České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Hons Dominik

Studijní program: Otevřená informatika
Obor: Softwarové inženýrství

Název tématu: C# knihovna pro správu časových razítek

Pokyny pro vypracování:

Výsledkem diplomové práce bude vytvoření knihovny pro jednoduché používání časových razítek. Seznamte se s principy časových razítek a jejich používání. Analyzujte existující nástroje určené pro práci s časovými razítky. Navrhněte a implementujte knihovnu pro C# umožňující správu časových razítek. Zvolte vhodný způsob otestování vzniklé knihovny a zvolené testy proveďte a zhodnoťte.

Seznam odborné literatury:

[1] PETERKA, Jiří. Báječný svět elektronického podpisu. Praha: CZ.NIC, c2011. CZ.NIC. ISBN 978-80-904248-3-8.
[2] BUDIŠ, Petr. Elektronický podpis a jeho aplikace v praxi. Olomouc: ANAG, 2008. edice právo. ISBN 978-80-7263-465-1.

Vedoucí: Ing. Jan Kubr

Platnost zadání do konce letního semestru 2017/2018

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
-4-
L.S.

prof. Dr. Michal Pěchouček, MSc.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 6.2.2017

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science and Engineering

Master's thesis

Management of timestamps C# library

Bc. Dominik Hons

Supervisor: Ing. Jan Kubr

Study Programme: Open Informatics , Master

Field of Study: Software Engineering

May 20, 2017

Aknowledgements

I would like to thank my supervisor Ing. Jan Kubr for his valuable advice and kind approach. I am also very thankful to my family for support throughout my study.

Declaration

Abstract

The goal of this work is to analyze current timestamping tools and to design and implement a C# library for managing the creation and verification of digital timestamps. The implementation follows the RFC 3161 timestamp protocol, which is based on the public key infrastructure. The resulting library provides a complete solution for managing timestamps. This library allows creating timestamps from one or more files, represented in different formats, using several hash algorithms. It also provides two output formats of resulting timestamp. The verification of a timestamp returns relevant information about the timestamp such as the generated time or certificate validity period. The library is publicly available as open source software.

Abstrakt

Cílem této práce je analyzovat existující nástroje určené pro práci s časovými razítky a následně navrhnout a implementovat knihovnu pro C# umožňující vytváření a ověřování časových razítek. Implementace knihovny se řídí razítkovacím protokolem RFC 3161, který je založen na infrastruktuře veřejných klíčů. Výsledná knihovna poskytuje kompletní řešení správy časových razítek. Knihovna umožňuje vytvářet razítka z jednoho či více souborů, v různých vstupních formátech, pomocí několika hashovacích algoritmů. Na výběr jsou také dva formáty výsledného razítka. Při ověřování razítka knihovna vrací důležité informace jako oražený čas nebo platnost certifikátu. Tato knihovna je veřejně dostupná jako open source software.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The phenomenon of the current world is digitalization. It influences almost every aspect of everyday life. People are starting to read digital books, listen to digital music, buy things in electronic shops and pay with digital money. This trend also applies to electronic documents. We use them everyday by sending e-mails, paying bills, writing word documents and in many more situations. As it usually is, new technology comes with new obstacles. The challenge is, how to sign a digital document so that it is undeniable who created the document and when it was created. This challenge is solved by using digital signatures and digital timestamps. This work is dealing with the latter one. Digital timestamping (further referred to as just timestamping).

A digital timestamp is a form of a digital signature which is used to capture a moment of time. The timestamp then guarantees that the timestamped document has been created somewhen before that moment of time. Security of digital timestamping is ensured by using publicly available and trusted providers called time-stamping authorities. It also means that nobody, not even author of the document, can temper with the captured timestamp[17].

This project aims to provide a complete set of timestamp management tools while remaining straightforward and easy to use. The library will guide a user through the process of creating, keeping and verifying timestamps. Creating timestamps is a process which at minimum involves providing data to be time stamped. As you can see in Figure 1.1, the provided data is hashed using a cryptographic hash function. This hash is then sent to the time stamping authority, which sends the complete timestamp back (more about this in a later chapter). We don't need the time-stamping authority to verify a timestamp. However, this step involves checking the digital signature used by the time stamping authority to sign a timestamp.

In my opinion, timestamping is quite complicated but interesting topic. Thanks to that, programming a timestamping library is a complex work involving all sorts of programming features. It implicates some knowledge ranging from cryptography to application protocols. Working on this project has broadened up my horizons in many fields. And I genuinely hope that this library will help to simplify timestamp management.

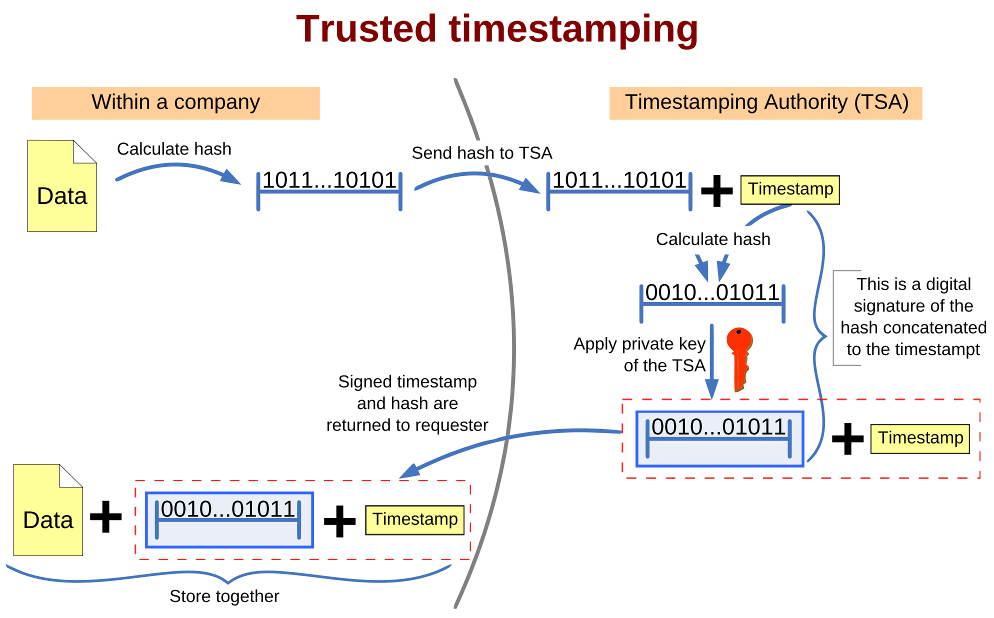


Figure 1.1: Diagram showing steps involved in creating a timestamp[15]

Chapter 2

Analysis

This chapter is going to explain the terminology and processes involved around time-stamping. I will clarify what the goals of the developing library are. Also, I will look into some other current alternatives for timestamping and explain the choices I have made.

2.1 What is a timestamp

As I have previously mentioned, a timestamp is a form of digital signature. It is used to guarantee that a document has been created at a particular point in time. An example of using a timestamp might be to certify that a payment has been made before its due date. Another example of timestamping is when some new technology is invented and needs to be patented before other competitors claim it.

2.1.1 Timestamping schemes

There are several strategies for timestamping, each of them with slightly different security goals. I will explain the advantages and disadvantages of these schemes and explain my choice.

The schemes I will talk about are:

- PKI-based scheme
- Linked-based scheme
- Transient key scheme
- Message authentication code scheme
- Decentralized timestamping using Bitcoin

2.1.1.1 PKI-based

PKI stands for public key infrastructure. It is a series of processes and standards used to ensure that the electronic transfer of information can be carried out securely; it is an industry standard approach used for e-commerce, Internet banking, confidential email and a range of other network activities[13]. In terms of cryptography, it is a technique that enables entities to securely communicate on an insecure public network, and reliably verify the identity of an entity via digital signatures[1].

There are several essential components of this infrastructure:

Certificate authority (CA) is a trusted, independent provider of digital certificates. A CA issues a public key to the sender to enable the data being transferred to be encrypted and a private key to allow the individual receiving the data to decrypt that information. The certificate authority has its own certificate (known as the root key or trusted certificate), public key and secret private key; the CA's trusted certificate is used to verify the digital signatures it issues. In addition to issuing certificates, a CA must also be able to revoke a certificate that has been lost or compromised and to make that information readily available[13].

Digital certificate serves as a confirmation about the identity of the owner of a public key. The public key is a part of the certificate itself. Additionally, the certificate proves that the owner of a public key located in a certificate is in exclusive possession of the corresponding private key. A certificate is associated with a period of time during which the certificate is valid. But the validity may be revoked before the expiration date. This feature serves as a way to deal with a non-standard situation, for example, if the private key is compromised and we can no longer trust the certificate[17].

Public and private key are specially created paired numbers that allow the digital signing of content. The private key positively identifies its owner and it needs to be kept secret. In contrast, the public key, as the name suggest, is publicly distributed and it serves to verify data signed by its paired private key[5].

All of this stands as a base for trusted timestamping. The timestamps are protected by digital signatures and verification of such timestamps consist of verification of digital signatures used to sign these timestamps.

Advantages - simple

Disadvantages - breach of CA[1]

2.1.1.2 Linked-based

Linked timestamping creates timestamp tokens which are dependent on each other, entangled into some authenticated data structure. Later modification of the issued timestamps would invalidate this structure. The temporal order of issued time-stamps is also protected by this data structure, making backdating of the issued timestamps impossible, even by the issuing server itself. The top of the authenticated data structure is generally published in some hard-to-modify and widely witnessed media, like printed newspaper. There are no (long-term) private keys in use, avoiding PKI-related risks[21].

Figure 2.1 illustrates the simplest linear hash chain-based timestamping scheme.

[1]https://www.schneier.com/academic/archives/2000/01/ten_risks_of_pki_wha.html

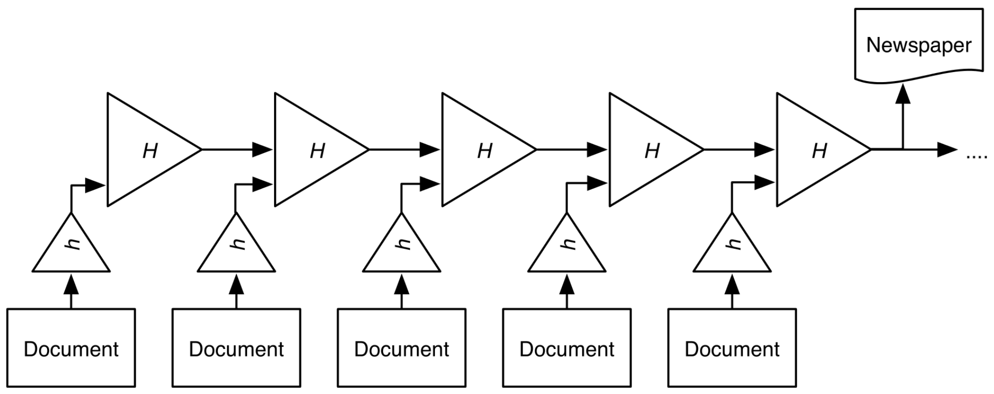


Figure 2.1: Diagram showing simple linked timestamping scheme[18]

The standard ISO/IEC 18014 describes the mechanisms for producing linked based timestamps[2].

2.1.1.3 Transient key

Transient key cryptography is a form of public-key cryptography wherein keypairs are generated and assigned to brief intervals of time instead of to individuals or organizations. Data encrypted with a private key associated with a specific time interval can be irrefutably linked to that interval. A keypair is active only for a few minutes, after which the private key is permanently destroyed. Therefore, unlike public-key systems, transient-key systems do not depend upon the long-term security of the private keys[23].

Whenever a time interval in a transient-key system expires, a new public/private keypair is generated, and the private key from the previous interval is used to digitally certify the new public key. This process is illustrated in Figure 2.2. The old private key is then destroyed. As an extra security measure, all requests for signatures made during an interval are stored in a log that is concatenated and is itself appended to the public key at the start of the next interval. This mechanism makes it impossible to insert new "signed events" into the interval chain after the fact. Transient-key cryptography is protected under US Patent #6,381,696 and has been included in the ANSI ASC X9.95 standard for Trusted Timestamping[23].

2.1.1.4 Message authentication code

In cryptography, a message authentication code (MAC) is a short piece of information used to authenticate a message. In other words, to confirm that the message came from the stated sender (its authenticity) and had not been changed in transit (its integrity). A MAC algorithm accepts a secret key and an arbitrary-length message to be authenticated as input and outputs a MAC. The MAC value protects both a message's data integrity as

[2]https://www.iso.org/standard/50457.html

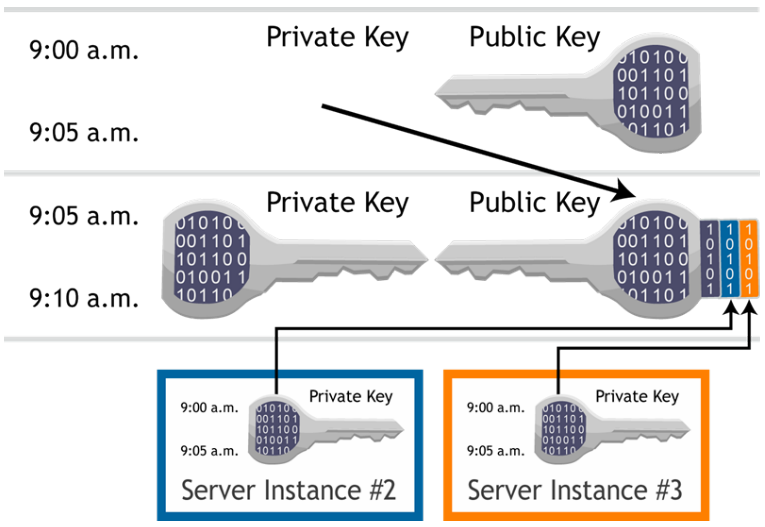


Figure 2.2: Creating new keypair in transient key timestamping scheme[20]

well as its authenticity, by allowing verifiers (who also possess the secret key) to detect any changes to the content of the message. MACs differ from digital signatures as MAC values are both generated and verified using the same secret key. This implies that the sender and receiver of a message must agree on the same key before initiating communications, as is the case with symmetric encryption. This also means that MACs do not provide the property of non-repudiation[3] offered by digital signatures[22].

2.1.1.5 Decentralized timestamping using Bitcoin

Crypto currencies can serve as decentralized trusted timestamping services if hash values of digital data are embedded into the transactions recorded in the block chain of the crypto currency. Service[4] based on this design has been created to allow users to hash files and store these hashes on the block chain. Users can then retrieve and verify the timestamps that have been committed to the block chain. The non-commercial service enables anyone, e.g., researchers, authors, journalists, students, or artists, to prove that they were in possession of certain information at a given point in time. Common use cases include proving that a contract has been signed, a photo taken, a video recorded, or a task completed prior to a certain date. All procedures maintain complete privacy of the user's data[9].

This approach sounds promising, however, in order to reduce operating costs, the service collects submitted hashes, concatenates them and generates a single aggregated hash. This hash is then submitted as a Bitcoin transfer, which implicates spending a small amount of currency. From personal observation, the service does that approximately every month, which makes it unsuitable for situations where timestamps must be created as soon as possible.

[3]It means that the sender cannot deny sending the message.
[4]http://www.originstamp.org

2.1.2 Timestamping standards

There are various standards covering timestamping. I will shortly talk about three of them, RFC 3161, X9.95 and ISO/IEC 18014.

2.1.2.1 RFC 3161

This standard was created by the Internet Engineering Task Force[5]. It is the most widely used timestamping standard. The text is fully available online at <`https://tools.ietf.org/html/rfc3161`> for free. It contains a detailed description of requests and responses between TSA and requester, the requirements of TSA and explains security considerations when dealing with timestamps. The developing library will be following this standard because it is the most widely used, many TSAs offer RFC 3161 compliant timestamps and also it is the only free standard out of these three.

2.1.2.2 X9.95

The standard, X9.95 describes the roles, responsibilities and requirements for users of trusted time stamps-time source entities, time stamp authorities, time stamp requestors, and time stamp relying parties. The standard also specifies data objects, processing flows, error handling, and message formats as well as defines technology methods for digital signature, message authentication code, linked token, and transient key. In addition, the standard offers a comprehensive set of time stamp control objectives to validate a trusted time stamp system for use by a professional audit practitioner. It also provides sample time stamp policy and time stamp practice statements[24]. At the time of writing, this standard is available for purchase at <`http://webstore.ansi.org/RecordDetail.aspx?sku=ANSI+X9.95-2016`> for 100 dollars.

2.1.2.3 ISO/IEC 18014

This standard was made by the International Organization for Standardization[6]. It consists of three parts. The first one gives general definitions of timestamping services. The second one specifies mechanisms producing independent tokens. And the third one deals with mechanisms producing linked tokens. All three parts are available in ISO store at <`https://www.iso.org/search/x/`>.

2.1.3 Creating timestamps

Timestamps are created by trusted third party called Time Stamping Authority (TSA). But we need to provide it with all necessary information. The process of creating timestamps is shown in Figure 1.1 and I will split it into several steps for an explanation.

[5]http://ietf.org/
[6]https://www.iso.org/about-us.html

1. **Hash calculation:** The first step on the road to getting a timestamp is to calculate a hash from the file we want to timestamp. This hash is also called message digest, and it can be seen as the fingerprint of a message, i.e., a unique representation of a message. The calculation is made by cryptographic hash function. This function has fixed output length, usually between 128-512 bits. It should be highly sensitive to all input bits. It means that small changes in input produce significant changes in output. Another important feature of a hash function is one-wayness. It must be computationally infeasible to find the original message from its hash[16]. MD5, SHA1, SHA256 and SHA512 are some of the widely used hash functions. Once the hash is calculated, it is then sent to the TSA.

2. **Timestamp creation** When the TSA receives the request, it does some checks to verify the correctness of the request and the hash itself. After that, it uses a trustworthy source of time to capture a moment of time into a timestamp. This timestamp is together with the sent hash signed by the TSA. It means that it uses a special digital certificate reserved for the purpose of timestamping. This signed timestamp is returned to the requester.

3. **Response verification** Upon receiving the response, the requester SHALL verify that what was time-stamped corresponds to what was requested to be time-stamped. The requester SHALL verify that the timestamp contains the correct certificate identifier of the TSA, the correct data imprint, and the correct hash algorithm. It SHALL then verify the timeliness of the response by verifying either the time included in the response against a local trusted time reference. Then, since the TSA's certificate may have been revoked, the status of the certificate SHOULD be checked. If any of the verifications above fails, the timestamp SHALL be rejected[2].

4. **Timestamp storage** The requester is now in possession of the original file, its hash and signed timestamp. Only the original file and the timestamp needs to be stored. He can throw away the calculated hash because it is going to be calculated again during verification in order to guarantee file integrity, but he needs to remember which hash algorithm was used so it can be applied again.

2.1.4 Verifying timestamps

The process of timestamp verification is very similar to verification of digital signatures. In fact, it is its superset because while verifying a timestamp, it is necessary to verify the digital signature used by the TSA to sign the timestamp. But let's start with the first step.

1. **File integrity check:** The first and relatively easy step in verifying a timestamp is to check that the file has not been tempered since the timestamp was issued. This step is based on the security of cryptographic hash function and asymmetric cryptography. We use the original file to recalculate its hash. Since the hash function has the properties described in the previous section, we can assume that if this hash is equal to the hash inside the timestamp, then the file has not been tempered. To get the original hash from a timestamp, the public key of

TSA needs to be applied to the timestamp. Schema of this process is depicted in Figure 2.3.

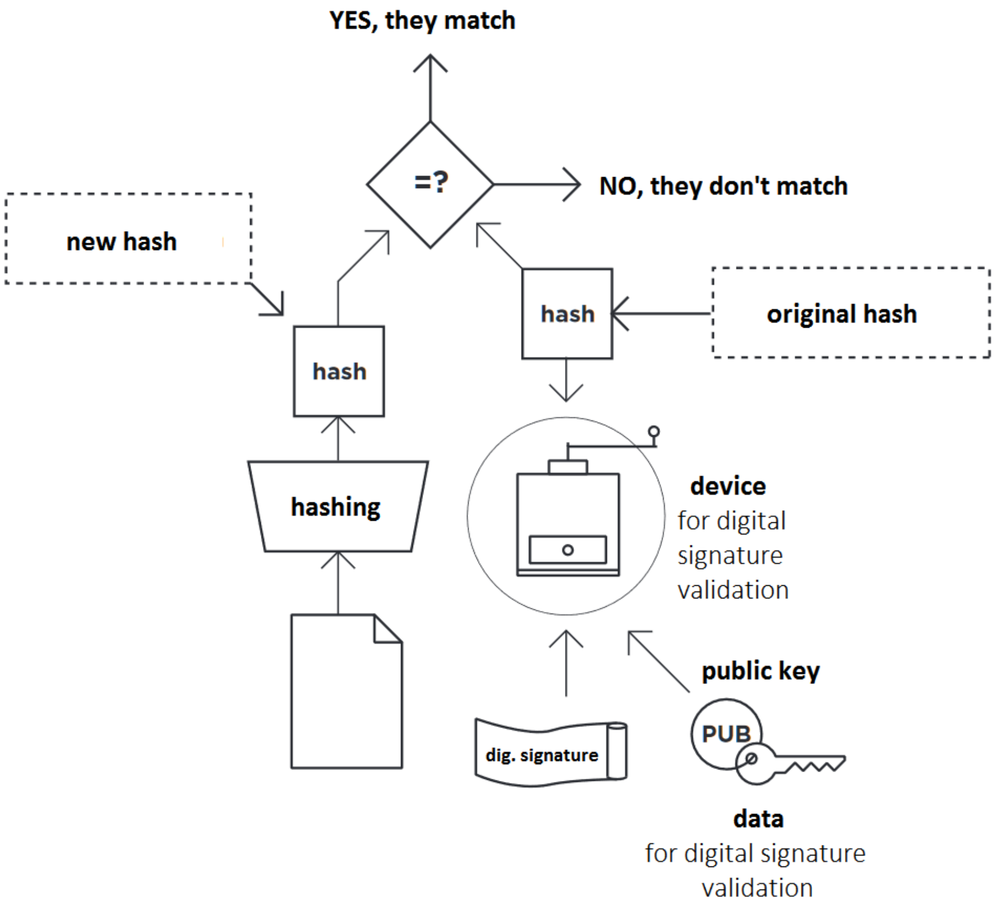


Figure 2.3: Verifying file integrity[17]

2. **TSA certificate check:** There are two parts in this step. The first one is to verify that the certificate's validation period has not expired. This check is done simply by comparing time from the timestamp with the information embedded in the certificate. The other part is a bit more complicated. Certificate's validation period may be expired prematurely, before the official time. This mechanism is called revocation. It serves as a protection against an unusual situation, such as private key compromising. I will explain more about revocation in section 2.1.6. For now, let's just say that the certificate can't be revoked. It is necessary to verify not only the certificate used by TSA but all of the parent certificates as well. This connection is called certificate chain, and it ends in root self-signed certificate[17].

In conclusion, there are three conditions to declare timestamp as valid:

1. File integrity has not been violated
2. No certificate in chain has been revoked before timestamped time
3. Certificate validation period has not been expired for all certificates in chain

However, if the first two conditions are fulfilled but some certificate has expired, then the timestamp is not considered invalid. It is in a "we don't know" state, and its invalidity would have to be proved. Nevertheless, there are some risks associated with expired certificates, so it is a good idea to keep timestamps valid. It is done by re-stamping and I will explain it in section 2.1.7.

2.1.5 Storing timestamps

The final timestamp is a signed data structure represented by a byte array. This byte array can be loaded back into object form in order to be used during verification. A timestamp can be stored on its own. There is, however, the possibility to store it together with the timestamped data in a container called ASIC-S. It is essentially a zip container that holds one or multiple files and their associated timestamp. If the timestamp is applied to multiple files, they have to be stored in a nested container inside the ASIC-S container. Figure 2.4 shows the resulting structure. This approach negates the risk that the signature becomes separated from the data to which it applies[6].

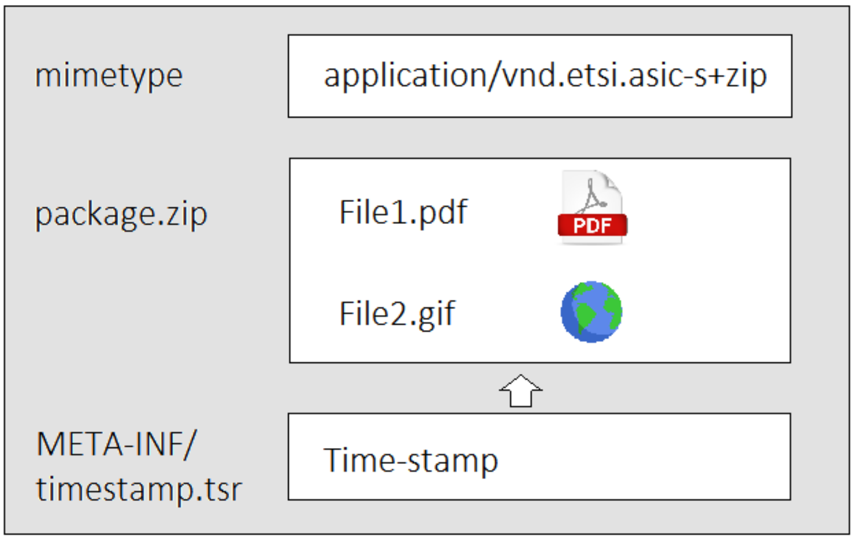


Figure 2.4: ASIC-S structure applied to a nested container

2.1.6 Revocation

The certificate used by TSA to timestamp our data may be revoked. It means that the private key could have been compromised or that there is some other reason to invalidate it.

The important thing is that this certificate is potentially dangerous and needs to be checked against the rules described in Figure 2.5. This verification is done because some revocation reasons don't necessarily mean that timestamps created with this certificate are invalid. For example, the TSA may be ending its operation, therefore revoking the certificate for future usage, but already created timestamps remain valid[2].

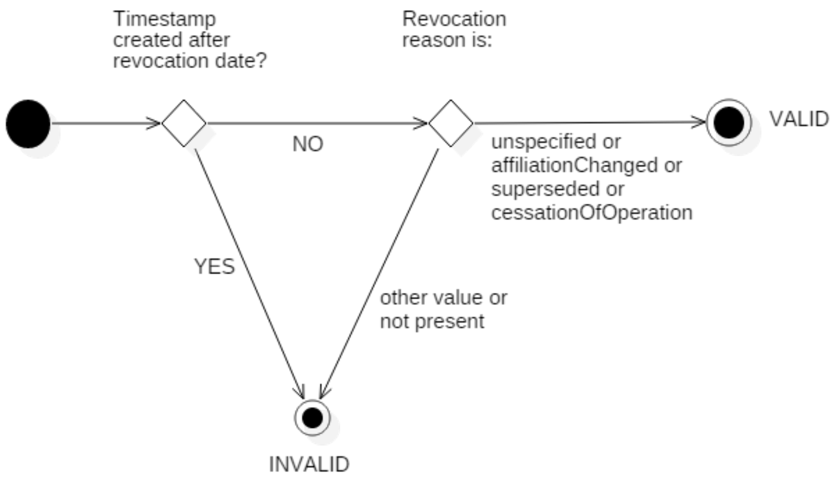

Timestamp created after revocation date?
Revocation reason is:
NO
YES
VALID
unspecified or affiliationChanged or superseded or cessationOfOperation
other value or not present
INVALID

Figure 2.5: Timestamp validity with revoked certificate

2.1.7 Re-stamping

Certificates used by TSA to timestamp our documents have limited period of validity (usually between one and five years). The reason is the fact that over time the development of cryptography or the computational force of computers may allow finding collision documents. This indicates that someone can pretend that this collision document is the original one. As depicted in Figure 2.6, two different documents have the same imprint, and therefore both of them can claim the original timestamp.

Re-stamping is one method how to keep our documents timestamped after the certificates expire. The document is being continuously timestamped again and again, right before the current certificate validity period is about to end. Repeated signing grants the ability to react to the aging of cryptographic algorithms. Each new stamp can use new, stronger and bigger ciphers. This way we can adequately increase the difficulty of finding collision documents[17].

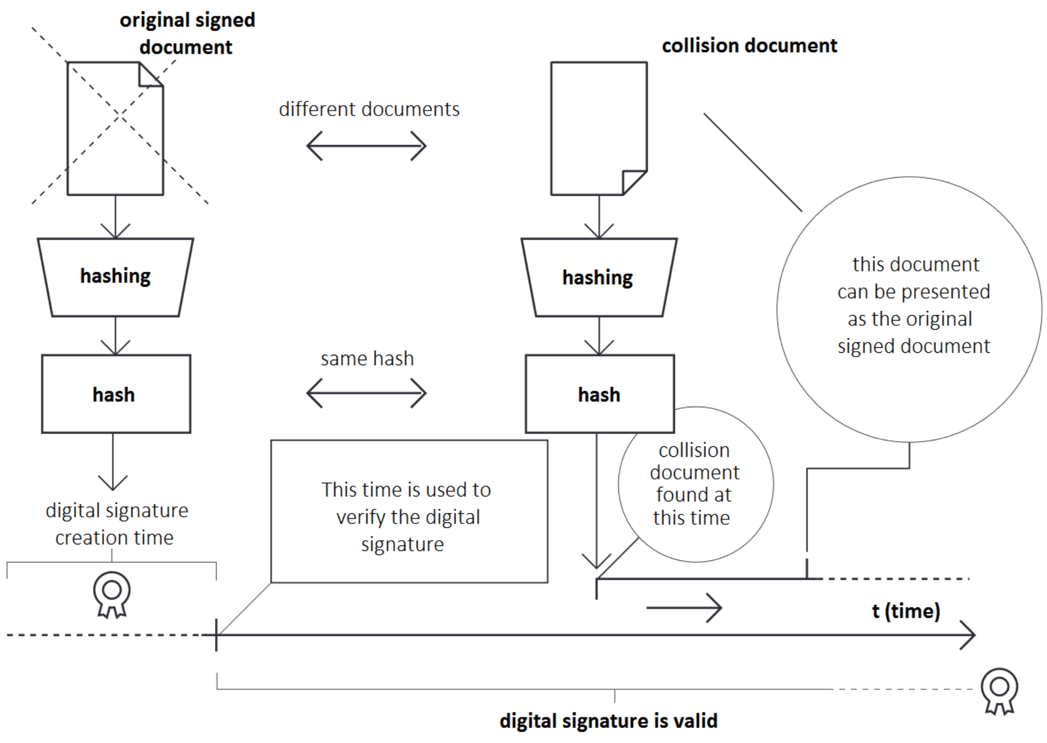

original signed document
collision document
different documents
hashing
hashing
this document can be presented as the original signed document
same hash
hash
hash
collision document found at this time
This time is used to verify the digital signature
digital signature creation time
t (time)
digital signature is valid

Figure 2.6: Possible collision document[17]

2.2 Current alternatives

This section introduces what the current alternatives for timestamp management are.

2.2.1 TimeStampClient

TimeStampClient is easy to use .NET RFC 3161 time-stamp client library and applications based on Bouncy Castle library and it is available on Github[7] and Nuget[8]. It is comprised of a C# library, command line application and a GUI application. The library provides tools for creating timestamps. The basic usage is shown in Listing 2.1. However, this library doesn't provide any timestamp verification.

2.2.2 TSA services

Many companies providing timestamping services have their own internal solutions that they offer to customers. It can be in the form of desktop or web application.

An example of a desktop application is TSA_klient from PostSignum. Demo version is available for free on their website <`http://www.postsignum.cz`>. They sell packages of

[7]<`https://github.com/disig/TimeStampClient`>
[8]<`https://www.nuget.org/packages/TimeStampClient/`>

Listing 2.1: Example of TimeStampClient library usage

```
using Disig.TimeStampClient;

...

TimeStampToken token = TimeStampClient
    .RequestTimeStampToken("http://localhost/tsa", "document.docx");

...
```

timestamps which are required in order to use the proper version. The application is a bit older (2009). It uses the Bouncy Castle library.

As an example of a web application, I have picked universign[9]. It is a company similar to PostSignum, which offers electronic signatures and timestamps. They provide five free timestamps that are available after signing up. The web application is simple to use by dragging and dropping files, and there is a small tutorial available at <https://www.universign.eu/en/timestamp/online/>.

[9]<https://www.universign.eu/>

Chapter 3

Used technologies

This chapter will describe the technologies used for this project. It includes programming language, framework, and packages. The main technologies, C# and .NET, were chosen by supervisor.

3.1 C#

C# is a modern object-oriented, general-purpose programming language, created and developed by Microsoft together with the .NET platform. C# is a high-level language that is similar to Java and C++. Because C# is developed by Microsoft as part of their modern platform for development and execution of applications, the .NET Framework, the language is widely spread among Microsoft-oriented companies, organizations and individual developers. The C# language is distributed together with a special environment on which it is executed, called the Common Language Runtime (CLR). This environment is part of the platform .NET Framework, which includes CLR, a bundle of standard libraries providing basic functionality, compilers, debuggers and other development tools. Thanks to the framework CLR programs are portable and, once written they can function with little or no changes on various hardware platforms and operating systems. C# programs are most commonly run on MS Windows, but the .NET Framework and CLR also support mobile phones and other portable devices based on Windows Mobile, Windows Phone and Windows 8. C# programs can still be run under Linux, FreeBSD, iOS, Android, MacOS X and other operating systems through the free .NET Framework implementation Mono, which, however, is not officially supported by Microsoft[11].

3.2 .NET framework

The C# language is not distributed as a standalone product – it is a part of the Microsoft .NET Framework platform. .NET Framework generally consists of an environment for the development and execution of programs, written in C# or some other language, compatible with .NET (like VB.NET, Managed C++, J# or F#). The .NET Framework is part of every modern Windows distribution and is available in different versions. It consists of:

- the .NET programming languages (C#, VB.NET and others)
- an environment for the execution of managed code (CLR), which executes C# programs in a controlled manner
- a set of development tools, such as the csc compiler, which turns C# programs into intermediate code (called MSIL) that the CLR can understand
- a set of standard libraries[11]

3.2.1 Cryptography library

One of the most useful .NET library in this project was the Cryptography library. It provides services such as hashing algorithms, that are crucial for timestamping. The cryptography namespace also includes X509Certificates library which presents resources for certificate management.

3.3 Bouncy Castle library

Bouncy Castle is a collection of cryptographic APIs. It started with Java and C# was added in 2006. It provides an implementation of many cryptographic algorithms. But most importantly, it provides means for generating and processing timestamp request. This functionality is following the RFC 3161 standard. Bouncy Castle is licensed under an adaptation of the MIT X11 license[10].

Chapter 4

Design

4.1 Workflow

This section is going to explain workflow of the two main library features, creating and verifying timestamps.

4.1.1 Creating timestamps

The workflow of creating timestamps is pictured in Figure 4.1. There are three roles in the process of creating timestamps, a user of the library, the library, and TSA. The process starts with the user. He has to provide all necessary inputs such as the URL and credentials of primary and secondary TSA, hash algorithm for calculating message digest and most importantly the data he wants to timestamp. The library then validates whether the input is complete. If so, it generates timestamp request, which is sent to primary TSA. If there is some exception during the communication with primary TSA (e.g. connection timeout, malformed URL), then the secondary TSA is used. TSA returns timestamp response which needs to be validated. If it is valid, all that's left to do is to validate the certificate used by TSA to sign the timestamp. The user is given the timestamp and other relevant information such as the time of timestamp, the issuer of the timestamp, certificate validity period or hash algorithm.

4.1.2 Verifying timestamps

The verification process starts by a user providing a timestamp and the data to which the timestamp applies. Then the library takes control. First, it validates that the provided input is complete. Then integrity of timestamped file is checked. And finally, all certificates in certificate chain are checked for validity. If the verification is successful, the library returns the timestamp with other information, same as it does after creating timestamp. Otherwise, an exception is raised. The complete process is depicted in Figure 4.2.

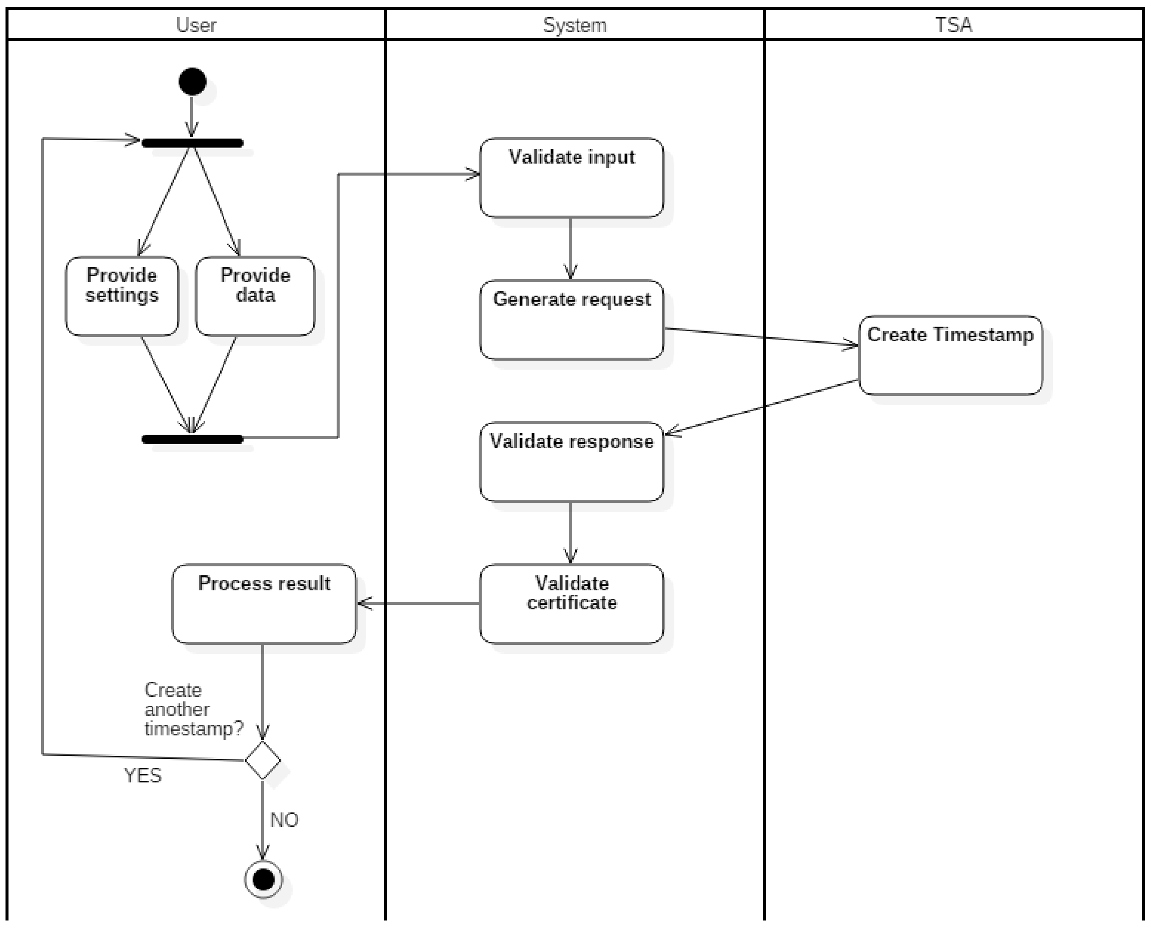


Figure 4.1: Diagram depicting workflow of timestamp creation

4.2 Data privacy

There will be occasions where the user wants to timestamp some private document. The user is cautious and doesn't want to give the private document to some unknown library. For this cases, there is the possibility to provide already calculated message digest instead of the original document. It means that the user hashes the file on his own and give the final hash to the library. The original file remains private to the user. However, this approach leaves a little bit more responsibility on the user.

4.3 API

This section provides the description of library's interface available to users. Once again, this is divided into two parts, creating and verifying timestamps. Mandatory items are

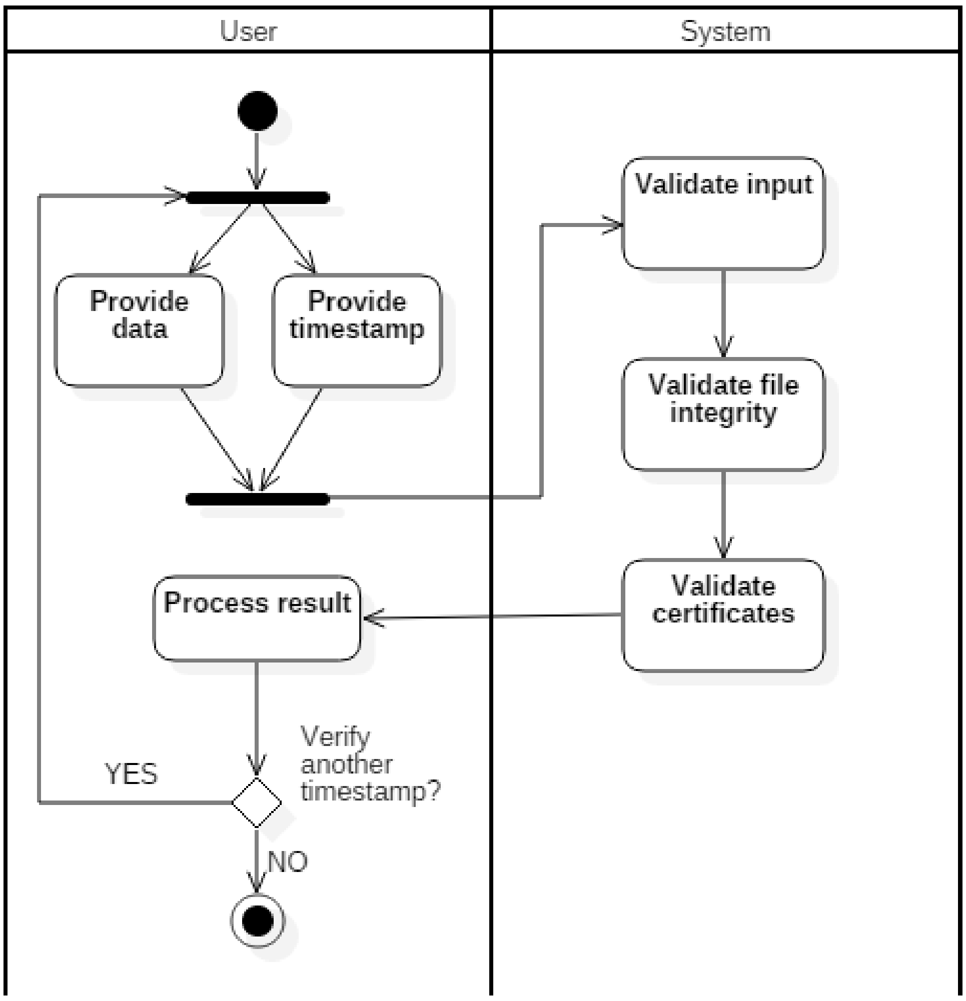


Figure 4.2: Diagram depicting workflow of timestamp verification

highlighted in bold. Some of the inputs can be provided through a configuration file which is described in section 4.4.

4.3.1 Creating timestamps

Inputs:

- **Primary TSA url**
- **Hash algorithm**
- **Data to be timestamped**

  - File
  - List of files
  - Message digest
- **Output format**
- Secondary TSA url
- Primary and secondary TSA credentials
- TSA connection timeout
- Minimum certificate validity period

Outputs:

- Timestamp
  - TSR format
  - ASIC-S format
- Additional info (generated time, certificate validity period, ...)

4.3.2 Verifying timestamps

Inputs:

- **Timestamp**
- **Hash algorithm**
- **Timestamped data**
  - File
  - List of files
  - Message digest
- Minimum certificate validity period

Outputs:

- Timestamp
- Additional info (generated time, certificate validity period, ...)

4.4 Configuration file

Several data, required for working with timestamps, can be provided through a configuration file. The purpose of this approach is to minimize the complexity of the code. Static data that doesn't change, such as credentials to access TSA, can be retrieved from this file. This way the user doesn't have to specify these settings every time he wants to create or verify a timestamp. All of the possible configuration options are described in Table 4.1. The configuration file will have the form of a simple text file. Each row contains exactly one key-value pair of settings separated with the equals sign (example: hash.algorithm=sha1). The configuration file should be easy to use. All of the settings are together in one file, available for a change. The file will be provided as a string containing the path to this file.

Configuration name	**Description**	**Value**
tsa.primary.url	Url address of primary TSA	
tsa.primary.username	Username for accessing protected TSA	
tsa.primary.password	Password for accessing protected TSA	
tsa.secondary.url	Url address of secondary TSA	
tsa.secondary.username	Username for accessing protected TSA	
tsa.secondary.password	Password for accessing protected TSA	
tsa.timeout	TSA connection timeout limit (milliseconds)	
hash.algorithm	Hash algorithm used to create message digest	MD5 SHA1 SHA256 SHA512
timestamp.output	Specifies format of timestamp that is returned	TSR ASICS
certificate.minimum.validity	Minimum time period when signing certificate has to be valid (days)	

Table 4.1: Table containing configuration options

4.5 Fluent interface

As you can see in the API section, the library offers a lot of different input options. This would require quite a complicated approach to building objects responsible for creating and verifying timestamps. There is a solution to alleviate this problem. It is the fluent interface approach. It is used to simplify the construction of complex object that would normally require having multiple constructors to satisfy all possible combination of provided parameters, or setting these parameters separately. This approach is primarily designed to be readable and to flow. The price of this fluency is more effort, both in thinking and in the API construction itself[8]. The best way to express the improved readability is to look at the example in Listing 4.1 and compare it to the traditional approach in Listing 4.2.

Listing 4.1: Creating timestamp using fluent interface

```
TimestampCreator creator = new TimestampCreator();
TimestampObject timestamp = creator
        .SetTsaPrimaryUrl(tsaUrl)
        .SetDataForTimestamping(data)
        .CreateTimestamp();
```

Listing 4.2: Creating timestamp without fluent interface

```
TimestampCreator creator = new TimestampCreator();
creator.SetTsaPrimaryUrl(tsaUrl);
creator.SetHashAlgorithm(HashAlgorithm.SHA1)
creator.SetOutputFormat(OutputFormat.TSR)
creator.SetDataForTimestamping(data)
TimestampObject timestamp = creator.CreateTimestamp();
```

Chapter 5

Implementation

The Implementation chapter describes the resulting library code. The library structure is shown below. It is comprised of interfaces, implementing classes, helper classes, enums and a configuration text file.

```
TimestampLibrary
├── ITimestampCreator.cs
├── ITimestampVerifier.cs
├── TimestampCreator.cs
├── TimestampData.cs
├── TimestampException.cs
├── TimestampObject.cs
├── TimestampVerifier.cs
├── Utils.cs
└── Enums
    ├── HashAlgorithm.cs
    └── OutputFormat.cs
```

5.1 TimestampCreator

TimestampCreator is the main class responsible for creating timestamps. It implements the ITimestampCreator interface. This interface defines the public API for creating timestamps. It is shown in Table 5.1. When TimestampCreator is instantiated it already contains properties that have been defined in the configuration file. These properties can be manually overwritten using setters. And these setters can be called fluently as it was described in section 4.5.

Data for timestamping can be provided in several different formats. The library accepts a file as a stream, a byte array or a string representing the path to the file. Each of these formats can also be supplied in an array in order to timestamp multiple files together. If the user doesn't want to provide the file, for example for privacy reasons, it can be exchanged for already calculated message digest. Provided file or files are handed over to TimestampData object which will be described later.

Type	**Method**
ITimestampCreator	SetTsaPrimaryUrl(string tsaUrl)
ITimestampCreator	SetTsaPrimaryCredentials(string username, string password)
ITimestampCreator	SetTsaSecondaryUrl(string tsaUrl)
ITimestampCreator	SetTsaSecondaryCredentials(string username, string password)
ITimestampCreator	SetTsaTimeout(int timeout)
ITimestampCreator	SetHashAlgorithm(HashAlgorithm hash)
ITimestampCreator	SetOutputFormat(OutputFormat format)
ITimestampCreator	SetMinimimCertificateValidityPeriod(int days)
ITimestampCreator	SetDataForTimestamping(string pathToFile)
ITimestampCreator	SetDataForTimestamping(string[] pathsToFiles)
ITimestampCreator	SetDataForTimestamping(Stream stream)
ITimestampCreator	SetDataForTimestamping(Stream[] streams)
ITimestampCreator	SetDataForTimestamping(byte[] data)
ITimestampCreator	SetDataForTimestamping(byte[][] datas)
ITimestampCreator	SetMessageDigestForTimestamping(byte[] digest)
ITimestampCreator	SetMessageDigestForTimestamping(byte[][] digests)
TimestampObject	CreateTimestamp()

Table 5.1: ITimestampCreator API

After all of the necessary data has been provided, it is time to invoke the CreateTimestamp method. First, it verifies that everything essential has been provided. Then, if necessary, the hash is calculated from the given file or files. This is done again by the TimestampData object. Now it is time to involve TSA. TimeStampRequest is generated from message digest and hash algorithm (because the TSA is required to check that the digest is of corresponding length to its hashing algorithm) and send over HTTP to the TSA. If necessary, credentials to access the TSA are included in the HTTP request. If any exception occurs during communication with TSA, the secondary TSA is used (if provided). Response from TSA needs to be validated for appropriate status and other controls. Some of this validation is handled by Bouncy Castle (for example it checks that the message imprints are identical in the request and the response). If the response passes this validation, all that's left is to validate the certificate and its predecessors in the chain. This is done by TimestampVerifier which is described in the next section. After the certificates are declared as valid, TimestampObject, containing the timestamp and additional info, is returned to the user.

In conclusion, the basic usage of TimestampCreator is shown in Listing 5.1. However, this can be even further simplified by providing repetitive settings through the configuration file.

Listing 5.1: Basic usage of TimestampCreator

```
TimestampCreator creator = new TimestampCreator();
TimestampObject timestamp = creator
      .SetTsaPrimaryUrl(tsaUrl)
      .SetHashAlgorithm(HashAlgorithm.SHA1)
      .SetOutputFormat(OutputFormat.TSR)
      .SetDataForTimestamping(data)
      .CreateTimestamp();
```

5.2 TimestampVerifier

This class is in charge of verifying timestamps. It includes the verification of file integrity and validation of certificate chain. Similar features can be found in TimestampVerifier and TimestampCreator, namely that both implement its own interface (ITimestampVerifier, ITimestampCreator), during instantiation properties are loaded from a configuration file and both use the fluent interface. The API of ITimestampVerifier is shown in Table 5.2.

Type	**Method**
ITimestampVerifier	SetTimestampedData(string pathToFile)
ITimestampVerifier	SetTimestampedData(string[] pathsToFiles)
ITimestampVerifier	SetTimestampedData(Stream stream)
ITimestampVerifier	SetTimestampedData(Stream[] streams)
ITimestampVerifier	SetTimestampedData(byte[] data)
ITimestampVerifier	SetTimestampedData(byte[][] datas)
ITimestampVerifier	SetMessageDigest(byte[] digest)
ITimestampVerifier	SetMessageDigest(byte[][] digests)
ITimestampVerifier	SetTimestamp(string pathToFile)
ITimestampVerifier	SetTimestamp(byte[] data)
ITimestampVerifier	SetTimestamp(Stream stream)
ITimestampVerifier	SetHashAlgorithm(HashAlgorithm hash)
ITimestampVerifier	SetMinimimCertificateValidityPeriod(int days)
TimestampObject	Verify()

Table 5.2: ITimestampVerifier API

Three things are necessary to verify a timestamp: the timestamp, the timestamped file or files and hash algorithm used to calculate message digest during timestamp creation. The timestamp is a binary representation of TimeStampResponse object. It is used to verify the file integrity in following way. File or files are used to recalculate message digest using the same hash algorithm as in creation. TimeStampRequest is generated from this hash and now it is time to validate it against the response. If it passes this validation, it means that file integrity has not been violated. All of the above mentioned is expressed in four lines as it is shown in Listing 5.2. The hash calculation is handled by TimestampData object and validation by Bouncy Castle's TimeStampResponse object.

Listing 5.2: File integrity verification

```
byte[] hashedData = timestampData.getHashedData(this.hashAlgorithm);
TimeStampRequestGenerator requestGenerator = new TimeStampRequestGenerator();
TimeStampRequest request = requestGenerator
    .Generate(new Oid(this.hashAlgorithm.ToString()).Value, hashedData);
this.timestampResponse.Validate(request);
```

Certificate validation is managed by a static method because this validation is needed during timestamp creation as well. This method takes TimeStampToken as a parameter, which is extracted from TimeStampResponse. This token contains the signing certificate, and at first, it is validated that it is actually signed by this certificate. Another step is to check if this certificate was valid at the time of timestamp creation. This is a necessary condition for timestamp validity. If the certificate was valid at that time but it has expired now, or it is going to expire soon, a warning is appended to the resulting TimestampObject.

Now it is time to validate the certificate's chain. It is done by .NET's X509Chain Build method. There are many reasons for this validation to fail such as untrusted root, partial chain or revoked certificate. All possible reasons are enumerated in X509ChainStatusFlags class. If any of these reasons occur, the validation is marked as failed, with the exception of the revoked reason. Even if there are any revoked certificates, there is still a possibility for the validation to succeed. Method IsValidAfterRevocation takes the certificate and validates it against the rules described in section 2.1.6. It does that by searching for revocation date and reason inside certificate revocation list. The evaluation of revocation date and reason is shown in Listing 5.3.

5.3 Additional classes

5.3.1 TimestampData

This class serves as a container for timestamped file or files. It has constructors allowing to accept all possible input formats (stream, byte array, string with the path to file). The class also contains private enum DataMode which specifies whichever input format is currently used to provide the input.

The most important functionality of this class is calculating message digest from the data it holds. This is used during timestamp creation and verification as well. The method GetHashedData accepts the name of the hash algorithm as an input and uses this name to create the corresponding instance of .NET's HashAlgorithm. In case there are multiple files given to timestamp, they need to be hashed together. HashAlgorithms in .NET implement ICryptoTransform interface. This interface provides two methods, TransformBlock and TransformFinalBlock. These methods allow calculating the hash block by block, in our case file by file[7]. Headers of these methods look complicated, but they are simply given the byte array representation of current file and its length. Code for this calculation process is shown in Listing 5.4.

Listing 5.3: Revoked certificate validation

```
/* All timestamps created after revocation date are invalid */
if (DateTime.Compare(timestampGenTime, revocationDate) > 0)
{
    return false;
}

DerEnumerated reasonCode = DerEnumerated
    .GetInstance(GetExtensionValue(revokedEntry, X509Extensions.ReasonCode));

/* If the revocation reason is not present,
    the timestamp is considered invalid */
if (reasonCode == null)
{
    return false;
}

int reason = reasonCode.Value.IntValue;

/* If the revocation reason is any other value,
    the timestamp is considered invalid */
if (!(reason == Org.BouncyCastle.Asn1.X509.CrlReason.Unspecified ||
    reason == Org.BouncyCastle.Asn1.X509.CrlReason.AffiliationChanged ||
    reason == Org.BouncyCastle.Asn1.X509.CrlReason.Superseded ||
    reason == Org.BouncyCastle.Asn1.X509.CrlReason.CessationOfOperation))
{
    return false;
}
```

Listing 5.4: Calculating hash from multiple files

```
private byte[][] datas;
...
for (int i = 0; i < datas.Length-1; i++)
{
    algorithm.TransformBlock(datas[i], 0, datas[i].Length, datas[i], 0);
}
algorithm
.TransformFinalBlock(datas[datas.Length-1], 0, datas[datas.Length-1].Length);
return algorithm.Hash;
```

5.3.2 TimestampObject

This class is just a placeholder for timestamp relevant data. It is returned to the user at the end of timestamp creation and verification. It contains the generated time, the name of the hash algorithm, the message imprint, the TSA's signing certificate and its validity period, a string containing warning message in case the certificate has expired or is going to expire and finally the timestamp itself. The timestamp is a byte array which corresponds to either TimestampRequest object or ASIC-S container depending on user's choice of output format.

5.3.3 Utils

Utils is a static class designed to help with common generic tasks, such as reading settings from a configuration file and creating zip containers. It contains a LoadConfigurationFile method which parses settings from the configuration file and saves these values into a dictionary. For creating the ASIC-S container structure, .NET's ZipArchive and ZipArchiveEntry classes were used.

Type	**Method**
void	LoadConfigurationFile(string pathToFile)

Table 5.3: Utils API

5.3.4 HashAlgorithm and OutputFormat

These enums are used as parameters inside the library. They enumerate possible choices of hash algorithm and output format that users can select. In addition, HashAlgorithm contains a mapping between algorithm name and its output length. This is used to verify the correctness of input when the user provides already calculated hash.

5.4 Exceptions

During the execution of timestamp creation or verification, many non-standard situations may occur. This ranges from user providing incorrect input (e.g. non-existing TSA URL, invalid path to file or wrong timestamp) over infrastructure errors (e.g. HTTP connection not working) to validation exceptions (e.g. invalid response status from TSA, expired signing certificate). In all of these situations, the library throws appropriate exception to inform the user about the reason of failure.

In order to distinguish between exceptions coming from outside and inside the library, custom exception called TimestampException has been created. This exception serves as a marker for user to know if there is something wrong with his input[19]. Whenever this exception is raised, the user should be aware that the problem is probably related to his actions.

All exceptions raised by the library and their description are listed below.

TimestampException: Custom made exception serving as a marker for exceptions occurring inside the logic of the library. This exception is raised in situations related to user's input such as incorrect path to a file, invalid message digest length, wrong TSA's URL.

ArgumentNullException: This exception is raised when any of the mandatory input has not been provided. In such case, the library can't proceed with execution because these inputs are essential for timestamp creation of verification. An example of these inputs is the hash algorithm, data for timestamping, TSA URL and timestamp output format.

TspException and TspValidationException: These exceptions come from the Bouncy Castle. They are thrown when exceptions happen while using objects or methods from Bouncy Castle. For example, method Validate on object TimestampResponse throws TspException if message digests don't match on request and response.

CryptographicException: Coming from .NET, this exception is thrown in case the certificate chain validation fails. It is enhanced by failure reason and sometimes there are multiple reasons for the validation to fail.

AggregateException: AggregateException is used to consolidate multiple failures into a single, throwable exception object[12]. It is used when exception occurs while accessing both primary and secondary TSA.

5.5 Evaluation of the implementation

Every library requirement and feature has been successfully implemented. The library provides all necessary tools to create and verify timestamps. Usage of the library should be very straightforward and also readable thanks to using fluent interface.

Creating a timestamp can be done in a few steps, by providing several essential inputs, such as the files that are going to be timestamped. And the library is able to process various different formats of input. It also offers two output formats of the final timestamp. One of these, the ASIC-S container, is accommodating the timestamp together with files to which it applies.

Timestamp verification is done very thoroughly and precisely in my opinion. The trickiest part was the certificate validation and in particular the special case of revoked certificate. Microsoft chain validation can find revoked certificates but doesn't give any additional information about the revocation date or reason. So it had to be achieved by getting and parsing the certificate revocation list. In conclusion, the verification process provides user with very detailed explanation of status of his timestamp.

Chapter 6

Testing

Testing is an essential part of software development. In this project, testing was integrated into every development stage. Three levels of testing were utilized:

1. Unit testing
2. System testing
3. Performance testing

6.1 Test-driven development

Test-driven development is a technique used in software development. Its main idea is to write small tests before developing corresponding functionality. This process ensures that the tests are correctly reacting to new functionality thus increasing developer's trust in the tests. These steps should be written to cover requirements from a specification. It forces the developer to fully understand the specification of functionality he's about to code. The development is done by incrementally repeating following steps:

1. Quickly add a test: A vital first step is to write a test. It can cover new function or improvement of a function, even just a small change. In reality, tests are usually written for whole functions, but it's nice to know that it is possible to tests even minor changes, because these tests come in handy when things are not working as they should. This first step assures that the developer is focused on the requirements before writing the code.

2. Run all tests and see the new one fail: Failure is progress. The new test should fail. If it doesn't fail then either the functionality is already developed, or the test is poorly written. This step gives programmer base for further development.

3. Make a little change: The next step is to write some code that makes the test to pass. It can be written in a very simple and non-elegant way. The purpose of this step is to just pass the test. The code will be refactored later. There are two strategies for quickly getting the test to pass:

- Fake It — return a constant and gradually replace constants with variables until you have the real code
- Obvious Implementation — type in the real implementation

When everything goes smoothly, the second strategy is a natural choice. However, it is much simpler to search for mistakes in a fake implementation and gradually refactor to get the real code.

4. **Run all tests and see them all succeed:** Running all tests should result in all of them passing. If not, then the new change probably broke some other functionality and needs to be revised. Passing this step also doesn't instantly mean that everything is finished. The test should be generalized to cover all different situations.

5. **Refactor to remove duplication:** This step is dedicated to refactoring code from step 3. The code passing the test could have been copied from a similar functionality or just written ineffectively. Object, function and variable names should clearly express their meaning[3].

Test-driven development was accomplished by utilizing unit tests. These tests were created during development. They were constantly changing, to include new features or to cover bigger section of possible input values. Nice example of this approach are the tests for TimestampData GetHashedData function. It is quite a complex function that has to deal with several different formats of data and non-standard situations when calculating the hash. So for example test called TestGetHashFromPath was created to cover the situation of calculating the hash from a file provided by path. After this test successfully passed, a new test called TestGetHashFromWrongPath was designed to cover the situation when the provided path is incorrect. These tests are demonstrated in Listing 6.1.

Listing 6.1: TimestampData tests for GetHashedData function

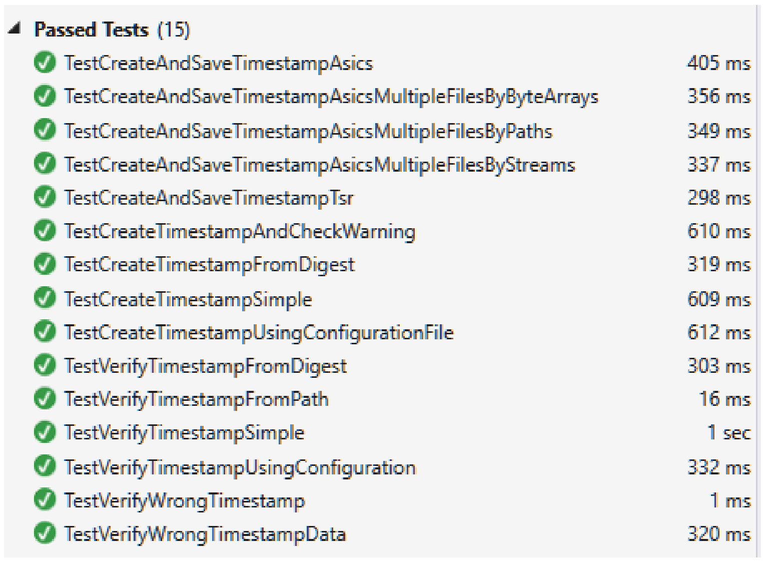
```
[TestMethod]
public void TestGetHashFromPath()
{
  TimestampData timestampData = new TimestampData(path);
  byte[] result = timestampData.GetHashedData(HashAlgorithm.SHA1);
  byte[] data = Encoding.UTF8.GetBytes(dataToTimestamp);
  byte[] expectedResult = new SHA1Cng().ComputeHash(data);
  Assert.IsTrue(result.SequenceEqual(expectedResult));
}

[TestMethod]
[ExpectedException(typeof(TimestampException))]
public void TestGetHashFromWrongPath()
{
  string path = "C:\\InvalidPath";
  TimestampData timestampData = new TimestampData(path);
  byte[] result = timestampData.GetHashedData(HashAlgorithm.SHA1);
}
```

6.2 System testing

System testing encompasses the entire system, fully integrated. Sometimes, as in installation and usability testing, these tests look at the system from a customer or end-user point of view. Other times, these tests stress particular aspects of the system that users might not notice but are critical to proper system behavior[4].

System tests were done to examine the library behavior in different situations. Timestamp creation and verification were tested with various settings. The varying parameters include different data formats, hash algorithms, output formats, using a configuration file, different validation results and more. All system tests are depicted in Figure 6.1.



```
◢ Passed Tests (15)
  ✓ TestCreateAndSaveTimestampAsics                               405 ms
  ✓ TestCreateAndSaveTimestampAsicsMultipleFilesByByteArrays      356 ms
  ✓ TestCreateAndSaveTimestampAsicsMultipleFilesByPaths           349 ms
  ✓ TestCreateAndSaveTimestampAsicsMultipleFilesByStreams         337 ms
  ✓ TestCreateAndSaveTimestampTsr                                 298 ms
  ✓ TestCreateTimestampAndCheckWarning                            610 ms
  ✓ TestCreateTimestampFromDigest                                 319 ms
  ✓ TestCreateTimestampSimple                                     609 ms
  ✓ TestCreateTimestampUsingConfigurationFile                     612 ms
  ✓ TestVerifyTimestampFromDigest                                 303 ms
  ✓ TestVerifyTimestampFromPath                                    16 ms
  ✓ TestVerifyTimestampSimple                                       1 sec
  ✓ TestVerifyTimestampUsingConfiguration                         332 ms
  ✓ TestVerifyWrongTimestamp                                        1 ms
  ✓ TestVerifyWrongTimestampData                                  320 ms
```

Figure 6.1: List of system test results

6.3 Performance testing

All tables in this section show comparison of execution time expressed in milliseconds. Tests to get these results were executed several times and the tables contain the averaged values. The hardware and software specifications of the computer used for the execution of these tests are shown in Table 6.1.

Operating system	Windows 10
Processor	Intel i5-4460 3.2GHz,
Memory	8GB DDR3 1600MHz
Storage	SATA SSD

Table 6.1: Specification of test executing computer

6.3.1 TimestampData

Variables:

- Hash algorithm
- File size
- File count

The performance testing of calculating message digest was done to find out if there is some bad combination of settings that would slow down this process. The variables for this test are the hash algorithm, file size, and file count. For each hash algorithm, there were four test case settings differing in file size and file count. From the data depicted in Table 6.2 it is clear that the stronger a hashing algorithm is, the more time it requires to operate. But all of the calculations were accomplished in a reasonable time. The numbers show that there is a linear growth for adding additional files. This means that there isn't any significant computational demand related to adding more and more files. This test was done by providing the TimestampData with an array of strings expressing the paths to files. This is the worst case scenario because each file needs to be loaded into memory first and then used for calculation. The bottleneck in this situation can be the storage in which these files are located. This is in opposition with providing the library with byte array representation of these files, which can be used directly for calculation.

	1x 1MB file	**1000x 1MB file**	**1x 100MB file**	**10x 100MB file**
MD5	6	2730	431	3011
SHA1	8	3379	361	3624
SHA256	14	12598	1384	12759
SHA512	52	46407	4662	46600

Table 6.2: TimestampData hashing performance table (time in milliseconds)

Figure 6.2 shows the comparison of hash calculation performance. The MD5 algorithm is used as a base for this comparison, and the scale shows multiples of increase in time execution. The figure illustrates various facts. First one is that the performance of using MD5 and SHA1 is very similar. The other one is that the results are ordered by total file size and not by total file count. It is most evident when looking at the SHA512 algorithm. The topmost and therefore the slowest performance is for thousand files of 1MB size, followed closely by ten files of 100MB size. Then there is a bigger gap going to one file of 100MB, and obviously, the best case is for one file of 1MB size. This means that the execution time of hash calculation will be primarily affected by the total size of files.

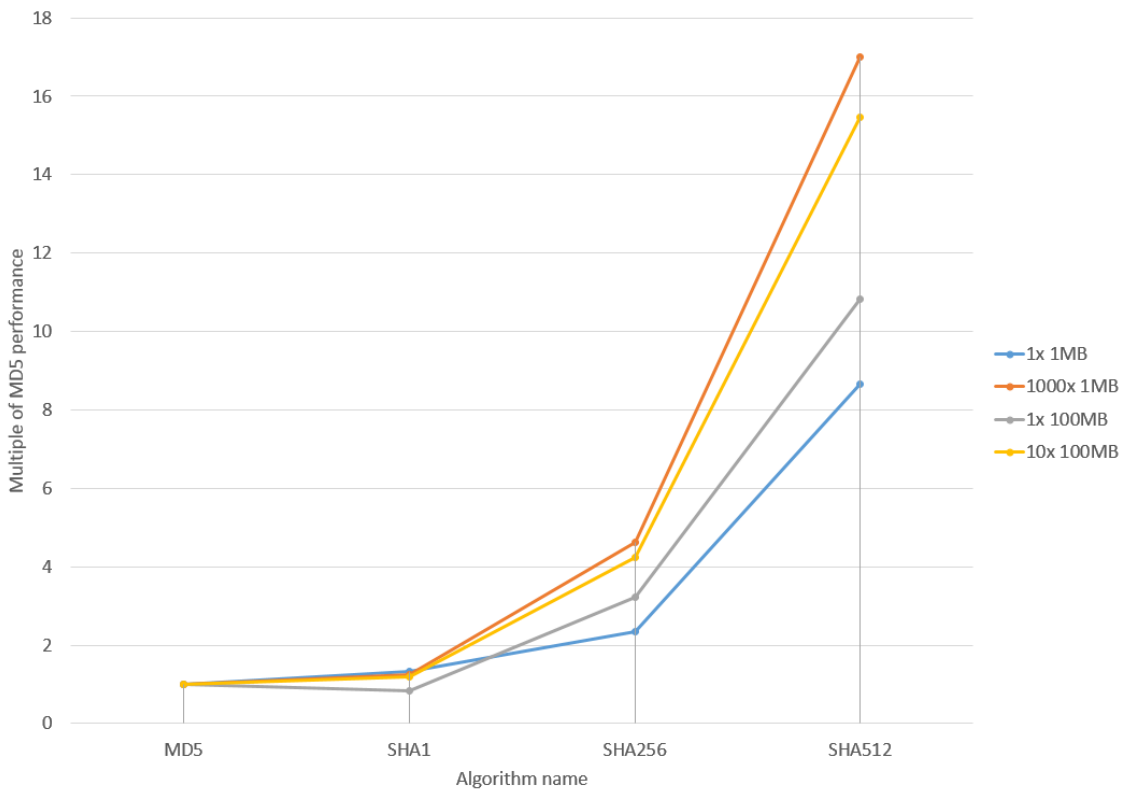


Figure 6.2: TimestampData hashing performance comparison

6.3.2 TimestampVerifier

Variables:

- Timestamp count
- Timestamp validity

Testing the performance of timestamp verification has a different set of variables than the hashing performance. It would be redundant to include different hash algorithms, file sizes, and counts as it is already tested in the previous section. For that reason, the variables for testing timestamp verification are timestamp count and timestamp validity. Timestamp validity means that the verification will result either positively or negatively. The reasons for negative verifications are file integrity violation and invalid certificate.

	Number of timestamps			
	1	**10**	**100**	**1000**
Valid	6	46	448	4110
File integrity violated	5	29	157	1267
Invalid certificate	15	43	391	3933

Table 6.3: TimestampVerifier performance table (time in milliseconds)

The data in Table 6.3 shows the results. It is clear that the shortest times occurred in cases when the file integrity was violated. It is due to verification workflow as it was described in 4.1.2. The worst case scenario is actually a valid timestamp. But even in this scenario, the performance of timestamp verification is around 250 timestamps per second.

Another assurance comes from looking at Figure 6.3. It contains three plots of data from Table 6.3. The graph's vertical scale is logarithmic allowing better demonstration of the linear growth.

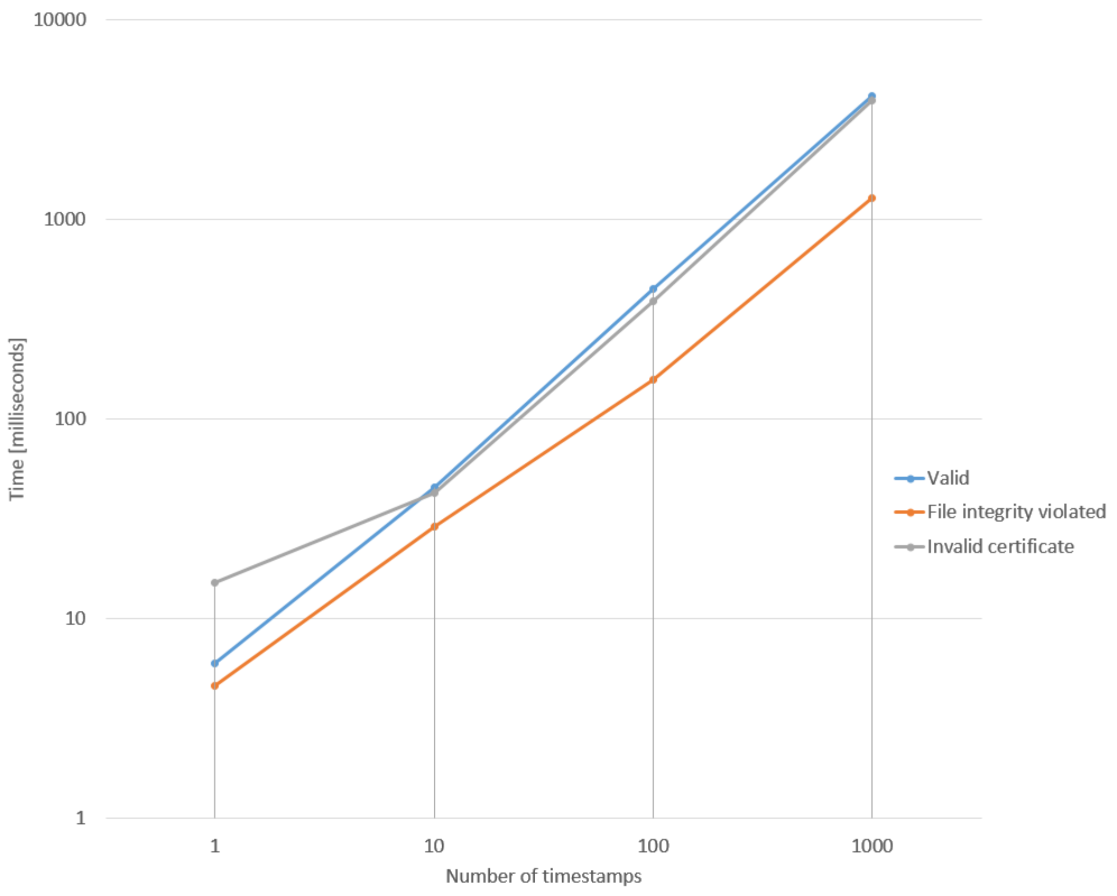


Figure 6.3: TimestampVerifier performance graph with logarithmic scale

6.3.3 TimestampCreator

Variables:

- Timestamp count
- File size
- Output format

There is one other variable, that isn't listed here. It is the TSA used for creating timestamps. Testing the performance of timestamp creation is dependent on the response time of TSA. Because of this, two TSAs were used to compare the results. It is more important to see the comparison of the different set of variables rather than the actual values.

The Graph 6.4 again shows the linear growth of additional timestamps. It means that creating one hundred timestamps takes roughly ten times longer than creating ten timestamps. More interesting fact is the comparison of timestamp output formats: TSR and ASIC-S. The difference becomes noticeable for the bigger file. Using ASIC-S results in approximately three times slower execution time. This is understandable as it is a zip container and requires to be run through a compression algorithm. Another fact coming from the graph is that the total file size is no longer the main factor. The last column, representing total file size of 1000MB, results in the same and even faster execution time than the third column, where the total file size is only 100MB. This is due to the TSA response time and the fact that timestamp creation includes timestamp verification (except for additional hash calculation). Therefore ten timestamps need only ten verifications, whereas 100 timestamps require verification to be run 100 times. This finding inspired another test but instead of creating a hundred timestamps of one file, there was only one timestamp for 100 files together and so on. This configuration eliminated previous effects, and total file size became the main factor again. The conclusion of this experiments is that users should consider grouping files together, instead of timestamping them separately, to achieve better performance.

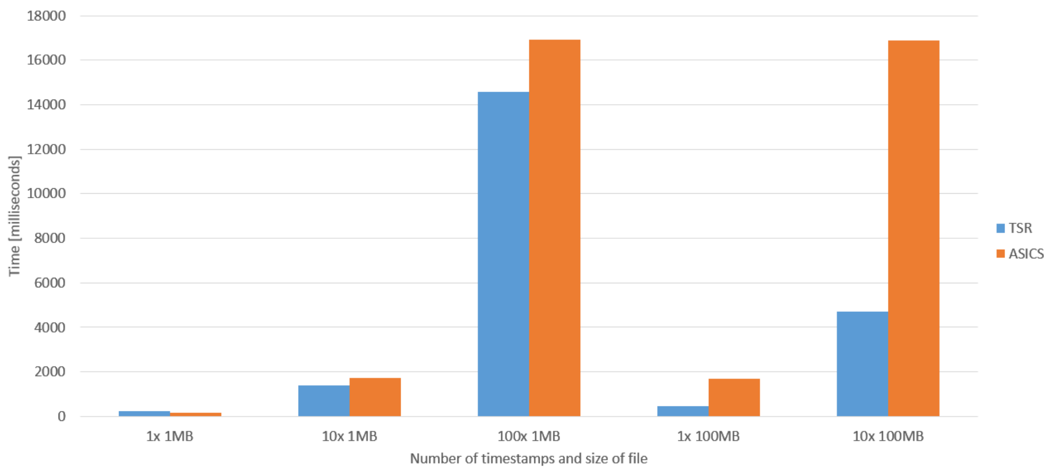


Figure 6.4: TimestampCreator performance graph

Another form of performance test was conducted in order to see how the library behaves under continuous load. The setup of this test was following:

- Duration: 20 minutes
- Data: rotation of 3 different files, provided as a file path (2x png, 1x txt)
- Hash algorithm: SHA1
- Output format: TSR

The results of this test are shown in Table 6.4. The library successfully completed this test without any performance issues.

Total number of timestamps	8088
Timestamps per second	6.74
Minimal creation time of a timestamp	116 ms
Maximal creation time of a timestamp	682 ms
Average creation time of a timestamp	148 ms

Table 6.4: TimestampCreator 20 minutes load performance test results

Chapter 7

Evaluation and future

7.1 Summary of the work

The work on this project went through several stages. The first stage was to analyze the current situation. Different timestamping schemes were considered and analyzed in section 2.1.1. Decentralized timestamping using Bitcoin sounded promising. The idea behind it is to use Bitcoin's block chain as a proof of time instead of traditional ways like using TSA. The concept is relatively new but it has some flaws, that made it unsuitable for this project. In the end, the PKI-based scheme was chosen as it is the most widely used, it is very well described in the literature and fairly straightforward to use.

It turned out that there aren't many solutions for managing timestamps programmatically. There was one option, TimeStampClient, which provides a library for creating timestamps. However, it is not very flexible to input and doesn't do enough timestamp verification. The result of this analysis was that a library offering flexible ways of creating timestamps and providing thorough timestamp verification is not available; thus it makes sense to create such library.

The second stage of work was the actual development and testing. Processes of timestamp creation and verification were separated. Class responsible for holding and processing the data for timestamping was created. The development was accompanied by testing thanks to following the test driven development principle. The developing code was constantly versioned using Git[1]. The development finished by system and performance testing.

7.2 Future

This project is going to be further developed as an open source software. It means that everyone can use it and change it for his needs. Additional features may be included into the library based on user's suggestions.

The source code is available on GitHub[2] and NuGet[3]. NuGet is the package manager

[1]<`https://gitlab.fel.cvut.cz`>
[2]<`https://github.com/honsdomi/AbsoluteTimestamp`>
[3]<`https://www.nuget.org/packages/Honsdomi.AbsoluteTimestamp/`>

for the Microsoft development platform including .NET. The NuGet client tools provide the ability to produce and consume packages. The NuGet Gallery is the central package repository used by all package authors and consumers[14].

Chapter 8

Conclusion

I have analyzed the current situation and came to the conclusion that there is a gap which can be filled by a library providing flexible tools for creating and verifying timestamps. The design phase separated the processes of timestamp creating and verification. Also, the library interface has been defined.

The result of implementation is a fully functional C# library for creating and verifying trusted timestamps. Timestamps are created by hashing provided files and sending requests to a time stamping authority. The authority responds with a timestamp, which can be stored and verified. This process follows the RFC 3161 time-stamp protocol which is based on public key infrastructure. The timestamp verification comprises of file integrity verification and certificate check.

The code was carefully tested throughout the implementation. System and performance testing were conducted after the implementation finished.

The library is publicly available as open source software for everyone to use. Integration into other projects is done easily, thanks to NuGet package manager.

Bibliography

[1] ADAMS, C. – LLOYD, S. *Understanding PKI: concepts, standards, and deployment considerations.* Addison-Wesley Professional, 2003.

[2] ADAMS, D. C. – PINKAS, D. Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP). RFC 3161, August 2001. Available from: <`https://rfc-editor.org/rfc/rfc3161.txt`>.

[3] BECK, K. *Test-driven development: by example.* Addison-Wesley Professional, 2003.

[4] BLACK, R. *Managing the Testing Process.* Wiley Publishing, Inc, 2009.

[5] BUDIŠ, P. *Elektronický podpis a jeho aplikace v praxi.* 1. Kollárovo nám. 698/7, Olomouc : Anag, 2008. ISBN 978-80-7263-465-1.

[6] European Telecommunications Standards Institute. Electronic Signatures and Infrastructures (ESI);Associated Signature Containers (ASiC). *Electronic Signatures and Infrastructures (ESI).*

[7] FARKAS, S. *Using the Hashing Transforms (or How Do I Compute a Hash Block by Block* [online]. 2004. [cit. 10. 4. 2017]. Available from: <`https://blogs.msdn.microsoft.com/shawnfa/2004/02/20/using-the-hashing-transforms-or-how-do-i-compute-a-hash-block-by-block/`>.

[8] FOWLER, M. *FluentInterface* [online]. 2005. [cit. 8. 4. 2017]. Available from: <`https://martinfowler.com/bliki/FluentInterface.html`>.

[9] GIPP, B. – MEUSCHKE, N. – GERNANDT, A. Decentralized Trusted Timestamping using the Crypto Currency Bitcoin. Newport Beach, CA, USA, March 24 - 27, 2015. Available from: <`http://ischools.org/the-iconference/`>.

[10] Legion of the Bouncy Castle Inc. *The Legion of the Bouncy Castle* [online]. 2017. [cit. 3. 4. 2017]. Available from: <`http://www.bouncycastle.org/`>.

[11] NAKOV, S. Fundamentals of Computer Programming with C# (The Bulgarian C# Programming Book). 2013.

[12] .NET Documentation. *AggregateException Class* [online]. 2017. [cit. 6. 5. 2017]. Available from: <`https://msdn.microsoft.com/en-us/library/system.aggregateexception(v=vs.110).aspx`>.

[13] Net security training. *What is a Public Key Infrastructure?* [online]. 2016. [cit. 26. 3. 2017]. Available from: <https://web.archive.org/web/20161009124244/http://www.net-security-training.co.uk/what-is-a-public-key-infrastructure/>.

[14] Nuget Team. *What is NuGet?* [online]. 2017. [cit. 7. 5. 2017]. Available from: <https://www.nuget.org/>.

[15] Original by Bart Van den Bosch, vector by Tsuruya. *Trusted timestamping* [online]. 2012. [cit. 2. 5. 2017]. Available from: <https://en.wikipedia.org/wiki/Trusted_timestamping#/media/File:Trusted_timestamping.svg>.

[16] PAAR, C. – PELZL, J. *Understanding cryptography: a textbook for students and practitioners*. Heidelberg : Springer, 2010. ISBN 3642041000;9783642041006;.

[17] PETERKA, J. *Báječný svět elektronického podpisu*. 1. Americká 23, 120 00 Praha 2 : CZ.NIC, z. s. p. o., 2011. ISBN 978-80-904248-3-8.

[18] RistoLaanoja. *Linked timestamping* [online]. 2009. [cit. 2. 5. 2017]. Available from: <https://en.wikipedia.org/wiki/Linked_timestamping#/media/File:Hashlink_timestamping.svg>.

[19] SEGUIN, K. *Foundations of Programming – pt 8 – Back to Basics: Exceptions* [online]. 2008. [cit. 14. 4. 2017]. Available from: <http://codebetter.com/karlseguin/2008/05/30/foundations-of-programming-pt-8-back-to-basics-exceptions/>.

[20] Skulvis. *Cross-Certification* [online]. 2007. [cit. 2. 5. 2017]. Available from: <https://en.wikipedia.org/wiki/Transient-key_cryptography#/media/File:Cross-Certification.png>.

[21] Wikipedia contributors. *Linked timestamping* [online]. 2017. [cit. 26. 3. 2017]. Available from: <https://en.wikipedia.org/wiki/Linked_timestamping>.

[22] Wikipedia contributors. *Message authentication code* [online]. 2017. [cit. 26. 3. 2017]. Available from: <https://en.wikipedia.org/wiki/Message_authentication_code>.

[23] Wikipedia contributors. *Transient-key cryptography* [online]. 2017. [cit. 26. 3. 2017]. Available from: <https://en.wikipedia.org/wiki/Transient-key_cryptography>.

[24] X9 Accredited Standards Committee. *New Standard Provides Time Stamp Security* [online]. 2005. [cit. 2. 4. 2017]. Available from: <https://web.archive.org/web/20070206014938/http://www.x9.org:80/news/pr050701>.

Appendix A

CD contents

```
CD
├── library
│   ├── TimestampLibrary
│   │   ├── ITimestampCreator.cs
│   │   ├── ITimestampVerifier.cs
│   │   ├── TimestampCreator.cs
│   │   ├── TimestampData.cs
│   │   ├── TimestampException.cs
│   │   ├── TimestampObject.cs
│   │   ├── TimestampVerifier.cs
│   │   ├── Utils.cs
│   │   └── Enums
│   │       ├── HashAlgorithm.cs
│   │       └── OutputFormat.cs
│   └── TimestampLibraryTests
│       ├── TimestampCreatorTest.cs
│       ├── TimestampDataTest.cs
│       └── TimestampVerifierTest.cs
└── text
    ├── source
    ├── honsdomi.pdf
    └── honsdomi.tsr
```