we can think of an active building block. The building blocks we will discuss, are:

- reordering
- inserting latency
- dummy traffic
- no replay
filtering

caching

broadcast

multiplexing

4.1.2.2 Pull based

In pull based models, the communication model is totally different. Here, a basic building block that receives a message does not take the initiative to forward it himself. Rather, other blocks waiting to communicate poll this block to check whether new communication data is available. As such, we can speak of a passive building block. We will discuss one building block in this category, i.e. the bulletin board.

4.1. Other

Some basic building blocks do not fit into the previous categories. We will discuss one such block that is used to process reply information.

Figure 4.1: Connection-level building block taxonomy
4.2 Properties

To describe and compare the different basic building blocks, we focus on the following properties.

4.2.1 Performance

Four different properties are used to evaluate the performance penalty of a block:

1. **Communication overhead**: the amount of extra communication introduced by this building block on top of the normal communication operation.

2. **Computational overhead**: the amount of computation performed by the block in order to execute its task. Encryption blocks for instance require much extra computation.

3. **Message delay**: the amount of time a message is delayed by a building block, i.e., compared to normal communication operation. Packet reordering for instance often involves a considerable message delay. Note that the result of this property and the previous one is the same: a delay of a packet. However, the reason for the delay is clearly different for both properties.

4. **Storage**: the requirements for and the consequences of the storage capacities of a block.

4.2.2 Security

Some blocks are designed especially to anticipate particular attacks. In that case, we describe the attacks.

4.2.3 Reliability

During the analysis of several blocks, we clearly noticed a difference in quality of similar building blocks. Two blocks can achieve the same goal, but differ in their exact algorithm or in specific parameters. This is what we mean by anonymity strength. To avoid ignoring this important information, we define a separate property for this purpose. Note that this is a subjective property, as you might notice in the rest of this chapter. We will try to formalize this property in the future.
CHAPTER 4. CONNECTION-LEVEL BASIC BUILDING BLOCKS

4.2.4 Block behavior

While blocks typically operate well under normal circumstances, usage at or even beyond the operational limits often results in unpredictable behavior. Since the latter must be avoided at all times, we research and describe what happens for a block at its boundaries. For message flow blocks for instance, we analyze its behavior for very few and very many messages.

4.2.5 Required/Defined

This property describes existential and/or operational requirements of building blocks. For instance, a reordering block might require a secure random generator. Or a caching block will require information about the return path, as known by the reply information block. Operational requirements often correspond to inter-block dependencies. In general, requirements typically consist of connection-level blocks, application-level blocks or other technological requirements.

4.2.6 Iell/Sev

Some building blocks require understanding of the communication that is processed. Others have access to more confidential information as a result of their functionality and are therefore subject to attacks. For instance, a building block that knows the sender and recipient of a message is much more sensitive than a building block that only inserts dummy traffic. With this property, we try to express the intelligence and the sensitivity, or the probability of attacks, in an informal manner.

4.3 Description of blocks

In general, the building blocks described in this chapter operate on the level of connections. A connection is a network level entity that is first explicitly constructed, after which streamed data can be forwarded over it. In order to guarantee the seamless integration of building blocks into an existing connection, they should hence fit into this streaming process. For this purpose, we have defined the interface of connection-level building blocks to be fixed: input, processing step and output.

To visualize the functionality of building blocks, we will use the following notation: see figure below. The left arrow denotes the incoming messages of the building block. The box contains the information that is necessary for execution. Some of this information might be stored by the building
block, which is written inside the block. The right arrow finally represents
the outgoing messages of the block.

Using this notation, we will abbreviate some common terminology, among
which:

A: Answer of a request
C: Content body of a message
M: Message
P: Padding information
R: Recipient
S: Sender
T: Time
D M /E M : Decryption/Encryption of a message M

4.1 Error

4.3.1.1 Description

Encryption is the ciphering and scrambling of information. This building
block encrypts or decrypts incoming messages before forwarding them on the
output stream. If a message is encrypted at some building block, another
building block must be able to decrypt the message.

An encryption system is called secure if seeing the encrypted message does
not give any partial information about the message, that is not known be-
forehand. Ideally, an encryption building block should use secure encryption.
Note that if we have only a number of possible messages a deterministic
public-key encryption system will not be secure, because the enemy may try
all the possible inputs. In those cases, random information must be added
to the message to be encrypted.
4.3.1.2 Properties

**Performance** Encrypting/decrypting messages includes some computational overhead and thus, some short delay. Sometimes encryption expands the length of the message leading to some communication overhead. No storage space is required.

**Attacks** Encrypted messages have some properties that are useful to set up an anonymous communication:

- **changing form** If a message is encrypted, the form of an incoming message differs from the form of an outgoing message. This makes it more difficult for an attacker to trace the route of a message.

- **information hiding** If the identity information of the message is encrypted, this information is not revealed towards attackers on the output stream.

- **replay attack** An attacker can replay a message to this building block. Using a non-secure encryption scheme, each replayed message appears on the output stream in the same form if encrypted with the same key. To decrease this kind of attack, encryption keys are changed at intermediate times.

**Strength** The strength of this building depends on the key length, the encryption algorithm and the information that is encrypted.

**Requirements** Keys must be secure and changed at regular times.

**Intelligence/Sensitivity** The collected keys must be trusted. The building block must know which key to use in order to encrypt the message.

4.3.1.3 Specific properties

**Computational hardness** No efficient algorithm can gain any partial information about the message. On the other hand, the encryption and decryption procedures are required to be efficient.

**Security with respect to any probability distribution of messages** The encryption system should be secure independently of the probability distribution of the messages we encrypt.
CHAPTER 4. CONNECTION-LEVEL BASIC BUILDING BLOCKS

Hardness against a-priori information  Even if the attacker has some a-priori information about the message it will not help him to achieve significant information about the message which he cannot get from the a-priori knowledge itself.

4.3.1.4 Examples

The encryption building block is present in nearly every system.

4.2 P dd

4.3.2.1 Description

\[ M = [S, C, R, P] \quad \rightarrow \quad M' = [S, C, R, P'] \]

Different messages have different sizes, which do not change during transmission. So, these messages can be traced by size. In some protocols, messages may increase/decrease in size by a small and approximately known amount at each hop. Even if a message is well mixed with the other messages in the mix, and even if they are all different sizes, they are still distinguishable.

To make outgoing messages unlinkable to incoming messages, padding information is added to or removed from the message before forwarding the message on the output stream. Two techniques exist:

fixed length messages  Padding information is used to keep the length of the message fixed.

- If information is added to the message at this entity e.g., return path info, the amount of padding decreases at this entity to keep the message length fixed.

  example: \[ P \rightarrow P' | where |P| > |P'| \quad Y = \text{return path info} \]

- If information is removed from the message at this entity e.g., next hop info, the amount of padding increases at this entity to keep the message length fixed.
example: \[ ZP \rightarrow [ P'] where |P| < |P'| \quad Z = \text{next hop info} \]

Remark 1: The length of an original message can be larger than the fixed length. Large messages are split up in shorter messages of equal length. This way, little messages do not have a lot of padding and very large messages can also be handled. These shorter messages must be brought together at the end.

Remark 2: Instead of using one fixed length for all messages, some fixed lengths can be placed: one length for little messages, one for normal messages and one for large messages.

**variable size messages** A random amount of padding is added to or removed from the message. This confuses an eavesdropper.

\[ P \rightarrow [ P'] where |P|, |P'| \text{ are random amounts of padding.} \]

### 4.3.2.2 Properties

**Performance** Padding information may increase communication overhead if messages grow. Larger messages require more storage space. The performance penalty is minimal if only one fixed message length is allowed and increases if more different message lengths are allowed. This building block does not include any computational overhead or message delay. The computational overhead is minimal. Inserting/removing padding information can be done in a minimum of time.

**Attacks** Fixed length messages make it impossible to trace the messages based on their length. So, this building block helps to provide untraceability of the messages.

**Strength** If all messages are equal length, it is impossible to trace a message based on the length. This is the ideal case. This case is not always realistic according to performance.

Sometimes, different message lengths are permitted. The anonymity strength decreases when the number of permitted message lengths increases.
If a random amount of padding is inserted/removed, the strength of the building block depends on the strength of the random generator.

**Intelligence/Sensitivity** The building block must calculate the amount of padding that must be added to the message in order to achieve a fixed message length. If more fixed lengths are allowed, the system may choose not to add a larger amount of padding than is really needed.

**4.3.2.3 Examples**

Type 2 remailers [10] define three length formats. Padding is appended to achieve one of these message lengths.

4. I form o b o

**4.3.3.1 Description**

\[ M = [S, C, R] \rightarrow \quad M' = [S', C, R'] \]

As the name already suggests, this building block substitutes information in order to hide the identity of some communicating party. Two variations exist:

**replacing identity information** The identity information is replaced by other information that hides the identity of that party.

**blanking identity information** The identity information of some party is blanked.

Information substitution can be provided both at connection-level and at application-level. We discuss both of them:

**at connection-level** The connection level information of some communicating party is substituted by other information. This hides the location of the party towards attackers.

**at application-level** The application level consists also information about the communicating parties. In current systems, only message headers are substituted. Remark that the message itself can also contain identity information. No system handles this problem. The initiator should
be aware that his message should not contain identity information. Although this is maybe possible, each author has his own characteristics to write messages. This way, attackers can maybe link messages based on this evidence.

The anonymity type of this building block depends on the way information is substituted. Two types of anonymity can be provided:

**persistent anonymity or pseudonymity** The identity information is substituted by other information that remains the same for all transactions made by the same party. This way, different transactions made by the same user can be linked but the real identity of that user is unknown after the real identity information is replaced.

**one-time anonymity** The identity information is substituted by other information that is different for each transaction made by the same party. This way, different transactions made by the same party cannot be linked any more after the information is substituted. Blanking the identity information also provides one-time anonymity.

### 4.3.3.2 Properties

**Performance** Substituting identity information does not affect the performance of the system if the identity information is substituted by random information one-time anonymity. If persistent anonymity is provided, the building block keeps a table that maps identities into pseudonyms. This table requires some storage space that increases if the amount of users increases. Finding the right corresponding pseudonym also takes more time if the table increases. This building block has practically no communication overhead.

**Attacks** This building block makes it more difficult to reveal the identity of the communicating parties after the information is substituted. However, if no other technique is used in addition to information substitution, an attacker can still trace the message from the initiator towards the responder.

**Strength** The anonymity strength depends on what information is substituted. For a good anonymity strength, all identity information at connection-level and at application-level must be substituted and the building block must be trusted.
CHAPTER 4. CONNECTION-LEVEL BASIC BUILDIN BLOCKS

requirements This technique should be used in combination with other techniques. If this building block flushes out messages immediately after the identity information is substituted, it is easy to link the incoming and outgoing message.

Intelligence/Sensitivity The mapping table forms the core element in case of pseudonymity. It is clear that this table may not be compromised.

4.3.3.3 Examples

All systems use this building block in some way to hide the party that wants to be anonymous.

4.4 Compressed

4.3.4.1 Description

```
\[ M \rightarrow \text{\text{map}} \rightarrow M_{\text{comp}} \]
```

A compression algorithm diminishes the size of a collection of bytes. While compression is typically used to decrease the size of files, it can also be used for messages in a connection. Between two connection routers, the contents of a message can be compressed. This changes the appearance of the message.

Some properties of compression algorithms should be discussed in this scope. First, compression can be performed lossy or lossless. It is clear that for this case, the latter is preferable. Second, algorithms sometimes allow choosing between speed of compression versus compressed size. This choice is certainly interesting. From a more generic point of view, compression algorithms can be designed to be tunable, which can improve the anonymity properties if the configuration is changed often enough.

4.3.4.2 Properties

Performance

This building blocks introduces no communication delay. On the contrary, by decreasing the message size it speeds up communication.
Depending on the specific algorithm, compression sometimes requires computation to construct and apply a useful transformation for a specific message. As a sidenote, it might be interesting to compare the computation overhead of encryption and compression. At first sight, it is hard to predict what will be most performant, since this heavily depends on specific algorithms, parameters etc.

Besides the previous computation, compression introduces no extra communication delay.

Most compression algorithms do require some storage during compression, but not between different compression rounds. Therefore, a compression building block will not have special storage requirements.

4. .5 Reorder
4.3.5.1 Description

As its name already suggests, this building block is used to change the order of messages before forwarding them from input to output in order to provide message untraceability. An attacker cannot assume that the first message in the input stream is the first message on the output stream. Although applicable in different scopes, it is normally used on the level of data connections, where its goal is to anticipate traffic analysis attacks. Using this block, it becomes harder to trace the message flow just by looking at the communication. Reordering always involves keeping some number of messages in the system at all times. These messages are called the message “pool”. Different strategies exist to reorder messages. We discuss three of them:

random sequence number This straightforward scheme consists of generating random sequence numbers for the messages in the input stream. A message leaves the message pool if it has been assigned the sequence number that is one higher than the last message that left the pool. A message that arrives earlier in the input stream than another one can be forwarded later and vice versa. The arrival of a message does not necessarily result in the leaving of a message. If the arrived message has been assigned the lowest sequence number, it can result in the leaving
of more than one message. Therefore, a small set of sequence numbers is used.

**one in one out** Another reordering scheme is to keep $N$ messages in the pool, and to send out one of the $N+1$ messages in the pool including the one that just arrived chosen at random. This implies that $N$ messages are always present in the message pool. In the worst case, a message is kept in the message pool forever. An upper limit can be set on the number of messages that leave the pool before a particular message.

**all but N out** Another reordering strategy periodically sends out all but $N$ messages in the pool rather than sending out one message from the pool each time a new message arrives.

While these strategies provide some anonymity service, they might have disadvantages for specific protocols. For example, the result of web browsing would look very different if the text and other content of the requested webpages would be mixed up by these reordering strategies. Therefore, more advanced reordering strategies could focus on such properties by for instance only reordering messages originating from different connections.

### 4.3.5.2 Properties

**Performance** This building block has no remarkable computational overhead. The reordering algorithms can be executed within a minimum of time. Second, reordering has no communication overhead. For each incoming message, only one outgoing message is generated. However, this building block needs some storage space for storing the message pool. The storage space increases if more messages are kept in the pool. Reordering also implies some delay. For some algorithms, the amount of delay is predictable; for others, it is not. The amount of delay depends on the incoming traffic. If the incoming traffic is very heavy, reordering can take place faster.

**Attacks** Reordering alone does not provide full untraceability towards everyone. For instance, attacks based on the contents of the ingoing and outgoing message stream can still trace the message flow. However, reordering resists some particular attacks. These attacks are outlined below. If it is used as part of a communication channel, this channel also helps to provide unlinkeability of sender and receiver with respect to external parties.
sequence based attack The main goal of reordering is to resist attacks based on the sequence of incoming and outgoing messages. Without reordering, the attacker knows that the first message in the input stream is the first message in the output stream. With reordering systems, an outsider cannot base his attack on this property. Each reordering system should resist this kind of attack. Moreover, some reordering techniques provide anonymity against some additional attacks.

spam attack Some reordering strategies also provide resistance against “spam” attacks in which an attacker sends many more than N messages to the system. In some schemes, these messages will displace all the real messages in the pool, leaving only messages which the attacker can recognize. If many more than N messages are sent to a reordering system then the pool of messages will contain only planted messages which can be recognized. Since the attacker can recognize his own message, yours will be obvious. Strategy 1 random sequence number is susceptible to this kind of attack. Strategy 2 one in, one out provides some protection against this attack because a message is possibly kept in the pool forever. Strategy 3 all but N out also provides some protection against this kind of tracing attack. If, during an average period, several real messages have arrived, then even if the pool of messages is flushed out, there will be more than just your message mixed in with the attacker’s messages.

Strength The strength of the anonymity provided by this building block depends heavily on the strategy used to reorder the packets. In order to be perfectly secure, it should depend on a secure random generator. Besides a secure random generator, the anonymity depends on three other factors:

size of message pool In some strategies, the size of the message pool has a fixed length. The anonymity strength fully depends on this size. If this size $N = 0$, messages in the input stream are set directly on the output stream. There is no reordering. The anonymity strength increases when $N$ grows.

amount of delay Strategy 3 all but N out keeps messages in the pool and reorders the incoming messages at regular times. If this period is long enough, more messages are present in the message pool. So, more messages can be reordered.

amount of traffic The strength of strategy 3 also depends on the amount of traffic. The amount of traffic influences the size of the message pool.
CHAPTER 4. CONNECTION-LEVEL BASIC BUILDIN BLOCKS

directly. If only one message arrives at the message pool during the period, no reordering takes place.

**Boundary behavior** This technique works well with normal and high traffic. However, if the amount of incoming traffic is very low, it is very difficult to reorder the message with other messages in real-time. In that case, the message can be mixed with dummy traffic. The communication overhead caused by dummy traffic will not be a bottleneck when the amount of traffic is low. However, reordering is very useful with a large amount of incoming traffic because this technique does not imply any communication overhead.

**requirements** Reordering building blocks are only useful if the incoming and outgoing messages have a different appearance. Otherwise, it is trivial to trace the message flow based on their content. Reordering relies on the existence of connection oriented mechanisms.

**Intelligence/Sensitivity** The message pool and the random generator may not be compromised. Otherwise, an attacker can map incoming and outgoing messages although they are reordered.

4.3.5.3 Examples

A lot of systems use reordering for setting up anonymous connections. However, only type 2 remailers [10] give an overview of some techniques that can be used for reordering.

4.6 Immediate

4.3.6.1 Description

This building block is used to store an incoming message for some time before it is forwarded on the output stream. In the normal execution flow without latency, an incoming message is immediately flushed out. This way, an attacker can trace the message based on this trivial property. Inserting latency decreases the possibility for the attacker to trace the message. The amount of latency must be chosen at random and must be different for each
message. If a fixed latency is chosen for all incoming messages, inserting latency is worthless. Generally, these techniques are used:

delay appended to the message The amount of time an incoming message must be kept before it is flushed out is appended to the message itself before it is sent by the initiator. The building block analyses the delay and keeps the message during the indicated time. For e-mail applications, the amount of delay that is appended to the message can be larger than for real-time applications such as web browsing.

delay calculated at arrival Another technique is to calculate a random delay when the message arrives. This way, the initiator cannot predict for how long the message is delayed. Because this technique has to work with all types of applications also real-time applications on top of it, the delay may not be large.

A combination of these two techniques should be to append an upper bound to the message for the amount of time an incoming message is kept before it is flushed out. For real-time applications, this upper bound will probably be smaller than for other applications.

Because the delay is a random chosen time, an implication is that messages are reordered. A message that arrives later than another message but has less latency can be flushed out earlier, and vice versa.

4.3.6.2 Properties

Performance This building block has no communication overhead. Computational overhead sometimes is needed to calculate the amount of delay. Additional storage space is needed to store the messages for some period of time. If the average amount of delay is large, more message will be stored at this building block.

Attacks Inserting latency is also a building block to provide untraceability of the message. Just like reordering, inserting latency does not provide full untraceability but is resistant against some particular attacks executed to trace the message. Inserting a random delay makes it difficult for an attacker to trace messages on the basis of their arrival time and departure time. If the upper bound of latency is known by the attacker, he could consider all messages that are flushed out within a given interval.
CHAPTER 4. CONNECTION-LEVEL BASIC BUILDIN BLOCKS  27

Strength  The anonymity strength depend on two factors:

random generator  The random generator must be perfectly secure. Otherwise, an attacker can predict the exact delay.

time range  The time range over which a message can be delayed. The larger this range, the more difficult it is to predict the leaving time of a message. One statement says: 'the delay must be longer than the time between message arrivals then it is impossible to know with certainty which incoming message corresponded to which outgoing message.'

Boundary behavior  If the amount of incoming traffic is very large, the amount of average delay can be shorter. If only one message a day arrives at this building block, the delay must be extremely large to be effective.

Requirements  Delaying building blocks have the same requirements as reordering. The incoming and outgoing messages should have a different appearance. Otherwise, it is trivial to trace the message flow based on their content. Delaying relies on the existence of connection oriented mechanisms.

Intelligence/Sensitivity  If the amount of delay is appended to the message, the originator of the message must append the amount of delay to the message. If the amount of delay is calculated by the building block, that building block must choose a good delay. The amount of delay for real time applications has to be small. It may be larger for other applications e.g. e-mail. The amount of delay can also be smaller if there is a lot of traffic and has to be larger if there is little incoming traffic.

4.3.6.3 Examples

Type 2 remailers [10] append the amount of latency to the message before it is sent. This latency can be some seconds, minutes or even hours dependent on the wishes of the initiator. Other systems frequently use latency but the amount of latency depends on the entity that keeps the message for some time.

4. .7 D mm  r ff

4.3.7.1 Description

dummy traffic insertion
CHAPTER 4. CONNECTION-LEVEL BASIC BUILDING BLOCKS 28

The intention of this building block is to send dummy traffic through the network in addition to normal messages in order to provide untraceability between the initiator and the responder. Its goal is to anticipate traffic analysis attacks. If dummy traffic is put on the output stream in addition to normal traffic, it is more difficult for an attacker to trace the route of normal messages. Inserting random traffic becomes more important when the amount of normal traffic decreases. A counterpart building block must be introduced to remove dummy traffic.

To make it possible for a receiver to distinguish between normal traffic and dummy traffic, MAC's are used in combination with this technique. If the receiver detects a bad MAC, the message is considered as dummy traffic. Rivest discusses this problem in [45]. An attacker may not distinguish between normal traffic and dummy traffic.

4.3.7.2 Properties

**Performance** This technique implies that more traffic is sent through the network than just the original messages. However, inserting dummy traffic has no real communication overhead because dummy traffic is only useful if the amount of traffic is very low for some time. In general, dummy traffic is added in order to keep the amount of communication constant. Other techniques e.g., reordering exist to provide untraceability when the amount of traffic is very high. In this case, very little or no dummy traffic is inserted. There is some computational overhead to decide when dummy traffic is useful. Additional storage space is not needed because dummy traffic can be flushed out immediately after generation. This building block decreases the delay of real messages because they can be mixed faster with dummy traffic.

**Attacks** The goal of inserting dummy traffic is to provide untraceability between the initiator and the responder. Just like reordering and latency
insertion, inserting dummy traffic does not provide full untraceability but is resistant to a particular attack. In the normal execution flow without random traffic, an attacker knows that each incoming message is followed by an outgoing message. This statement is not true if random traffic is inserted.

**denial of service attack** An attacker can deny all traffic in the network except the message he wants to trace. Without random traffic, there will only be one outgoing message that corresponds with the traced incoming message. When random traffic is inserted, more outgoing messages are generated and, if additional techniques are applied e.g., encryption of the real message, the attacker can not distinguish between the random messages and the real message.

**Strength** The strength of this building block depends on the amount of dummy traffic that is inserted. The more dummy traffic inserted, the more difficult it comes for the attacker to distinguish between normal messages and dummy traffic.

**Boundary behavior** This technique behaves well if the amount of incoming traffic is low for some period. Then, reordering is not very useful or results in too large delays. Dummy traffic should be avoided if there is a lot of incoming traffic.

**Requirements** Inserting dummy traffic without other anonymity mechanisms makes no sense. Additional mechanisms must be provided that make it impossible for attackers to distinguish between normal traffic and random traffic. Moreover, a receiver must be able to distinguish between normal traffic and dummy traffic. MAC's are appended to messages to solve this problem.

**Intelligence/Sensitivity** The building block must decide when it is useful to generate dummy traffic i.e. when the amount of incoming traffic is very low. Moreover, an attacker may not be able to distinguish between real traffic and dummy traffic. They must both have the same format.

### 4.3.7.3 Examples

4. No re l
4.3.8.1 Description

To trace a message, the attacker captures your message and sends many copies of it to the mix. Many identical messages will then emerge from the mix. This bump in mix traffic will show the route of the message. To prevent this attack mixes must refuse to send any message more than once. This can be done by including a random ID number for each hop, which the system records. Unfortunately this places large storage demands on the mix, but the impact can be limited by one of the following techniques:

expire time An expire time is appended to the message before it is sent by the initiator. The message is valid only during that period of time. At message arrival, the expire time is checked. If the message is not within the valid time interval, it's ignored. If the message is within the valid time interval, the system looks if it is not a duplicate. If so, it's also ignored. If not, the message is flushed out and the message identifier is kept until expire time. The system has to hold some information about the message until the message has expired. Then the message is not sent any more because it's not valid anymore.

changing keys the mixes' key is changed periodically. At this point the list can be cleared because the outgoing message appears in a different form in the output stream. This supposes that keys are used to encrypt messages.

e-cash A third solution is to require anonymous e-cash postage in each layer. If the message is resent, then the cash has been doubly spent, and the mix would refuse to send it. This also has the benefit of making spams expensive, and of motivating mix operators to provide better services. In this solution, the e-cash is the anonymous identifier.

We further concentrate on the first two techniques that are often used in combination.
4.3.8.2 Properties

Performance This building block has to check if an incoming message is a duplicate. This requires some computational overhead. Storage space is required for storing messages that have not expired. More storage space is required if the table grows. Therefore, the keys must be changed at regular times. Then the tables can be cleared. But exchanging new keys also takes some time. There is no delay or communication overhead with this building block. Because duplicates are not flushed out, the communication performance increases.

Attacks Anonymous identifiers offer protection against entities that try to trace messages by replaying the original message to the mix. Untraceability and unlinkability against these entities is provided if the system does not flush out replays in the same form.

Strength The anonymity strength depends on the implementation of the entities that handle the anonymous identifiers:

if this entity simply removes old identifiers without changing the key at intermediate times, a message replay is possible and messages can flushed out in the same form at the output stream.

if keys are changed at regular times, the same message can be flushed out but in a different form. It’s impossible for the attacker to link the original message and the replay. The anonymity strength is optimal if the tables are removed only if the keys are changed.

Requirements This building block must be used in combination with other techniques in order to provide untraceability between the initiator and the recipient.

Intelligence/Sensitivity The initiator of a message includes an expire time. This expire time must be large enough. Otherwise, it may happen that the message has expired before it is actually delivered to the recipient i.e. denial of service. If the expire time of all messages is too large, the table will be very large. Therefore, this building block must search for a good tradeoff between the length of the table and the exchange of new keys.
4.3.8.3 Examples
An interesting example is onion routing [34]. Reply onions contain an expire
time. This means that it is only possible to reply a message for a limited
period of time.

4.9 Filter
4.3.9.1 Description

This building block is used to filter incoming messages. If an incoming
message reveals the identity of some communicating party, it is denied.

4.3.9.2 Properties
Performance This building block implies no communication overhead or
storage space. Filtering messages may include some computational overhead
to check if an incoming message reveals the identity of some communicating
party. The delay of this computation is minimal.

Attacks The goal of this building block is to hide the identity of some
party. Because messages that reveal the identity are denied, an attack on
the contents of outgoing messages may be very difficult.

Strength Filtering messages provides a very strong anonymity. The price
we have to pay is that some messages will be denied.

requirements One requirement is that this technique must be used in
combination with a technique that changes the form of the outgoing messages.

Intelligence/Sensitivity This building block may not deny too much
messages. Therefore, this building block must do a good check on what
identity information is revealed in each message.

4.3.9.3 Examples
Zero-Knowledge [44] builds 'Personal Firewalls' to filter messages.
CHAPTER 4. CONNECTION-LEVEL BASIC BUILDIN BLOCKS

4. \( .10 \) Ch

4.3.10.1 Description

\[
M = [S_i, C_i, E, R] \\
M' = [E, R, A_j, S_j]
\]

table \( C_i, A_i \)

This building block is used to hide a responder that wants to remain anonymous. A request towards an anonymous responder implies setting up an anonymous path between the initiator and the responder. If this initiator performs some successive real-time requests towards the same responder, an attacker could reveal the identity of the responder because each request towards the responder is followed by a reply. Caching data at intermediate entities in the path towards the responder makes it less likely that all requests will actually get forwarded all the way back to the responder. If the data is found at the intermediate entity in the path, the request is not forwarded towards the responder.

4.3.10.2 Properties

**Performance** Some computational overhead is required for matching a reply to a request in the cache. If replies are often found in the cache, communication performance increases because the full path towards the responder does not have to be set up. Very large disk space will be required if the cache wants to be effective.

**Attacks** The goal of this building block is to provide full anonymity towards responders against attackers that try to execute successive requests.

**Strength** The strength of anonymity depends on the storage space on the intermediate entity on the path. If the storage space is small, only few information can be cached and the information will be cached during a small period. Besides, a good algorithm is needed to decide what information to throw away.

**Boundary behavior** If the cache space is very small, nearly each request will follow the whole path towards the responder. If the cache space is very large, almost each reply to a request will be present in the cache.
requirements  Caching is only useful for keeping static data. For instance, in web browser systems different users could make the same request. Caching maps each similar request to the same answer. If the answer data change very often, the cached data may be expired. Moreover, this technique makes no sense if the identity is revealed in the original request. This identity information must be hidden using some encryption technique.

Intelligence/Sensitivity  If the cache is full at some intermediate node, an algorithm is applied to decide what information should be thrown away. One technique is to delete the information that is not requested for some period of time. A successive request for this information will be forwarded all the way back to the responder.

4.3.10.3 Examples

TAZ servers [20] use caching. Successive requests of the same information do not follow the full path towards the responder. This makes it more difficult to detect the location of the responder.

4.11  Broadcast

4.3.11.1 Description


Broadcast means that a message is sent to a group. All entities that receive the message scan the message. Only the intended receiver can read the content of the message. If the message is only forwarded to one user, i.e. the intended receiver, the building block has no sense. If each message is forwarded to each entity in the network, too much bandwidth will be used.

4.3.11.2 Properties

Performance  Broadcast requires communication overhead because several outgoing messages are generated for one incoming message. No delay or additional storage space is required. Some delay will happen when the recipient decrypts incoming messages.
CHAPTER 4. CONNECTION-LEVEL BASIC BUILDIN BLOCKS

Attacks The goal of this building block is to hide the receiver of a message. This way, the receiver and the sender cannot be linked. Remark that the sender and the receiver can be different from the initiator and the responder of the message. Intermediate entities can also act as senders/receivers.

Strength The anonymity strength depends on the amount of users to which the message is broadcasted. The probability that a sender and a receiver can be linked decreases when the amount of users increases.

Boundary behavior This technique does not work well if the amount of traffic is high.

requirements One requirement is that only the intended receiver can actually read the contents of the message. For this purpose, the message is for instance encrypted with the public key of the receiver.

Intelligence/Sensitivity The building block must choose the number of entities to which it will forward the message. This number of entities grows if there is little incoming traffic. If there is a lot of traffic, other techniques such as reordering will be more effective to avoid traceability.

4.3.11.3 Examples

BABEL [23] uses broadcasting for reply messages. Instead of delivering a reply directly to the initiator of the original message, the responder can deliver it to a local newsgroup with a special number tag. The initiator scans this newsgroup for replies matching that number tag.

4. .12 Multlex

4.3.12.1 Description

multiplexer

\[
\begin{array}{c}
M_1 M_2 \ldots M_n \\
\end{array}
\xrightarrow{}
\begin{array}{c}
M'=[M_1 M_2 \ldots M_n] \\
\end{array}
\]

demultiplexer
Multiplexing is sending several messages at the same time in the form of a single message. Several messages use the same connection. This makes it harder for an attacker to trace a message. A counter part building block must be introduced to demultiplex messages.

4.3.12.2 Properties

Performance  It is not needed to set up a connection for each particular message. Thus, the computational and communication performance increases if more messages are multiplexed along the same socket connection. This technique implies no delay or extra storage capacity.

Attacks  This building block helps to provide untraceability because it is harder to distinguish the path of a particular message.

Strength  The anonymity strength depends on the number of messages that are multiplexed along the same socket connection.

requirements  This technique has no sense if the form of messages is not changed. Some other building blocks such as encryption must be applied in addition to multiplexing.

Intelligence/Sensitivity  The building block must know which messages can be multiplexed. Only messages that follow the same partial route can be multiplexed. Moreover, messages should also be demultiplexed at the end.

4.3.12.3 Examples

Onion Routers [34] multiplex different messages.
4. Bulletin Board

4.3.13.1 Description

A bulletin board can be thought of as an intermediate message pool. Messages sent by someone are stored until some user pulls it from the board. An example of a bulletin board is an Internet newsgroup. All entities that can be the receiver of that message scan the messages posted in that newsgroup at regular times. Only the intended receiver can read the content of the message. Some techniques exist to remove a message from the bulletin board:

removal by bulletin board The bulletin board removes messages after some fixed period. Thus, receivers must scan the board at regular time intervals because messages are removed after that fixed period. Variation: an expire time is appended to the message by the sender. The building block removes the message after that period.

removal by sender The sender can always decide to remove messages it placed on the bulletin board.

removal by receiver After the receiver has pulled the message from the bulletin board, that receiver informs the bulletin board that the message may be removed from the board. To achieve this anonymously, the sender of the message could append some credit to the message. Only the receiver can use the credit to remove the message from the bulletin board.

4.3.13.2 Properties

Performance This building block implies some communication overhead. All entities have to check the bulletin board at regular times to see if they are the intended receiver of some message. After they have pulled a message from the board, they have to decrypt it with their private key. Some storage space must be available to put messages on the board. The amount of delay depends on the frequency that a user consult the bulletin board.
CHAPTER 4. CONNECTION-LEVEL BASIC BUILDIN BLOCKS  38

Attacks  The goal of this building block is to hide the receiver of a message. This way, the receiver and the sender can not be linked. Remark that the sender and the receiver can be different from the initiator and the responder of the message. Intermediate entities can also act as senders/receivers.

Strength  The anonymity strength depends on the amount of users in the group. The probability that a sender and a receiver can be linked decreases when the amount of users increases.

requirements  One requirement is that only the intended receiver can actually read the contents of the message. Therefore, the message is encrypted with the public key of the receiver.

Intelligence/Sensitivity  The entity that pulls the message from the board has to know what messages on the board belong to him.

4.3.13.3  Examples

BABEL [23] is a system that mixes e-mail through a chain of entities. In typical implementations of chains, an intermediate mix on the forward path can discover the identity of the previous and the next mix hop. To prevent this knowledge of the mix, mix addresses are omitted in the anonymous message, and each intervening mix posts the message to a newsgroup periodically scanned by all mixes instead of sending the message to the next hop.

4.4  Summary

The following table shows an overview of the connection building blocks and their performance characteristics.
<table>
<thead>
<tr>
<th>Name</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Communication</td>
</tr>
<tr>
<td>$g$</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>$g$</td>
<td>0</td>
</tr>
<tr>
<td>$g$</td>
<td>0</td>
</tr>
<tr>
<td>$\text{ffi}$</td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>+</td>
</tr>
<tr>
<td>$g$</td>
<td>0</td>
</tr>
<tr>
<td>$g$</td>
<td>++</td>
</tr>
<tr>
<td></td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>+</td>
</tr>
<tr>
<td></td>
<td>--</td>
</tr>
</tbody>
</table>
Chapter 5

Application-level basic building blocks

5.1 Introduction

In this chapter we describe several building blocks that can be used to provide anonymity at the application level.

Most of them are techniques that have been developed to add anonymity to a particular type of application—typically electronic commerce or electronic elections.

Many of the described building blocks are rather complex, given that they comprise protocols, algorithms and several resources. They are not easy to combine because they have been conceived to provide a complete solution. On the other hand, some other building blocks are a basic block that is used to design more complex systems—e.g., blind signatures.

Note that to achieve anonymity we need to use the building blocks described at the connection level. Otherwise the communication will be traceable, despite all our efforts to try to provide anonymity at the application level.

5.2 Properties

We give in this section a brief description of the properties that are discussed for each building block.

This properties will make easier the comparison between different techniques, and will help to choose the most appropriate solution for a particular problem.
5.2.1 Model Requirement

Most systems make some assumptions or require certain components in order to work. In some cases, these can be mathematical assumptions, e.g., assuming a problem is hard to solve, and in other cases other building blocks are required, e.g., private channels.

It is interesting to know the assumptions and requirements of a building block, to decide whether a technique can be implemented with the available resources or not.

5.2.2 U o d o l o m

Most of the discussed building blocks are useful to provide unconditional anonymity. Nevertheless, some building blocks that provide conditional anonymity have been included to provide a broader scope.

Some building blocks can also be implemented to provide conditional or unconditional anonymity.

5.2. Tr

There is in many systems an entity that is trusted to do something in an honest way. In some cases, the entity is trusted to perform certain actions for example distributing keys, and in other cases the entity is trusted not to disclose secret information.

Some systems distribute the trust over several entities to achieve robustness. In those cases, the faulty behavior of one or several trusted entities does not prevent the system from working in a correct and secure way.

The last section, describes in detail the functions of a general trusted party. This section has a different structure, given the large variety of characteristics a trusted party may have.

5.2.4 Efficiency

This property tries to give an idea of the performance of the discussed building blocks, in order to be able to compare different techniques and see whether a theoretical solution can also be practical or not.

5.2.5 Security

With this property we give information about the strength of the security of the building block. In many cases, the security of the scheme can be reduced to a known problem that is hard to solve.
5.2.6 Verifiability

For some applications, e.g., electronic elections, it is interesting to know if the building block allows participants or even passive observers to check that the result is correct.

5.2.7 k

Here we mention attacks that can be carried out to break the security of the building block, and also those attacks against which the scheme is protected. Most systems are designed with an attack model in mind, thus it is interesting to know towards which type of attacks the system is protected.

5.3 Blind Signatures

Blind signatures were proposed by Chaum in [8]. The basic idea can be easily explained with a real world analogy: the user introduces the document to be signed in an envelope, together with carbon paper. He gives the closed envelope to the trustee, who signs it. The carbon paper leaves a carbon copy of the signature on the paper within the envelope. The signer does not see the content of the information of the document.

The following three functions make up a blind signature cryptosystem:

1. A signing function \( s' \) known only to the signer, and the corresponding publically known inverse \( s \), such that \( ss' x = x \), and \( s \) gives no clue about \( s' \).

2. A committing function \( c \) and its inverse \( c' \), both known only to the user, such that \( c' s' c x = s' x \), and \( c x \) and \( s' \) give no clue about \( x \).

3. A redundancy checking predicate \( r \), that checks for sufficient redundancy to make search for valid signatures impractical.

The way these functions are used is reminiscent of the way the carbon paper lined envelopes were used in the example described above:

The user chooses \( x \) such that \( r x \), forms \( c x \), and supplies \( c x \) to the signer.

The signer signs \( c x \) by applying \( s' \) and returns the signed matter \( s' c x \) to the user.
The user strips the signed matter by application of $c'$, yielding $c' \ s' \ c \ x \ = \ s' \ x$.  

Anyone can check that the stripped matter $s' \ x$ was formed by the signer, by applying the signer’s public key $s$ and checking that $r \ s \ s' \ x$.  

The blind signature system has the following security properties comprising the mentioned functions and protocols:

Digital signature: anyone can check that a stripped signature $s' \ x$ was formed using the signer’s private key $s'$.

Blind signature: the signer knows nothing about the correspondence between the elements of the set of stripped signed matter $s' \ x_i$ and the elements of the set of unstripped signed matter $s' \ c \ x_i$.

Conservation of signatures: the user can create at most one stripped signature for each thing signed by the signer.

5.1 Pro er e

Assumptions and requirements One-way functions exist.  

In order to preserve the anonymity of the user, we need untraceability of the communication. Therefore, an anonymous communication channel is required in at least one of the phases of the protocol i.e. either during the blind signing protocol or in the phase where the user sends the unblinded information. If no anonymous channel is used in both phases, an eavesdropper may link the two phases for example, looking at the IP address.

Usually, it is more interesting to use an anonymous channel in the second phase actual payment, ballot cast... because this way we make unlinkable the user and the payment or vote. If we use an anonymous channel in the first phase, we prevent eavesdroppers to get information about withdrawals.

These requirements are, though, application dependent, therefore specific requirements will be needed depending on the service provided and the implemented protocols.

Unconditional Anonymity With a blind signature scheme we can create systems that provide unconditional anonymity. This is because no party, including the signer, is able to link the blinded and the unblinded information.

It is important to note that the same information will always lead to the same blinded message. Therefore, the user should include some random information in the message to get different blinded messages.
CHAPTER 5. APPLICATION-LEVEL BASIC BUILDIN BLOCKS

If we want to provide conditional anonymity, mechanisms to revoke anonymity must be implemented.

**rust** In this scheme, no party has to be trusted to keep the private information secret.

**Efficiency** If we want to use blind signatures for payments, and we want to prevent double spending, the trustee or signer must remain on-line during the second phase of the protocol payment phase. In such an implementation the efficiency of the system is low.

More efficient realizations than the one proposed by Chaum can be found in [18, 26]. In [26], the user performs two on-line exponentiations and one offline inverse; the signer computes one exponentiation; and the verifier needs to compute two exponentiations.

**Security** The proofs of the security of digital signatures can be classified into two major categories: Complexity-based proofs and proofs based on a random oracle model.

The random oracle model has been used to prove that digital blind signatures are as secure as factorization [32]. This has been proven against single and parallel attacks.

The first complexity-based theoretic proof for security of blind signatures was produced by Juels, Luby and Ostrovsky [28]. They made use of a one-way trapdoor to prove that blind signatures are secure. Their proof is secure against an adaptive interleaved chosen-message attack. An adaptive interleaved chosen-message attack is an attack where the attacker is allowed to run many protocol execution attacks in parallel. In other words blind signatures are as secure as factoring.

**Verifiability** Any party can verify the validity of the signature using the public key of the signer.

**Attacks** In a system based on blind signatures with on-line trustee a denial of service attack may lead to the unavailability of the system.

5.2 Ex m le

Blind signatures have a wide range of application in systems that provide anonymity. The most important fields of application are electronic payments traceable and untraceable electronic cash and electronic elections. Many of
the implemented or proposed systems that provide those services make use of blind signatures.

5.4 Fair Blind Signatures

Blind signature schemes provide perfect unlinkability between the blinded and the unblinded information. In systems where anonymity control is desirable to prevent abuse the scheme needs to be modified.

Fair blind signature schemes have the property that a trusted entity can deliver information allowing the signer to link his view of the protocol and the message-signature pair. The system has been proposed by Stadler, Piveteau and Camenisch in [40].

The players of a blind signature protocol are the sender and the signer. In a fair blind signature scheme two protocols are used: one performing the blind signature, \( g \rightarrow g \), involves the signer and the sender; the other, called \( T \), involves the signer and the trusted party.

There are two types of fair blind signature schemes, depending on the information the judge receives from the signer during the link-recovery protocol.

- \( T \): Given the signer's view of the signing protocol, the trusted party delivers information that enables the signer to efficiently recognize the corresponding message-signature pair.

- \( T \): Given the message-signature pair, the trusted party delivers information that enables the signer to efficiently identify the sender of that message.

This scheme does not protect individuals from blackmail. The criminal can force the victim to use a blind signature that does not have an identity revealing protocol built into it. A warning system built into the protocol that could be triggered undetected could solve this problem.

In [40] several protocols are proposed to realize fair blind signatures.

**Trusted-Based Fair Blind Signature** Trustee-based tracing is another method of fair blind signatures [6]. Here, the user provides trustees with information that allows the trustees to recognize the electronic notes of the user. Each of the trustees receive a part of this information that makes the electronic messages of the user traceable. So, when the law needs to trace the messages of a user all the trustees get together, and with their individual pieces they are able to identify the users electronic notes and are therefore able to trace the user.
Fair Blind Signatures without trustees In [5], another method to implement anonymity revocation for double-spenders in a payment system based on blind signatures without a trustee is proposed. In this scheme the identity of the user is encoded in the coin, and it is not revealed unless the user spends the coin twice.

5.4.1 Pro e e

Assumptions and requirements One-way functions exist.

Anonymous communication channels are required, as in the blind signature scheme.

Conditional Anonymity Conditional anonymity will be discussed in future documents, but we include this technique here as a different flavor of blind signatures to give a broader view on this technique.

The system provides anonymity and unlinkability for honest users. If a user is dishonest, his anonymity can be revoked. This scheme enables anonymity control.

rust In the general case, the judge and the signer are trusted not to collaborate to revoke the anonymity of a honest user.

In the Trustee-Based scheme, the trustees are trusted not to collude and disclose users' private information without a judge's order.

In the scheme proposed in [5] there is no trusted party that keeps information about the user. In order to prevent double spending, a user that tries to spend a coin twice will reveal his identity.

Efficiency Dishonest users that abuse the system will loose their anonymity. Because of the anonymity revocation, the signer does not need to check that a coin has already been spent, and it does not need to be contacted on-line. This improves the efficiency of the protocol with respect to systems that require on-line verification.

An example of fair blind signatures would be using a blind signature scheme together with a registration protocol. The overhead of the registration protocol consists of three messages exchanged between the user and the trustee, one exponentiation computed by the user and another one computed by the trustee, and two digital signatures computed by the trustee.

User efficient fair blind signature schemes have been proposed, as in [19].
CHAPTER 5. APPLICATION-LEVEL BASIC BUILDIN BLOCKS 47

Security The security of the scheme is similar to the one described for blind signatures.

Verifiability Any party can verify the validity of the signature using the public key of the signer.

Attacks A powerful attacker that is able to break the security of the trustee would have access to useful information to link the messages sent by the user, and learn private information.

5.4.2 Example

The applications of fair blind signatures are, for example, payment systems, electronic elections, or electronic auctions. In general, any system where un-linkability between two messages containing the same information is desired, while anonymity revocation is also needed.

5.5 Group Signatures

A group signature scheme [3] allows a group member to sign messages anonymously on behalf of the group. In contrast to ordinary signatures they provide anonymity to the signer, i.e., a verifier can only tell that a member of some group signed.

A group signature scheme is a digital signature scheme comprised of the following five procedures:

SETUP: On input a security parameter, $l$, this probabilistic algorithm outputs the initial group public key, $PK$, and the secret key, $S$, for the group manager.

JOIN: A protocol between the group manager and a user that results in the user becoming a new group member. The user’s output is a membership certificate and a membership secret.

SIGN: A probabilistic algorithm that on input a group public key, a membership certificate, a membership secret, and a message $m$ outputs group signature of $m$.

VERIFY: An algorithm for establishing the validity of an alleged group signature of a message with respect to a group public key.
CHAPTER 5. APPLICATION-LEVEL BASIC BUILDING BLOCKS

OPEN: An algorithm that, given a message, a valid group signature on it, a group public key and a group’s manager’s secret key, determines the identity of the signer.

5.5.1 PROPERTIES

Assumptions and Requirements The strong RSA and the decisional Diffie-Hellman assumptions are made.

The scheme relies also on the Fiat-Shamir heuristic random oracle model.

Conditional Anonymity In case of a dispute, the identity of a signature’s originator can be revealed only by a designated entity.

For any other entity than the group manager, identifying the actual signer or deciding whether two different valid group signatures were computed by the same group member is computationally hard.

Trust The group manager can a group signature. Therefore, it is an entity trusted not to reveal the identity of the signer in other circumstances. The group manager cannot misattribute a valid group signature. He cannot sign a message on behalf of other users.

Efficiency During the JOIN protocol the new member has to provide efficient proofs of knowledge of discrete logarithms. It can be realized with binary challenges.

The SIGN algorithm computes 14 exponentiations and 2 divisions.

The VERIFY algorithm computes 11 exponentiations and 2 divisions.

The scheme in [3] is claimed to be very efficient.

Security The scheme in [3] is proven secure and coalition-resistant under the strong RSA, the decisional Diffie-Hellman and the Fiat-Shamir assumptions.

The protocols proposed in [3] can be proven zero-knowledge in an honest verifier model.

Verifiability Any party can check the validity of the signature, given the public key.
CHAPTER 5. APPLICATION-LEVEL BASIC BUILDING BLOCKS

Attacks The scheme is provably coalition-resistant against an adaptive adversary. That is, a colluding subset of group members even the entire group cannot generate a valid signature that the group manager cannot link to one of the colluding group members.

5.5.2 Example
Group signatures are useful for applications such as voting and bidding.

5.6 Threshold cryptosystems

In a public key threshold cryptosystem the private key is shared among several users, in such a way that a subset of users of smaller size than the threshold \( t \) cannot get any information about the secret.

Depending on the use of the key pair, we can distinguish between threshold signatures and threshold decryption.

**Threshold Signatures** Threshold signatures are motivated by the need that arises in organizations to have a group of employees who agree on a message before signing and by the need to protect the group private key from the attack of internal and external adversaries.

The goal of a threshold signature scheme is to increase the availability of the signing authority and, at the same time, the protection against forgery or key stealing by making it harder for the adversary to learn the group secret key.

**Threshold Decryption** In a public key threshold cryptosystem [17] the private key is shared among \( n \) users. A number of users \( t, t < n \), is defined as the minimum number of users that can decrypt a message encrypted with the public key.

The secret key is distributed in such a way that if less than \( t \) users collaborate they get no information about the secret. Each user owns a share of the secret, and it is called in some of the literature a threshold share.

Each shareholder will calculate their share separately and transmit the result to a designated individual. The designated individual will be able to decrypt the information using these partial results.

A group of \( t \) or more users can decrypt the message. After this, the secret shares of the private key are not revealed, and the secret does not have to be updated.
Threshold decryption is the most interesting application of threshold cryptosystems to provide anonymity. It is useful to distribute the trust over several authorities in a voting system or to revoke anonymity in case of fraud.

5.6.1 Pro er e

Assumptions and requirements The public key cryptosystem used RSA, ElGamal, etc. is secure.

If we want to implement a threshold cryptosystem without a trusted party that generates and distributes the shares of the key, a special protocol for selecting and distributing the shares without relying on a trusted party is needed [30].

Conditional Anonymity This system may be used to implement anonymity revocation. The user encrypts the data that allows the revocation of his anonymity with a public key, and the corresponding private key is distributed over \( n \) authorities. Then, a group of at least \( t \) authorities should agree to revoke the anonymity of the user.

trust Threshold cryptosystems are useful to distribute the trust. If several authorities have to work together to revoke the anonymity of a user, the failure of at most \( t - 1 \) authorities can be tolerated.

A group of \( t - 1 \) or less authorities is unable to get any information about the encrypted message. Therefore \( n - t + 1 \) authorities are trusted not to collude.

Depending of the implemented protocol to generate and distribute the private key, there might be a trusted entity that knows the private key. This is not desirable in our case, in [30] a protocol to select and distribute the key is introduced, with no need of trusted party.

Efficiency Some practical schemes have been proposed in [17]. In this system, a non-interactive solution is proposed.

To perform threshold encryption based on ElGamal cryptosystem, the sender needs to compute two exponentiations. To decrypt the message, the receiver has to compute one exponentiation.

Security A proof of security using the concept of zero-knowledge is presented in [17]. It implies that if the discrete logarithm is hard, then the calculation of the partial results will also be hard.
Verifiability The can verify that their of the key are valid [30].

Attacks If \( t \) or more shareholders are corrupted and they collude, then they can reconstruct the private key.

5.6.2 Example

This scheme can be very useful to distribute trust. In case conditional anonymity is required, threshold cryptosystems can be used to reduce the possibility of unauthorized disclosure of private data by forcing several authorities to agree before revealing private information.

The other field of application of this scheme would be electronic elections. In this case the system is useful to enhance the security of the system by making the system more robust: a malicious behavior of at most \( n - t + 1 \) authorities can be tolerated.

5.7 Multi-Party Computation

The goal of secure multi-party computation is that \( n \) players compute an agreed function of their inputs in a secure way, where security means guaranteeing the correctness of the output as well as the privacy of the player’s inputs, even when some players cheat.

A collection of \( n \) players can efficiently compute the value of an \( n \)-input function, such that everyone learns the correct result but no other new information.

5.7.1 Pre er e

Assumptions and requirements To achieve cryptographic security, the assumption that trapdoor one-way functions exist has to be made.

Less than \( n/2 \) players are dishonest in some cases the limit is \( n/3 \).

Communication is assumed to be synchronous.

Private channels between every pair of players are required to achieve unconditional or information-theoretic security.

Many proposals are based on \( r \quad s \quad r \quad s \quad r \quad g \), i.e., a protocol allowing a dealer to securely distribute a secret \( s \) among the players, where the dealer and some of the players may be cheating.

If the communication model is cryptographic, then a broadcast channel is required either by physical means or implemented by a protocol.
Unconditional Anonymity  The attacker, even if he can influence the input of the corrupted players, never finds out the input of honest players. The privacy of the inputs of honest players is never revoked, so the system can be used to provide unconditional anonymity.

Trust  A key tool for multi-party computation is verifiable secret sharing. In this scheme [13, 14] a trusted party (dealer) is needed to distribute the shares of the secret $s$ among the players.

Efficiency  Depending on the adversary structure considered, the multi-party computation protocol complexity might be polynomial in $n$ [14].

Security  If the considered communication model is cryptographic, security can only be guaranteed in a cryptographic sense (relying on the difficulty of solving certain mathematical problems).

If secure channels can be assumed, then the system is secure in the information-theoretic sense.

Verifiability  All players can verify that the output of the function is correct.

Attacks  These protocols can be proved secure against a polynomial time bounded static active adversary who can corrupt a set of less than $n/2$ players (in some systems the limit is set to $n/3$). When a player is corrupted, the adversary gets all the data held by this player, including complete information on all actions and messages the player has received in the protocol so far.

5.7.2 Examples

Multi-party computation systems can be used to implement an electronic election.

5.8 Homomorphic Encryption

Homomorphic encryption is a special type of cryptography in which an operation (sum, multiplication) of two encrypted values is equal to the encrypted sum of the values.

For a majority of cryptographic algorithms, this does not hold true. In most cases, it is undesirable because it may help reveal information which
can be used to break the encryption. However, this is a desirable property if one wishes to have the sum of a group of encrypted values verified without revealing those encrypted values. In voting protocols, this is used to verify the tally of the ballots without revealing what those ballots are.

If the encryption function is homomorphic then the addition (or multiplication) of encrypted messages is the same as the encryption of the sum of the messages in cleartext. More formally:

if $e_1$ is the encryption of $v_1$, and $e_2$ is the encryption of $v_2$, then $e = e_1 + e_2$
is the encryption of $v = v_1 + v_2$.

5.8.1 Properties

Assumptions and Requirements A cryptosystem with the homomorphic property is required.

The homomorphic property is introduced in [37, 25, 1] for the ElGamal and Elliptic Curve cryptosystems.

Unconditional Anonymity An homomorphic encryption scheme is useful to build a system that provides unconditional anonymity (such as electronic elections). If only the sum of the messages is disclosed, we get no information about individual inputs, that is, all users are indistinguishable.

Trust The parties that hold the private key are trusted not to decrypt the particular messages. In electronic elections, this private key is very often distributed over several authorities, and it is never reconstructed. Threshold cryptosystems [30] are used to decrypt the result of the election without revealing the private key.

Efficiency The efficiency of the encryption function will be that of the cryptosystem that has the homomorphic property (ElGamal, Elliptic Curves, etc.).

Security In the general encryption case, it is not advisable to use an homomorphic encryption scheme, because this property may help revealing information to an attacker. On the other hand, this is a very useful property when the individual values are to be protected, but not the sum of them, as it is the case in an election scheme.

So far there are no known provably secure algebraic homomorphic encryption functions.
CHAPTER 5. APPLICATION-LEVEL BASIC BUILDING BLOCKS  54

The security of the protocol proposed in [25] is specified with respect to a threshold \( t \), where the correctness of the tally is guaranteed if at least \( t \) authorities remain honest during the whole protocol execution, and privacy is guaranteed as long as no \( t \) or more authorities pool their information.

Verifiability The homomorphic property is very useful to achieve universal verifiability in electronic elections, because it allows parties to check the sum of the votes without getting information about individual votes.

5.8.2 Examples

Homomorphic encryption is useful in electronic voting schemes. Using it we can decrypt the final count of votes of the tally without disclosing the individual votes. In [25] Hirt and Sako describe a receipt-free voting system based on this technique; in [1] Adler et al. show how an election system can achieve privacy and universal verifiability using this property.

Other examples of electronic election systems that use homomorphic encryption can be found in [11, 37].

5.9 Secret Sharing Schemes

In a secret sharing scheme we have a group of participants that all get a “share” of the secret we wish to distribute among these participants. The goal of the scheme is to give each participant a piece of the secret. The different pieces (or shares) are constructed in such a way that some subsets of the participants can reconstruct the secret and others cannot.

The access structure (or concurrence scheme) on the group of participants is a specification of subsets that are qualified to reconstruct the secret, and of the subsets that are forbidden to obtain any additional information about the secret. If we use the notation of [43] then the set of participants is denoted by \( \mathcal{P} \), and an access structure on \( \mathcal{P} \) is a pair \( (\Gamma, \Delta) \), where \( \Gamma \) and \( \Delta \) are collections of subsets of \( \mathcal{P} \). \( \Gamma \) consists of all qualified groups and \( \Delta \) consists of all forbidden groups.

The most common secret sharing schemes are \((m,n)\)-threshold schemes. In this case any subset of at least \( m \) participants can combine their shares to obtain the secret.

Proactive Secret Sharing This type of secret sharing schemes [24] offer a better protection for the storage of long-lived and sensitive secrets (e.i.
a master key). In these systems, the shareholders renew periodically their shares, without changing the secret, by executing a rerandomization protocol.

An adversary who wants to learn the secret has to compromise more that $k$ participants during the same time period, i.e. between consecutive executions of the rerandomization protocol. All information the attacker may get during one time period becomes useless after the shares have been renewed.

### 5.9.1 Properties

**Assumptions and Requirements**  The shares are distributed in a secure way (only the intended receiver learns the content of the share).

A secure channel is required to distribute the shares.

**Unconditional Anonymity**  There are a number of ways in which Secret Sharing can be used to provide anonymity. For example, in a $(2, n)$-threshold scheme, any two people out of $n$ can sign a document together, without revealing their identity.

**Trust**  In this system, parties have to be trusted not to reveal their shares to other parties.

**Efficiency**  The computational and communicational cost of secret sharing schemes grows with the number parties. Most systems become very inefficient with large groups and are no longer practical.

**Security**  Many secret sharing schemes are provable secure, i.e. in a $(m, n)$-threshold scheme, $m - 1$ shares reveal *no* information about the secret.

**Verifiability**  Universally Verifiable (vs Non-verifiable): Everybody can verify the validity of the different shares.

Perfect (vs Non-Perfect): An unauthorized group of participants cannot gain any information about the secret.

Ideal (vs Non-Ideal): The size of the shares is equal to the size of the secret (in bits).

**Attacks**  Since most secret sharing schemes are provable secure, there are no attacks on the schemes themselves.

Weaknesses in the specific implementation are always possible and can be exploited (for example the distribution of the shares).
5.9.2 Examples

Let the set of participants be $\mathcal{P} = \{1, 2, 3\}$, the set of qualified groups be $\mathcal{G} = \{\{1, 2\}, \{2, 3\}, \{1, 2, 3\}\}$, the set of forbidden groups be $\Delta = \{\{1, 3\}\}$, and the secret $s$. We can share the secret $s$ with the following secret sharing scheme:

- Participant 1 gets the share $(s \oplus a_1)$,
- Participant 2 gets the share $(a_1)$,
- Participant 3 gets the share $(s \oplus a_1)$.

It is easy to verify that participant 2 can obtain the secret if he combines his share with participant 1 or 3, and participants 1 and 3 cannot obtain any additional information about the secret $s$. Furthermore, no one of the participants alone has any information about the secret. Shamir’s Secret Sharing Scheme [39] is an example of a $(m, n)$-threshold scheme.

5.10 Untraceable Broadcast

The scheme, proposed by Pfitzmann and Waidner in [31], combines privacy (secrecy of the inputs of individual participants) with fault tolerance.

Goal of the system: It should enable each participant to broadcast an arbitrary number of messages from an arbitrary finite message set. Privacy, in this case, means that a protocol must not reveal any information about the senders’ identities.

The system provides unconditionally untraceable broadcast and unconditional fault tolerance.

5.10.1 Properties

Assumptions and Requirements Reliable broadcast assumption: if a sender broadcasts a message, then all honest participants agree on a message $v$, and if the sender is honest, $v$ is the message the sender meant to send. It can be implemented using the Byzantine agreement protocol (BAP).

The requirements for the system are:

- Synchronous network which enables each pair of honest participants to communicate securely.
- Secure channels are necessary to achieve fault tolerance.
CHAPTER 5. APPLICATION-LEVEL BASIC BUILDING BLOCKS  57

- Private channels between each pair of honest participants.

The untraceable broadcast protocol is based on three protocols:

- A DC-Protocol with collision resolution
- Pseudo-signatures
- Unconditional Byzantine Agreement

**Unconditional Anonymity**  The private inputs of honest users are unconditionally untraceable, and there is no trusted party that has the power to revoke anonymity, therefore it provides unconditional anonymity.

**Trust**  There is no trusted party in this scheme.

**Efficiency**  The protocol is polynomial in the number of participants, $n$, and the security parameter. The complexity is quite high.

**Security**  The protocols are unconditionally untraceable with an exponentially small error probability.

**Verifiability**  The correct output can be verified by all participants, since all messages are published in a reliable broadcast channel.

**Attacks**  The protocols proposed in [31] tolerate any number of attacking participants and are secure in an information-theoretic sense regarding privacy.

The protocols described in [31] do not guarantee simultaneity, dishonest voters can choose their votes dependent on the honest voters’ votes.

The proposed protocol is not completely fault-tolerant. If the attacker is the last participant to publish his local sum, he can manipulate the output by choosing the appropriate local sum. This can be solved if in each round all outputs are made completely simultaneously.

### 5.10.2 Examples

The scheme is useful in the context of transactions via public networks that require untraceability (like untraceable payment systems).

In [31] a secret ballot election system is proposed based on this scheme.
5.11 Bulletin Board

A bulletin board [11, 37] is a broadcast channel with memory. Any party (including passive observers) can see the contents of the bulletin board, and each active participant can post messages by appending the message to his own designated area. No party can erase anything from the bulletin board.

The user posts a single encrypted message accompanied by a compact proof that it contains valid information.

Digital signatures are used to control access to the sections of the bulletin board.

5.11.1 Properties

Assumptions and Requirements The bulletin board behaves as a public broadcast channel, and all the users can see the contents of it.

The user sends an efficient non-interactive proof of validity together with the encrypted message to prove that the encryption contains valid information.

The ElGamal cryptosystem is secure.

A Public Key Infrastructure is required to generate and distribute key pairs to the users so they can authenticate themselves.

To build an electronic election system, homomorphic encryption is required. The privacy of the users relies on the fact that individual messages are never decrypted (instead, the sum of those messages is decrypted). In [11] ElGamal cryptosystem is used.

There is a key generation protocol to generate the private key jointly by the authorities and a decryption protocol to jointly decrypt the ciphertext without explicitly reconstructing the private key.

There are a set of replicated servers implementing Byzantine agreement, such that access is never denied as long as at most a third of the servers is compromised.

Unconditional Anonymity If at least \( n - t + 1 \) authorities remain honest the anonymity of the users is never revoked. The systems provides unconditional anonymity.

Trust If \( t \) or more authorities collude, they could reconstruct the private key and decrypt individual messages. Therefore, at least \( n - t + 1 \) authorities are trusted to be honest.
CHAPTER 5. APPLICATION-LEVEL BASIC BUILDING BLOCKS  59

Efficiency  The time and communication complexity is independent of the number of authorities. The protocol proposed in [11] requires little work from voters (they only need to send a single encrypted message).

Security  The system provides computational privacy protection. Private channels are needed to achieve information-theoretic security.

The system is robust with respect to malicious users. This is achieved by means of the soundness of the proof of validity, which ensures that users cannot submit bogus ballots.

Robustness with respect to at most $n-t$ malicious authorities is inherited from the robustness of the key generation and decryption protocols.

Vote-duplication is prevented due to the fact that the proofs of validity are made voter-specific.

The security of the scheme relies on the ElGamal cryptosystem.

Verifiability  When the bulletin board is used to realize a voting system, it provides universal verifiability: any observer can check the proofs of validity for the ballots.

Attacks  Denial of service attacks are excluded due to the fact that users can only append messages to their own designated area.

5.11.2 Examples

This building block has been designed as part of a secret-ballot election.

In the scheme presented in [11] there is one public key for which the matching private key is shared among the authorities using threshold cryptography techniques. The private key is never reconstructed, and only used implicitly when the authorities cooperate to decrypt the final tally.

The privacy of voters is ensured as long as less than a certain number of authorities collude. The key used to decrypt the final tally is distributed over $n$ authorities, and a minimum of $t$ ($t < n$) authorities must collaborate in order to compute the final tally. Due to the homomorphic property of the votes, it is not needed to decrypt single votes, instead, the votes are added and the final result is decrypted.

5.12 Interactive Proofs and Zero-knowledge

Informally, an interactive proof is a protocol between two parties in which one party, called the prover, tries to prove a certain fact to the other party,
CHAPTER 5. APPLICATION-LEVEL BASIC BUILDING BLOCKS  60

called the verifier. An interactive proof usually takes the form of a challenge-
response protocol, in which the prover and the verifier exchange messages
and the verifier outputs either “accept” or “reject” at the end of the pro-
tocol. Apart from their theoretical interest, interactive proofs have found
applications in cryptography and computer security such as identification
and authentication. In these situations, the fact to be proved is usually
related to the prover’s identity, such as the prover’s private key.

It is useful for interactive proofs to have the following properties, espe-
cially in cryptographic applications:

Completeness. The verifier always accepts the proof if the fact is true and
both the prover and the verifier follow the protocol.

Soundness. The verifier always rejects the proof if the fact is false, as long
as the verifier follows the protocol.

Zero knowledge. The verifier learns nothing about the fact being proved
(except that it is correct) from the prover that he could not already
learn without the prover, even if the verifier does not follow the protocol
(as long as the prover does). In a zero-knowledge proof, the verifier
cannot even later prove the fact to anyone else.

In order to explain the zero-knowledge property more in depth, we will
take a brief look at an easy example: logging on to a computer network
through the use of a private password and (public) user name. When logging
on, the user types his user name and password, this is sent to the host, who
checks it against a stored list.

The security problems with this are many and well known. Let us con-
centrate here on the obvious problem that if an adversary eavesdrops the line,
he can pick up the password, and then impersonate the user. When trying to
solve this, we might propose that we transport the password in a protected
way. Perhaps encrypted or something?

Although this might be sufficient in some cases, we should ask ourselves
first what the original purpose of the protocol was. The purpose is not to
send the password from the user to the host, but to identify the user through
the use of his secret password. The only thing the host needs to know is
whether the user knows his password or not. The host does not have to learn
the actual password.

A zero-knowledge approach to this problem would look like this: the
user and the host conduct a interactive protocol, where in the end, the host
can compute a one-bit answer saying whether the user was successful in
proving himself or not. In this case this would mean that the only information
transmitted from the user to the host is the answer to the following question: “I know the secret corresponding to user name xxx”, and nothing more than that.

This leads us to this loose definition of zero-knowledge protocols: a protocol is zero-knowledge if it communicates exactly the knowledge that was intended, and no (zero) extra knowledge.

For more information on zero-knowledge protocols, see for example [16].

5.12.1 Properties

Assumptions Most efficient zero-knowledge schemes are designed for one purpose (like the example above), and they all have different assumptions. In the example below the assumption is that RSA or another public key crypto system is secure. Other possible assumptions are: one-way functions exist, symmetric ciphers are secure, secure channel, etc.

Requirements Depending on the application at hand. Since this is truly “zero-knowledge”, normally no special measures have to be taken to protect the knowledge of the prover.

Trust Normally no party has to be trusted in zero-knowledge schemes. In a secure scheme the prover cannot prove something he does not know, and the verifier cannot do anything to reveal (part of) the prover’s secret. So no party needs to trust the other.

Efficiency Generic zero-knowledge systems are very inefficient and unpractical to use. Special purpose schemes can be efficient and thus used in a real system (cfr. example below).

Security It is possible to prove the “zero-knowledge” property of good zero-knowledge schemes. These schemes are the provable secure, using some assumptions like RSA is secure, etc. (see assumptions).

Attacks Since zero-knowledge schemes are provable “zero-knowledge”, no attacks on the schemes themselves are possible. Of course the systems (encryption, MAC, etc.) that are used by the schemes can be attacked.
5.12.2 Examples

Zero-knowledge protocols can be used as subprotocols in larger constructions such as voting schemes, key distribution protocols or in general any multiparty computation.

To conclude the discussion on the log on procedure, we will give a true zero-knowledge protocol for the problem stated above:

1. If the prover (user) claims to be $A$, the verifier (host) chooses a random message $M$, and sends the ciphertext $C = P_A(M)$ to the prover.
2. The prover decrypts $C$ using $S_A$ and sends a commitment to the result $\text{commit}(r, M')$ to the verifier.
3. The verifier sends $M$ to the prover.
4. The prover checks if $M = M'$. If not he stops the protocol. Otherwise he opens the commitment, i.e. he sends $r, M'$ to the verifier.
5. The verifier accepts the identity of the prover if and only if $M' = M$ and the pair $r, M'$ correctly opens the commitment.

[In this protocol $P_A, S_A$ is public/private key pair; and $\text{commit}(r, M)$ is a bit (word) commitment to $M$ using random choice $r$.]

Another example of zero-knowledge techniques is the class of Cut-and-choose protocols. Cut-and-choose protocols work in the way, that one failure means the failure of the whole protocol (i.e. that the prover is not legitimate), but you can keep working on the protocol as long as you want, if the prover is legitimate. After you reach the level of confidence you need without being cut off, the protocol is successful. To illustrate this we will give an example of this technique as it is used in some eCash systems:

Suppose we have a system that uses digital coins of the following format: $\text{coin} = [\text{ID}]\text{amount}$. In order to prevent counterfeiting of these digital coins, we present them to a bank that digitally signs the coins. There is, however, a big privacy problem with this: if the bank can see the ID numbers of the coins during the sign operation, it can later trace the coins back to its owner after the owner purchased something with these coins. Therefore the coins are first blinded before they are signed by the bank (see the section of Blind Signatures for more details). But now, of course, there is another problem: people can tell the bank that the blinded coins that they present are coins of, say, 1 Euro, while the amount field of the coins actually contain, say, 1000 Euro. The bank will not be able to see this because the coins are blinded. To solve this, we can use a cut-and-choose protocol:

1. We prepare 100 identical coins of the same amount and with unique ID's. We blind the coins and present them to the bank.
2. The bank chooses randomly 99 out of the 100 blinded coins and asks us to open them. The bank checks the amount of the coins. If one of them does not contain the correct amount field, the coins are discarded (and we are taken to the authorities). If all 99 of them are correct, the bank accepts the last coin and digitally signs it.

In this protocol the counterfeiters have a success rate of 1%. If this is not sufficient (for example for large amounts), the bank can use the same system with a larger number of coins.

5.13 Pseudonyms

Pseudonym systems have been used for a long time in the real world. Informally speaking, we could say that a pseudonym is an identity for a user, that is used in a particular context. The pseudonym can be unlinkable to the real identity, and then, by using a pseudonym, the user keeps secret his real identity.

Pseudonym systems were introduced in the electronic world by Chaum [9] in 1985, as a way of allowing a user to work anonymously with multiple organizations.

In [29], Lysyanskaya et al. propose a pseudonym system model that achieves anonymity. Some variations of the model are discussed to make it suitable for different real-life scenarios. In the model, individuals interact with different organizations using different pseudonyms and, at the same time, an individual in a pseudonym system can prove a statement to an organization about his relationship with another organization remaining anonymous to both. By providing such a statement, no information other than the statement itself is revealed to the receiving organization.

In the model proposed in [29] the identity of a user is a well defined concept: a user is an entity that possesses a certified public key. Each user has a master public key whose corresponding secret key the user is highly motivated to keep secret.

The user opens accounts with different organizations using different, unlinkable pseudonyms. However, all pseudonyms are related to each other: a user can authenticate a valid pseudonym only if he possesses the master secret key that was used to create this pseudonym.

Pseudonyms are a powerful and flexible tool to provide anonymity in different systems. The simpler systems rely on Trusted Third Parties, and the more sophisticated ones [29] use public key cryptography, one-way functions and more complex protocols.
CHAPTER 5. APPLICATION-LEVEL BASIC BUILDING BLOCKS 64

5.13.1 Properties

Assumptions and Requirements  One-way functions exist.

There is a public key infrastructure with a trusted CA that certifies the
public key of the users. In systems where the digital identity does not need
to correspond to one and only physical identity, the CA is not needed.

The requirements for a pseudonym system in order to achieve the goals
are:

- Each authenticated pseudonym corresponds to a unique user.

- Security of the user's master secret key: whatever can be computed
  about the user's secret key as a result of the user's interaction with the
  system, can be computed from his public key alone.

- Credential sharing implies master secret sharing.

- Unlinkability of pseudonyms: The nyms of a user should not be linkable
  at any time better than by random guessing.

- Unforgeability of credentials: A credential may not be issued to a user
  without the organization's cooperation.

- Pseudonym as a public key for signatures and encryption: This is an
  optional but desirable feature: the ability to sign with one's nym, as
  well as encrypt and decrypt messages.

(Un)Conditional Anonymity  This technique can be used to provide
conditional and unconditional anonymity: if we want a system with anonymity
control (conditional anonymity) then we must create a link between the
pseudonym and the real identity. In order to provide anonymity to honest
users, this link should be disclosed only if some authority (judge) requests it.

It is important to remark that all actions made under the same pseudonym
are linkable. That is, even if we do not know the real identity of the user we
can get very useful information by linking together his actions. For example,
a user opens an email account in a system like hotmail or yahoo, even if
he does not provide information about his real name, address, etc., we can
observe his behavior over the time and see with what people he is communi-
cating. If we want to prevent this, different pseudonyms should be used each
time.
Trust  There is a Certification Authority (CA) that certifies the public keys of the users and guarantees that users in the system can be trusted.

Users are trusted not to share their pseudonym. This is achieved by linking the use of the pseudonym to the private key, so users cannot share the pseudonym without sharing the private key.

Efficiency  In [29] a construction is proposed based on non-interactive proofs of knowledge to issue single-use credentials.

In this proposed protocol, to issue a credential, the effort required by the organization is 25 exponentiations; the work the user must do is 4 exponentiations.

To transfer a credential, the user computes 2 exponentiations and the organization has to compute 4 exponentiations plus the verification of the transcripts.

More efficient realizations could substitute zero-knowledge proofs by other proofs based on general assumptions as one-way functions and trapdoor permutations.

Security  Pseudonyms are unlinkable in an information theoretical sense.

Attacks  A dishonest CA can introduce invalid users into the system.

5.13.2  Examples

Pseudonyms can be used to provide anonymity in nearly all types of applications. Pseudonym systems have been proposed to achieve anonymity in electronic payments, anonymous email systems, etc.

An interesting example where pseudonyms can be used is the medical data protection. Medical records constitute sensitive information, and it is not desirable that employees, insurance companies and the hospital can link their records. On the other hand, the patient must be able to convince one organization of his relationship with another (for example, he wants to prove to the hospital that he has paid the medical insurance).

5.14  Deniable encryption

Described by Canetti, Dwork, Naor and Ostrovsky in [7]. In a deniable encryption scheme the sender can generate 'fake random choices' that will make the ciphertext 'look like' an encryption of a different cleartext.
CHAPTER 5. APPLICATION-LEVEL BASIC BUILDING BLOCKS 66

In deniable encryption each ciphertext has unique decryption, and at the same time can be opened in several ways for an adversary.

Deniable encryption can be classified into sender-deniable, receiver-deniable and sender-and-receiver-deniable, depending on the parties that may be coerced. We can also distinguish between shared-key schemes and public-key deniable encryption schemes. Another issue is the time at which the coerced party must decide on the fake message: at the time of attack or at the time of encryption.

It provides a tool to make unlinkable the entity and the data sent by this entity.

5.14.1 Properties

Assumptions and Requirements Some assumptions are made for the particular algorithm, such as: “Trapdoor permutations exist”, or “The unique shortest vector problem is hard in the worst case”.

It is required the availability of a (publicly known) faking algorithm, several proposals can be found in [7].

Unconditional Anonymity Deniability can be useful to provide unconditional anonymity. A third party who wishes to decrypt the contents of a message cannot know if the cleartext presented by the entities in the system in the real one or not.

Trust There is no trusted party in this scheme.

Security The system is secure if:

- Public key encryption: For any two messages, \( m_1 \) and \( m_2 \), the sender can send to the receiver, for the specific protocol, the communication for sending \( m_1 \) and \( m_2 \) encrypted with a public key are computationally indistinguishable.

- Shared key encryption: For any \( m_1 \) and \( m_2 \) and for a shared key \( k \) chosen at random, the encryptions of \( m_1 \) and \( m_2 \) with \( k \) are computationally indistinguishable.

Verifiability The goal of the scheme is that no external party can verify the decryption of a message.
**Attacks** In [7] an attack is described that suggests that no one-round scheme can enjoy negligible $d(n)$.

### 5.14.2 Examples

This technique is useful in electronic voting schemes, because it can be used to prevent vote-buying.

In a multi-party computation where parties use internal data to compute a common function deniable encryption allows parties to keep their internal data private in the presence of a coercer. In this kind of computation it is also a solution in the presence of an *adaptive* attacker.

### 5.15 Trusted Third Parties

A *TTP* (Trusted Third Party) is an entity that is *trusted* by the other entities of the system.

Any degree and type of trust is possible, and the characteristics will depend on the requirements of the application.

The role of a TTP is different in systems that require anonymity control and systems that do not.

**Conditional Anonymity** In services that require anonymity control (*conditional anonymity*) the task of a TTP is to revoke the anonymity of a user under *special* circumstances (i.e. when it is required by a judge). Ideally, this entity should not have any other power in the system.

Legal conditions to revoke anonymity are examined in part two of the general legal report (deliverable 4) regarding the legal conditions to provide on-line anonymity services. The obligations of service providers, faced with orders of preservation or disclosure of personal data are more closely examined.

**Unconditional Anonymity** In *unconditional anonymity* systems, the TTP is an entity that is trusted to keep the private data secret (*informed provider*).

**Distributed TTP** In many systems (implementing both conditional and unconditional anonymity), the TTP may be distributed. In this case the faulty behavior of one (or more) TTPs does not lead to a disclosure of private information.
5.15.1 Properties

The TTP is a flexible approach; we could say that it can be used to solve nearly any problem, at least in theory. The properties of the TTP solution are very much application-dependent. At the same time, the variety of possible services with very different characteristics that can be implemented using the TTP approach makes impossible a detailed summary of the properties.

The basic characteristic of such a solution is that the TTP is trusted to do what it is supposed to do. Some systems rely completely in a trusted entity, and some others use a TTP during some phases to perform specific operations.

There is an important drawback in the TTP approach that advises to look for alternatives to this easy solution: it is very vulnerable. An attacker that is able to gain control over the TTP will break the security of the system, and this is something that can happen in the real world.

In any case, many systems use a component that is trusted to some degree, for example to distribute keys, to keep certain data secret, etc. It is advisable, though, to reduce the power of the trusted entities to the minimum, if we want a system that gives confidence to the users.

This trust is in some cases distributed over several TTPs, in such a way that even if the attacker corrupts some of the trusted entities, the system keeps on working correctly.

5.15.2 Examples

The TTP approach can be used to provide any anonymity service, and it is, in practice, a very commonly used solution.

Systems that use a TTP as main building block to provide anonymity are, for example, some payment systems, rewebbers, remailers, etc.
Chapter 6

Composition of building blocks

The functionality of the basic building blocks described in the previous chapters is often too limited to be used in a stand-alone manner. This does not mean that they are not useful. On the contrary, the functionality of existing anonymity services is often a combination of several building blocks. Hence, in order to build more powerful anonymous services, we should enable building block cooperation in some way. In this chapter, we will tackle the first step towards operational and possibly even fully-automated block composition by describing the rationale and the mechanisms to do so.

Block composition combines the functionality of two or more blocks. Hereby, the result often gains additional anonymity properties. For instance, think of a reordering block and an encryption block. When used separately, they do not conceal in any way the path of the message. However, their combination does pretty well hide the traversed path.

As in the previous chapters of this deliverable, to describe block composition, we distinguish between connection- and application-level.

6.1 Connection-level composition

6.1.1 Goal

The primary responsibility of anonymity services at connection level regards both the privacy of the information contained within the body of a message and the anonymization of any other identifying information related to the message. To ensure the former, encryption of the message body is an adequate measure, which can be applied at application or at connection level. In fact, most of the building blocks that change the appearance are possible candidates for this requirement.
CHAPTER 6. COMPOSITION OF BUILDING BLOCKS

For the latter, there are two different sources of information that require attention. First, a message contains certain sensitive attributes such as the identity of the sender and receiver, the traversed path, etc. This information is available either explicitly (e.g. in message header or body) or implicitly (e.g. through connection identifiers). Attackers should not be able to read or deduce this information in any way. Notice that this type of information closely relates to the goal of one of the basic categories in the connection-level taxonomy: changing form. Second, normal network operation makes it possible to track down the traversed path of a message by following the message along the different network components. This is normally achieved by eavesdropping on the input and output channels of the components. Therefore, to protect this kind of information, it should be impossible to link input and output of intermediate entities involved in the connection. More formally stated, given input messages $M_1, M_2, \ldots$, one should not be able to link any of the output messages $M_{j1}, M_{j2}, \ldots$ to its corresponding input message. This requirement closely relates to the second important category in the connection-level taxonomy: changing flow.

6.1.2 Composition strategy

Based on the requirements described in the previous section, we can deduct the primary rule for the construction of connection-level anonymity services:

Always combine both form changing and flow changing building blocks.

Compliant to this rule, probably the most obvious composition is the combination of all building blocks. However, besides its lack of performance, some of the blocks can be removed from this composition without losing any important anonymity properties. For instance, latency and reordering blocks have more or less the same functionality (they both change the message flow using the same type of algorithm). Therefore, composing them does not result in much added value.

The specific algorithm to choose out of all building blocks a set that primo meets the anonymity requirements and secundo is as minimal as possible, is not clear to us yet. It depends on several factors, among which:

attacker model Depending on the capabilities and the strength of the attackers, the anonymity service should be more or less powerful. Active attackers for instance will be able to execute much more powerful attacks that passive ones.
CHAPTER 6. COMPOSITION OF BUILDING BLOCKS

anonymity requirements Are there, besides compliance with the primary composition rule, any specific anonymity requirements that must be met? For instance, anticipation of a particular anonymity attack might be a very specific requirement.

block properties Based on the requirements, the algorithm must choose a set of blocks primarily based on their type and exact functionality. Other properties such as sensitivity might come into play when trying to optimize the composition.

dependencies & order The algorithm must respect dependencies between blocks, if any. If the correct execution of a certain block requires results from another block, both must be included into the composition. Hereby, the algorithm must also take into account ordering requirements. If a block depends on another block, then the former must probably be executed later than the latter.

security vs performance There is often a tradeoff between security strength and performance. This also comes into play when there are different candidate blocks to choose from. Moreover, specific block parameters (such as randomness, key length, ...) might also be involved in this tradeoff.

As will be clear by now, composition of building blocks is far from straightforward. In our opinion, the ultimate goal of this part of the project is to come up with an algorithm that is able to perform this task automatically.

6.1.3 Composition setup

The previous section dealt with choosing an appropriate combination of blocks based on their functionality. Once a set is selected, the blocks can be composed using different setups. During our research, we have identified different kinds of setups. Hereby, we distinguish between local and global setups based on their granularity.

Note that for now, we have only identified several settings. We feel that the choice of setup might also influence the anonymity properties. In that case, this factor should also be part of the selection process. Research on this topic will be part of future work.

6.1.3.1 Local setup

A local setup deals with composing blocks into one execution unit. We have identified three different setups:
CHAPTER 6. COMPOSITION OF BUILDING BLOCKS

Chaining

In this setup, blocks are executed after each other whereby the input of a block is the output of the previous block in the chain. For instance, combining an encryption block and an reordering block will typically be performed with chaining. For this example, both orderings can make sense. This setup is also often called serial execution (as opposed to the next setup).

Parallel

Sometimes, the functionality of two blocks is totally unrelated. In that case, both blocks can be executed in parallel. One requirement for parallel execution is that at least one of the blocks does not change the contents of a message. If both blocks would do so, the result of parallel execution would be unpredictable. As an example, a caching block and a reordering block can be composed as such.

Nested
In this setup, the execution of one block is interrupted by the execution of another block. This type of setup is useful for advanced message transformations or block dependencies.

6.1.3.2 Global setup

To build powerful anonymizing services, local setups can again be combined. We distinguish between at least two:

Central server Many existing applications consist of one server that provides certain anonymity services. Applications can perform a certain task through this server. Most remailer and rewebber services work like this. This approach has two important disadvantages. First, since the server is a single point of failure, the anonymity service can be disabled quite easily. And second, since the server performs all the anonymizing tasks, he should be highly trusted.

Chaining Another type of global setup is to use several identical system in a chain. This setup is similar to the local chaining setup, except for its level of application and the fact that in this case, normally only identical systems are combined.

This setup solves the trust problem of the previous setup. Each particular entity in the chain has not enough information to reveal the identity of the communicating parties. The identity of the initiator is known to the first entity in the chain. The identity of the recipient is known by the last entity in the chain. Intermediate entities only know the previous and next entity in the chain. Thus, the identity of the communicating parties is revealed only if all entities in the chain are compromised.

To construct a chain, several techniques are used:

fixed chain All messages follow the same path in the chain.

initiator composes chain The initiator looks up the available entities that can be used for chaining. The initiator himself composes a data structure that contains enough information to set up the anonymous connection through a chain of entities towards the recipient. This case is useful if the initiator wants to remain anonymous.

recipient composes chain The recipient composes a data structure that contains enough information for the initiator to set up anonymous connection through a chain of entities. The recipient publishes this data
structure. If an initiator wants to connect to the recipient, he uses the data structure that the recipient published for this purpose.

**chain composition at set up time** The initiator forwards the message to an entity. This entity decides either to forward the message to the recipient or to forward it to a next entity that will be part of the chain.

**combination** The initiator or recipient composes the chain. An intermediate entity in the chain can still decide to embed a new chain towards the next entity in the original chain.

### 6.1.4 Case study: Onion Routing

Onion routing [34] provides anonymous connections that are strongly resistant to both eavesdropping and traffic analysis. Unmodified Internet applications can use anonymous connections by means of proxies. These proxies can also anonymize the communication by removing identifying information from the data stream. Onion routing has been implemented for web browsing, remote logins, and e-mail.

#### 6.1.4.1 Overview

The onion routing network is accessed via proxies. An initiating application creates a connection to an application specific proxy. That proxy defines a route through the onion routing network by constructing a specific layered data structure called an *onion*. To set up a connection, the proxy sends the onion through the network (actually to the first onion router in the connection). Each layer of the onion defines the next hop in a route. An onion router that receives an onion peels off one layer, identifies the next hop, and sends the embedded onion to the next onion router. After connection setup, the initiator's proxy sends data through the anonymous connection.

The last onion router forwards data to another type of proxy on the same machine, called the responder's proxy, whose job is to pass data between the onion network and the responder. Some literature makes no difference between the last onion router and the responder proxy. Funnels are the entry and exit points of the onion routing network.

An example onion routing network and anonymous socket connection is illustrated in figure 1.

In addition to carrying next hop information, each onion layer contains key seed material from which keys are generated to encrypt the data that is sent forward or backward through the anonymous connection.
CHAPTER 6. COMPOSITION OF BUILDING BLOCKS

Figure 6.1: An example onion routing network

Once the anonymous connection is established, it can carry data. Before sending data over an anonymous connection, the initiator's onion router adds a layer of encryption for each onion router in the route. As data moves through the anonymous connection, each onion router removes one layer of encryption. As such, it arrives at the receiver as plaintext.

6.1.4.2 Components

**Application specific proxy** The application specific proxy is the interface between the client application and the core proxy. If the application specific proxy accepts a new request, it creates a socket to the core proxy's well known port. The client proxy then sends a standard structure to the core proxy, followed by the ultimate destination address.

**Onion proxy** If the onion proxy accepts the standard structure, it proceeds to build the anonymous connection to the responder proxy (i.e. the last core onion router) using the standard structure, sends the standard structure to the responder proxy over the anonymous connection, and passes all future data to and from the application specific proxy and the anonymous connection. Figure 2 denotes the execution flow at an onion proxy. An number of encryptions are executed to both the onions and the data that is passed.
CHAPTER 6. COMPOSITION OF BUILDING BLOCKS

Core onion router (COR) The execution flow for onions and data is similar. They are both transferred as cells. Each connection between two core onion routers is link-to-link encrypted and multiplexed. To move a cell through the system, an onion router peels of the outermost layer and checks the freshness (not expired and not replayed). Moreover, padding is inserted to fix the length of all messages and the messages are reordered. Figure 3 shows the execution flow.

The last core onion router in the chain acts as responder proxy. It reads the standard structure that is the first data sent across the anonymous socket connection and establishes a connection to the ultimate destination. After this, it will blindly forward data between the anonymous connection and the connection to the responder machine.

Entry and exit funnel Funnels are the entry and exit points of the onion routing network. They mostly run on the same machine as the core onion routers. The entry funnel is responsible for multiplexing data and encrypting it (link-to-link encryption). The exit funnel is the link between the onion routing network and the responder.
Figure 6.3: Onion router execution flow

Figure 6.4: Entry/Exit funnel execution flow
6.2 Application-level composition

Most of the discussed application-level building blocks are meant to be complete solutions to realize an electronic election scheme or an electronic payment scheme. This means that the proposed solutions are not as application-independent as it would be desirable.

The complexity of the proposed schemes is rather high compared to the complexity of the techniques available at the connection-level: many of the proposed schemes present a model, and they solve the problem of anonymity combining protocols, algorithms and resources (i.e., reliable broadcast channels). Therefore, the studied building blocks are themselves, in some cases, a composition of several smaller building blocks, which do not offer anonymity by themselves.

Another problem we find when trying to combine application-level building blocks is that the functionality of the available building blocks is, in many cases, too different.

For the reasons explained above, the combination of different building blocks at this level may be very difficult in some cases. Nevertheless, the discussed techniques may be used as subprotocols, between different entities in different phases, to construct a complete system.

There are some systems that combine building blocks, not at the implementation level but at the mathematical level. We present in this section two examples: multiparty computation from threshold homomorphic encryption and fair blind threshold signatures.

It is also important to remark that the application-level building blocks should be, in most cases, combined with anonymous connections. Otherwise, the communication may be traced, despite of the effort made on providing anonymity at the application level.

6.2.1 Multi-party Computation from Threshold Homomorphic Encryption

In this scheme a collection of $n$ players can efficiently compute the value of an $n$-input function, such that everyone learns the correct result, but no other new information.

With such a system the private input of each player is not revealed to the others. This technique can be used to keep a user and his data unlinkable.

The threshold decryption is done using the shares of the secret key held by the players.
6.2.1.1 Properties

Assumptions and Requirements The protocol described in [12] assumes the existence of sufficiently efficient threshold cryptosystems. It requires a secure threshold encryption scheme with homomorphic properties.

Computational assumptions are made in some systems, like the quadratic residuosity assumption and the decisional Diffie-Hellman assumption are true.

In [12] it is assumed the existence of a semantically secure threshold public-key system, where the public key is known by all the players, while the matching private decryption key has been shared among the players, such that each player holds a share of it.

For the protocol proposed in [12] the existence of three secure (and efficient) sub-protocols (the two first ones rely on zero-knowledge proofs) is required:

- Proving you know a plaintext
- Proving multiplications correct
- Threshold decryption: the protocol computes securely the decryption of a ciphertext from the shares of the players.

Unconditional Anonymity The system keeps secret the inputs of the users, therefore it is useful to provide unconditional anonymity.

Trust Some systems require the existence of a trusted party in an initial phase where keys for the threshold cryptosystem are set up.

Efficiency The system is easily scalable. It can handle scenarios where a large group of players supply inputs, and a smaller group does the actual computation.

The threshold homomorphic encryption scheme avoids the need to VVS (Verifiable Secret Sharing) all values handled in the computation, and leads to more efficient protocols: supplying input values to the computation consists essentially of sending encryptions of these values.

Claimed to be the most efficient general Multi-party Computation protocol (proposed to date) for active adversaries [12].

Security The protocols [12] can be proved secure in the cryptographic model without relying on the random oracle model.

To achieve unconditional security, private channels are needed.
Verifiability The correctness of the end result is publicly verifiable.

Attacks This technique uses the properties of homomorphic encryption to build a multi-party computation system that is secure against an active static adversary that corrupts any minority of the players.

6.2.1.2 Examples
An electronic voting system may be implemented using this technique, as shown by Cramer, Damgard and Nielsen in [12].

6.2.2 Fair Blind Threshold Signatures
This scheme [27] combines the properties of fair blind signatures and threshold cryptosystems, and it is based on discrete logarithm.

The signing process is similar to the one described for threshold signatures: it is needed that at least $t$ out of $n$ signers collaborate to sign the message.

6.2.2.1 Properties
Assumptions and Requirements Any $t$ out of $n$ signers in a group can represent the group to sign fair blind threshold signatures.

Computing discrete logarithms and factoring are problems hard to solve.
Anonymous channels are required to make communication untraceable towards eavesdroppers.

Conditional Anonymity Any of the $t$ signers can collaborate with the trusted party to link the message-signature pair with the signer view of the protocol. The scheme is useful when we want to implement a system with conditional anonymity.

Trust In the scheme proposed [27] there is a trusted party (judge) that in collaboration with any of the $t$ signers can link the blinded and unblinded information.
At least $(n - t + 1)$ of the $n$ possible signers are trusted not to conspire with each other.
Efficiency  The size of a fair blind threshold signature is the same as that of an individual signature, and the signature verification process is equivalent to that of an individual signature.

In [27] a table with the number and type of operations required is shown.

Security  The security of these schemes relies on the difficulty of computing discrete logarithms or on the hardness of factoring.

Verifiability  The verification process is simplified by the use of a group public key and, if required by a judge, any of the \( t \) signers can link his view of the signing protocol to the message-signature pair with the information delivered by the judge.

Attacks  An attacker will successfully match the blinded and unblinded information if he corrupts the judge and one of the signers.

Anonymity will be lost by a user if the attacker is able to trace his messages.

An attacker who can corrupt \( t \) signers is able to create money in an electronic payment system.

6.2.2.2 Examples

Blind threshold signature schemes are useful in a distributed environment, where several signers work together to sign a blind threshold signature. These schemes allow \( t \) out of \( n \) participants in a group cooperating to sign a blind threshold signature without the assistance of a single trusted authority.

The fair blind threshold signature schemes can be directly applied to e-cash systems, while providing anonymity control. In this case, the power to create money is distributed over several signers, who need to collaborate to produce coins. This makes the system more robust against dishonest or absent signers.
Chapter 7

Conclusion and Future Work

To enable advanced evaluation and comparison of existing anonymity and privacy technologies, we have decomposed them into smaller, reusable components called basic building blocks. These blocks typically have one specific task and are often designed to anticipate one particular attack. Evaluation of blocks with limited functionality is much easier than dealing with complex systems. Existing systems can then be compared based on the set of blocks they are composed of.

Different types of building blocks exist. First, they can be separated based on the specific level in the application they are deployed at. For each level, different types of tasks must be performed. In order not to loose the overall picture, we presented a block taxonomy. Based on this categorization, we described in detail the functionality and various other properties of each building block. For now, this is based on informal properties.

To build complex anonymity systems, several building blocks must be composed. For this purpose, we first described the requirements. Furthermore, we presented different ways of how this can be achieved. Hereby, we distinguished between application- and connection-level blocks.

The results presented in this deliverable open up new research. In the future, we will focus on defining a more formal model for anonymity properties of building blocks, which will allow us to exactly analyze and describe the characteristics of existing techniques. Furthermore, this will be used to build custom-tailored anonymity systems. We consider building a generation tool that will help us to automate this process.
Bibliography


BIBLIOGRAPHY


[34] M.G. Reed, P.F. Syverson, D.M. Goldschlag, “Anonymous Connections and Onion Routing”.


BIBLIOGRAPHY


