Ing. Mark Robert Bergmoser, BSc

Secure collaborative editing of shared documents on untrusted servers: Preventing a third party from reading your data

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Software Development And Business Management

submitted to

Graz University of Technology

Supervisor

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

Institute of Applied Information Processing and Communications (IAIK)

Advisor

Dipl.-Ing. Florian Reimair

Graz, October 2017

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

SIGNATURE INFORMATION	**Signatory**	Ing. Mark Robert Bergmoser
	Date/Time-UTC	2017-10-31T10:28:32+01:00
	Verification	Information about the verification of the electronic signature can be found at: https://www.signaturpruefung.gv.at
Note	This document is signed with a qualified electronic signature. According to Art. 25 para. 2 of the Regulation (EU) No 910/2014 of 23. July 2014 ("eIDAS-Regulation") it shall have the equivalent legal effect of a handwritten signature.	

Date

Signature

Abstract

Exchanging information among team members in a fast, easy and secure way is, especially for companies, essential for maintaining competitiveness. For this purpose, so called text collaboration tools hosted on a server in the cloud, are used on a large variety of devices. The main drawback of common tools is that data is always readable for the server itself. Since servers are considered as honest but curious, data might be read and analysed. Furthermore, servers might get compromised and attackers might be granted access to sensitive data. Available tools which support encryption of data are often restricted to single user usage only or implement critical steps such as sharing keys among clients. Preventing servers from being able to decrypt the processed data along with an improved key management might enhance data privacy. In the context of this thesis, the software STeCT has been developed, aiming to present a prototypical text collaboration tool addressing these issues in existing solutions. STeCT consists of a server, which stores the document and provides an algorithm capable of merging data solely by its metadata. The Server furthermore provides software to the clients enabling them to send encrypted data, which is not decipherable by the server itself. For encryption and decryption as well as for solving the problem of key distribution, CrySIL is used by the clients. The tool was successfully tested and first performance tests showed a high applicability of STeCT.

Kurzfassung

Der Austausch von Information innerhalb von Teams, speziell innerhalb von Firmen, muss schnell, einfach und sicher erfolgen um Wettbewerbsfähigkeit aufrecht zu erhalten. Für diesen Zweck werden sogenannte Text Collaboration Tools verwendet, die auf einem Server in der Cloud gehostet und für viele verschiedene Geräte zur Verfügung gestellt werden. Der Nachteil dieser Tools ist, dass der Server die Möglichkeit hat, die Daten der Benutzern zu lesen und zu analysieren. Server können somit als ehrlich, jedoch auch als neugierig angesehen werden. Ferner können Server von Kriminellen kompromittiert werden, welche dann möglicherweise auch Zugriff auf sensible Daten erhalten. Verfügbare Tools, welche die Daten verschlüsseln, sind oft nur von einem Benutzer gleichzeitig nutzbar oder sie implementieren kritische Vorgänge wie zum Beispiel den Schlüsselaustausch. Wenn für Server nicht mehr die Möglichkeit bestehen würde, die verschlüsselten Daten entschlüsseln zu können und das Schlüssel-Management verbessert werden würde, könnte die Datensicherheit stark verbessert werden. In dieser Arbeit wurde das Tool STeCT entwickelt. Dabei handelt es sich um ein sicheres Text Collaboration Tool, welches die oben genannten Probleme löst. STeCT besteht aus einem Server, welcher das Dokument speichert und einem Algorithmus, der nur anhand der Metadaten von verschlüsselten Daten Änderungen von Benutzern in das Dokument einpflegen kann. Ferner stellt der Server die Software für die Benutzer bereit, um diesen das Bearbeiten des Dokuments zu ermöglichen und die gesendeten Daten zu verschlüsseln. Für die Ver- und Entschlüsselung sowie für das Schlüsselmanagement wird CrySIL verwendet. Das Tool wurde erfolgreich getestet und erste Leistungstest zeigen eine hohe Eignung von STeCT.

Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

ACL	Access Control List
AES	Advanced Encryption Standard
CBC	Cipher Block Chaining
CPU	Central Processing Unit
CrySIL	Crypto Service Interoperability Layer
HDD	Hard Disk Drive
HOM	Homomorphic encryption
HSM	Hardware Security Module
HTTPS	Hypertext Transfer Protocol Secure
IDE	Integrated Development Environment
IV	Initial Vector
JSON	JavaScript Object Notation
NIST	National Institute of Standards and Technology
OS	Operating System
OT	Operational Transformation
P2P	Peer-to-Peer
RAM	Random Access Memory
REST	Representational state transfer
SQL	Structured Query Language
SSH	Secure Shell
UPS	Uninterruptible Power Supply
WLAN	Wireless Local Area Network

1 Introduction

Collaboration among team members within a company, among customers, vendors or business partners is very important for companies to remain competitive. Especially in a digital environment it is necessary to share, exchange, edit and distribute data and knowledge in a fast, easy and secure way. Text documents (in the following just called documents) are one way to satisfy these requirements.

With classical document software like *Microsoft Office* [35], *Libre Office* [18] or *Open Office* [17], only one person at a time is able to edit the content of a document. Sharing the latest version of the document might be a challenge since every user has to have access to the location of the document which is used to be a shared folder on a network drive. This might be an issue if a team member is not in the local network of the company. Also merging different versions of the document is a very difficult process, since most of these documents are not containing pure text. Furthermore, every member of the project needs a local installation of the software on the computer in order to be able to read and edit the document. This increases the costs of licences and maintaining the software on each computer. Since classical document software is often only available for desktop computers, smartphones and tablets cannot be used or their use is limited to editing this kind of documents. In summary, using available document software has some drawbacks regarding collaborative text editing. However, they are still often used in companies since until now there is no secure and applicable alternative. As the study of Dimensional Research [12] shows, collaboration is very important for business professionals. To overcome the mentioned issues, text collaboration tools like *Google Docs* [24] or *Word Online* [36] have been developed.

1 Introduction

Text collaboration tools allow any number of users to simultaneously edit the same text document in real-time on any device having access to the Internet and supporting a web browser, independent of the used Operating System (OS) and the used browser. This can be achieved by moving the logic of the text collaboration tool into the cloud. At the login, the client receives the latest version of the document, called the master document, from the server. When a client edits the document, only a changeset is sent to the server. A changeset describes a set of changes (some kind of delta), *e.g.,* a word which was added or deleted by the user. In addition, the position of the text, the version of the change, etc. is included in the changeset. Figure 1.1 illustrates the basic architecture of a typical text collaboration tool. The server in turn merges the changeset into the latest version of the document and distributes the change to all connected clients. Under these conditions, every client has the latest version of the document at any time. If two or more clients are editing the same passage within a document, the server takes care of merging the changesets so that no changeset gets lost.

An advantage of text collaboration tools is that they do not need a local installation of the software since they can be executed directly in the client's web browser. This means, the client always uses the latest version of the software. It follows that text collaboration tools can also be used on smartphones and tablets because they do not need a local installation. This kind of document editing is very attractive to companies since they have less software to maintain and the distribution of documents is very easy. Also the requirements are very low. A web browser and an Internet connection are sufficient to run the text collaboration tool on any device.

When outsourcing the documents to the cloud, some issues arise concerning, *e.g.,* data security, availability, access permissions, etc. The provider of the cloud service has to be trustworthy since the server may be able to read the content of the document and may analyse it. This is due to the fact that for most standard collaboration tools only the connection from the user to the

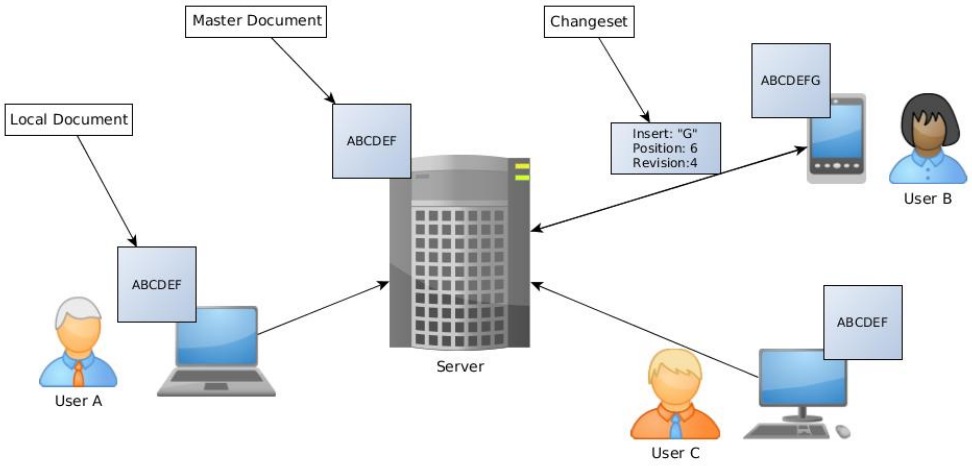


Figure 1.1: Principle architecture of text collaboration tools. The server always holds the latest version of the document.

server is encrypted and not the content itself. Even if the content is encrypted, the server has to be capable of performing a decryption in order to be able to merge the changesets from the clients. For companies, this might be a problem in case the document contains sensitive data. Often, depending on the terms of service, the provider reserves the right to access and analyse the content of the document in order to enable, *e.g.*, personalised advertising or to improve the service. Further, some states may be able to carry out a lawful interception, like the Unites States of America. Because of the *USA PATRIOT Act* [37], the government of the United States of America is able to legally retrieve information of documents stored by cloud providers, independent of the terms of service. Also changing the cloud provider may constitute a security issue. The company has to trust the old cloud provider to securely delete the data, so that it is impossible to restore the data for anyone. Another problem arises if the cloud provider gets compromised and the document gets copied by criminals. If the document contains sensitive data, this may cause the economic ruin of the company.

Basically, there are two options to avoid a third party from analysing and/or stealing data from documents stored in a cloud. The first option is to host an own cloud. However, this has some important disadvantages like maintenance of the hardware and software, costs for the server, availability, prevention from unauthorized accesses, etc. There are many aspects to be considered when hosting a cloud on one's own. The other option is to encrypt the content of the document in such a way, that only the authorized clients are able to read the encrypted documents. Logically, the server should not be able to read the documents but it should be able to merge the changesets of the different clients although the content is encrypted. There are some solutions for encrypting the changesets of the document, like the Browser add-on from Adkinson-Orellana *et al.* [1], but they are not supporting collaboration of multiple users because merging encrypted content is not an easy task.

The aim of this thesis is to develop a strategy for merging encrypted changesets on the server side. Neither the server nor any third party should be able to decrypt the content of the document at any time, only authorized clients should be able to decrypt and read the document. To achieve an appropriate level of security, CrySIL [39] should be part of the user management concept and act as a central cryptographic provider for encrypting and decrypting data.

The content of this thesis is structured as follows. Chapter 2 gives some background information on the used technologies. Chapter 3 describes CrySIL, which is a remote key storage and cryptographic provider for heterogeneous platforms. In Chapter 4, state of the art concepts and tools are described. Chapter 5 presents the developed tool STeCT which is able to merge encrypted content without the need to decrypt the data first. A security evaluation and a performance test of STeCT are outlined in Chapter 6. Chapter 7 shows further research opportunities and concludes this thesis.

2 Background

This chapter gives an introduction into text collaboration tools. The idea is to get familiar with the principal workflow and the components which are necessary to operate a text collaboration tool.

Section 2.1 describes the components and elements which are necessary to execute a text collaboration tool. Section 2.2 explains what merging is and how it is used for collaboration purposes. Section 2.3 introduces the typical workflow of a text collaboration tool. Section 2.4 gives a summary of this chapter.

2.1 Components and Elements

In this section, the basic components of a text collaboration tool and the basic elements are described (see Figure 1.1). This aims to help in getting a better understanding of each component's function and their interconnections.

2.1.1 Server

The server represents the central element of the necessary infrastructure around text collaboration tools. It is responsible for a range of tasks, the most important ones are described in the following:

- **Hosting the master document**: In a usual environment, the server is the only component which stores the master document. Typically, the clients only store a local copy of the master document as long as they are connected to the server. Therefore, the server is responsible for saving the master document securely. Furthermore, it is responsible for the distribution of the document.
- **Merging of changesets**: If a client is editing the document, it sends changesets to the server in order to notify it about the local changes done by the user. The server in turn merges this changeset into the master document to keep it up-to-date. A detailed description of the merging process is provided in Subsection 2.2.
- **Distribution of changesets**: After the server received and merged the changesets into the master document, it also distributes the changesets to all connected clients. It follows that all connected clients always receive the latest version from the server.
- **Hosting software**: Since most text collaboration tools are web-based, the server hosts the software which is executed at the client to run the web-application. After the login, the software (*e.g.,* written in *JavaScript*) is sent to the client and executed locally at the clients device as long as the client is connected to the server. This ensures that the client always runs the latest version of the software. Also the distribution of software is much easier without any local installation.

Hosted vs Self-Hosted

In general, there are two ways for offering a public web service on the Internet: a hosted server or a self-hosted server. Possessing a hosted server means to rent an infrastructure, *e.g.,* a virtual machine with an operating system to run applications. In contrast, a self-hosted server means owning and operating the server in a private infrastructure. Both variants have advantages and disadvantages in costs, security, etc. and will be discussed in the next section.

Hosted: Nowadays, there are many companies which are offering servers or virtual machines for rental, like *Amazon Elastic Compute Cloud (Amazon EC2)*[1], *Rackspace*[2], *Google Cloud Platform*[3] etc. Of course there are many more providers. It is very easy to rent and operate a hosted server, it is just a few clicks. The user first configures her virtual machine, like the used OS, number of Central Processing Unit (CPU) cores, size of Random Access Memory (RAM), size of Hard Disk Drive (HDD), etc. Afterwards, the user accepts the monthly price, agrees to the terms of service and the virtual machine is ready to go. The user can cancel the contract at any time with compliance to the period of notice if the virtual machine is not needed any more.

According to Ziff Davis Enterprise [15], a particular advantage of a hosted server is its scalability. If the user needs more computing power , *e.g.,* more RAM or more CPU cores, the virtual machine can easily be reconfigured to the customer's needs. Furthermore, the customer does not need to care about the infrastructure of the server, the availability, Internet connection or other issues like backups, maintenance of hardware, Uninterruptible Power Supply (UPS), etc.

In contrast to the advantages of a hosted server, there are also some disadvantages which have to be considered. Omer Tene [52] states in his article that most of the cloud providers have a clause in their terms of service which allows them to legally read, analyse and even change the data of the user. Often cloud providers justify it with improvements of their services. This trend is raising concerns. Not only cloud providers, also governments are able to legally read the data stored by the cloud providers, independent of their terms of service. For example, the United States of America amongst others released a law called *USA PATRIOT Act* [37] in 2001 allowing them to read data from cloud providers. Furthermore, Ristenpart *et al.* describe how it is possible to extract information from one virtual machine by another virtual

[1] `https://aws.amazon.com/ec2/`
[2] `https://www.rackspace.com/`
[3] `https://cloud.google.com/`

machine whereby both are running on the same physical hardware. Further, Maurice *et al.* [33] demonstrated that it is possible to create a robust cache covert channels between different virtual machines. Under these circumstances an attacker might be able to extract sensitive data from other users using the same cloud provider and being hosted on the same physical server. For most of the cloud providers the user cannot choose in which country the server is hosted meaning where the data is stored.

In summary, a hosted server has many advantages like its scalability or that the user does not need any infrastructure. There are also some drawbacks like privacy and data protection against third parties. In other words, if the data are stored in an encrypted way at the server, a hosted server is a good choice.

Self-Hosted: In contrast to a hosted server, a self-hosted server means to own the server and the corresponding infrastructure for running applications. Owning a server brings along many advantages but also comes with some drawbacks.

First of all, owning a server means that the owner can decide which hardware should be used , *e.g.*, which CPU, how much RAM, size of HDD, etc. The server can be customized independently. The applications on the server can be executed natively, which often brings along an improvement of the performance. Basically, there is no need for a virtual machine except the user has special use cases.

The drawbacks of a self-hosted server are versatile. One major drawback in contrast to the hosted server is the scalability. Once it is configured and up and running, it is difficult to change the hardware. For example, if the server needs more CPU cores, maybe also the motherboard has to be exchanged. Apart from this issue, the server and the service running on it need to be shut down during the time of maintenance if there is no backup server which also hosts the service in parallel. According to Koomey *et al.* [31] there are

many cost factors which have to be considered. It is not only the costs of the hardware, there are also other cost factors like electricity (for the server, cooling, UPS, lights, losses, etc.), hardware maintenance, wiring, personnel costs (IT staff, security staff, facility staff, etc.), software, facility costs, etc. Another issue is the security of the server. The server needs a firewall, antivirus software, a certificate for a secure and trusted client-server communication, regular updates of the OS and the used software, etc.

In summary, the self hosted server has some advantages concerning the privacy of data and the control of the hardware, but it is very expensive, needs a lot of configuration effort and requires maintenance.

Summary: Table 2.1 summarizes the most important advantages and drawbacks of a hosted and self-hosted server which have been discussed in this section. Although there are drawbacks with privacy and data confidentiality, a hosted server might be the right choice if the user does not want to investigate much effort in configuring and maintaining the server. To overcome the privacy issues, a new concept like introduced in this thesis or some of the state-of-the-art tools described in Chapter 4 *State Of The Art* could be used.

2.1.2 Client

The client is another central component in a text collaboration tool. It is used by the users who actually create and edit a shared document stored on the server. As the server, the client has some important tasks to do, like merging changesets into the local version of the document, creating changesets when the user edits the document, etc. Simultaneously, the client has to satisfy a lot of user requirements regarding security, simplicity, supported devices, etc. In the following, this tasks and requirements are described in detail:

	Advantages	Drawbacks
Hosted server	• Scalability • No infrastructure needed • Fixed costs per month • 24/7 service	• Privacy/data protection • U.S. government can access the data • Security issues like getting data from other VMs • No control of hardware
Self-hosted server	• Full control of hardware • Native execution of software • Hardware matches requirements	• Hight costs • High configuration effort • High maintenance effort

Table 2.1: Advantages and drawbacks of hosted and self-hosted servers.

- **Merging of changesets:** The merging process of the client is different to the merging process of the server because the client only receives changesets from the server and not from other clients. As a result, the process of merging in this case is just applying the changeset sent by the server to the local document of the client to keep it up to date. By updating the local version, also the revision number is updated to the one sent by the server. The role of the revision number is explained in 2.2 *Merging*.
- **Creation of changesets:** Every time the user edits the document, a new changeset is created and sent to the server. This changeset contains all necessary information about the change itself, *e.g.*, what has changed, at which position the change happened, on which revision number the change based on, etc. Section 2.1.3 *Changesets* provides a detailed description of a changeset. The revision number of the changeset is very important for the server, because it is used to identify merge conflicts (two changes are based on the same revision number) between clients.

- **Easy to use:** The client is a software which is typically used by the end customer. Therefore, it should be very easy to use. Ideally, the software is self-explanatory and the user does not have to read a manual. This includes a simple and clear design. Furthermore, the client software should support as many OS and devices as possible.
- **Support of different devices:** Nowadays, however, the requirements for a software are completely different than just a few years ago. Today, beside the classic desktop computer, there are a lot of mobile devices (especially smartphones or tablets) with limited hardware. As a result, applications cannot be ported directly to mobile devices. Furthermore, a lot of different OS exists, like *Windows*[4], *Linux*[5], *Mac OS*[6] for desktop computers or *Android*[7], *iOS*[8], *Windows 10 Mobile*[9], *Symbian*[10] for mobile devices. For a software, especially for mobile devices, it is not easy to support many OS [57]. This is due to the fact that a developer needs knowledge about different programming languages and runtime environments like J2ME for Android, Objective C for iOS or C++ for Symbian [34].

  To overcome these issues, a text collaboration tool is mostly based on HTML [54] and JavaScript [56], which is supported on nearly every OS and device. In fact, the application only has to be developed once and can be used on a variety of different devices with different OS. As a result, the developing and maintaining costs of the software can be reduced enormously.

[4] https://www.microsoft.com/en-us/windows/
[5] https://www.linux.com/
[6] http://www.apple.com/macos
[7] https://www.android.com/
[8] http://www.apple.com/ios/
[9] https://www.microsoft.com/en/mobile/windows10/
[10] http://www.nokia.com/en_int

2.1.3 Changesets

Changesets are used to edit text in a standardized way within a collaborative environment and transmit information about what has been changed in a document between the server and the client.

A changeset is used to edit the master document, i.e., it is possible to insert, delete or format data with it. The insertion of deletion of data may be combined with a formatting, but this depends on the implementation of the text collaboration tool. In general, a changeset is a combination of data and the corresponding operation. Moreover, the data not only consists of the data which should be edited, the data also contains the position of the change in the document and the revision number it is based on. The revision number is used to detect merging conflicts (see Section 2.2 *Merging*). Some text collaboration tools like *EtherPad Lite* [19] are also adding the author to the changeset. Listing 2.1.3 shows an example of a real world changeset from *EtherPad Lite*. This example inserts the word "Hi" into the master document. For security reasons, the changeset may also contain a hash or a *signature* [10] of the changeset itself in order to detect undesired changes caused by transport errors or data manipulation by attackers. In summary, every changeset must contain at least following attributes:

- The **Text** which should be inserted or deleted
- An **Operation** like insertion or deletion
- A **Position** where to insert/delete the text
- The **Revision Number** on which the changeset is based on

```
...
"data":{"baseRev":0,"changeset":"Z:dn>2|7=dm*0+2$Hi"},
"apool":{numToAttrib":{"0":["author",␣"a.0h14LrJ1959Kyvz1"]}}
...
```

Listing 2.1: Example of the relevant part of a changeset from *EtherPad Lite*.

2.1.4 Master Document

The document which is stored at the server is called master document. It always has the latest revision number and is downloaded by the client at login. After the download, the client has a local copy of the current master document. If a client is editing the local document, a changeset is generated by the client and sent to the server. The server in turn merges the changeset into the master document, increments the revision number and distributes the latest change to all connected clients. This guarantees that the server version of the document is always the latest one.

The master document can be seen as an empty document with a list of changesets attached [3]. As a result, the client has to download the whole list of changesets and applies them to an empty document locally. Downloading and applying thousands of changesets may take some time and is therefore not suitable for business software. To improve the performance, the server creates so called checkpoints after, *e.g.*, 50 changesets or some defined interval. This means, the server creates a new document with the data from the last, *e.g.*, 50 changesets including the data of the previous checkpoint. The previous or empty document is exchanged by the new generated checkpoint. This avoids the changeset list from becoming too large and therefore decreasing the performance.

2.1.5 Real-Time Text Collaboration

A text collaboration tool allows multiple users to edit a shared document with different devices independent of the actual location of the user. In principle, there are two types of text collaboration tools: real-time and non real-time collaboration. The difference between this two types is that with a real-time text collaboration tool multiple users can edit a shared document at the same time, whereas with non real-time text collaboration tools users are not editing a shared file at the same time (these tools are similar to a revision system like *git*[11] or *svn*[12]).

As described by Ellis *et al.* [14], a real-time collaboration tool can be characterized by the following attributes:

- **highly interactive:** The response time must be kept as short as possible.
- **real-time:** The notification time should be as close to the response time as possible.
- **distributed:** Several users can work with different devices over the Internet together.
- **volatile:** Users can login or logout at any time.
- **ad hoc:** Users do not have to follow a given way of working.
- **focused:** There is a high probability of merging conflicts because many users are editing the same document at the same time.

According to Romanowski *et al.* [44], there are two important parameters: The first one is the *response time* and the second one is the *notification time*. The response time is the time needed to display the changes made by the user to the users interface. This includes, *e.g.*, time for computing the user input, refreshing the screen, etc. The notification time is the time needed to propagate the changes made by a user to all other users over the network. This includes, *e.g.*, network latency.

[11] `https://git-scm.com/`
[12] `https://subversion.apache.org/`

The entire complexity of real-time collaboration tools is a result of communication latency in computer networks. Obviously, if there would be no latency in computer networks, a text collaboration tool would have the same complexity as a single-user editor. This is due to the fact that merging conflicts only occur if two users are editing the same revision of a document without knowing (because of network latency) that another user is also editing the document at the same time. Without any network latency, every user would notice the changes of other users immediately and therefore will always have the latest version of the document instantly. As a result, merging conflicts would not be possible any more.

2.2 Merging

In general, merging means to combine or unite two or more entities to a single one. It is an essential process of collaborative working. Without this process, it would not be possible to operate collaboration tools in general. In the case of text collaboration tools, merging means to integrate a changeset, created by a user, into the master document on the server. As a result, the server increases the revision number of the document which is now representing the latest version of the document and distributes the changes to all connected clients. Figure 2.1 illustrates a simple merging process.
When multiple users are working with a text collaboration tool, especially a real-time one, sooner or later merge conflicts will occur. Simply put, a merge conflict means that two clients are editing text based on the same revision number. This does not necessarily mean that both are editing text at the same position. Figure 2.2 demonstrates an example of a merge conflict. As a result, both users have different versions of the document (”AB” and ”BA”) which results in inconsistency. This is because of unpredictable network latency when sending the changeset to the server. Thus, the goal is to have a merging algorithm on the server which provides consistency of the master document and the local version of all clients.

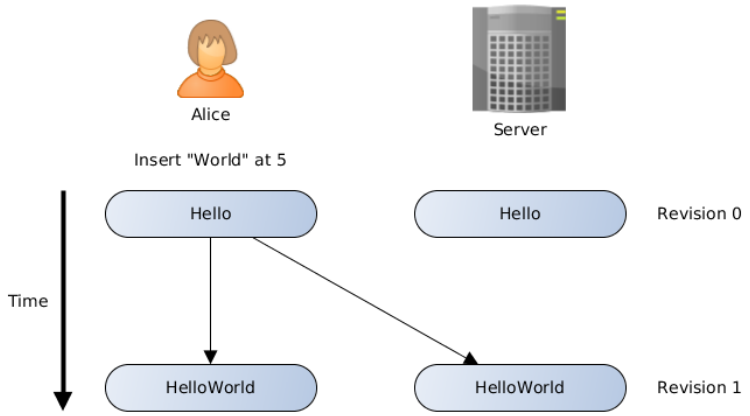


Figure 2.1: Example of merging a changeset into the master document on the server.

Creating a merging algorithm is not an easy task. Many things have to be considered to get it to work properly. Section 2.2.1 *Inconsistency Problems* introduces some inconsistency issues a merging algorithm has to take care of.

2.2.1 Inconsistency Problems

To keep consistency of a shared document, concurrency control is needed to overcome issues caused by the latency of the network. If no such mechanism is active, every user might have a different version of the document. Figure 2.3 illustrates a scenario with three users editing and sharing a document without concurrency control. In this scenario, operation $O1$ is created by user Alice and $O2$ and $O3$ are created by user Bob. The operations will be applied to the current local copy of the respective user. According to Sun *et al.* [50], three major issues caused by the unpredictable latency of the network can be identified when using a real-time text collaboration tool. [50]

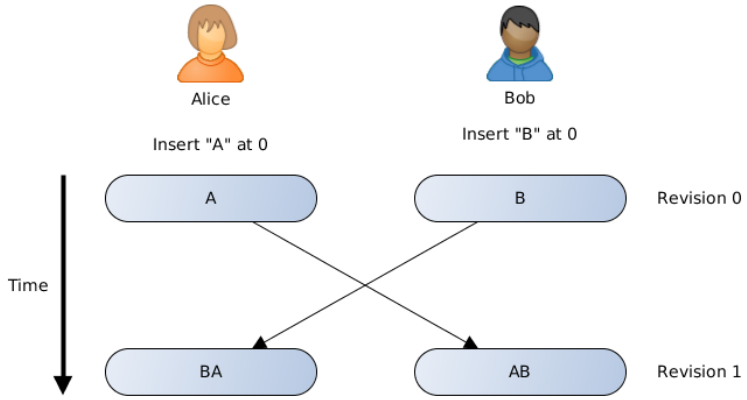


Figure 2.2: Example of a merge conflict when two users are editing the document at the same position based on the same revision. Because of unpredictable network latency, both users have a different version of the document.

- **Divergence problem:** Since the operations are executed in the order they are arriving, they are not in the same order for all users. Alice's execution order will be $O1$, $O2$, $O3$, Bob's $O2$, $O1$, $O3$ and Charlie's $O2$, $O3$, $O1$. As a result, every user has another version of the document. Assume that $O1$ inserts an "A", $O2$ a "B" and $O3$ a "C". The document for Alice would be "ABC", "BAC" for Bob and "BCA" for Charlie.
- **Causality violation:** Bob executes his operation $O3$ after he received operation $O1$ from Alice. Therefore, it might be possible that $O3$ depends on $O1$, *e.g.*, $O1$ is a question and $O3$ is the answer. Since Charlie receives $O3$ before $O1$, he might be confused by having the answer before the question.
- **Intention violation:** Because $O1$ and $O2$ are created at the same time by different users, both operations are independent of each other. When Alice applies $O2$ to her local document, she has another base document than Bob had when he created $O2$. By implication, the result may be different to the intended one of Alice or Bob, respectively. For example,

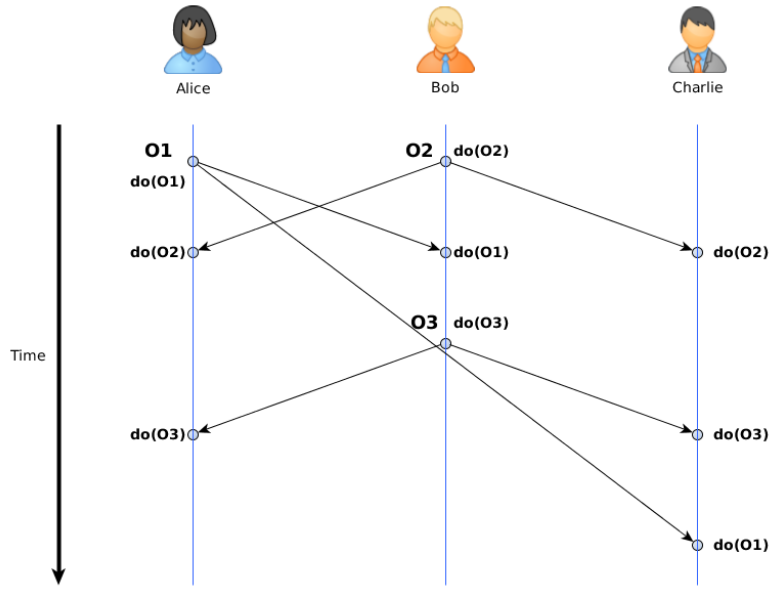


Figure 2.3: Scenario of a text collaboration tool without concurrency control. The changes are applied on the local document of the user.

assume that the shared document contains the text "ABCDEF" before $O1$ and $O2$ are executed. Now, Alice wants to insert two characters ("12") at position 1 which should result in the text "A12BCDEF" and Bob wants to delete two characters at position 2 ("CD") which should result in the text "ABEF". After executing $O1$ and $O2$, the intended result at all users should be "A12BEF". Because of missing concurrency control, the result in Alice's document is "A1CDEF", which is neither the expected result of Alice, nor of Bob. This is due to the fact that $O2$ is executed on another base text ("A12BCDEF" instead of "ABCDEF") As a result, the text "2B" is deleted instead of "CD".

To overcome these inconsistency problems in collaborative working, a lot of algorithms and protocols implementing different strategies have been developed. The following list gives a short overview of existing protocols:

- **Turn-taking protocols:** Only one user is allowed to edit the document at a time. [26, 48]
- **Lock-based protocols:** The data object is locked before updating. [27, 29]
- **Transaction-based protocols** as proposed by [30, 5]
- **Optimistic execution protocols** as proposed by [28, 49]

2.3 Workflow

The prior chapters described only single parts of the workflow of text collaboration tools. To put all the pieces together, this chapter describes the general workflow from a high level view. This should give an idea of how a text collaboration tool works in general.

In principle, every user can perform three different actions: receiving the current master document from the server, creating and deleting text on the local document and sending and receiving changesets from the server. Although there exist solutions for text collaboration tools without the use of a server, most of them are using a server as central component for storing the master document. In general, the client only communicates with the server and not with the other connected clients.

After a successful login, the client receives either an empty document with a list of changesets or a so called *checkpoint document* with a list of changesets from the server. Figure 2.4 visualises the login process. A checkpoint document is used to increase the performance of creating the local document at the client side. This is due to the fact that the client does not have to apply all changesets from the beginning until the latest change

on an empty document (the more changesets the client applies the more time it needs), instead the client receives a checkpoint document and a list with considerable less changesets. Figure 2.5 visualises the difference between an empty document and a checkpoint document. This checkpoint document contains all changesets from the beginning to a defined amount of changesets. It is generated automatically by the server, *e.g.*, when the list of changesets exceeds a certain number of changesets. After creating this checkpoint document, the list of changesets is empty again. There might be several checkpoint documents, but only the latest one is downloaded by the client. All changesets from the beginning on are stored and will not be deleted by the server. By doing so, the server holds a history of the document and the clients are able to restore any revision of the document.

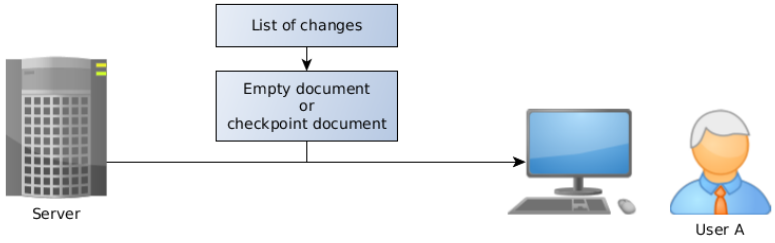
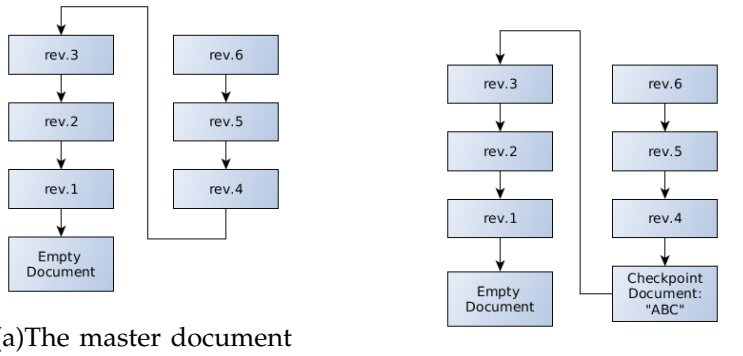


Figure 2.4: At the login process, the client receives either an empty document or a checkpoint document and a list of changesets.

After downloading the latest version of the document, the client can start editing the document. Typically, the text collaboration tool checks periodically, *e.g.*, every second, if the user has done some changes to the local document and sends them as a changeset to the server. The server in turn checks the validity of the changeset and applies it to the master document with a new revision number. If necessary, a merge (see chapter 2.2 *Merging*) has to be done by the server. After this process, the new changeset is distributed to all connected clients.



(a)The master document consists of an empty document and a list of changesets. The client has to apply each single changeset to the empty document.

(b)The client receives the checkpoint document and has to apply only few changesets which increases the performance a lot.

Figure 2.5: Difference between a document with and without a checkpoint.

When the client receives a new changeset from the server, it is applied to the local document of the client (see Figure 2.6). This mechanism ensures that the client always has the latest version of the document to prevent merging conflicts with other clients.

2.4 Summary

In this chapter, first the components and elements of a text collaboration tool were explained to get a basic knowledge about the topic. Next, the merging process was explained. It is important to understand why a merging conflict occurs and how it can be solved. The next section covered the typical workflow of a text collaboration tool. It explains how the document is stored on the sever, how it is distributed to the connected users, which actions a user can perform and how to deal with changesets.

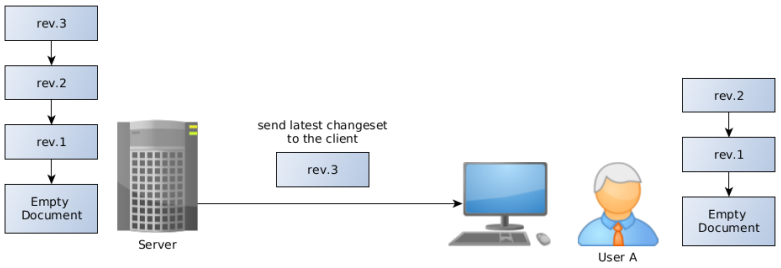


Figure 2.6: After the server applied the latest changeset (rev.3) to the master document, it sends it to all connected clients which in turn apply the changeset to their local document.

3 CrySIL

Nowadays, a high heterogeneous landscape exists for developing applications for different devices. Above all, the diverse operating systems for mobile devices like smartphones, tablets and smart watches differ a lot from each other, *e.g.,* for each mobile OS, applications have to be written in different programming languages (Android uses Java, iOS uses Objecive C and Windows Phone uses C# and C++). Further, each mobile OS has different concepts for dealing with, *e.g.,* security, applications, performance, etc.

Especially security related aspects may suffer from this amount of heterogeneous platforms like cryptographic protocols. The same protocol has to be implemented several times in different programming languages in order to cover all kinds of systems. This makes the use of cryptography really hard. Reimair *et al.* [40] defined three problems with heterogeneous systems related to security:

- Managing cryptographic keys on multiple platforms might be difficult because of missing key storage facilities.
- Some cryptographic protocols might not be available on every platform
- Because of the complexity of today's cryptographic protocols, the likelihood of bugs in the implementation increases [13].

To overcome this issues, Reimair *et al.* [39] developed the so called Crypto Service Interoperability Layer, also known as *CrySIL*. In this thesis, CrySIL is used as a secure central cryptographic key storage and key manager, as a cryptographic provider and as an authentication service. These three tasks are explained in detail in the next sections.

3.1 Cryptographic Key Storage

In order to perform cryptographic operations on data, so called cryptographic keys are used to, *e.g.*, encrypt/decrypt data or to calculate a digital signature. Usually, the cryptographic key is stored on the device doing the cryptographic operation or on a portable medium like a smartcard. As long as the user uses just a single device, this course of action does not cause any problems. If the user uses several different devices, it gets complicated. Especially, if keys are added and deleted frequently the synchronisation of the cryptographic keys is not an easy task. The synchronisation can either be done with shared cloud storage or by exchanging passwords. Both are not recommended. Especially for users who are not familiar with the field of IT security. Furthermore, not all devices are supporting cryptographic operations with smartcards, *e.g.*, smartphones or tablets.

To solve the issue with the key distribution for several devices, the idea is to move the cryptographic keys into the cloud, in the case of this thesis to CrySIL. In combination with a Secure Hardware Module (HSM), CrySIL is a secure storage for cryptographic keys. Figure 3.1 visualises its concept. It is no longer necessary to store the cryptographic keys on each device. In addition to the secure storage of CrySIL, the cryptographic key is available everywhere and at any time for the user as long as she is connected to the Internet. Furthermore, if the user wants to share encrypted data with other users, this can be done easily by using CrySIL as a key storage. For example, if the user wants to share the encrypted data (which is, *e.g.*, stored on a cloud service like Dropbox[1]), she just has to grant the other users access to the cryptographic key. Because the cryptographic key never leaves CrySIL, the other users are only able to decrypt (the actual decryption of the data is done by CrySIL, see 3.3 *Cryptographic Provider*) the data which are encrypted with this key. As a result, all users are able to read the data without the need for disclosing any cryptographic keys. Figure 3.2 illustrated this use case.

[1] `https://www.dropbox.com`

In summary, moving the cryptographic keys to the cloud brings advantages compared to a local storage.

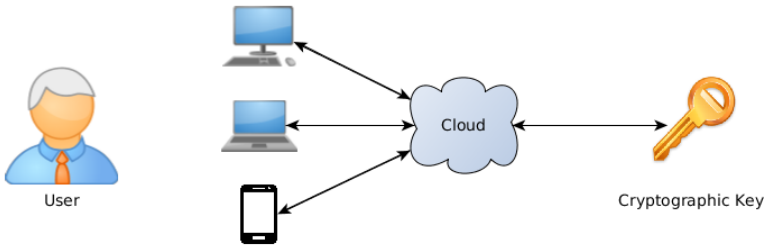


Figure 3.1: The user can use her cryptographic keys on all devices without the need to synchronize them because they are only stored once in the cloud.

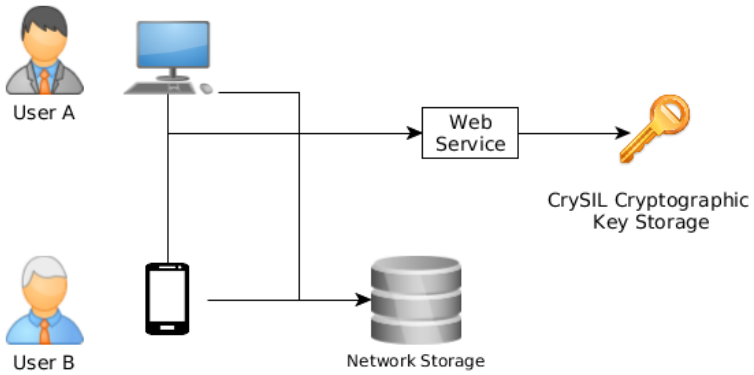


Figure 3.2: Both users are able to see the content without exchanging the cryptographic key.

3.2 Cryptographic Key Management

Creating a cryptographic key is not as easy as it might look like. A lot of different things have to be considered, like enough entropy for the cryptographic key generation, the length of the resulting cryptographic key, whether the cryptographic key should be derived from a user password or another key or some random value, etc. If the cryptographic key generation is not performed correctly or with weak parameters, the cryptographic key can be compromised easily by attackers. The National Institute of Standards and Technology (NIST) [4] gives a recommendation on how to generate cryptographic keys.

By moving the key management into the cloud, in this case to CrySIL, the user does not have to care about a secure key generation any more. This is all done by CrySIL. It takes care of a secure cryptographic key generation and additionally, as described in Section 3.1 *Cryptographic Key Storage*, takes care of storing them securely. CrySIL is able to create different types of cryptographic keys, i.e., symmetric or asymmetric cryptographic keys.

3.3 Cryptographic Provider

As already discussed prior in this section, there are a lot of different implementations for the same cryptographic algorithms and protocols. This is due to the fact that every OS has different concepts and uses different programming languages for the implementation. Further, due to the complexity of algorithms, the likelihood of implementation failures is always given. Furthermore, on some platforms specific protocols and/or algorithms may not exist.

When moving the encryption/decryption process and the corresponding protocols to the cloud, in this case to CrySIL, there is only one implementation which is available for every client with any device independent of the used OS. In other words, the client sends data to the cloud and

receives the encrypted or the decrypted data, respectively. This concept has a great advantage: the cryptographic key, which is generated and stored by CrySIL, never leaves the cloud. The user is not able to obtain the cryptographic key. This is an additional security feature of CrySIL, eliminating possible risks caused by compromised user devices. The report of Kaspersky Lab [32] from 2015 shows that attacks from the Internet are very common.

When several users need access to shared data, *e.g.*, with a secure text collaboration, all of them are able to read the encrypted content of the document because all are using the same cryptographic key. If one member leaves the group, the administrator only has to restrict further access to the cryptographic key by CrySIL instead of reencrypting the whole document with a new cryptographic key and distributing it to all other clients.

Regarding security, the connection to CrySIL is secured via HTTPS which is supported by every common web browser. The actual data are encoded as JavaScript Object Notation (JSON) [7] and sent via HTTPS to the server or to the client, respectively.

3.4 Authentication Service

CrySIL can also be used as an authentication service. In other words, in case of, *e.g.*, a secure text collaboration tools, CrySIL authenticates the user and grants or denies the access to the cryptographic service. It is important to note, that the whole authentication process is done by CrySIL. The client only needs to send the credentials to CrySIL which in turn grants or denies the access.

3.5 Summary

In this chapter, some properties of CrySIL have been explained which are useful for a secure text collaboration tool. The following list summarizes these properties:

- Cryptographic keys can be stored securely by CrySIL in the cloud.
- CrySIL provides a secure key generation with state-of-the-art parameters and algorithms like *AES* [47], *RSA* [43], etc.
- Data encryption and decryption is performed by CrySIL. Due to this fact, only one implementation of each cryptographic algorithm is necessary and thus reduces the likelihood of implementation flaws.
- The user authentication can be performed by CrySIL.
- The service of CrySIL is platform independent. As a result, CrySIL can be used by any operation system and any device which supports a common web browser.

Because of its properties like platform independence, key storage and manager, CrySIL is suitable for developing a secure text collaboration tool which can be operated on different operating systems and different devices like computers, smartphone, tablets, etc.

4 State Of The Art

This chapter gives an overview of state-of-the-art concepts and tools which allow to operate a secure text collaboration tool. In this case, secure means that the content of the file stored on the server is encrypted and the server is not able to read or analyse its content. Only the clients are able to decrypt and edit the content.

First, Section 4.1 *Concepts* gives an overview of concepts which can be used as a basis for a secure text collaboration tool. Next, Section 4.2 *Tools* gives an overview of current solutions.

The disadvantage of all tools presented in Section 4.2 *Tools* is that none of team is really a real-time secure text collaboration tool and some are only for single user purposes.

4.1 Concepts

There are many different concepts how to create a secure text collaboration tool. Some of them are using a server as a central element, other do not need a server at all and only use clients. This section gives an introduction to the most important concepts which are currently used. Also the advantages and the flaws of the concepts are discussed.

4.1.1 Content Cloaking

Content Cloaking is a simple approach to achieve additional security when using a cloud as a central storage for documents with sensible data. Most times, this concept is realised by using third-party-applications, like plug-ins for web browsers (see Sections 4.2.1 *SeGoDocs* and 4.2.2 *SafeGDocs*). As the name suggests, the content of the message is cloaked before sending it to the server. This does not prevent the server from reading the data of the user, but the server is not able to process the data any more because they are encrypted.

The basic functionality of content cloaking is rather easy. Before the data is sent to the server or the cloud, it is encrypted with a state of the art crypto algorithm like the Advanced Encryption Standard (AES) [47]. When receiving the data, it is decrypted before it is passed to the respective application. Figure 4.1 illustrates the workflow of content cloaking.

To the user, the whole process of encrypting and decrypting data is completely transparent. Because of the hardware implementation of the AES standard, modern CPUs can encrypt and decrypt data very fast. Due to this fact, the delay caused by the encryption/decryption process is very small and therefore not noticeable for the user.

Content cloaking is a good concept to protect sensible data from third parties. It encrypts data before they are sent to the server and decrypts the data before processing them. Because it can be implemented as a plugin for web browsers, it is easy to use and the process is completely transparent to the user. Content cloaking can be applied to already existing text collaboration tools like *Google Docs*[1].

The drawback with content cloaking is that it cannot be used simultaneously by many users because it only can encrypt and decrypt data. Since the merging algorithm is the original one from the cloud provider, it does not

[1] `https://docs.google.com`

support merging of encrypted content (as long as the server is not able to decrypt the content). Further, the cryptographic key has to be shared among all users having access to the document.

In summary, content cloaking is a good and easy concept to protect data against unauthorised data access, but it is not suitable for simultaneous editing of the document.

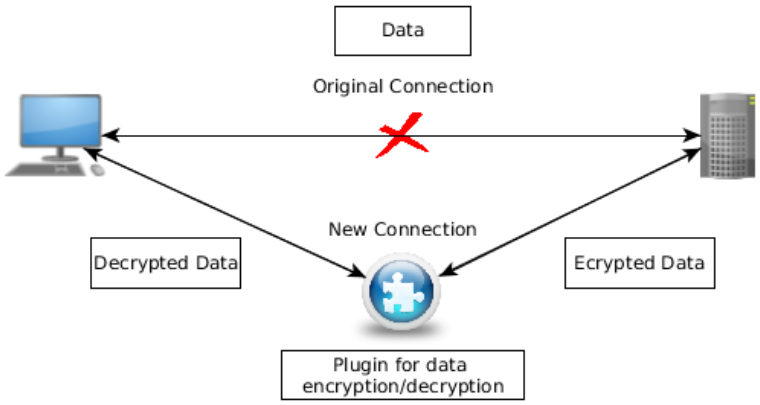


Figure 4.1: Principle workflow of content cloaking. It acts like an additional layer between the server and the client. The data is encrypted before sending it to the server and decrypted before the data is processed by the client.

4.1.2 Peer-to-Peer Networks

In contrast to the most other solutions of secure text collaboration tools, Peer-to-Peer (P2P) networks are working without having a central server to increase the security. This is due to the fact that servers are often high-value targets for attacks[2,3]. For example, tools like *SubEthaEdit* [53] and *CoWord* [41] are using such a P2P network. Figure 4.2 illustrates an example of a typical P2P network.

[2] https://www.nytimes.com/2016/09/23/technology/yahoo-hackers.html?_r=0

[3] http://www.informationisbeautiful.net/visualizations/worlds-biggest-data-breaches-hacks/

Every client has a local copy of the document. All changes on the document are exchanged with all connected clients. Because of a missing server, this is not an easy tasks. To overcome issues with the synchronisation, one peer of all connected ones becomes a so called *super-peer*. This super-peer takes over the role of the server (*e.g.,* detecting merging conflicts, distributing changesets, etc.) and has to be present during the whole session. Logically, this might cause a problem if the super-peer is offline for some reason. To solve this issue, a central storage provider can be introduced, like Dropbox[4], Google Drive[5], SugarSync[6] or Box[7]. In contrast to a server based solution, the server solely stores data instead of detecting merging conflicts, distributing changesets, etc. As a result, the data is stored securely (in the sense of backups) on the server, but has to be encrypted by the peers before sending it to the cloud. Otherwise the storage provider might be able to read and/or analyse the content of the document.

The advantage of a peer-to-peer network is that no server is needed, neither a hosted nor a self-hosted one. This may save a lot of money because there is no need to maintain a server infrastructure.

The drawbacks within this solution are diverse. The key management and distribution of the cryptographic key is not easy. Because of a missing server, a random client takes care of the merging process and may store the document in the cloud. As a result, the round trip time increases because now there are two instances involved in the merging/storing process. This leads to a bad performance. Further, the document has to be reencrypted with a new cryptographic key if a user leaves the text collaboration tool.

To sum up, a P2P network is no real alternative to a server-client concept. The disadvantages are strongly overbalanced within this solution.

[4] `https://www.dropbox.com/en/`
[5] `https://www.google.com/drive/`
[6] `https://www.sugarsync.com/en/`
[7] `http://www.box.com/`

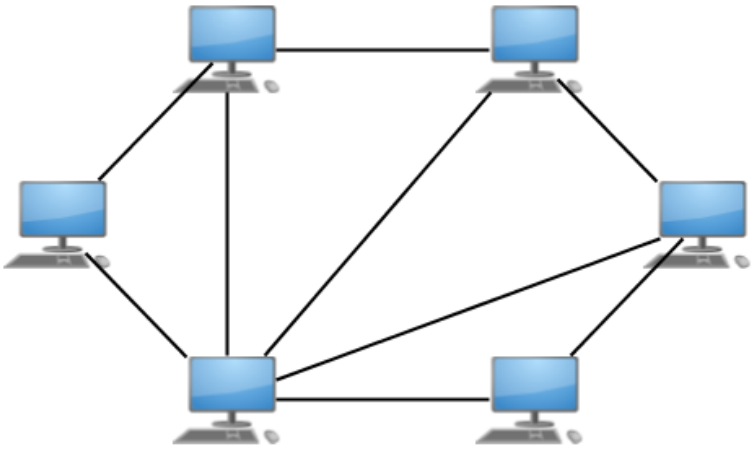


Figure 4.2: Example of a P2P network with 6 connected users. One of the connected peers becomes a super-peer which acts like a server.

4.1.3 Homomorphic Encryption

Homomorphic encryption (HOM) is a completely different approach how a secure text collaboration tool can be realised. The basic idea of HOM is to execute operations on encrypted data without the need to decrypt them first. The operations can directly be applied to the data as they would be in plain.

In general, the term *Homomorphic* describes a property of an encryption scheme. This property allows calculations or operations on encrypted data without the need to decrypt the data first. But not every encryption scheme has homomorphic properties. The drawback with HOM, especially fully homomorphic encryption, is that it is currently rather slow [9]. When using HOM in practice, it often supports only one operation like multiplying or adding values. The HOM scheme of Boneh *et al.* [6] supports two different operations, but only one at a time.

To get a better understanding in how HOM works, two examples are given below. The first one is *ROT13* [8], which is a special case of *Caesar cipher*, and the second example is unpadded RSA [43]. In the following, the encryption will be written as *E(plaintext)* and the decryption as *D(cipher text)*.

The *Caesar cipher* is a substitution cipher and one of the oldest encryption schemes known and named after the roman politician and general Julius Caesar. It is at least 2000 years old. Julius Caesar used this cipher to encrypt his private correspondence. The caesar cipher is the first recorded substitution cipher, but other substitution ciphers are known to have been used earlier [46]. The principle of this cipher is rather simple. Each character of the plaintext is replaced by an other fixed character from the alphabet. This is done by shifting the alphabet. Equation 4.1 shows how the encryption works and equation 4.2 shows how the decryption works. In the formula, p represents the current character of the plaintext (more precisely the number of the character in the alphabet), n the amount of shifts and c the resulting ciphertext. This formula has to applied to each single character of the plaintext to get the ciphertext. Equation 4.3 shows an example for the decryption of the text "Hello World".

$$E_n(p) = (p + n)\ mod 26 \tag{4.1}$$

$$D_n(c) = (c - n)\ mod 26 \tag{4.2}$$

$$E_3(Hello\ World) = Khoor\ Zruog \tag{4.3}$$

Homomorphic properties of ROT13

ROT13 (rotate by 13 places) is a special kind of the caesar cipher. The shift of each character is always 13. Of course it is known that ROT13 is not a secure encryption scheme because it can be easily broken with letter analyses, but for explaining HOM it is useful. First, the two words *HELLO* and *WORLD* will be encrypted with ROT13. The result will be:

$$E_{13}(HELLO) = URYYB \tag{4.4}$$

$$E_{13}(WORLD) = JBEYQ \tag{4.5}$$

Imagine that user Alice writes the encrypted text *URYYB* into a plain document and user Bob concatenates his text *JBEYQ* with Alice's text. The result will be *URYYBJBEYQ*. When this text gets decrypted, the result will be *HELLOWORLD* (see equation 4.6). This means, Bob added some text to an encrypted text without decrypting it first. It follows that ROT13 has homomorphic properties with respect to the concatenation.

$$D_{13}(URYYBJBEYQ) = HELLOWORLD \tag{4.6}$$

Homomorphic properties of RSA

The unpadded RSA encryption scheme has also homomorphic properties with respect to multiplication. Given two arbitrary numbers a and b, it does not mater if someone calculates $E(a) * E(b)$ or $E(a * b)$. Equation 4.7 shows the general encryption formula for RSA and equation 4.8 proves the correctness of the homomorphic property. The public key modulus will be written as m, the exponent as e, the plaintext as p.

$$E(p) = x^e \ mod \ m \tag{4.7}$$

$$E(a) * E(b) = a^e * b^e \ mod \ m = (a * b)^e \ mod \ m = E(a * b) \tag{4.8}$$

As a result, two numbers can be multiplied without the need of decrypting them first. Of course, unpadded RSA cipher is not secure and is limited to the multiplication of two numbers. There are many other HOM schemes like the Paillier cryptosystem [38] or the Goldwasser–Micali cryptosystem [23]. To be applicable in a real world scenario, a HOM should support any operation without any restrictions. Such HOMs might be fast, but not yet practicable for daily business. A HOM cryptoscheme which supports any operation on data is called *fully homomorphic encryption*.

In contrast to HOM, fully homomorphic encryption (fully HOM) supports any kind of operation on encrypted data. Such a fully HOM already exists. Gentry Craig wrote a PhD thesis [21] about this topic and also published a paper [22]. Although fully HOM works in theory, it is impractical with today's computational power. Craig estimated, that a Google search query with fully HOM would increase the amount of computing time by about a trillion. According to Moore's law [45] and given Craigs estimation holds, it will take another 40 years until fully HOM will be as efficient as today's search.

In summary, some cryptographic algorithms have homomorphic properties, *e.g.,* RSA, which can be used with today's computing power. But these properties are often limited to one calculation, for example multiplying. Fully HOM already exists for any kind of operation, but it is not applicable for today's computing power. It is estimated that it will take about another 40 years before fully HOM can be used in our daily life. Fully HOM would be a perfect solution for STeCT, but it is not practicable yet.

4.1.4 Summary

This section presented different concepts of how a secure text collaboration tool can be built. Each of them has its own advantages and disadvantages. The common problem with all these concepts is the cryptographic key management. Each solution assumes that the client encrypts/decrypts the data locally and the cryptographic key has to be shared among all clients. Further, none provides a solution for an easy and secure key exchange between the clients and only few of them support multi-user functionality. To overcome these issues, a new concept has to be developed which benefits from the advantages and overcomes the drawbacks of the presented concepts.

4.2 Tools

Many tools exists which can be used for secure text collaboration. Most of them are implemented as a plug-in for web browsers. This makes them available on many different platforms. The main drawback with current solutions is that they do not support multiple users and real-time collaboration at the same time. This chapter describes some of the current solutions which are working properly for single user purposes.

4.2.1 SeGoDocs

The tool *SeGoDocs* (short term for "Secure Google Docs") is a prototype developed by D'Angelo *et al.* [11]. It is a lightweight, cryptographic, client-side solution browser plug-in for Firefox web browser [8] which supports encryption of Google Docs [24] files, in particular for the word processor, using the content cloaking technique as described in 4.1.1 *Content Cloaking*.

After SeGoDocs successfully performed the login for Google Docs, it starts two observers which are monitoring the HTTP [55] traffic. Because Google Docs uses AJAX [20] and a specific protocol, SeGoDocs is able to identify the requests. When an observer recognises incoming or outgoing data, the message gets intercepted. Depending on whether data is incoming or outgoing, the data gets decrypted or encrypted, respectively. For encryption/decryption, SeGoDocs implements only the AES [47] algorithm written in pure JavaScript [56].

This tool is designed for single-user purposes. The authors do not address multi-user collaboration except that the key distribution between several users is difficult.

[8] `https://www.mozilla.org/en-US/firefox/products/`

In summary, one major advantage of *SeGeDocs* is that it can be used with the existing collaboration tool Google Docs. The drawback is that it can be assumed that the tool is not applicable for multi-user support. Further, the key management is difficult and has to be done by the user itself.

4.2.2 SafeGDocs

SafeGDocs [42] is a Firefox web browser plug-in offering additional security by encrypting the content of a Google Docs document before sending it to the cloud. It is developed by GRADIANT [9] (Galician Research and Development Center for Advanced Telecommunications) and is available for everyone.

Like SeGoDocs, SafeGDocs is also based on the concept of content cloaking (see 4.1.1 *Content Cloaking*), but the workflow is different. SafeGDocs does not send any information to the server when the user edits the document. Instead, all changes are cached locally. When the user clicks on the save button in the plug-in, the whole document gets encrypted (the standard algorithm is AES with a 128 bit key) and is send to the server. SafeGDocs supports some standard encryption algorithms which can be chosen by the user for encryption of the document.

The cryptographic key for the document is encrypted with AES 256 bit derived by a master password and stored as a hidden file called *SafeGDocs.doc* in Google Drive. This means, that the cryptographic key is never stored locally on the clients device.

The developers do not mention multi-user collaboration. Due to the fact that the whole document gets encrypted when the user saves the document, it is very unlikely that more than one user can work simultaneously on the document. Also the key exchange is difficult because the encrypted cryptographic key is stored in the Google Drive of the document's creator.

[9] `https://www.gradiant.org/?lang=en`

Another disadvantage is that not all features of Google Drive are supported yet, like headers, footnotes, page number and page count, comments, etc. Clearly, this limitations are decreasing the usability of the tool.

To sum up, this tool can be used to easily store an encrypted document in the cloud, but it is not practicable for a multi-user collaboration. The user can only save the whole document at once by clicking a button which triggers the encryption of the document.

4.2.3 SPORC

Feldman *et al.* developed a generic framework called *SPORC* [16], which can be used for a lot of collaborative applications using untrusted servers as a storage location for documents. They also developed, among other software, a prototype for a browser-based collaborative text editor.

The only role of the server in this approach is to store the encrypted document and to distribute the changesets to all connected clients. The clients are provided with a local copy of the document which is downloaded at the login. Due to performance issues, the server also stores so called *checkpoints* (see 2.1.4 *Master Document*). The server is not able to read the content because it is encrypted by the clients and the key is never distributed to the server at any time. Therefore, the server can be untrustworthy. To provide further security, to each changeset of the client the global sequence number and a client-specific sequence number is added along with a hash value. This hash value contains the hash value of the previous changeset chain and the current changeset of the client. This ensures the integrity of the data over the lifetime. The clients are able to detect any modification on the data stored at an external server or if an operation went wrong. In addition, each changeset is signed by the client. Therefore, each client can check whether the received changeset is authentic and of integrity or not.

Each client can perform changes at any time without locking the document on the server. If a merge conflict occurs, it is resolved automatically by

the clients by applying a technique called *Operational Transformation* (OT) [14, 51]. SPORC also supports offline changes on the document.

Each document in SPORC has an Access Control List (ACL) which defines the access rights of each user. Three types of users can be distinguished:

- **Reader**: The reader is only allowed to read/decrypt the document. She is not allowed to carry out modifications.
- **Editor**: The editor is allowed to edit the document, *e.g.,* delete or insert new text. She is not allowed to change the ACL of the document.
- **Administrator**: The administrator has full control over the document, i.e., editing the document and the ACL.

Another great feature of SPORC is that clients are able to detect if a server manipulates data. If this happens, the clients can switch automatically to a new server and restore the document. After switching the server, the clients can continue their work. SPORC allows multiple users to collaborate at the same time on the same document via the web browser. The changes of other users arrive in nearly real-time. According to Feldman *et al.,* the user experience of SPORC is similar to Google Docs [24] or EtherPad [19]. For the implementation of SPORC, Feldman *et al.* reused some code from the meanwhile suspended project *Google Wave* [25], which is now called *Apache Wave* [2].

To sum up, SPORC is a comprehensive tool which supports many features like switching the server on the fly, etc. As in the other tools, the main weakness of SPORC is the implementation of the key management. The administrator has to create a key and share it with all participating users which might be critical.

4.2.4 Peer-to-Peer Network

Zhang *et al.* [58] developed a Peer-to-Peer proof-of-concept using the cloud provider Dropbox[10] as a central storage. Due to the use of an external storage provider, it provides storage integrity, but it does not guarantee low-latency.

To avoid merging conflicts, the document on the server is split into several pieces (see Figure 4.3). This is done automatically by the clients. Every piece of the document on the server is stored in an encrypted way in order to prevent the server from reading the data. If a client edits a piece of the document, it is locked for the other ones. As a result, only one client can edit a piece of the document at a time. By doing this, the tools achieves quasi-real-time collaboration. Further, relaxing the real-time requirements reduces the amount of required resources and eases the achievement of document consistency. If the client does not release the lock it is automatically unlocked after a defined time-out.

By locking pieces of the document, the performance of the system gets decreased. If the pieces are chosen too small (*e.g.*, a single line or a few words), a lot of overhead is produced. If the pieces are too large (*e.g.*, a whole paragraph), the probability that two users want to edit the same piece of text increases and users have to wait until the locked piece gets unlocked.

In summary, a Peer-to-Peer network is not the best choice for a text collaboration tool as already mentioned in Section 4.1.2. Further, locking resources will decrease the performance of the tool. Storing the document in form of n pieces is a good idea because, *e.g.*, it can support redo-operations, provides a history of the changes, etc.

[10] `https://www.dropbox.com/en/`

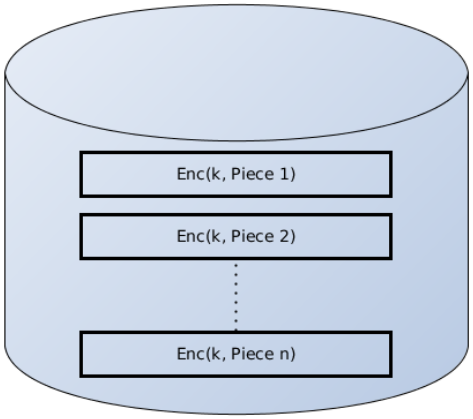


Figure 4.3: The document is divided into n encrypted pieces. Only one client is allowed to edit a piece at a time.

4.2.5 Summary

This section presented some state-of-the-art secure text collaboration tools. Most of them support a multi-user collaboration, *e.g.,* SPORC or the Peer-to-Peer network. The drawback within all this solutions is the cryptographic key management. Each client has to store the cryptographic key locally and has to perform the encryption/decryption itself. Due to different implementations of the same algorithm for different operating systems, this might cause problems discussed in Section 3.3.

5 STeCT: Secure Text Collaboration Tool

In Chapter 4 *State Of The Art*, some state-of-the-art concepts and existing solutions for a secure text collaboration where shown, but all of them have some drawbacks, *e.g.*, a missing cryptographic key management. This chapter presents a solution to securely store data even if the server is considered to be insecure. Further, users do not have to care about a cryptographic key and user management, since CrySIL [39] is used. To achieve the goal of only using metadata for merging, an algorithm was developed which is capable of merging encrypted data without the need to decrypt them first. Further, a web-based prototype was implemented as a proof of concept for this algorithm.

In Section 5.1, the architecture and the concept of STeCT are presented. In Section 5.2, the components of STeCT are described. The communication between all components is shown in Section 5.3. The merge algorithm is explained in Section 5.4 and Section 5.5 gives an overview of the security measures.

5.1 Architectural Overview

Existing solutions for text collaboration tools have limitations regarding the security of data stored at the server. Either the data on the server is not encrypted at all or the server holds a key for decrypting the data. If the server gets compromised, *e.g.*, due to an attack, the attacker is able to read and analyse the data.

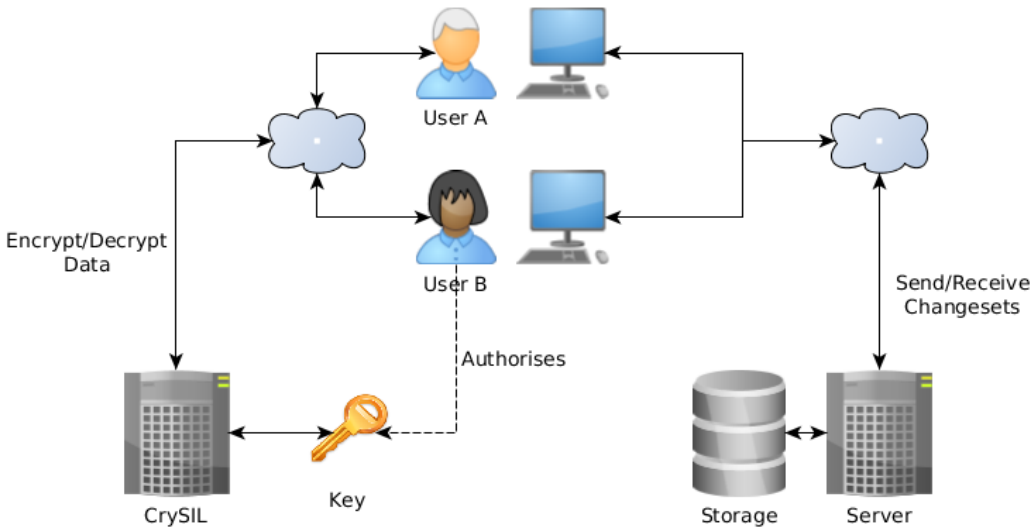


Figure 5.1: Overview of the STeCT architecture and its components.

To overcome this issue, data from the clients have to be encrypted and under no circumstances the server should be able to encrypt the data. To meet these requirements, an algorithm was developed which is able to merge encrypted data. This algorithm represents the core of STeCT and is executed by the server in the cloud. Therefore, the document is securely stored at the server even if the server is considered as honest but curious. The clients are exchanging changesets with the server to modify the document or to receive updates from other clients. CrySIL has the role of the central cryptographic provider. It stores the cryptographic key and encrypts/decrypts the messages from the client. Figure 5.1 visualises the architecture of STeCT.

5.2 STeCT Components

This section describes the role of each component in STeCT. It gives information about the purpose of each component and its relationships to others.

5.2.1 Server

The server is the central component in STeCT. In this thesis, the server is treated like one hosted by a third party because of the reasons discussed in Section 2.1.1 *Hosted vs Self-Hosted*. Because of the drawbacks of a hosted server (privacy protection, no control of hardware, security issues like getting data from other VMs, etc.) the server is considered as honest but curious in sense of data protection. It can be assumed that a third party is able to read the data.

One task of the server is to store the document securely in the sense of data confidentiality. Another task is to give the clients the possibility to access and change the stored document simultaneously at any time. Therefore, the client software is hosted by the server in form of a downloadable web application which is executed by the client's web browser. Only with this software, the clients are able to access and change the document.

If the document exceeds a predefined length (amount of changesets) since the last checkpoint, the server advises a random client to generate a so called *checkpoint* (the communication object *checkpoint* is discussed in Section 5.3.1). A checkpoint is defined as the result of applying a defined amount of changesets on an empty document or on another checkpoint. In other words, the checkpoint contains all changes from the prior changesets and the client only has to apply one checkpoint instead of a list of changesets in order to get the resulting document.

Furthermore, the server takes care of the integrity of the document. It especially ensures that the revision number of each changeset is strictly monotonically increasing without any gaps. A gap within the revision

number would result in an invalid document. The revision number is used to order the sequence of the changesets for building the document. At first the changeset with revision number zero is applied to the local document, followed by the changeset with the revision number one, etc.

5.2.2 Client

The client, or more specifically the client software, is downloaded from the server and executed by the user's web browser. Due the fact that the client software only requires an up-to-date web browser like Chrome[1], Firefox[2], etc. and an active Internet connection to operate, it can be used on a large bandwidth of devices like desktop computers, smartphones or tablets.

In general, the client allows the user to read and edit the document and displays changes from other users. It communicates directly with the server. When the client is started, it sends the current revision number (at startup this is always zero) to the server. The server in turn sends all necessary changesets to the client so that the client is able to build the current document by applying the changesets. When the user types in or deletes some text, a changeset is sent to the server at which the based revision number is the revision number of the last received changeset (this means the current change is based on revision number x) and the new revision number is the based revision number plus one.

To receive updates from other clients, the client polls the server periodically with a so called *Update Client* communication object (see Section 5.3.1 *Update Client*) containing the client's current revision number. If the revision number of the server is higher than the client's one, the client receives the missing changesets and applies them to its local document to achieve the same document version as on the server.

[1] `https://www.google.com/chrome/index.html`
[2] `https://www.mozilla.org/en-US/firefox/new/`

To build *checkpoints* (see Section 5.3.1 *Checkpoint*), the client periodically sends a request to the server. If there is the need to build such a checkpoint, the server sends all changesets since the last checkpoint to the requesting client, otherwise the server does not send an answer to the client. The client in turn applies all the changesets to a checkpoint-changeset and sends it back to the server.

Before the client sends or applies a changeset, the payload of the changeset has to be encrypted or decrypted, respectively. Therefore, the client first sends the changeset to CrySIL for data encryption and afterwards to the server. When receiving a changeset, it has to be decrypted first by CrySIL before the client can apply it to the local document.

5.2.3 CrySIL

In STeCT, CrySIL acts as a central cryptographic provider and performs the login of the clients. To be logged in means that CrySIL provides the cryptographic keys and the cryptographic operations to the client for encrypting or decrypting data. Without being logged in, anybody may be able to download the document, but is not able to decrypt it. Further, CrySIL provides a defined interface for the encryption and decryption operations. It is completely independent from the server and the client. Detailed information on CrySIL can be found in Chapter 3 *CrySIL*.

5.2.4 Master Document

The document in STeCT is, in principle, a list of changesets (see Section 5.3.1 *Changeset*) and checkpoints (see Section 5.3.1 *Checkpoint*). Figure 5.2 visualises how a document looks like. The document is ordered by the so called *revision number*. The first element in the list has the revision number zero, the second one one, the third one two, etc. The revision number has to be strictly increasing without any gaps, otherwise the client is not able to generate the resulting text out of the document. How the changesets are applied to the local document is explained in Section 5.4.5 *Applying Changesets*.

The server is the only component with the permission to manipulate the document. It can insert changesets and checkpoints, but it is not supposed to delete any changeset or checkpoint.

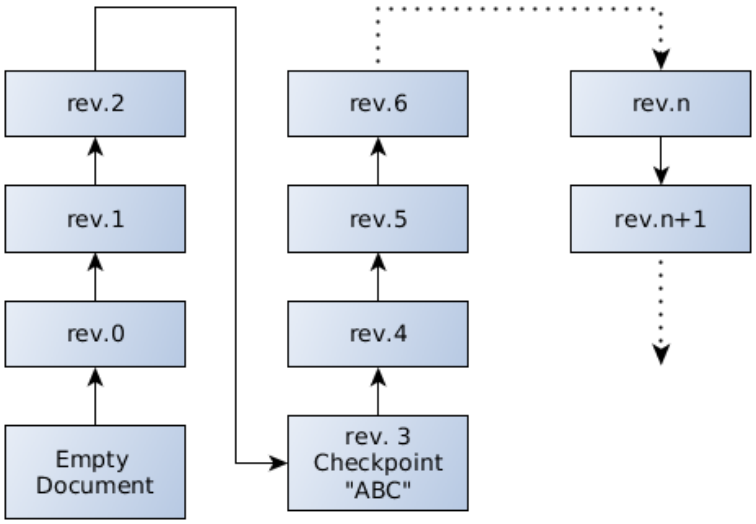


Figure 5.2: A document consists of a list of changesets and checkpoints with a strictly monotonically increasing revision number. Applying all changesets will result in the final document.

5.3 Communication

This section describes the communication in STeCT . Section 5.3.1 describes the communication objects which are used to exchange information between the STeCT components. Section 5.3.2 gives details on the communication between the STeCT components.

5.3.1 Communication Objects

In STeCT there are 4 different communication objects which are used to send and receive data from the client, server and CrySIL. These are changesets, checkpoints and build checkpoint.

Changeset

The changeset is used to exchange data between the server and the client. It is used for inserting or deleting text from the document and for updating the client. The changeset consists of the following fields:

- *int basedRevisionNumber*: Specifies the revision on which this changeset is based on.
- *int revisionNumber*: The new revision number of this changeset according to the client. The revision number might be modified by the merge algorithm.
- *operation*: The operation can have four different values:
  - INSERT: Used by the client. By applying an *insert*-changeset, text is added to the document.
  - DELETE: Used by the client. By applying a *delete*-changeset, text is removed from the document.
  - BUILD_CHECKPOINT: Used by the server to advise the client to build a new checkpoint.
  - INITIAL: Used as the initial changeset which creates an empty document at the client.
- *int position*: The position in the local document where the current operation should be applied. The position has to fulfil the following requirement: $0 <= position <= datalength$
- *String data*: Contains the (encrypted) payload of the changeset. This value is null if the operation is not *INSERT*.

- *int numberOfCharToDelete*: Defines how many characters should be deleted. This field is only used if the operation is *DELETE*.
- *boolean checkPoint*: If the flag is true, this is a *Checkpoint*. A description of the checkpoint can be found in Section 5.3.1 *Checkpoint*.
- *int lockBegin*: Start index of lock. Further details are given in Section 5.4 *Merge Algorithm*.
- *int lockEnd*: End index of lock. Further details can be found in Section 5.4 *Merge Algorithm*.
- *String userName*: The unique username is a random String generated by the client application. It is not used in the current version but may be important for implementing e.g. a document history.
- *int dataLength*: Length of the decrypted payload (field *data*).

Checkpoint

If the *checkpoint*-flag of a changeset is set to true, the this changeset defines a checkpoint. The checkpoint consists of the same subset of fields as a changeset, but not all of the fields are used. If the client receives a checkpoint, then it clears the local document (all changesets are removed from the list) and applies the checkpoint to the local document. In the following, only the used field are described:

- *int revisionNumber*: The revision number indicates the state, meaning the current revision's content, of the document. It is always the based revision number plus one.
- *int basedRevisionNumber*: Specifies the revision on which this changeset is based on.
- *String data*: Contains the (encrypted) payload of the previous changesets since the last checkpoint including the old checkpoint.
- *boolean checkPoint*: In case of a checkpoint, this flag is true.
- *int dataLength*: Length of the decrypted payload (field *data*).

Update Client

This communication object is used by the client for updating the client to the latest reversion. If there are newer changesets, the server will send all of them to the client, otherwise an empty String is returned to the client. The *Update Client* objects consists of the following fields:

- *String userName*: The unique username of the user.
- *int clientRevision*: The current revision of the client. It is used to determine whether there is a newer revision at the server.

Build Checkpoint

The client periodically sends a *Build Checkpoint* to the server. If the client should build a checkpoint, the server sends all necessary changesets to the client so that it is able to build a checkpoint. This object does not need any fields because the server decides if there is the need for building a new checkpoint.

5.3.2 Workflow Communication

In the prior section the communication objects which are used to exchange data between all STeCT components were described. This section describes on which channels the communication objects are sent from A to B and who communicates with whom. Figure 5.3 visualises the communication paths between all STeCT components. The security of the communication is described in Section 5.5 *Security Considerations*.

Client

The client in STeCT is communicating with the server and CrySIL. Before the client sends a changeset to the server, it sends an encryption request containing the plain payload data to CrySIL. After receiving the encrypted data, the changeset is forwarded to the server. When the client receives a changeset from the server, it sends a decryption request to CrySIL before the

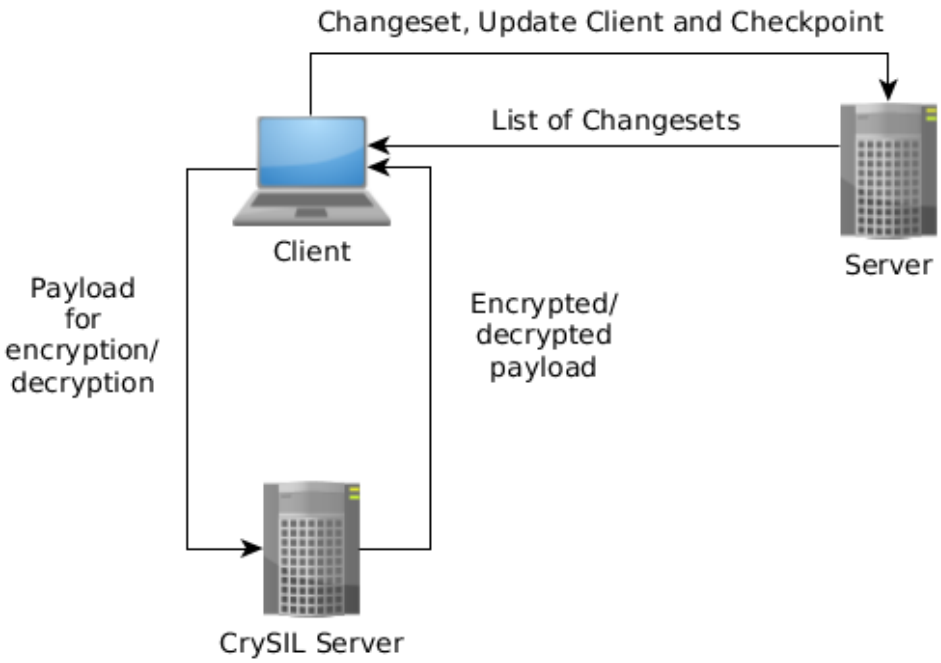


Figure 5.3: The communication paths in STeCT. The client communicates with CrySIL and the server. There is no communication between CrySIL and the server.

changeset can be applied to the local document.

The client periodically sends an *Update Client* and *Checkpoint* object to the server in order to check whether there is a new revision number which is required for updating the client's local document to the latest version or whether building a new checkpoint changeset for the server is necessary.

Server

The server communicates only with the client. It can receive *Changeset*, *Update Client* and *Checkpoint* objects from the server and always responds with a list of changesets, containing zero to n objects.

When receiving a changeset, the result of the merging algorithm determines the changesets being sent to the client. The result of the merging algorithm can have the following values:

- *merged*: The current changeset was successfully merged.
- *merge_split*: The current changeset has to be split into two independent changesets (only possible for the operation *DELETE*). These changesets have to be processed again by the merging algorithm.
- *already_deleted*: The text of the current changeset has already been deleted by another user. The changeset can be ignored.
- *checkpoint_inserted*: A new checkpoint has been added to the list.
- *merge_failed*: An error occurred during the merge process.

If there is no merge needed and the changeset can be applied directly to the master document, the same changeset is sent back to the client. If the result of the merge algorithm is *merged*, *merge_split*, *already_deleted*, *checkpoint_inserted* or *merge_failed*, the client receives all changesets since the latest checkpoint including the latest checkpoint. This ensures that the client is always up-to-date.

The *Update Client* object is used to update the client's local version if there are new changes at the master document (changesets with a higher revision number than the current revision number of the client). If there are changesets in the master document with a higher revision number, the server sends all of them to the client to update the local document to the latest version. If the client is already up-to-date, an empty String is returned.

The *Checkpoint* object is used to determine whether the client should build a checkpoint for the server or not. If it is necessary to build a checkpoint, the server sends all changesets since the latest checkpoint including the latest checkpoint to the client. Otherwise an empty String is returned.

5.4 Merge Algorithm

The merge algorithm is the main component of each text collaboration tool. It is responsible for solving conflicts such as described in Section 2.2 caused by inconsistency problems due to parallel modification of the master document by different users. Development of such a merge algorithm has hardly been done before. A lot of aspects have to be considered such as taking care of the consistency of the master document, avoiding multiple deletion of the same text by different users, etc. It is even more challenging to develop a merge algorithm which is able to merge encrypted data without decrypting them first. This section describes the merge algorithm developed and used by STeCT. This algorithm is intended to run on honest but curious servers, therefore it is able to merge encrypted data solely by using metadata.

First, the preconditions for the merge algorithm are explained in Section 5.4.1. Next, Section 5.4.2 describes the merging process of STeCT. Section 5.4.3 explains the basic idea of the algorithm and the prerequisites. Next, section 5.4.4 shows some implementation details of the algorithm. Section 5.4.5 describes how a changeset is applied by the client.

5.4.1 Preconditions

As already described in Section 5.2.4 *Master Document*, the master document consists of a list of changesets and checkpoints. By applying all changesets from the list to an empty String, the client can build a human-readable presentation of the current master document.

The server is able to read and modify all fields of the changeset except the field *data*, which contains the encrypted payload from the client. In order to perform a merge, the server needs to modify some fields like *revision number*, *position* or *lockBegin*. However, the server is not allowed to change the field *data* at any time. Further, the algorithm is not allowed to edit changesets which are already merged into the master document.

When applying a changeset to the master document, the algorithm is only allowed to add it at the end of the list, checkpoint changesets might also be inserted at any valid position. The revision numbers of all changesets in the list have to be in ascending order whereby no gaps are allowed in order to ensure a consistent master document.

5.4.2 Merging

Merging in STeCT means to append a changeset of a client to the end of the master document. If a conflict occurs, the client's changeset has to be modified in such a way that it won't have an effect on the changesets created by other users having a higher revision number than the client's based revision. After these modifications, the changeset is merged into the master document. For simplification, the process of merging and resolving a conflict is simply called *merging*. For example, if the master document contains the characters "abc" and user A inserts the characters "123" at index zero (result should be "123abc") and user B deletes three characters an index zero (delete "abc") at the same time, the algorithm should prevent user B from deleting the characters "123" from user A if the changeset of user A was processed first. The algorithm should modify the changeset of user B in such a way that it deletes the intended characters "abc" and not "123". At the end of each merge process, the modified changeset is appended to the master document's list.

5.4.3 Basic Idea

When a client sends a changeset to the server, the first step of the algorithm is to determine whether a merge is necessary or whether the changeset can directly be applied to the master document. Therefore, the latest revision number of the master document is compared with the *based revision number* of the client's changeset. If they are identical, the changeset can be applied to the master document without any changes. If the based revision number is lower, then the server needs to merge with the master document. If the based revision number is higher, an error is logged at the server side and the client will be updated to the latest version of the master document.

The basic workflow of how to merge a changeset into the master document is the following: At first, only the changesets of the master document with the same or a higher revision number have to be considered. This is due to the fact that both, the new changeset from the client and the changesets from the master document with a lower revision number, are based on the same data basis. Next, the algorithm has to iterate through all changesets from the master document whereby the changesets are ordered by their revision number in an ascending manner. For each changeset the algorithm checks whether there is a merge conflict within the current changeset (more details are described in Section 5.4.4). Figure 5.4 visualises how a merge conflict between two changesets is detected. If there is a conflict, the algorithm solves the conflict by updating some fields of the client's changeset like *position* etc. Otherwise it does not change the client's changeset. Section 5.4.4 describes in detail, which fields are updated. If the algorithm iterated through the list of changesets and solved all occurring merge conflicts, the modified changeset from the client can be applied to the master document. Algorithm 1 illustrates the work flow of the merge algorithm.

5.4.4 Merging Changesets

When a merge conflict has been detected, it has to be resolved by the algorithm. This is done by updating fields in the client's changeset or by splitting the changeset. After the merging process, the changeset is appended to the master document.

If the operation of a changeset is *INSERT*, it means that the new inserted text begins at index *lockBegin* and ends at the index *lockEnd*. These two fields are always set by the client. It follows that the changeset to be merged is not allowed to modify the text between the index *lockBegin* and *lockEnd*. In contrast, if the operation is *DELETE*, the fields mark the indices where text has been deleted. This is necessary in order to prevent multiple deletion of text, which would result in a wrong behaviour of the algorithm. In general, the fields *lockBegin* and *lockEnd* are used to detect a conflict between two

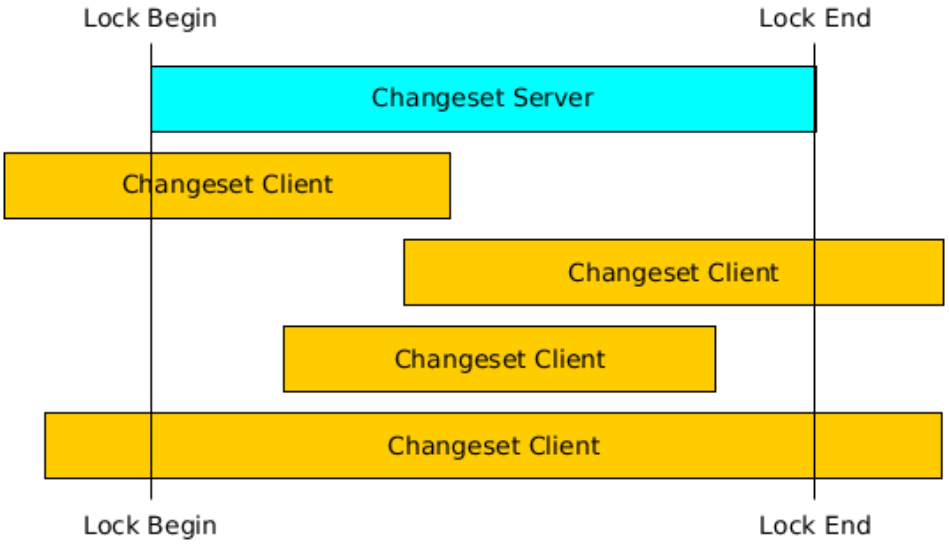


Figure 5.4: This image illustrates how a merge conflict between two changesets can be detected. There are four different types of merge conflicts.

changesets. Further, these two fields provide some kind of *locking* for future conflicts.

When merging a changeset, four different types have to be distinguished. Each of them requires an individual solution. Table 5.1 shows an overview of these four cases. In the following, these four types are explained in detail aiming to generate a better understanding of how the algorithm is able to resolve the merge conflict and how the client's changeset is merged into the master document. In the following, the changeset of the client which has to be merged is denoted as *Client Changeset (CS)*. The current element of the list with higher revision numbers than the one from CS is donated as *Master Changeset (MC)* and the list containing these changesets is just called *list*.

Algorithm 1: This pseudo code shows how a merge conflict is solved in STeCT.

```
Data: List of changesets from the master document with an equal or
      higher revision number than the based revision number of the
      client in ascending order
while not at end of the list do
    /* ignore checkpoint changesets for merging          */
    if isCheckpoint then
        continue;
    end
    if check for merge conflict then
        solveConflict();
    end
end
add Changeset to master document;
```

MC Insert & CC Insert (Case1)

Case1 is the simplest case of solving a merge conflict. At first, the algorithm checks whether the position of CC is before or after MC. In the case CC is after MC, the index of CC has to be adapted because there is an insert before CC.

The next step is to determine whether there is a merge conflict between these two changesets. A merge conflict in this case is defined as overlapping lock-indices. If a conflict is detected, the position of CC has to be adapted to the *lockEnd* index of MC again. In other words, the CC will be inserted after MC. If no conflict is detected, the algorithm has finished processing the current MC. Algorithm 2 shows the implementation as pseudo code.

	MC INSERT	MC DELETE	CC INSERT	CC DELETE
MC INSERT	x	x	Case 1	Case 2
MC DELETE	x	x	Case 3	Case 4
CC INSERT	Case 1	Case 3	x	x
CC DELETE	Case 2	Case 4	x	x

Table 5.1: This table shows the four different possibilities of a merge conflict. For each case, a different solution exists.

Algorithm 2: Pseudo code of solving a merge conflict for Case1

```
Data: MC (current changeset of list), CC (changeset of client)
if CC.position >= MC.position then
    CC.position += MC.dataLength;
end
if merge conflict then
    CC.position = MC.lockEnd;
end
continue with next changeset from list;
```

MC Insert & CC Delete (Case2)

The second case is similar to Case1, but in case of a merge conflict the changeset has to be split into two changesets. The first new changeset deletes the text before MC and the second one the text after MC. This prevents the current CC from deleting text being inserted by MC.

The two new changesets get the revision number of MC and, among other fields, the field *numberOfCharToDelete* is updated. In case of a split, the algorithm terminates and starts again with these two changesets. They are treated as they have been received by a client. Algorithm 3 shows the implementation as pseudo code. If it is only necessary to delete text, *e.g.*, in front of MC, the field *numberOfCharToDelete* of the second changeset is zero.

The algorithm will ignore such changesets because they won't have any effect on the master document.

Algorithm 3: Pseudo code of solving a merge conflict for Case2

```
Data: MC (current changeset of list), CC (changeset of client)
if CC.position >= MC.position then
    CC.position += MC.dataLength;
end
if merge conflict then
    Changeset c1 = CC.clone();
    Changeset c2 = CC.clone();

    c1.setnumberOfCharToDelete(MC.getLockBegin() -
    CC.getPosition());
    update lock-indices of c1;
    c1.setBasedRevisionNumber(MC.getRevisionNumber());

    c2.setPosition(MC.getLockEnd());
    c1.setnumberOfCharToDelete();
    update lock-indices of c2;
    c2.setBasedRevisionNumber(MC.getRevisionNumber());

    terminate algorithm and start it again with c1 and c2;
end
continue with next changeset from list;
```

MC Delete & CC Insert (Case3)

The challenge within Case3 is the correct handling of the deleted text and not to mix up existing text. The first step is to check whether a conflict exists. If there is a conflict, CC is either located before or after MC. In case CC follows MC, the position of CC is reduced by the amount of characters to delete from MC. If CC is located before MC, the position of CC does not have to be updated. Algorithm 4 shows the implementation as pseudo code.

Algorithm 4: Pseudo code of solving a merge conflict for Case3

```
Data: MC (current changeset of list), CC (changeset of client)
if merge conflict then
    CC.setPosition(MC.getPosition());
else
    if CC.getPosition() >= MC.getPosition() then
        if MC.getLockEnd() < CC.getPosition() then
            CC.setPosition(CC.getPosition() -
            MC.getNumberOfCharsToDelete());
        else
            CC.setPosition(MC.getLockBegin());
        end
        update lock-indices of CC;
    end
end
continue with next changeset from list;
```

MC Delete & CC Delete (Case4)

The challenge in Case4 is to detect and prevent multiple deletions of the same text. If a text has already been deleted by MC, CC should be ignored. If only a part of MC is deleted, CC should be split up (like in Case2) in order to delete the remaining text.

At first, the algorithm checks whether the text has already been deleted by MC. If this is the case, the algorithm terminates. The next step is to check for a conflict. If there is a conflict, three cases are possible:

- delete text before prior delete
- delete text after prior delete
- delete text before and after prior delete

If there is no conflict, the algorithm executes the same code as in the else-condition of Algorithm 4. Algorithm 5 shows the respective pseudo code.

Algorithm 5: Pseudo code of solving a merge conflict for Case4

```
Data: MC (current changeset of list), CC (changeset of client)
/* check if CC is already deleted by MC                          */
if MC.getLockBegin() <= CC.getPosition() && Mc.getLockEnd() >=
(CC.getPosition() + CC.getNumberOfCharsToDelete()) then
    terminate algorithm;
end
if merge conflict then
    /* delete text before MC                                     */
    if CC.getLockEnd() < MC.getPosition() then
        update lock-indices of CC;
    /* delete text after MC                                      */
    else if MC.getLockEnd() < CC.getPosition() then
        CC.setPosition(CC.getPosition() -
        MC.getNumberOfCharsToDelete());
    /* delete text before and after MC → split CC                */
    else
        Changeset c1 = CC.clone();
        Changeset c2 = CC.clone();

        c1.setnumberOfCharToDelete(MC.getLockBegin() -
        CC.getPosition());
        update lock-indices of c1;
        c1.setBasedRevisionNumber(MC.getRevisionNumber());

        c2.setPosition(MC.getLockEnd());
        c2.setnumberOfCharToDelete(CC.getNumberOfCharsToDelete()
        - c1.getNumberOfCharsToDelete() -
        MC.getNumberOfCharsToDelete());
        update lock-indices of c2;
        c2.setBasedRevisionNumber(MC.getRevisionNumber());
    end
```

```
else
    if CC.getPosition() >= MC.getPosition() then
        if MC.getLockEnd() < CC.getPosition() then
            CC.setPosition(CC.getPosition() -
            MC.getNumberOfCharsToDelete());
        else
            CC.setPosition(MC.getLockBegin());
        end
        update lock-indices of CC;
    end
end
continue with next changeset from list;
```

5.4.5 Applying Changesets

One task of the client is to display the document to the user. For this purpose, the client has to apply the changesets to the local document, which is in fact a simple String. If the operation of the received changeset is *INSERT*, the client inserts the payload at the given position in the master document. If the operation is *DELETE*, the client deletes *numberOfCharToDelete* characters at the given position from the master document. In the following, a simple example is given for each case.

- *INSERT*
  - Local document contains the text *HelloWorld*
  - Insert the text *New* at position 5
  - Result is *HelloNewWorld*
- *DELETE*
  - Local document contains the text *HelloNewWorld*
  - Delete 3 characters at position 5

– Result is *HelloWorld*

5.5 Security Considerations

In this chapter, the security considerations of STeCT are presented. First, Section 5.5.1 gives details about the security mechanisms, *e.g.*, how the connection to the server is secured. Next, Section 5.5.2 explains how the payload of the changeset is encrypted. Finally, Section 5.5.3 gives an idea of what can be learned from analysing metadata.

5.5.1 Security Mechanisms Of STeCT

STeCT has to be protected against a variety of attack vectors like man-in-the-middle-attack, Spoofing, etc. Therefore, some security mechanisms are used. The following list gives an outline of how STeCT is secured:

- The *payload* of the changeset is encrypted and decrypted by CrySIL. The advantage is that the cryptographic key never leaves CrySIL. Even if the client gets compromised, the attacker is not able to steal the cryptographic key. The master document does not have to be re-encrypted with a new key if a user is not allowed to use STeCT any more.
- The *encryption* of the payload is done with AES and a 512 bit key. The size of the payload is limited to 512 bit. Detailed information on the encryption can be found in Section 5.5.2
- To protect the *communication* between the server and the client and between the client and CrySIL, a standard HTTPS connection is used. HTTPS ensures a secure end-to-end encrypted connection which protects against man-in-the-middle-attacks, data manipulation, etc.

5.5.2 Payload Encryption

Encrypting data is the most critical part in STeCT. The security of the master document relies on a secure encryption of the data. Therefore, some points have to be considered regarding the encryption:

- Encryption algorithm
- Length of the cryptographic key
- Size of the payload
- Entropy of the plaintext

For data encryption and decryption, the symmetric encryption algorithm AES [47] is applied in Cipher Block Chaining (CBC) mode whereby a 256 bit key is used. The size of the payload should be the same as the key length, except for the payload of checkpoint changesets. Because the ciphertext is at least as long as the cryptographic key, the encryption is reasonably secure. This means, the security depends on the cipher. It follows that the payload size for normal changesets is limited to 256 bit, whereby at most 128 bits (4 characters with UTF-8 encoding) are used as payload and at least 128 bits are used as a kind of Initial Vector (IV). This should increase the entropy of each encrypted payload. Further, this should avoid having two times the same cipher text when encrypting the same plaintext.

When using the UTF-8 encoding and each character uses 32 bit, at most 4 characters can be stored in the payload of a single changeset. The entropy of these 4 characters can be calculated as follows (for simplification purposes, only the characters a-z, A-Z, 0-9 and 10 special characters are considered):

$$\begin{aligned} 26\ (a-z) + 26\ (A-Z) + 10\ (0-9) + 10\ (\textit{special characters}) \\ = 72\ \textit{combinations per character} \end{aligned} \tag{5.1}$$

As calculation 5.1 shows, there are 72 possibilities for each character. To calculate the entropy, this number has to be exponentiated by the number of possible characters:

$$72^4 \approx 2.67 * 10^7 \tag{5.2}$$

According to calculation 5.2, there are approximately $2.67 * 10^7$ different combinations for the user data of the payload of each changeset. Additionally, 128 bits IV are added before the payload gets encrypted. The final entropy is calculated using the following calculation:

$$2.65 * 10^7 * 2^{128} \approx 9.14 * 10^{45} \tag{5.3}$$

According to calculation 5.3, there are about $9.14 * 10^{45}$ different possibilities for the payload before it gets encrypted. It should be noted that $9.14 * 10^{45}$ is the lowest possible entropy. If the text is, *e.g.*, reduced to 3 characters, the entropy raises to $72^3 * 2^{160} \approx 3.93 * 10^{55}$.

5.5.3 Learning From Metadata

Because only the payload is encrypted, the server or third parties may be able to read the metadata of communication objects, *e.g.*, the position, kind of operation or based revision number. If an attacker is able to read the metadata, she might extract some information about the master document depending on the kind of communication object. The following sections discuss, which information might be extracted by analysing metadata of different communication objects.

Learning from changeset metadata

For each changeset, only the payload is encrypted. This is due to the fact that the server needs the metadata to resolve conflicts and merge changesets into the master document. Therefore, the changeset contains the most information of metadata. An attacker might be able to extract the following information from the changesets:

- If it is possible to analyse the metadata of all changesets, the resulting length of the master document can be calculated.
- Because of the field *userName* it can be analysed which user currently edits which part of the document. Further, some kind of document history can be build.
- It can be determined at which *position* an *operation* (insert or delete) is executed and how many characters are added or removed, respectively.
- If the *operation* of the changeset is *INITIAL*, then the document is reset to an empty document.

Because at least 128 bit of the 256 bit payload are random, the possibility that the same plaintext results in the same ciphertext is vanishingly low. Due to this fact, a frequency analyses of the characters is impossible.

Learning from checkpoint metadata

The checkpoint uses the same fields as the changeset, but not all of them are used. The fields of the checkpoint are described in Section 5.3.1. Further, only the field *payload* is encrypted. Following information might be extracted by an attacker:

- Each time an attacker analyses the metadata of a checkpoint, she is able to determine the length of the resulting document.
- By comparing the *length* field of different checkpoints, it can be determined whether text was added to or deleted from the document.
- The *revision number* gives information on how many changesets are necessary to trigger a client to build a checkpoint.

Learning from update client metadata

The update client only contains the fields *userName* and *clientRevision*. The only information an attacker can extract is which user has which local version of the master document.

Learning from build checkpoint metadata

The build checkpoint only contains the field *buildCheckpoint* which is set to true if the client should build a checkpoint for the server. It is not possible to extract any useful information from this communication object.

5.6 Summary

This chapter described the general idea of developing STeCT, a secure text collaboration tool. At first, an overview of the architecture was given. In this context, the components and their roles were described in order to generate a better understanding in how STeCT works. Next, the communication itself and the used communication objects were explained, *e.g.*, which components are communicating with each other and which communication objects are exchanged. The basic idea of the merging algorithm with all limitations and preconditions was explained in detail. Further, the four possible types of merge conflicts were explained and solutions for solving each of them was presented. At the end of this chapter, some security aspects were discussed. This also involved the description of the implemented security mechanisms within STeCT. Further, the encryption method of the payload was explained and the entropy of each payload was calculated in order to show that the encryption is reasonably secure. This chapter also includes a discussion about what can be learned from analysing the metadata of the different communication objects.

6 Evaluation

The previous chapter contains the implementation details of STeCT. This chapter evaluates the security and the performance of STeCT. For this purpose, this chapter is split into two parts. The first Section 6.1 *Security Analysis* describes the methodology, assumptions, the model and assets. Further, threat agents and threats are defined and described. Section 6.2 *Performance Test* describes some performance test done along with the implementation.

6.1 Security Analysis

The main reason for performing a security analysis is not to detect implementation flaws. The security analysis is used to proof whether the concept of exchanging and processing data is secure against different attacks or not. Therefore, first the methodology defines how the security analysis is done for STeCT. Next, some assumptions are made which limit the scope of the security analysis. Then, a model is defined which is used to identify assets and threats. Also, threat agents are defined.

6.1.1 Methodology

Before starting the security analysis, the methodology used throughout this section is explained. First, some assumptions have to be made in order to create a basis under which the security analysis can be performed. Second, an abstract model of STeCT is defined which is used to identify assets, threats, etc. Third, the assets of STeCT are identified. Fourth, threat agents are identified and explained. Fifth, the impact of the threats is analysed.

6.1.2 Assumptions

It is assumed that all the encryption algorithms used for this implementation have been evaluated by others. Also CrySIL [39] is assumed to be evaluated and secure. The security of STeCT is not given if any of the following assumptions is violated:

SA-1 CrySIL, as proposed by Reimair *et al.* [39], is assumed to be secure. The cryptographic key for encrypting/decrypting the data never leaves CrySIL.

SA-2 All used cryptographic algorithms are considered to be secure if they are used as proposed by the NIST[1]. The NIST defines standards for different processes like encryption schemes.

SA-3 Because HTTPS is a standard technology which provides confidentiality and integrity, the connection between the server, client and CrySIL is secure if configured properly.

SA-4 Used web browsers are always up to date and are assumed to be secure. This means that they have no implementation flaws and are not compromised by , *e.g.,* faked root certificates or malicious plug-ins.

SA-5 The operating system is up-to-date and has to provide protection mechanisms which prevent third party applications from interacting with STeCT in any way.

SA-6 For mobile operating systems it is assumed that the device is not rooted and sandboxing is enforced between the different applications.

6.1.3 Model

The abstract model shown in Figure 6.1 is used to identify potential security threats. Furthermore it is possible to derive assets, threat agents and threats from this model. The identified security threats are discussed in the next sections.

[1] `https://www.nist.gov/`

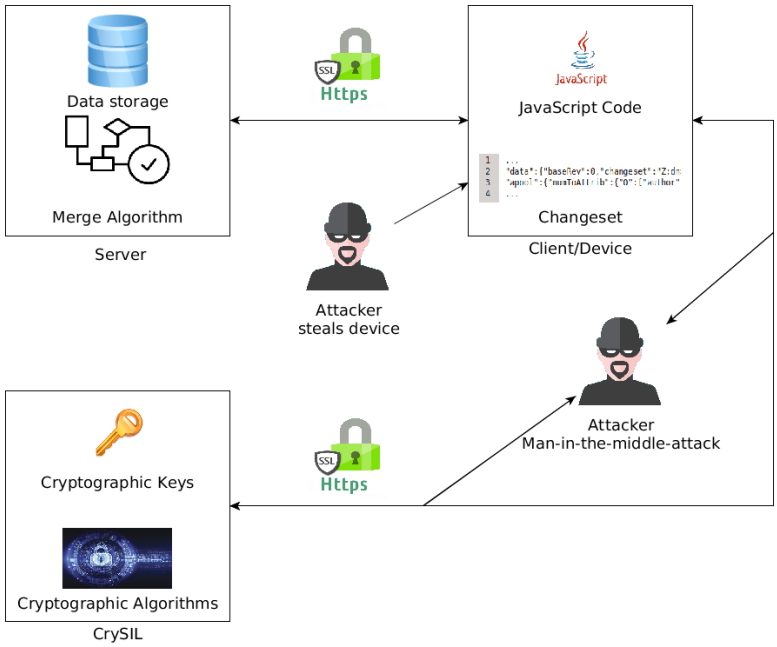


Figure 6.1: The abstract model with all components of the implemented secure text collaboration tool STeCT. This model helps to identify assets, threat agents and threats.

6.1.4 Assets

From the model in Section 6.1.3, the following assets are identified and have to be protected:

A-1 The *data storage* of the server contains all changesets which are necessary to build the resulting document.

A-2 The *cryptographic keys* are used to encrypt and decrypt the payload of the changesets. These keys are securely stored by CrySIL and are never leaving CrySIL.

A-3 The *code* executed by the browser is responsible for creating the changesets, encrypting/decrypting the payload, etc.

A-4 The *changeset* contains all necessary information about the modifications of the master document.

A-5 The *credentials* are necessary for performing the login for CrySIL.

6.1.5 Threat Agent

Threat agents are individuals or groups which can perform a threat. A threat agent may be an attacker, user, etc. Following threat agents can be identified:

TA-1 The *Observer* is able to monitor all data sent between the client and the server.

TA-2 An *Interceptor* has full access to the communication and is able to modify the sent and received data.

TA-3 The *Server* itself might be a threat agent if it gets compromised.

TA-4 A *Natural Person* which is able to, *e.g.*, steal devices or manipulate devices.

TA-5 An *Attacker* may take control over a device or manipulate software on the device.

6.1.6 Threats

From the model in Section 6.1.3, a lot of threats can be identified against which STeCT needs protection. In the following, the identified threats are discussed:

T-1 *Man-in-the-middle attack*

An attacker is able to read and modify the data between two communication partners. As a result, the attacker may modify the changesets sent to the server and manipulate the master document.

Affected assets: A-1, A-4

Involved threat agents: TA-2

Mitigation: To maintain integrity, a digital signature can be added to the changeset. The server will ignore changesets with invalid signatures.

Residual risk: The attacker is still able to read the changesets. However, due to the design of the changeset only metadata can be analysed by an attacker. The payload is still protected by an AES encryption.

T-2 *Replay attack*

If an attacker is able to record a changeset, she might send it to the server several times in order to modify the master document.

Affected assets: A-1, A-4

Involved threat agents: TA-1

Mitigation: The client can retrieve a unique random number from the server and add it to the changeset in a secure way. This random number is only valid for a short period of time. The server will ignore changesets if the random number is used multiple times or if the random number has not been assigned yet.

Residual risk: None.

T-3 *Device theft*

The device of the user (smartphone, laptop, etc.) might get stolen by an attacker. If the credentials for CrySIL are stored within the web browser, the attacker has access to the document.

Affected assets: A-1, A-5

Involved threat agents: TA-4

Mitigation: Use a two-factor authentication, do not store credentials in an insecure way.

Residual risk: The stolen device is used for the two-factor authentication.

T-4 *Compromised server*

There are many ways to compromise a server, *e.g.,* due to security flaws or cross-VM attacks. As a result, an attacker may have access to the data storage.

Affected assets: A-1

Involved threat agents: TA-3, TA-5

Mitigation: The server is able to decrypt the changeset, but it is not able to decrypt the payload. as long as CrySIL does not collide.

Residual risk: None.

T-5 *Compromised cryptographic provider*

If the cryptographic provider CrySIL gets compromised, an attacker might be able to steal the cryptographic keys.

Affected assets: A-2, A-5

Involved threat agents: TA-5

Mitigation: CrySIL can securely store the cryptographic keys in a dedicated HSM.

Residual risk: None.

T-6 *Cross-Side Scripting (XSS)*

An attacker might be able to inject some malicious code into the web client's software.

Affected assets: A-1, A-3, A-4, A-5

Involved threat agents: TA-2, TA-3, TA-5

Mitigation: Signing the code of the software.

Residual risk: None.

T-7 *Spoofing*

An attacker might get access to the changesets and modify the parameters or the payload. Spoofing can be achieved by, *e.g.,* a man-in-the-middle-attack or by manipulating the network traffic inside the local network.

Affected assets: A-1, A-4

Involved threat agents: TA-2, TA-5

Mitigation: The changeset can be signed by the client. If the changeset has been manipulated, the server would just reject it.

Residual risk: None.

6.1.7 Summary

The chapter 6.1 *Security Analysis* described the assumptions on which this security analysis is based on. Further, the model described in Chapter 6.1.3 was used to define assets and threat agents which are used to describe possible threats on STeCT. Table 6.1 shows a summary of the relationship between threats, assets and involed threat agents.

As shown in Table 6.1, Asset 1 (data storage) is the most critical asset to be protected against attacks. Also Asset 4 (changeset) and Asset 5 (credentials) are often at risk of being attacked. By encrypting the payload, Asset 1 and Asset 4 are protected against threats. Asset 5 remains a problem of the user requiring her to securely store her credentials. Table 6.1 also shows

that TA-2 (interceptor) and TA-5 (attacker) are the most common threats of a secure text collaboration tool.

Although a lot of security mechanisms are implemented in STeCT, still some residual risks exist. However, they are out of scope of STeCT. The user credentials have to be protected by the user herself. If the users looses her credentials, an attacker might be able to perform a login and gain access to the document. Another problem might be a stolen device if the user has the auto-login function enabled in the web browser and the attacker is able to get access to the stolen device. Further, an attacker might be able to read and analyse the metadata of communication objects and extract some information about the document. This is discussed in Section 5.5.3 *Learning From Metadata*.

Threats	Assets					Threat Agents				
	A-1	A-2	A-3	A-4	A-5	TA-1	TA-2	TA-3	TA-4	TA-5
T-1	☒	☐	☐	☒	☐	☐	☒	☐	☐	☐
T-2	☒	☐	☐	☒	☐	☒	☐	☐	☐	☐
T-3	☒	☐	☐	☐	☒	☐	☐	☐	☒	☐
T-4	☒	☐	☐	☐	☐	☐	☐	☒	☐	☒
T-5	☐	☒	☐	☐	☒	☐	☐	☐	☐	☒
T-6	☒	☐	☒	☒	☒	☐	☒	☒	☐	☒
T-7	☒	☐	☐	☒	☐	☐	☒	☐	☐	☒

Table 6.1: Relationship between threats, assets and threat agents.

6.2 Performance Test

Because performance is an important factor influencing the acceptance of a software, this section describes the performance of STeCT. At first, the test setup is described which contains the infrastructure as well as the used software. Next, the execution of the test and the results are presented. Finally the performance test is summarized.

6.2.1 Test Setup

The test setup consists of two computers connected via Wireless Local Area Network (WLAN). One computer is the server on which STeCT is running and the second one is the client.

The server hardware is a Lenovo W530 Workstation with an Intel i7-3630QM processor and 8GB RAM. The used operating system is Kubuntu 14.04 LTS (Trusty Tahr) with latest updates. The used Java version is 1.7.0_80. For the development of STeCT, the Integrated Development Environment (IDE) Eclipse was used with the version *Kepler Service Release 2*.

The client hardware consists of a desktop computer with an AMD Phenom™|| X6 1055T processor and 8GB RAM. The operating system is Windows 10 Version 1703 with latest updates. The used Java version is 1.7.0_79. The used web browser is Google Chrome with the version 62.0.3202.75.

The used router to establish the network connection between the server and the client is a FRITZ!Box 6840 LTE with the software version 06.84. The used WLAN standard is 802.11n+g+b with 2.4GHz.

6.2.2 Test Execution And Results

Especially for text collaboration tools the loading time of the document and single changesets is crucial. To determine the loading times of STeCT, the following described test was performed.

At first, the loading time of the whole document was determined whereby the document consists of a varying amount of changesets without the usage of checkpoints. For the first measurement, the document consists of 1.000 changesets, for the second measurement the document consists of 2.000 changesets and so on up to 10.000 changesets. The second test was performed like the first one with the modification that the client created a checkpoint after every 300 changesets.

The time measurement was performed using timestamp differences. The first timestamp was taken after the client's startup when the client made the first update request to the server. The second timestamp was taken after the call had finished. This also includes the processing of all received changesets. Figure 6.2 visualises the results of the measurement and Table 6.2 holds the exact values of the measurement.
As can be seen in Figure 6.2, without the usage of checkpoints the loading time seems to increase exponentially with increasing amount of changesets. In the test environment it took 460 ms to load 1000 changests. This loading time might barely be accepted by the users. With 2000 changesets, the loading time increases to nearly one second which might not be accepted by the users.

With the usage of checkpoints, the loading time of the document stays below 200 ms independent of the amount of changesets. This is due to the fact that the client has to process at most 300 changesets which leads to a better performance. Reducing the threshold to, *e.g.*, 100 changesets for creating checkpoints might not be affective because a loading time of at most 200 ms is an acceptable value and the overhead bears no relation to the improved performance.

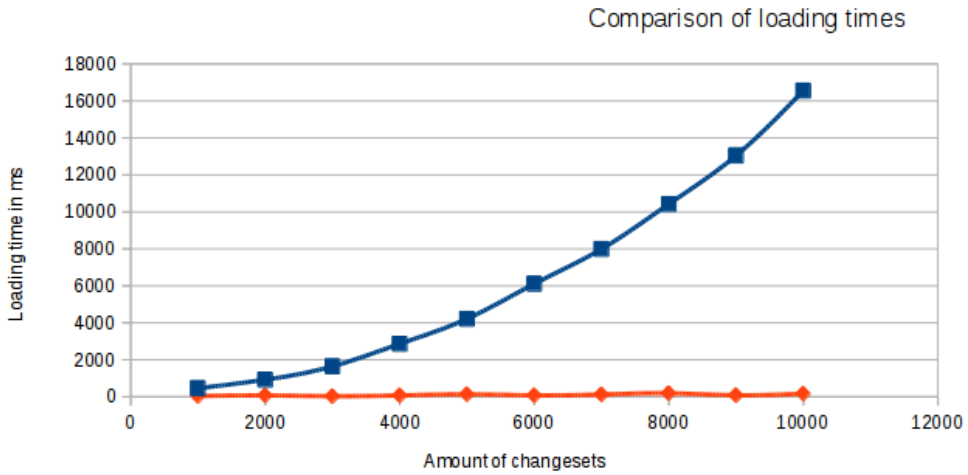


Figure 6.2: Comparison between the loading times of different setup of STeCT. The *blue* line shows the loading time without encryption and without the usage of checkpoints. The *red* line shows the loading time without encryption and checkpoint creation after every 300 changesets.

Independent of the usage of checkpoints, an update request to the server needs about 15 to 20 ms if the client already has the latest version of the document and no changeset has to be applied by the client.

6.2.3 Summary

Figure 6.2 clearly shows that there is a need for checkpoints in STeCT. Without checkpoints, the loading time will increase with every new changeset. With the use of checkpoints, the loading time can be maximized to a distinct value dependent on the frequency of the checkpoints. Too many checkpoints will increase the overhead and too less checkpoints lead to a bad performance. As the performance test in Section 6.2.2 showed, creating a checkpoint after every 300 changesets is a good trade-off between performance and overhead.

Amount of changesets	**Loading time without checkpoints [ms]**	**Loading time with each 300 changesets a checkpoint [ms]**
1000	460	42
2000	925	73
3000	1645	31
4000	2864	73
5000	4214	133
6000	6110	78
7000	7991	124
8000	10422	194
9000	13044	83
10000	16554	170

Table 6.2: Comparing the loading times of different setups of STeCT.

7 Conclusion And Outlook

This work presented STeCT which can be used as a secure alternative to, *e.g.*, Google Docs[1] because the payload of each changeset is encrypted and the server is not able to decrypt the data. Due to this fact, STeCT allows users to share sensible documents on untrusted servers which are considered as honest but curious. Because the server hosts the document and is responsible for merging and distributing changesets, a new algorithm was developed which allows the server to merge changesets only with usage of its metadata. Because STeCT closely operates with *CrySIL*, the users do not have to care about the cryptographic key management compared to other existing solutions. Each user has her own credentials and CrySIL authorises the user to use certain cryptographic functionality. As a proof of concept of STeCT, a prototype of the client was implemented and tested successfully.

Because the client of STeCT is a prototype, it consists of a simple text field which does not support any code formatting like bold, italic, text colours, etc. However, these are restrictions of the client, not of the merge algorithm. Further, this prototype does not support a login functionality for CrySIL yet. For simplification reasons, the update strategy of the client's local document is polling. Together with a login procedure for CrySIL, polling can be exchanged with an asynchronous session management.

The changeset is used to add, remove or modify the master document at the server. Because the payload contains sensitive data, it is encrypted by CrySIL before it is sent to the client and decrypted by CrySIL before the client processes the data. Altough the payload is encrypted, an attacker might

[1] `https://docs.google.com`

be able to read and modify the metadata of the changeset. As a counter measure, the client might sign the changeset. This would guarantee integrity of the data over the entire life-cycle. The server rejects changesets with an invalid signature and requests the client to resend the changeset. Signing is also applicable for the other communication objects.

Another possibility is to additionally encrypt the changeset with another cryptographic key. The drawback with this solutions is that the server also needs this cryptographic key to be able to read the metadata. If the server gets compromised, also an attacker might be able to read the decrypted metadata.

To mitigate replay attacks for all communication objects, the client can request a random or unique number from the server which gets signed with the communication attack. If the same number is used more than once, the communication object is rejected by the server.

The prototype implementation of the client is downloaded from the server and executed locally by the client's web browser. An attacker might modify the code in such a way that the encryption/decryption calls to CrySIL are prevented and the payload is not encrypted. To mitigate code modifications due to a compromised server or by code injections, the code of the client might also be signed. This ensures that the original code is executed at all times.

Bibliography

[1] Lilian Adkinson-Orellana et al. "Privacy for google docs: Implementing a transparent encryption layer." In: *Proceedings of the 2nd International Conference on Cloud Computing* (2010), pp. 20–21 (cit. on p. 4).

[2] Apache Wave. *The Apache Software Foundation*. Available online at `http://incubator.apache.org/wave/`. 2014 (cit. on p. 40).

[3] Etherpad Foundation AppJet Inc. "Etherpad and EasySync Technical Manual." In: Mar. 2011. URL: `https://github.com/ether/etherpad-lite/raw/master/doc/easysync/easysync-full-description.pdf` (cit. on p. 13).

[4] Elaine Barker and Allen Roginsky. "Recommendation for cryptographic key generation." In: *NIST Special Publication* 800 (2012), p. 133 (cit. on p. 26).

[5] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, 1987. ISBN: 0-201-10715-5. URL: `http://research.microsoft.com/en-us/people/philbe/ccontrol.aspx` (cit. on p. 19).

[6] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. "Evaluating 2-DNF Formulas on Ciphertexts." In: *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*. 2005, pp. 325–341. DOI: `10.1007/978-3-540-30576-7_18`. URL: `http://dx.doi.org/10.1007/978-3-540-30576-7_18` (cit. on p. 33).

[7] Tim Bray. "The javascript object notation (json) data interchange format." In: (2014). URL: `https://tools.ietf.org/html/rfc7159.html` (cit. on p. 27).

[8] Chris Christensen. "Review of *Modern Cryptanalysis: Techniques for Advanced Code Breaking* by Christopher Swenson." In: *Cryptologia* 33.1 (2009), p. 5. DOI: 10.1080/01611190802293397. URL: http://dx.doi.org/10.1080/01611190802293397 (cit. on p. 34).

[9] Michael Cooney. "IBM touts encryption innovation - New technology performs calculations on encrypted data without decrypting it." In: (June 2009). URL: http://www.computerworld.com/article/2526031/security0/ibm-touts-encryption-innovation.html (cit. on p. 33).

[10] Ian Curry. "An Introduction to Cryptography and Digital Signatures." In: (Mar. 2001). URL: https://netrust.net/docs/whitepapers/cryptointro.pdf (cit. on p. 12).

[11] Gabriele D'Angelo, Fabio Vitali, and Stefano Zacchiroli. "Content cloaking: preserving privacy with Google Docs and other web applications." In: *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC), Sierre, Switzerland, March 22-26, 2010*. 2010, pp. 826–830. DOI: 10.1145/1774088.1774259. URL: http://doi.acm.org/10.1145/1774088.1774259 (cit. on p. 37).

[12] Dimensional Research. *Collaboration Trends And Technology - A Survey Of Knowledge Workers*. Available online at https://www.alfresco.com/sites/www.alfresco.com/files/dimesional-research-collab-survey-findings-report-082415.pdf. Aug. 2015 (cit. on p. 1).

[13] Manuel Egele et al. "An Empirical Study of Cryptographic Misuse in Android Applications." In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*. CCS '13. Berlin, Germany: ACM, 2013, pp. 73–84. ISBN: 978-1-4503-2477-9. DOI: 10.1145/2508859.2516693. URL: http://doi.acm.org/10.1145/2508859.2516693 (cit. on p. 23).

[14] Clarence A Ellis and Simon J Gibbs. "Concurrency control in groupware systems." In: *Acm Sigmod Record*. Vol. 18. 2. ACM. 1989, pp. 399–407 (cit. on pp. 14, 40).

[15] Ziff Davis Enterprise. "The Pros and Cons of Self-Managed vs. Hosted Solutions." In: 2010. URL: http://i.dell.com/sites/doccontent/public/solutions/k12/en/Documents/pros-cons-self-managed-vs-hosted.pdf (cit. on p. 7).

[16] Ariel J. Feldman et al. "SPORC: Group Collaboration using Untrusted Cloud Resources." In: *9th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2010, October 4-6, 2010, Vancouver, BC, Canada, Proceedings*. 2010, pp. 337–350. URL: http://www.usenix.org/events/osdi10/tech/full_papers/Feldman.pdf (cit. on p. 39).

[17] The Apache Software Foundation. *Open Office*. Available online at https://www.openoffice.org/. June 2016 (cit. on p. 1).

[18] The Document Foundation. *Libre Office*. Available online at https://www.libreoffice.org/. June 2016 (cit. on p. 1).

[19] The Etherpad Foundation. *Etherpad*. Available online at http://etherpad.org/. June 2016 (cit. on pp. 12, 40).

[20] Jesse James Garrett et al. "Ajax: A new approach to web applications." In: (2005) (cit. on p. 37).

[21] Craig Gentry. "A fully homomorphic encryption scheme." crypto.stanford.edu/craig. PhD thesis. Stanford University, 2009 (cit. on p. 36).

[22] Craig Gentry. "Fully Homomorphic Encryption Using Ideal Lattices." In: *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*. STOC '09. Bethesda, MD, USA: ACM, 2009, pp. 169–178. ISBN: 978-1-60558-506-2. DOI: 10.1145/1536414.1536440. URL: http://doi.acm.org/10.1145/1536414.1536440 (cit. on p. 36).

[23] Shafi Goldwasser and Silvio Micali. "Probabilistic encryption and how to play mental poker keeping secret all partial information." In: *Proceedings of the fourteenth annual ACM symposium on Theory of computing*. ACM. 1982, pp. 365–377 (cit. on p. 35).

[24] Google. *Google Docs*. Available online at https://docs.google.com. June 2016 (cit. on pp. 1, 37, 40).

[25] Google. *Google Wave*. Available online at https://wave.google.com/wave/. 2012 (cit. on p. 40).

[26] Saul Greenberg and David Marwood. "Real Time Groupware As a Distributed System: Concurrency Control and Its Effect on the Interface." In: *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*. CSCW '94. Chapel Hill, North Carolina, USA: ACM, 1994, pp. 207–217. ISBN: 0-89791-689-1. DOI: 10.1145/192844.193011. URL: http://doi.acm.org/10.1145/192844.193011 (cit. on p. 19).

[27] Irene Greif, Robert Seliger, and William E. Weihl. "Atomic Data Abstractions in a Distributed Collaborative Editing System." In: *Proceedings of the 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*. POPL '86. St. Petersburg Beach, Florida: ACM, 1986, pp. 160–172. DOI: 10.1145/512644.512659. URL: http://doi.acm.org/10.1145/512644.512659 (cit. on p. 19).

[28] A. Karsenty and M. Beaudouin-Lafon. "An algorithm for distributed groupware applications." In: *[1993] Proceedings. The 13th International Conference on Distributed Computing Systems*. May 1993, pp. 195–202. DOI: 10.1109/ICDCS.1993.287708 (cit. on p. 19).

[29] Michael J. Knister and Atul Prakash. "DistEdit: A Distributed Toolkit for Supporting Multiple Group Editors." In: *Proceedings of the 1990 ACM Conference on Computer-supported Cooperative Work*. CSCW '90. Los Angeles, California, USA: ACM, 1990, pp. 343–355. ISBN: 0-89791-402-3. DOI: 10.1145/99332.99366. URL: http://doi.acm.org/10.1145/99332.99366 (cit. on p. 19).

[30] Michael J. Knister and Atul Prakash. "Issues in the Design of a Toolkit for Supporting Multiple Group Editors." In: *Computing Systems* 6.2 (1993), pp. 135–166. URL: http://www.usenix.org/publications/compsystems/1993/spr_knister.pdf (cit. on p. 19).

[31] Jonathan Koomey et al. "A Simple Model for Determining True Total Cost of Ownership for Data Centers." In: 2007. URL: http://www.premiersolutionsco.com/wp-content/uploads/2010/12/Total-Cost-Of-Ownership-For-Data-Centers.pdf (cit. on p. 8).

[32] Kaspersky Lab. "Kaspersky Security Bulletin 2015 - OVERALL STATISTICS FOR 2015." In: (2015). URL: https://securelist.com/files/2015/12/KSB_2015_Statistics_FINAL_EN.pdf (cit. on p. 27).

[33] Clémentine Maurice et al. "Hello from the other side: SSH over robust cache covert channels in the cloud." In: *NDSS, San Diego, CA, US* (2017) (cit. on p. 8).

[34] Tom Melamed and Ben J. C. Clayton. "A Comparative Evaluation of HTML5 as a Pervasive Media Platform." In: *Mobile Computing, Applications, and Services - First International ICST Conference, MobiCASE 2009, San Diego, CA, USA, October 26-29, 2009, Revised Selected Papers*. 2009, pp. 307–325. DOI: 10.1007/978-3-642-12607-9_20. URL: http://dx.doi.org/10.1007/978-3-642-12607-9_20 (cit. on p. 11).

[35] Microsoft. *Microsoft Office*. Available online at https://products.office.com/en-US/word. June 2016 (cit. on p. 1).

[36] Microsoft. *Word Online*. Available online at https://products.office.com/en-US/office-online/. June 2016 (cit. on p. 1).

[37] U.S. Government Publishing Office. *Patriot Act*. Available online at https://www.gpo.gov/fdsys/pkg/PLAW-107publ56/pdf/PLAW-107publ56.pdf. Oct. 2001 (cit. on pp. 3, 7).

[38] Pascal Paillier. "Public-key cryptosystems based on composite degree residuosity classes." In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1999, pp. 223–238 (cit. on p. 35).

[39] Florian Reimair, Peter Teufl, and Thomas Zefferer. "WebCrySIL - Web Cryptographic Service Interoperability Layer." In: *WEBIST 2015 - Proceedings of the 11th International Conference on Web Information Systems and Technologies, Lisbon, Portugal, 20-22 May, 2015*. 2015, pp. 35–44. DOI: 10.5220/0005488400350044. URL: http://dx.doi.org/10.5220/0005488400350044 (cit. on pp. 4, 23, 43, 70).

[40] Florian Reimair et al. "MoCrySIL - Carry Your Cryptographic Keys in Your Pocket." In: *SECRYPT 2015 - Proceedings of the 12th International Conference on Security and Cryptography, Colmar, Alsace, France, 20-22 July, 2015*. 2015, pp. 285–292. DOI: 10.5220/0005547902850292. URL: http://dx.doi.org/10.5220/0005547902850292 (cit. on p. 23).

[41] Advanced Collaborative Technology Research. "Codoxware: Connecting people and documents." In: (). URL: http://www.codoxware.com/ (cit. on p. 31).

[42] GRADIANT (Galician Research and Development Center for Advanced Telecommunications). "SafeGDocs." In: (2013). URL: http://safegdocs.com/ (cit. on p. 38).

[43] Ronald L Rivest, Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems." In: *Communications of the ACM* 21.2 (1978), pp. 120–126 (cit. on pp. 28, 34).

[44] Andrzej Romanowski, Pawel Wozniak, and Juliusz Gonera. "Simplified Centralized Operational Transformation Algorithm for Concurrent Collaborative Systems." In: *IJCSA* 9.3 (2012), pp. 47–60. URL: http://www.tmrfindia.org/ijcsa/v9i34.pdf (cit. on p. 14).

[45] Robert R Schaller. "Moore's law: past, present and future." In: *IEEE spectrum* 34.6 (1997), pp. 52–59 (cit. on p. 36).

[46] Simon Singh. *The Code Book: The Evolution of Secrecy from Mary, Queen of Scots, to Quantum Cryptography*. 1st. New York, NY, USA: Doubleday, 1999. ISBN: 0385495315 (cit. on p. 34).

[47] National Institute of Standards and Technology (NIST). "Advanced Encryption Standard (AES)." In: (Nov. 2001). URL: http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf (cit. on pp. 28, 30, 37, 65).

[48] M. Stefik et al. "WYSIWIS Revised: Early Experiences with Multiuser Interfaces." In: *ACM Trans. Inf. Syst.* 5.2 (Apr. 1987), pp. 147–167. ISSN: 1046-8188. DOI: 10.1145/27636.28056. URL: http://doi.acm.org/10.1145/27636.28056 (cit. on p. 19).

[49] C. Sun, Y. Zhang, and Y. Yang. "Distributed Synchronization of Group Operations in Cooperative Editing Environments." In: *Concurrent Engineering* 4.3 (1996), pp. 293–302. DOI: 10.1177/1063293X9600400308. URL: http://dx.doi.org/10.1177/1063293X9600400308 (cit. on p. 19).

[50] Chengzheng Sun et al. "A consistency model and supporting schemes for real-time cooperative editing systems." In: *Australian Computer Science Communications* 18 (1996), pp. 582–591 (cit. on p. 16).

[51] Chengzheng Sun et al. "Achieving Convergence, Causality Preservation, and Intention Preservation in Real-time Cooperative Editing Systems." In: *ACM Trans. Comput.-Hum. Interact.* 5.1 (Mar. 1998), pp. 63–108. ISSN: 1073-0516. DOI: 10.1145/274444.274447. URL: http://doi.acm.org/10.1145/274444.274447 (cit. on p. 40).

[52] Omer Tene. "What Google Knows: Privacy And Internet Search Engines." In: 2008. URL: http://epubs.utah.edu/index.php/ulr/article/viewFile/136/118 (cit. on p. 7).

[53] TheCodingMonkeys. "SubEthaEdit: Collaborative text editing." In: (). URL: http://www.codingmonkeys.de/subethaedit/ (cit. on p. 31).

[54] W3C. *HTML and CSS*. Available online at https://www.w3.org/standards/webdesign/htmlcss. 2016 (cit. on p. 11).

[55] W3C. *HTTP - Hypertext Transfer Protocol*. Available online at https://www.w3.org/Protocols/. 2016 (cit. on p. 37).

[56] W3C. *JavaScript Web APIs*. Available online at https://www.w3.org/standards/webdesign/script. 2016 (cit. on pp. 11, 37).

[57] Spyros Xanthopoulos and Stelios Xinogalos. "A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications." In: *Proceedings of the 6th Balkan Conference in Informatics*. BCI '13. Thessaloniki, Greece: ACM, 2013, pp. 213–220. ISBN: 978-1-4503-1851-8. DOI: 10.1145/2490257.2490292. URL: http://doi.acm.org/10.1145/2490257.2490292 (cit. on p. 11).

[58] Chunwang Zhang et al. "Secure Quasi-Realtime Collaborative Editing over Low-Cost Storage Services." In: *Secure Data Management - 9th VLDB Workshop, SDM 2012, Istanbul, Turkey, August 27, 2012. Proceedings*. 2012, pp. 111–129. DOI: 10.1007/978-3-642-32873-2_8. URL: http://dx.doi.org/10.1007/978-3-642-32873-2_8 (cit. on p. 41).