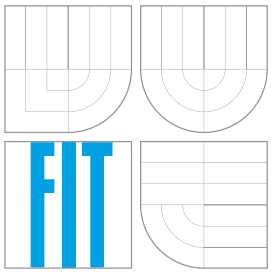
BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ



FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

THE CRYPTOGRAPHIC PROTOCOL FOR MANAGEMENT AND APPROVAL OF DOCUMENT VERSIONS

KRYPTOGRAFICKÝ PROTOKOL PRO SPRÁVU A SCHVALOVÁNÍ VERZÍ DOKUMENTŮ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE

Bc. PETER LACKO

SUPERVISOR
VEDOUCÍ PRÁCE

Ing. IVAN HOMOLIAK

BRNO 2016

Abstract

This work deals with design and implementation of the system for document management and versioning. The first part contains description of related work. In the second part, information security concepts and security model, upon which application is build, is discussed. Third part contains description of designed system and its typical use in a form of sequence diagram. Fourth part introduces cryptographic protocol used in this work. Next follows the description of implementation and security analysis of developed system. The output of this work is cryptographic protocol for document management and versioning, and client-server application implementing this protocol.

Abstrakt

Tato práce se zabývá návrhem a implementací systému pro spravování a verzování elektronických dokumentů. V první části jsou popsány aplikace se stejným nebo podobným zaměřením. Druhá část obsahuje popis bezpečnosti informace a představuje bezpečnostní model nad kterým je aplikace vybudována. Třetí část popisuje navržený systém a jeho typické použití formou sekvenčního diagramu. Ve čtvrté části je představen kryptografický protokol použitý v této práci, postavený na kryptografii veřejných klíčů. Dále následuje popis implementace a analýza bezpečnosti navrženého systému. Výstupem práce je kryptografický protokol pro správu a verzování dokumentů a aplikace typu klient-server implementující tento protokol.

Keywords

document management, versioning, cryptography, PKI, RMIAS

Klíčová slova

správa dokumentů, verzování, kryptografie, PKI, RMIAS

Reference

LACKO, Peter. *The Cryptographic Protocol for Management and Approval of Document Versions.* Brno, 2016. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Homoliak Ivan.

The Cryptographic Protocol for Management and Approval of Document Versions

Declaration

I hereby certify, that this thesis is presentation of my original research work, done under guidance of Ing. Ivana Homoliak.

.....
Peter Lacko
May 27, 2016

Acknowledgements

I thank my master thesis supervisor, Ing. Ivan Homoliak, for his support and guidance through whole development of this work.

Contents

Chapter 1

Introduction

In almost every working industry people need to communicate with each other, share their knowledge, collaborate on projects or exchange information regardless of they are located in the same office or at opposite sides of the world. In many cases it is done over the internet and this form of the communication is growing continuously. E-mail, VoIP services, instant messaging or social networks are some of most popular ways. For more specific usage like multimedia sharing, software development or exchange of an arbitrary data, powerful specialized systems has been developed. Some of them are utilized for collaboration on contract documents where security plays a crucial role. The typical example of this group are legal documents.

Many online solutions are currently available and in use (not only) for collaboration on legally binding documents[1]. However, many of them do not meet security requirements necessary for documents of such importance. Some of them do not provide digital signature functionality or two-factor authentication, lack effective file and user permission management, document versioning, history tracking or quick document comparison. Another common way for collaboration on legal documents is sending their printed copies to other parties of contract by mail or personal delivery, both of which can be rather slow and expensive. As a result, sensitive documents are usually either vulnerable to information disclosure by unprivileged person or their management is chaotic or they consume too many resources (manpower as well as time and money).

The goal of this thesis is to design the cryptographic protocol for secure document exchange and signing as well as development of the application implementing required functionality. The application will contain simple but feature-rich interface, containing support for document management, versioning and signing. The system could find its application in law firms, real estate agencies or in other companies where circumstances require an agreement of many parties on single document or set of documents.

The text of this work is structured as follows: Chapter 2 gives an overview of related projects. Chapter 3 discusses general security concepts. In Chapter 4, core features of the application are described in detail with use case provided. Chapter 5 provides overview of the public key cryptography which is crucial to proposed cryptographic protocol, proposes a scheme of public key infrastructure, and describes protocol more formally. Chapters 6 and 7 discuss details of implementation and analyze system's security. Chapter 8 discuss possible extensions and improvements, and finally Chapter 9 concludes the thesis.

[1] Legally binding document is, according to http://thelawdictionary.org, a lawful action (e.g. an agreement) consciously agreed by two or more entities, establishing lawful accountability.

Chapter 2

Related Work

In recent years, many systems providing content management, document versioning, collaboration and signing have emerged. Although, there is no official categorization of these systems, for the purpose of this work, we divide them by type of content they manage into: textual vs. binary data, documents, multimedia files, etc.; by targeted users: from personal use through small businesses to big corporations, and by type of storage they provide to centralized (cloud/remote server) or local storage. Based on a functionality they provide, we can further divide them into four categories: systems for legal content management; Version Control Systems (VCS); general cloud solutions; and document signing and managing systems. Selected software from each category is reviewed with regards to a functionality of a system being developed.

For quick review, Table 2 summarizes some of most important features of described application, in regard to designed application.

2.1 Systems for Secure Document Sharing and Management

This category contains systems designed specifically for legal content management – `Closing Table` and `Effacts` – and general purpose systems applicable also in other areas – `Fluix`, `WatchDox`, `ExperDocs` and `DocuXplorer`.

Closing Table

`Closing Table` [27] is commercial, document and transaction management system primarily used for document sharing, negotiation and archiving. `Closing Table` is aimed at lawyers/attorneys, brokers and other real estate and business firms. It supports organization of documents into deals containing list of authorized users for current deal (no per-document access granting is possible) and list of documents with further categorization (i.e. leases, loans, closing documents), document storing, versioning and archiving. Users are able to upload new documents into deal, update existing documents (by uploading new versions), sign documents using digital signatures, comment on documents and their revisions and review history of particular document. Personal web page is generated upon user's registration for easy linking person to his account.

Website does not provide explicit information on underlying technologies used, nor on features as two factor authentication, comparing different versions of same document, PDF generation, searching on documents or email/sms notifications and since no trial version is available, there's no way to verify their presence.

Name	Digital Signatures	Versioning	Multi-Factor Authentication	Collaboration History	Document Diff	End-To-End Encryption
Closing Table	✓	✓	?	✓	?	?
Fluix	✓	?	✓	✓	?	✗
WatchDox	?	?	✓	✓	?	✓
Effacts	✓[1]	?	✓	?	?	?
DocuXplorer	?	✓	?	✓	?	?
ExperDocs	✗	✗	✗	✗	✗	✗
DocuSign	✓	✗	✓	✗	✗	✗
E-Sign	✓	✗	✓	✗	?	?
DocVerify	✓	?	✓	?	?	✓
Open eSignForms	✓[2]	✓	✗	✓	✗	✗
Git	✓	✓	✓	✓	✓	✗
Subversion	✗	✓	✗	✓	✓	✗
Google Docs	✗	✓	✓	✓	✓	✗
Office365	✓	✓	✓	✓	✓	✗

Table 2.1: Summary of core features support in described applications.

[1] In form of integration with DocuSign only.
[2] Only one key-pair per deployment.

Fluix

`Fluix` [14] is cloud based, collaborative document management software with focus on portable devices like tablets and smartphones. It is designed for construction industry, but examples of its use cases extend far beyond it. `Fluix` provides functionality for a document storing and sharing, collecting signatures for them, automating workflow, viewing document of (almost) any type, templates of documents, putting annotations, tracking changes and even modifying PDF files. Documents can be signed by digital signature as well as via tablet or smartphone with pen or finger. When signing by hand, the software keeps also information about dynamics of movement so in the case of need, a signature can be later forensically inspected. The software also supports two factor authentication for signing into the system by providing an user name with password together with unique pin code sent to a mobile phone.

`Fluix` does not have support for document versioning and comparison, only one user at time can be requested to sign a document and user interface is less intuitive in web version. Software is closed source, therefore no closer technical details of underlying system and protocol are available.

WatchDox

`WatchDox`[4] is a data-centric security solution for a document sharing and synchronization from BlackBerry. It supports basic password and multi-factor authentication as well as integration with MS Active Directory and other identity management systems. Each document is encrypted with unique key using 256-bit AES encryption with possibility to use hardware security module[1]. Access rights to a document can be set individually for a user, group, or based on a role. User can annotate and edit documents, put watermarks, set expiration or even wipe out a document from all locations (i.e. also after sharing with other users). Activity of each document is tracked as well, providing information about who, when and where accessed a document, and what action he made. Moreover, advanced digital rights management can be applied to a document, restricting how documents are used, e.g. forwarding, printing or downloading. For its public cloud services, `WatchDox` utilizes Amazon Web Services (AWS) but provides also others, on-premise deployment options. `WatchDox` makes its services available through a web browser or as a mobile application on all platforms and allows integration with another cloud solutions, primarily MS SharePoint. It is also compliant with HIPAA/HITECH[2] healthcare security program, ITAR/EAR[3] and FedRAMP[4] certified.

Website does not provide information about features as a document signing, versioning or comparison.

Effacts

`Effacts`[12] is the plug-and-play legal management software for lean legal departments to manage contracts, entities, claims, compliance and more. System provides tools for a document management and collaboration with external parties. Every document is encrypted, activity of users and documents are logged, custom system notifications are supported. As extra tools are offered integration with *DocuSign* and advanced authentication methods in form of Single Sign On (SSO)[5] and two factor authentication. Specialized modules for easing contract, claim, entity, compliance and meeting management are available as well. Service runs by default in `Effact's` private cloud but can also be deployed on a custom server. All datacenters are SSAE 16 (SOC1)/ISAE 3402 Type II[6] audited.

No information about document versioning, commenting, user management and access control is available.

DocuXplorer

`DocuXplorer` [10] is another feature-rich system for document management accessible through both web interface and as an standalone application. System supports tracking of document versions, document indexing and searching (including binary files as PDF), OCR for scanned documents, text extraction from PDF and is also highly integrated with MS Office

[1] Hardware security module is a hardware device for safe storing, processing and management of cryptographic keys.

[2] http://www.hhs.gov/hipaa/for-professionals/special-topics/HITECH-act-enforcement-interim-final-rule/index.html

[3] http://www.pmddtc.state.gov/regulations_laws/itar.html

[4] http://www.fedramp.gov/

[5] Using SSO, user is able to gain access to connected, but different systems with single ID and password.

[6] http://ssae16.com

suite. DocuXplorer support organization of documents into cabinets, drawers, folders and index fields. User or group permissions can be set by administrator on any of these levels.

DocuXplorer does not provide any information about document commenting, signing and two factor authentication.

ExperDocs

ExperDocs [2] is another tool for a document management and collaboration accessible through both web interface and stand-alone desktop application. ExperDocs' main features are advanced file organization with tagging feature, sharing, publishing and collaboration on documents, users' permission control, commenting on arbitrary file, encryption on server and workstation.

Software however miss many important features like document versioning, comparing and signing and two factor authentication.

2.2 Document Signing and Management Systems

Since digital signatures are utilized in many areas, numerous systems for dealing with electronic signatures in documents or other electronic content have been developed. Two of the most popular solutions are described here – DocuSign and E-Sign – together with DocVerify, and the only open source software described here, Open eSignForms.

DocuSign

Most widely used software in this category is DocuSign [9] allowing users to simply prepare and sign their documents by pasting a signature – prepared hand-written like signature (i.e. stamp) – onto specified place in a document and then digitally signing hash of a document with user's private key. Signature on single or multiple documents can be requested from an arbitrary number of users who can either sign, postpone or decline with clarification. When sending document for a signature, sender can additionally verify recipient's identity by asking for the secret code they communicate with each other, without use of the application. For example, by entering a secret code received in SMS, by requesting recipient to answer the call and providing a secret code or to answer questions about themselves based on data available in public records.

No document versioning is available as well as no document comparison. Commenting is available in a form of email. DocuSign is, among others, ISO 27001 [16] certified and xDTM [59] compliant. As DocuSign is also closed software, no detail technical specification is available.

E-Sign

The goal of E-Sign [11] is to provide easy to use and intuitive service for collecting signatures on an arbitrary electronic document or any other type of a file. During registration, an identity is verified via e-mail. After successful registration, user is immediately able to start his work flow. A user can import a document from his computer or cloud storage and then digitally sign it by himself or request signature from other users. After signing a document, the system generates a unique QR Code which is attached to a document. A QR code

ensures quick access to a document and easier signature verification[7]. Upon confirming signature, the application collects information such as date and time, IP address, location and web browser & system information.

The application of `E-sign` is available via web browser and via mobile apps for iOS and Android. It provides simple document management without organizing into hierarchical structures (i.e. folders) and no document versioning is possible either.

Accounts in both applications – `DocuSign` and `E-sign` – are created for single users, so documents can only be shared in means of sending them between each other.

DocVerify

`DocVerify` is another system for collecting electronic signatures on a single document or batch of documents at once. System supports multi-factor authentication and authentication over phone, where phone number must be verified prior its usage. During the call, user's voice is recorded, and fingerprinted record is later attached to the document. In cases where identity verification is necessary, DocVerify utilizes knowledge based authentication[8] over third party services.

Each page of a signed document is stamped with a date and time the document was created, page number, unique bar code, and dynamically embedded watermark. Digital signatures are FIPS 186-4 [39] compliant and in accordance to ISO 14533 [17], to assure long term authenticity and validity of signatures. DocVerify utilizes PKI in accordance to ISO/IEC 9594-8 and ITU-T X.509 [15, 53] standards. Secure document time-stamping as described in `RFC 3161` [1] is used as well, together with sequential time-stamping, making forged document injection into the system extremely difficult. Moreover, all actions are recorded in a tamper-proof audit trail.

Every document is encrypted with it's own 256 bit AES key, symmetric keys are then encrypted with 512 bit user's private key (supposedly Elliptic Curve based). System is also ISO/IEC 27001 certified and SSAE16 audited.

Enterprise version provides more functionality, such as document templating, integration into custom application or notary features.

Open eSignForms

Another tool Open `eSignForms` for document signing is not so widely used, but rich in functionality [60]. It is an open source software developed by Yozons, Inc and is primarily aimed for document management, templating, versioning and signing. It contains built-in text editor with support for pre-made forms and form editor. Documents can be in a testing and production stage, i.e. documents with work in progress and their final versions. Documents are organized into the libraries and access to them is granted on per user basis. Users do not have their own key pair or certificate. Instead, each document is signed by private key of `Open eSignForm` installation containing IP address of signer, timestamp, user's name, email and ID. Software supports only work with structured documents (in XML format) of built-in text editor. Structured documents can be generated from forms or PDF files of original documents.

[7] After scanning a QR Code, a user is taken to the website, where he can verify signature validity.
[8] Authentication based on knowledge of secret personal information

2.3 Version Control Systems

Version Control Systems [42] are systems primarily developed and used for a software source control, versioning and collaboration, but can be also used for collaboration in any other type of project. These systems usually provide managing of text files, but also support managing of binary files. We can split these systems into two main categories, based on scheme they use for content management to centralized (client-server) and decentralized (distributed).

Centralized VCS

Centralized VCS use a concept of central repository holding main copy of a project together with project's whole history and serves as a reference copy for all users. Users can push their files or changes into existing files (also called changesets) and other users can view those changes and incorporate them into their own project repository. Some of the most popular centralized VCS are `SubVersion` (SVN) [18], `Concurrent Versioning System` (CVS) [21] or `Team Foundation Server` [36].

Distributed VCS

As opposed to a centralized Version Control Systems, in Distributed VCS there is no central repository, since all repositories hold full history of a project (though one repository can be chosen as a main, but it is not a rule). Probably two, most popular decentralized VCSs are `Git` [55] and `Mercurial` [32].

All VCS have many features in common. Among the most popular ones belong possibility of branching a project and work in a new branch, viewing differences between different file revisions, going back through history of project or commenting on changes being made. Some systems like Git or Mercurial have support for digital signing of changsets using PKI, so other users can verify their authenticity.

VCS are great tool for a collaboration on various types of projects, but they require a lot of initial learning (have „steep learning curve“) which can discourage, mostly, non technical users from their utilization. They are also inconvenient for use within law companies for absence of built-in support for structured text documents as `OfficeOpenXML`, `Open Document Format` or `Rich Text Format`.

2.4 Cloud Solutions

Another category of systems aimed for a collaboration are general purpose cloud solutions, e.g. Google with its service `Google Docs` or Microsoft with their `SharePoint` which in connection with Microsoft Office suite gives its users powerful tool for content management, document storing, sharing, presenting, project planning, managing etc. MS SharePoint's typical use case is deployment within single company, requiring server farm to install into, and an IT person (or team) to manage such infrastructure. Because of this reason, SharePoint is not suitable for individuals or smaller companies and thus `MS Office 365`, providing most of SharePoint functionality but in cloud, will be described.

Google Docs [23] and **Microsoft Office 365** [35] are very similar in a functionality they offer. They are both cloud services, so users need only web browser to start using them. Both ones offer applications for creating and editing structured text documents,

spreadheets, presentations or to keep track of your notes, versioning of documents with visual diffs (Google Docs only) and restore function. Inserting comments into documents is supported as well, however, commenting on the whole document is not possible. Both applications support adding digital signatures into documents, but only Microsoft Office 365 supports signing files and messages with your private key through `Digital ID`[9] which is provided upon user's request. Document management is very similar to the management of typical file systems with possibility to share documents with other users for a collaboration purpose. Collecting signatures for single document is not built-in in either of them, since it is not intended use case. Both ones support two factor authentication for their services. Transmitted and stored data are encrypted with perfect forward secrecy [24, 34], which means that compromising key in single session does not affect data security in previous or future sessions.

The biggest advantages of described (and similar) cloud solutions are in complexity and variety of tools and functions they provide with 100% portability, since everything is available online. Negotiating on single document version can be, however, cumbersome, since no simple solution for collecting signatures on documents is available.

[9] https://support.office.com/en-us/article/get-a-digital-ID-0eaa0ab9-b8a2-4a7e-828b-9bded6370b7b

Chapter 3

Information Security Concepts

Since desired system for document versioning and signing can contain documents with highly confidential information, providing high information security must be the one of the top priorities regarding system design and implementation.

Chardetseva and Hilton [7] define **Information Security** as a multidisciplinary area of study and professional activity which is concerned with the development and implementation of security countermeasures of all available types (technical, organizational, human-oriented and legal) in order to keep information in all of its locations (within and outside the organization's perimeter) and, consequently, information systems where information is created, processed, stored, transmitted and destructed, free from threats. In other words, Information Security is concerned with protection of information of any kind (i.e. electronic, paper or verbal) and its critical elements including the systems and hardware which use, store and transmit information.

Closely related to the Information Security is **Information Assurance**, defined in similar way [7] as multidisciplinary area of study and professional activity which aims to protect business by reducing risks associated with information and information systems by means of a comprehensive and systematic management of security countermeasures, which is driven by risk analysis and cost-effectiveness. Information Assurance can be seen as a superset of Information Security with inclusion of other fields as well, as depicted in Figure 3.1.

Combined Information Assurance & Security form a knowledge area which incorporates the knowledge acquired from both fields, including all actions directed at keeping information secure as well as the management of these actions.

3.1 CIA Triad

Information Security concepts are historically based on three characteristics of information (also know as three pillars of Information Security) that give value to organizations: confidentiality, integrity and availability – CIA Triad. Many authors put another information characteristic – non-repudiation – to the same level with CIA Triad. CIA Triad and non-repudiation has for long been basis for evaluating and implementing information security regardless of underlying system [58].

Integrity of data is achieved when it is whole, complete and incorrupted. Corruption can occur while information is being stored or transmitted and can be caused by technical issue (corrupted medium, noise in transmission) or intentionally by (un)authorized user. By

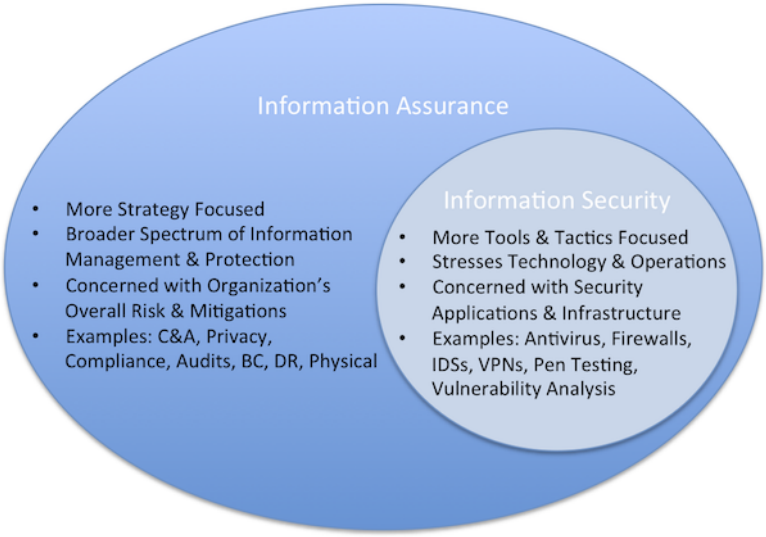


Figure 3.1: Relationship between Information Security and Information Assurance [25].

assuring integrity of information, we can ensure that modification was done by authorized users only, in the way each modification should be, moreover, traceable and revertible. If modification of data by unauthorized user would occur, then the modification could be easily detected. For ensuring data integrity, file is typically distributed with its unique message digest – hash (see Section 5.1 in Chapter 5).

Confidentiality ensures only users with knowledge of a key (i.e. authorized users) are able to access information encrypted by this key. If unauthorized access to data occurs, confidentiality is breached. Example of confidentiality breach is intentional or accidental disclosure of private information or theft, achieved by breaking into system by an unprivileged user. Various methods can be utilized to achieve high information confidentiality. Among most common are data encryption, strong passwords, two factor authentication or even physical data isolation, e.g. by storing data on disconnected device or as a hard copy only. Proper staff training can also highly improve information confidentiality.

Availability means that data are safely stored or transmitted in a way, that infrastructure is robust enough to deal with power outages, hardware failures, users' mistakes as well as with intentional attacks against it. In other words, availability ensures that data are always available upon user request in required (or specified) format. Availability can be best ensured by correct maintenance of deployed hardware and software, storing data redundantly and/or on geographically isolate locations.

Non-repudiation means that user cannot deny action that he has intentionally done before. User can for example sign document and later claim, that his private key or account has been compromised. Non-repudiation in system is difficult to achieve, but if system is well designed, then it can be harder for user to repudiate his own actions. Well design of a system require e.g. timestamping or strong authentication. Another approach to prevent repudiation is by deploying trusted third party (TTP). As a TTP, a forensic specialist to analyze signature, or a notary to witness act of signing can be employed.

3.2 From CIA Triad to IAS Octave

Over time, researchers and security specialist have pointed out [58, 44] that CIA triad is not sufficient anymore to address current trends in the information security and proposed its extension. In 2013 after interviewing many leaders in information security area, Chardetseva and Hilton [6] proposed extension of CIA triad by following four principles:

- **Auditability** – an ability of a system to monitor and log all actions performed by every user as well as by machine. Auditability enables users and administrators to trace arbitrary transaction in the system and perform checks of system behavior.
- **Accountability** – an ability of a system to keep users responsible for their actions. Accountability in a system can be improved by keeping log of users' actions and by providing means to easily trace or search over them when necessary.
- **Autenthicity/Trustworthiness** – an ability of a system to verify identity and establish trust in a third party and information it provides. This can be for example achieved by restricting access to a system to users that are verified and trusted by existing system users.
- **Privacy** – A system should obey privacy legislation and it should enable individuals to control their personal information (user-involvement). User's personal information should be used only for internal purposes and mustn't be exposed without explicit permission.

3.3 Reference Model of Information Assurance & Security

A Reference Model (RM) is an abstract framework for understanding significant relationships among the entities of some environment. It enables the development of specific reference or concrete architectures using consistent standards or specifications supporting that environment. A reference model consists of a minimal set of unifying concepts, axioms and relationships within a particular problem domain, and is independent of specific standards, technologies, implementations, or other concrete details [6].

Reference Model of Information Assurance & Security (RMIAS) described in this work servesss as a conceptual framework for designing and implementing secure cryptographic protocol and application built on it.

The RMIAS is depicted in Figure 3.2 and contains four dimensions:

Information System Security Life Cycle Dimension depicts security in various stages of IS life cycle. RMIAS emphasis the need to address security through all stages of IS development.

Information Taxonomy Dimension describes the nature of information being protected and defines four information attributes that are crucial to correctly estimate and achieve security goals and countermeasures to be taken in order to avoid threats whose goals addresses. *Form of information* can be electronic, paper or verbal. *Information sensitivity* may change over system's life cycle, for example publishing a file which was before used only for internal purposes (e.g. disclosing application's source code). *Information location* varies often over time and may be either controlled – information is under full

control of particular organization; partially controlled – information is physically stored at party having contractual relationship with the organization (e.g. cloud provider, business partner); or uncontrolled – information is not in controlled, nor in partially controlled location. *Information state* defines state of information in every specific moment of its life cycle. These states are creation, transmission, processing, storing and destruction. Security of information should be handled equally at each state.

Information category is defined by combination of attributes form, state, sensitivity and location, and serves as basis for the specification and selection of security goals and countermeasures. If an employee creates an electronic document aimed for use in higher management (sensitivity is internal use only), then this document is within controlled location. If document is accidentally emailed out of organization, then it enters uncontrolled location and since sensitivity remains for internal use only, information falls into dangerous category. For such a case, countermeasures should be planned to prevent information falling into this category (e.g. by warning user when private document is emailed to address not in organization space).

Security Goals Dimension lists security goals applicable to system, where security goal is desirable ability of system to resist specific category of threats. Set of security goals that RMIAS tries to achieve is built upon IAS Octave described above. It should be noted, that this list of security goals is not fixed and may change to reflect future changes in system and possible threats.

Security Countermeasures Dimension contains four types of security countermeasures: organizational, human-oriented, technical and legal. *Technical countermeasures* refer to security aspects of system implemented in software or hardware. Examples are cryptographic features, firewall, antivirus or biometric devices. *Organization countermeasures* are meant to be an administrative activities which build secure environment, e.g. following a security strategy, procedures, best practices, physical security etc. *Human-oriented countermeasures* are intended to explain rationale behind given security instructions and relates to users' education, awareness, motivation, ethics etc. *Legal countermeasures* refer to use of legislation for the purpose of information protection, for example to prove information ownership, for enforcement of application of legal agreement over information, or copyright laws.

Relationship Between RMIAS Dimensions

Figure 3.2 depicts RMIAS and relationship between its dimensions. Beginning from the top left quadrant we first define current stage of Security Development Life Cycle. Then, we have to consider every category of information relevant to our system followed by information cataloguing, which together with prioritizing goals, contributes to higher completeness of Information Security Policy Document (ISPD[1]). Security goals are then prioritised for each information category which is based on the conducted risk analysis. Selection of security countermeasures is then performed with emphasis on cost-effectiveness and efficiency. Identified countermeasures should be then traced with consistency throughout all stages of the security life cycle. Described model should be used in iterations for each stage of the life cycle.

[1]ISPD is document containing rules and guidelines that must be followed to meet security requirements

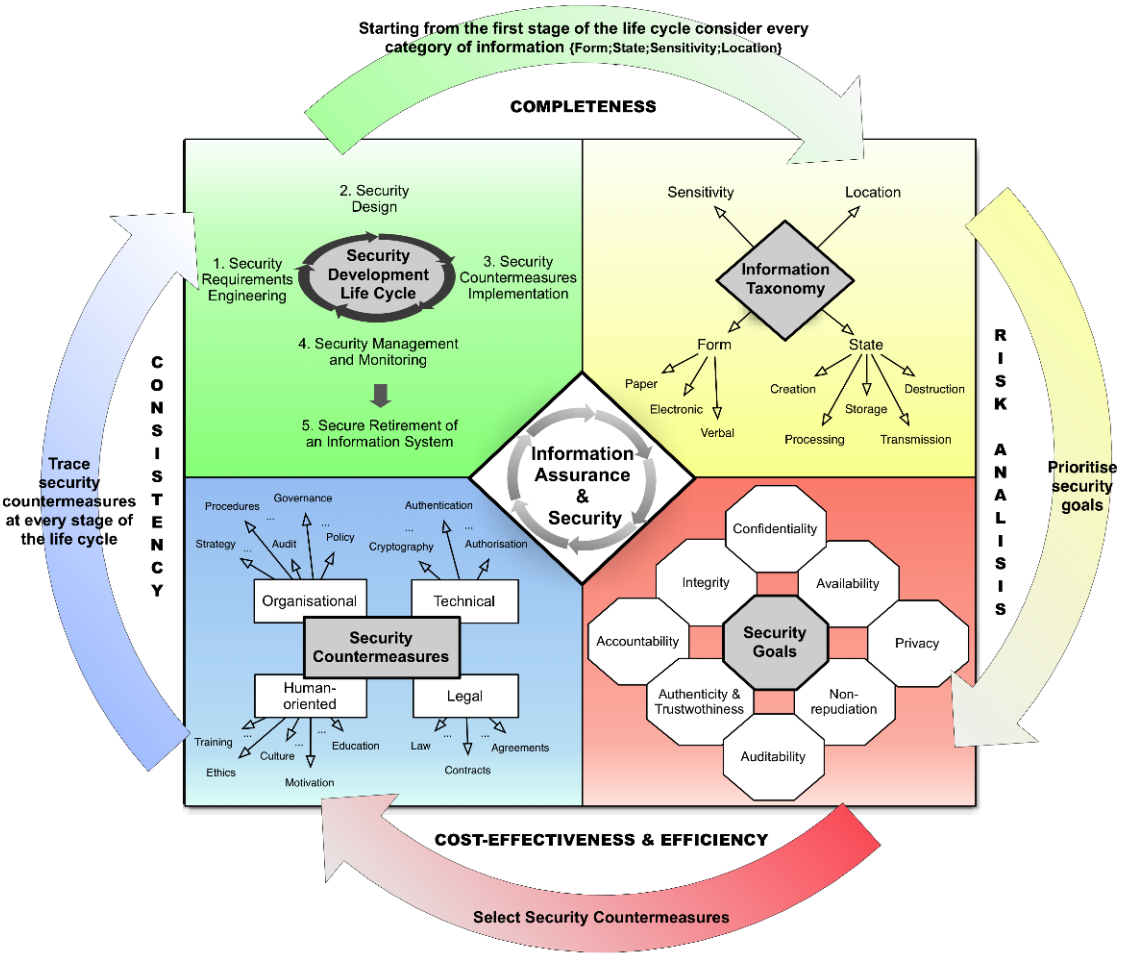


Figure 3.2: Reference Model of Information Assurance & Security [6].

Chapter 4

Application Design

This chapter acquaints reader with core feature requirements for designed application and a typical use case of the system in form of a sequence diagram.

Application is designed as a client-server one, where a client, represented by web browser, will communicate with server over a secure channel. All the sensitive information will be processed on the client side, such as key generation, document encryption or signing. Server will serve as a storage for documents, signatures and other necessary data, and as a user management system with additional functionality for comfortable collaboration on documents.

For better understanding of the system design, it is necessary to introduce three core entities present in the system and relationships among them: organizations, users and documents. Organization is in the system created by system administrator, who will also create[1] it's first user – Organization Administrator (OA). OA can then invite more users to the system and assign them arbitrary permissions. OA can also assign permissions to existing users of the system so they can participate in collaboration as well. Organization can have one or more administrators (thus at least one user). User can be bound to the zero or more organizations. Every document is bound to exactly one organization and can not be transferred to another. Every user with bonding to an organization, can access any document bound to this organization, if he has sufficient permissions to do so. User permissions are described in greater detail in section User Management later in this chapter.

As the project is in early stage, the list of features described here may change and further develop over time.

4.1 Document Management

Documents will be organized in a typical hierarchical structure, well known from file systems, allowing arbitrary depth for directory nesting. Every document in the system will be encrypted with unique symmetric key, thus providing end-to-end encryption. This key will be shared between all document's collaborators, encrypted by user's public key. No other user, not even a system administrator will be able to access document in clear-text, unless this feature is requested by a customer (e.g. to prevent loss of a document, when only single user has access to it, and he forgets his key password). When user is removed from collaboration on a document, his access to document is restricted and document key encrypted by his public key is removed as well.

[1] By creating and sending an invitation.

Till the end of this section we will assume that user is authorized for every action taken, unless not stated otherwise.

Versioning and History. Negotiation of each document will run in iterations, where each time new document version is uploaded, new iteration will start. In single iteration, users can either sign or comment current version of a document. Every document can be in one of three, non overlapping phases: `In Progress`, `Approved` or `Rejected`. When new document is uploaded to the server, it is automatically labeled as In Progress. When document reaches sufficient number of signatures (or other criterion is met), the negotiation is over, current revision is marked as final and document as Approved. If document is no longer required, owner, or other user with sufficient permissions, can mark it as Rejected, and optionally also remove it from the system. Document can also be archived, so it will no longer appear in the main directory structure, but will still be available to review in the archive.

To prevent simultaneous modifications of a document by multiple users, user who want to modify it will be able to temporarily lock it for updates from other users. Lock will be visible to other users as well, to indicate that someone else is working on it. Lock can be removed only by lock's owner or after its expiration.

Users will also be able to comment individual versions of document, so document related communication can be kept on single place.

For each document, either in process of negotiation, approved, or rejected, the system will keep document's history, representing all revisions of each document, list of contributors and signatures, comments and all relevant metadata (time-stamps, access list and others). History will be available for a review to each authorized user.

Signing and Signature Verification. Each user upon registration into the system, receives his personal certificate which enables him to sign latest revision of any document he has permission to sign. User may as well be marked as a mandatory signatory for a document, which in effect causes, that document can not be marked as Approved untill it is not signed by all mandatory signatories. User consents with content of document by signing it. It goes without saying that all actions are transparent to a user, so he does not have to deal with a key management at all.

Notifications. To speed up the process of negotiation, a notification will be sent to a user after specific event occurs, over channel of user's preference. For example, when document has been signed or commented, user has been invited for collaboration, document has been approved or rejected, and many others.

4.2 User Management

For the participation in a document negotiation process, it is necessary for a user to have account in the system and sufficient access permissions for document.

User Permissions. When user is invited to collaborate to an organization, default set of permissions is assigned to him. These permissions will be applied to every new document that appears in directory structure. Permissions can however be changed at any time by one of organiztion's administrators or overwritten on per-document or per-directory basis.

Set of permissions specifying actions that user can perform on document is as follows:

- **View** – user can only access documents for viewing;
- **Comment** – user can view and comment documents;
- **Sign** – user can view and sign documents;
- **Modify** – user can view and upload new version of document;
- **Remove** – user can view and remove document from the system.

Permissions can be arbitrarily mixed, but it is obvious, that some combinations doesn't have practical use.

For more practical access control, Role Based Access Control could be deployed, this is however not supported by current design.

Apart from document permissions, user can also be allowed to invite new user for collaboration. If user is labeled as an administrator, he can perform any action allowed by design, without restrictions. Only administrator has power to modify default user's permissions. Per-document or per-directory permissions can be modified by its owner as well.

User Authentication. Simple authentication via user name and password is not sufficient for such sensitive information as a legal content. Therefore, additional form of authentication is necessary. On each sign in to the system, a text message with a code will be sent to a user's phone. Received code together with user name and password will be used to verify his identity by two factor authentication (knowledge, possession). Except login password, user is also required to remember his key password, i.e. password by which his private key is encrypted and stored on the server. On user registration, identity of a user must be verified even more thoroughly – through a third trusted party.

User Registration. For a user being able to register into the system, he receives an invitation email containing URL to registration form with pre-filled information. To access this form, user must first enter secret code he receives to his mobile phone in SMS. To ensure that only user with valid code can perform registration, registration form contains hidden field signed by server, that is checked after submitting the form. User is then prompted to enter additional information (full name, address), login password and key password. After finishing the registration process, user is able to sign in to the system. Registration process is also depicted on Figure 4.1.

Password Loss and Recovery. Situations, when user forgets his password are very common. Mechanism, that would allow password recovery, is thus necessary. Similarly to registration, login password can be recovered by sending an email to the user (and secret code in SMS), containing URL to password recovery form. After confirmation, new password will be set for the user.

If user forgets his key password, it is not possible to recover it, and new certificate must be issued for him. Again, email containing URL for new certificate request (and secret code in SMS), will be sent to the user, and after completion he will receive new certificate. At this point, user doesn't have access to any document, since no one has encrypted document keys of documents he is supposed to access, with his public key. It is thus necessary to ask

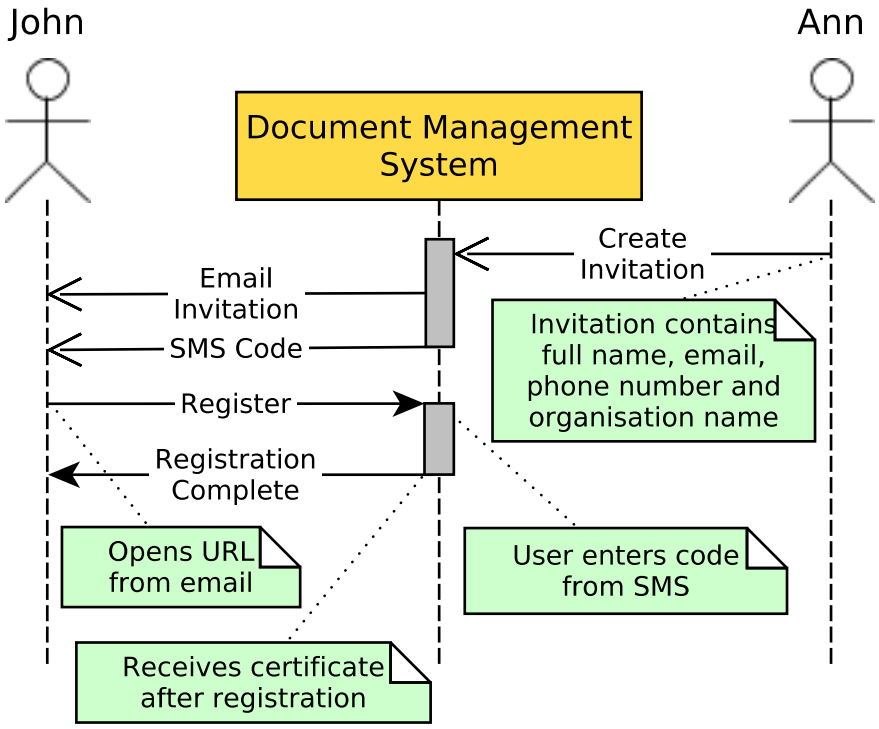


Figure 4.1: Sequence diagram of the user registration process.

one of Organization Administrators to perform this task (or system administrator, if he has access to document keys).

4.3 Use Case

Typical use case for the system is illustrated in Figure 4.2 and goes as follows:

In the use case, we assume that both Ann and John are legitimate registered users of the system and are logged into it. Imagine Ann needs to get a signature from John on document D1. Ann creates a new case folder – Case1. Then, she invites John for collaboration on the documents stored inside the folder and uploads D1 into Case1. John receives notifications about the actions that Ann just did in his preferred way (email, SMS, phone call), enters the system and downloads D1.

John sees that the document requires numerous changes, so he locks D1 and starts to edit it. Notification about John's activity is sent to Ann. Meanwhile, Ann also made some changes to the D1, but since she didn't lock the document, she is not able to update it (even though she is the owner of it). Instead, she puts a comment to D1 which is forwarded to John so he can incorporate her changes as well. John suddenly receives a notification from the system, that his lock on D1 is about to expire and he is prompted to either release the lock, enabling other users to work on D1, or upload its revised version. John needs little more time for editing, therefore he renews the lock and then uploads revised version of the D1. Ann is notified about the new revision, therefore she downloads the last two revisions to see the differences. As she accepts the changes that John has made, she signs the document with her private key.

System then notifies both users collaborating on D1 about the approval of D1 by all required parties (Ann and John) and the final version of D1 is considered legally valid and signed. All collaborators can now (or at *any* point in time) review all document revisions, history of signatures and comments.

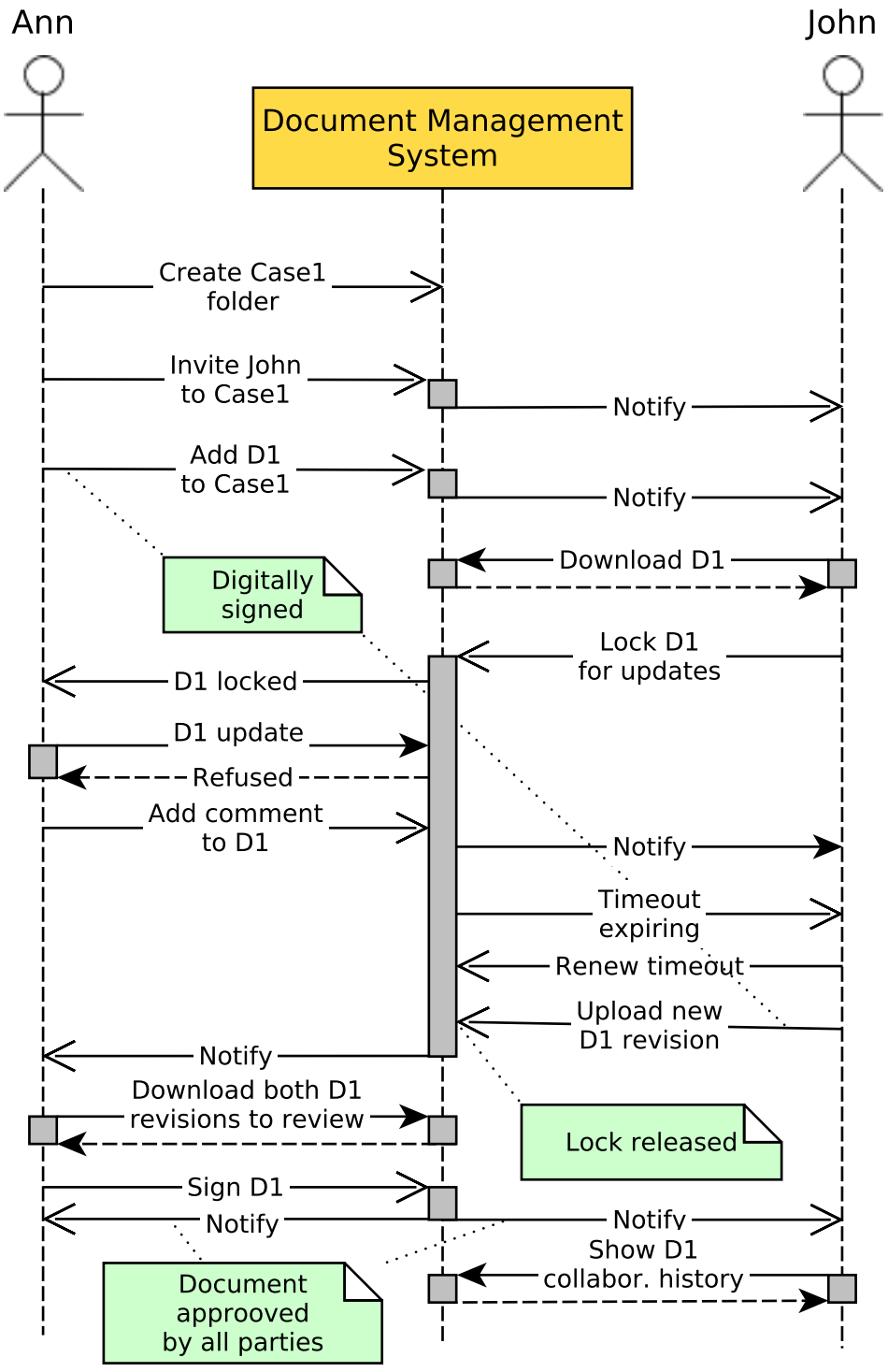


Figure 4.2: Use case of two users collaborating on single document in form of sequence diagram.

Chapter 5

Cryptographic Protocol Design

For long time, handwritten signatures have been used as a proof of authorship or agreement on paper document. With emerge of computers, the ability to sign electronic documents has emerged as well. In electronic documents, however, it is much easier to forge user's signature, as one can simply copy and paste signature from original signed document or modify original document without leaving any trace of modification [50]. This chapter has two goals. First one is to describe in high level Digital Signatures and their use in Public Key Infrastructure (PKI) as a mechanism for digitally signing and verifying signatures of an electronic documents. Second is to formalize designed protocol by presenting it in common syntax.

5.1 Digital Signatures in Public Key Infrastructure

We use *Digital Signatures* for the purpose of document signing in digital environment. Directive *1999/93/EC* [13] of the European Parliament and of the Council defines digital signature as a data in electronic form which are attached to or logically associated with other electronic data and which serve as a method of authentication. The most algorithms for digital signatures are based on public key cryptography considering each user has pair of keys: public and private. Private key is known only to the owner and it serves mainly for the purpose of signing documents. Public key is available to anyone for verifying authorship of signature and document's integrity.

For proving association of public keys to their owners, the electronic document known as a *Digital Certificate* is issued and made publicly available. Digital certificate contains user's public key, identity information, signature of an authority that verified user's identity and identification of issuing *Certification Authority*, date of certificate revocation and much more information defined by used standard[1].

In the most regions and countries around the world, legislation gives digital signatures same weight and legal effect like for handwritten signature. For example in European Union it is aforementioned *Directive 1999/93/EC*, while in USA *Electronic Signatures in Global and National Commerce Act.* [52] from 2000 and *Uniform Electronic Transactions Act* [38] from 1999.

[1]Currently most popular standard in use is `ITU-T X.509` which contains not only certificate format specification but complete set of recommendations for designing and implementing Public Key Infrastructure.

5.1.1 Proposed PKI Approach

Public Key Infrastructure systems are complex distributed systems based on the public/private key cryptography, which are responsible for giving users enough information to make reasonable trust judgments about each other [5, 33]. Elementary entities present in Public Key Infrastructure according to Scheirer [49] are:

- **Certificate Authority** (CA) is an entity that issues certificates to users by signing users' public keys by CA's own private key; Certificate of this CA can be either signed by higher level CA or by itself (self-signed certificate);
- **Registration Authority** (RA) is an entity which verifies user's identity and sends her public key and other information required by a standard to CA to sign;
- **Verification Authority** (VA) – upon request it provides valid certificate of a person in query or notify user whether an error or non standard situation occurs, e.g. if a certificate does not exist for given user, then certificate expired or has been revoked; and
- **End user** is an authorized user of a system.

The same entities will be considered in PKI described in this work. Single entity can take a place of all three authorities, Certificate, Registration and Verification Authority, as is the case of designed application.

Simplified procedure for digitally signing and verifying document in described PKI of designed system is depicted in Figure 5.1 and contains following steps:

1. Alice generates public/private key pair and sends her public key and credentials (SMS code, full name, email and other required information) to Registration Authority in form of signed Certificate Signing Request.
2. RA verifies provided credentials (i.e. via SMS and email) and sends Alice's request to CA for signing .
3. Certificate Authority creates and signs Alice's certificate with its own private key and sends signed certificate to the Alice and VA.
4. Alice queries the VA for Bob's certificate.
5. Assuming Bob is registered user and system holds his valid certificate, VA sends the certificate to the Alice.
6. Alice then:
   (a) creates a document;
   (b) using cryptographic hash function [45], e.g. `MD5` or `SHA-256` she creates a unique message digest of a document (document hash) and encrypts it using her private key – creates digital signature;
   (c) generates symmetric key and encrypts document using this key;
   (d) encrypts symmetric key using Bob's public key obtained from his certificate, and continues to the next step.

7. Alice attaches encrypted hash and encrypted symmetric key to the encrypted document and stores it to the shared Document Repository.
8. Bob downloads encrypted document, signature and encrypted symmetric key from the repository.
9. Bob queries the VA for Alice's certificate.
10. Assuming Alice is registered user and system holds her valid certificate, VA sends the certificate to the Bob.
11. Bob then:
    (a) decrypts symmetric key using his private key;
    (b) decrypts document using decrypted symmetric key;
    (c) applies the same hash function to the document as Alice did, then decrypts document's signature using Alice's public key, compares the results and continues to the next step.
12. If both digests have the same value, then Bob can assume Alice is really the author of the document and it has not been tampered, assuming that Alice's private key and account have not been compromised.
13. If the digests differ, then either the document or signature or both have been modified by an attacker. In this case, Bob will not trust the document.

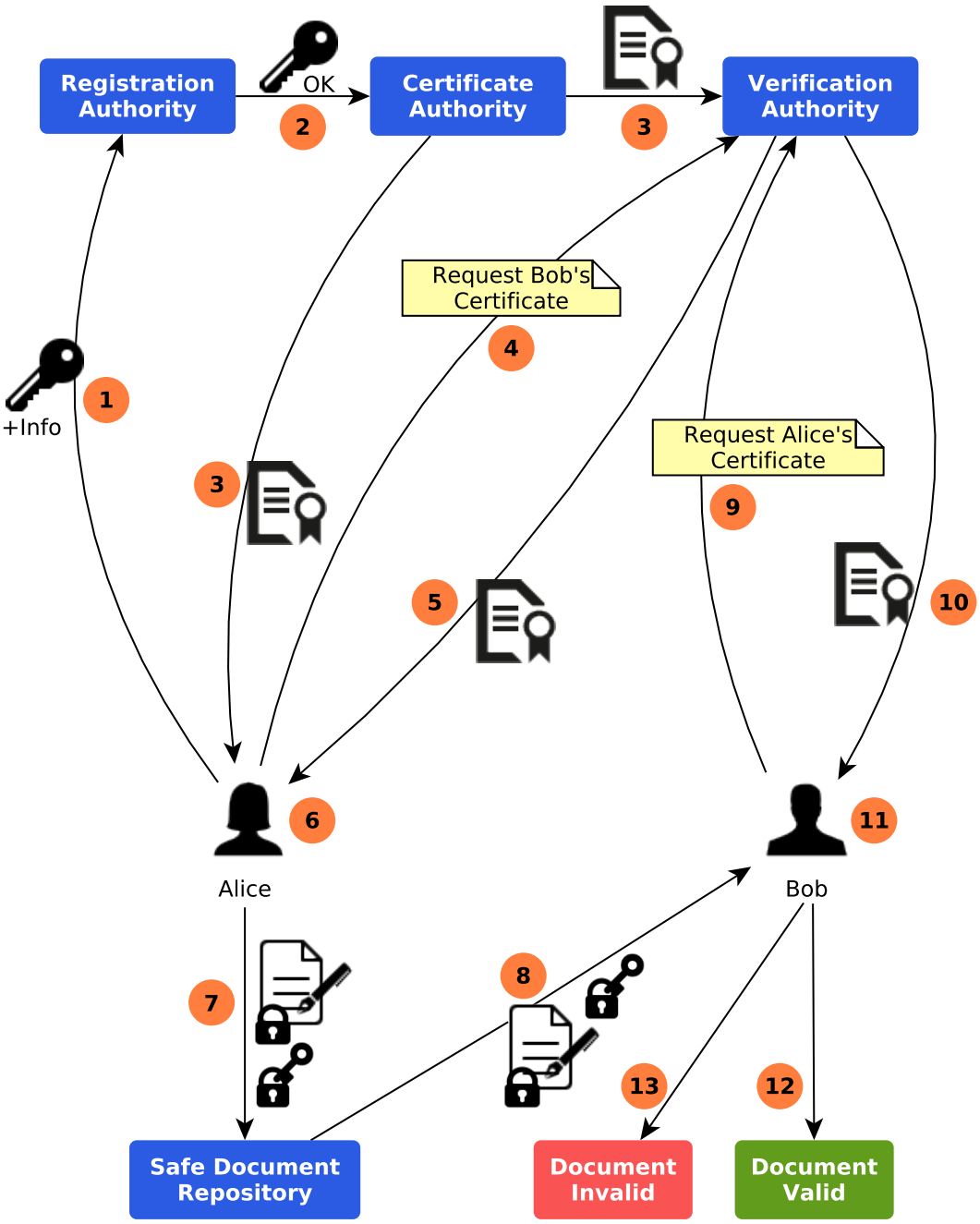


Figure 5.1: High level scheme of Public Key Infrastructure.

5.2 Formal Protocol Design

Cryptographic protocol is described in notation inspired by common syntax [51]. Meaning of symbol used in notation is as follows:

A, B	users of the system
D	arbitrary document, D_v its particular version v
U	all users in the system
U_D	users that collaborate on document D; $U_D \subseteq U$; A, $B \in U_D$
$U_{D_v}^{sig}$	users that signed D_v, $U_{D_v}^{sig} \subseteq U_D$
S	server storing documents, certificates and signatures
T	time-stamp
KP_A	public-private key pair of user A, KP_A^{public} its public (or private) part
K_D	symmetric key for encryption/decryption of associated document D
K_A^P	symmetric key for encryption/decryption of $KP_A^{private}$, derived from password P
IV_{D_v}	initialization vector, unique for each version v of associated document D
IV_A	initialization vector associated with K_A^P
C_A	certificate of user A
CSR_A	Certificate Signing Request of user A
$H(X)$	hash of the value X
$\{X\}K$	value X encrypted by symmetric key K
$\{X\}KP^{public}$	value X encrypted by public key KP^{public}
$\{X\}KP^{private}$	value X encrypted by private key $KP^{private}$
$Q(subject)$	query the server for data specified in the subject, written in free syntax

For simplicity we will assume that all users are in the same organization and they have all necessary permissions.

5.2.1 User Management

Registration

1. $A \rightarrow S : CSR_A$, $\{H(CSR_A)\}KP_A^{private}$, $\{KP_A^{private}\}K_A^P$, IV_A
2. $S \rightarrow A : C_A$

Explanation

1. Assuming user A received valid invitation, he generates his own key-pair, KP_A, creates Certificate Signing Request, CSR_A, and its hash, $H(CSR_A)$. User A creates initialization vector IV_A and provides secret password, P, from which key K_A^P is derived. K_A^P and IV_A are then used, to encrypts A's private key, $KP_A^{private}$. K_A^P is derived from P every time user needs to decrypt his private key. A then sends CSR_A; its encrypted hash $\{H(CSR_A)\}KP_A^{private}$; encrypted private key $\{KP_A^{private}\}K_A^P$, and initialization vector IV_A, necessary for its decryption, to the server.

2. Server generates new certificate, C_A, from CSR_A, stores it with rest of the received data in its database, and returns C_A to the user A.

Additional Notes

- CSR_A is in accordance to PKCS#10 [40] standard, containing following attributes:
  - `commonName`=<full user's name>;
  - `emailAddress`=<user's email address>;
  - `countryName`=CZ;
  - `localityName`=Brno; and
  - `organizationName`=<organization's name>.

  It further contains single extension, `subjectAlternativeName`=<email address>, and user's public key, KP_A^{public}.
- C_A is generated from CSR_A and follows X.509 standard, where Subject Name's fields contain previously mentioned attributes.
- KP_A is public/private RSA[2] key-pair and it's modulus length is always 2048 bit.
- K_A^P is 128 bit symmetric Advanced Encryption Standard cipher in Counter Block Chaining mode (AES-CBC) [20]. It is derived from password P provided by user, using Password-Based Key Derivation Function 2 (PBKDF2), as defined in PKCS#5 [30].
- H uses Secure Hash Algorithm producing 256 bit long fingerprint (SHA-256).
- For enveloping private key material, PKCS#8 [31] is used, which is futher enveloped into PKCS#12 [37] structure.

5.2.2 Document Management

First Document Version Upload

1. $A \rightarrow S : Q$(list of all users)
2. $S \rightarrow A : U$
3. $A \rightarrow S : Q$(certificates of all users in U_D, where U_D is selected from U by A)

[2]Name is derived from initial letters of its creators' surnames: Ron Rivest, Adi Shamir and Leonard Adleman

4. $S \rightarrow A : C_u \forall\, u \in U_D$
5. $A \rightarrow S : \{D_1\}K_D,\ IV_{D_1},$

$$\{K_D\}KP_u^{public} \forall\, u \in U_D$$

Explanation

1. User A queries the server for list of all users.
2. Server returns list of all users, U.
3. User A select collaborators on the document D from U as U_D, and queries server for their certificates.
4. Server returns certificates of all users specified in U_D.
5. A generates new document key, K_D, initialization vector IV_1 and encrypts D_1 using K_D and IV_{D_1}; A encrypts K_D for all users in U_D with their public keys obtained from their certificates. A then sends $\{D_1\}K_D$, its respective initialization vector, IV_{D_1}, and K_D, encrypted by each collaborator's public key, to the server S, which subsequently stores received data.

Any Subsequent Version Upload

1. $A \rightarrow S : Q$(necessary data to upload new version of document D)
2. $S \rightarrow A : \{KP_A^{private}\}K_A^P,\ IV_A,\ \{K_D\}KP_A^{public}$
3. $A \rightarrow S : IV_{D_v},\ \{D_v\}K_D$

Explanation

1. User A queries the server S for data necessary to upload new version of document D.
2. Server returns encrypted private key of user A, $\{KP_A^{private}\}K_A^P$, initialization vector IV_A necessary for its decryption; and encrypted document key, $\{K_D\}KP_A^{public}$. A decrypts his private key and document key, generates new initialization vector, IV_{D_v}, and encrypts D_v using K_D and IV_{D_v}.
3. A sends initialization vector IV_{D_v} and encrypted document $\{D_v\}K_D$, to the server, which saves it to the database.

Additional Notes

- K_D is a 128 bit symmetric AES key in Galois Counter block Mode (AES-GCM) [48], providing data authenticity and confidentiality.
- Encryption and decryption operations are performed using RSAES-OAEP algorithm, as defined in PKCS#1 [47].

5.2.3 Signature Management

Sign Document

1. $A \rightarrow S : Q$(necessary data to sign document D_v)
2. $S \rightarrow A : \{D_v\}K_D$, $\{K_D\}KP_A^{public}$, IV_{D_v},

   $\{KP_A^{private}\}K_A^P$, IV_A
3. $A \rightarrow S : \{H(D_v), T\}KP_A^{private}$

Explanation

1. User A queries server S for all the necessary data to sign document D_v.
2. S returns encrypted document, $\{D_v\}K_D$ with initialization vector IV_{D_v}; encrypted document key, $\{K_D\}KP_A^{public}$; and encrypted private key of user A, $\{KP_A^{private}\}K_A^P$ with IV_A.
3. A decrypts his private key, document key, and document; creates hash of D_v and timestamp T, and encrypts both using his private key, as $\{H(D_v), T\}KP_A^{private}$, effectively creating signature of document D_v. A then sends signature to S which stores it.

Verify Document Signatures

1. $A \rightarrow S : Q$(necessary data to verify all signatures of document D_v)

   $S \rightarrow A : \{D_v\}K_D$, $\{K_D\}KP_A^{public}$, IV_{D_v},

   $\{KP_A^{private}\}K_A^P$, IV_A,

   $\{H(D_v), T\}KP_u^{private}$, $C_u \, \forall u \in U_{D_v}^{sig}$

Explanation

1. A queries server S for all necessary data to verify all signatures of D_v
2. S returns encrypted document, $\{D_v\}K_D$ with initialization vector IV_{D_v}; encrypted document key, $\{K_D\}KP_A^{public}$; encrypted private key of user A, $\{KP_A^{private}\}K_A^P$ with IV_A; and signatures of all the signatories of document D_v and their certificates.

   A then decrypts his private key, document key, and document; decrypts all signatures with signatories' public keys, obtained from their certificates; creates hash of D_v, $H(D_v)$, and compares it with each hash from decrypted signatures. If any of hashes doesn't match, either document, or signature, or both has been tampered by an attacker.

Additional Notes

- T contains current date and time in Coordinated Universal Time (UTC)
- H uses Secure Hash Algorithm producing 256 bit fingerprint (SHA-256)
- Signing and verification operations are performed using RSASSA-PKCS1-V1_5 algorithm, as defined in PKCS#1.
- Document signatures are detached, and enveloped using Cryptographic Message Syntax (CMS)[26].
- Validity of signatures is currently verified only against user's certificate, verification against chain of certificates (chain of trust) is thus not possible.

Chapter 6

Implementation

System is implemented as a client-server application, communicating using traditional request-response schema. Client runs from web browser (also called thin client) and its functionality is implemented in JavaScript. Thanks to this, it is theoretically possible to use application from any device or operating system with graphical interface and web browser supporting JavaScript and Web Cryptography API.

On server side, Python application accessible over HTTP server and Web Server Gateway Interface (`WSGI`) is running. WSGI is universal interface enablig communication between python application and arbitrary web server, as `Apache`, `IIS` or `nginx`. Schema of such system is depicted on Figure 6.1.

Communication is performed over secured HTTP channel, synchronously and asynchronously.

Asynchronous JavaScript and XML (AJAX) requests are used for most operations, except loading new page or reloading current one. Uploading or signing a document are typical examples of asynchronous requests, where user is notified upon operation's success or failure. All AJAX requests are sent to URL, whose path is prefixed with `resources/` keyword. Every path specifies certain resource, to which client can request access. HTTP requests can be of type `GET`, `POST`, `DELETE`, or `PUT`. Messages in asynchronous communication are formatted as JavaScript Object Notation (JSON)[1] formatted data, with exception of form loading. All responses contain mandatory keys `status` and `message`, specifying result

[1] JSON is a lightweight, easily human-readable data-interchansge format, consisting of key:value pairs

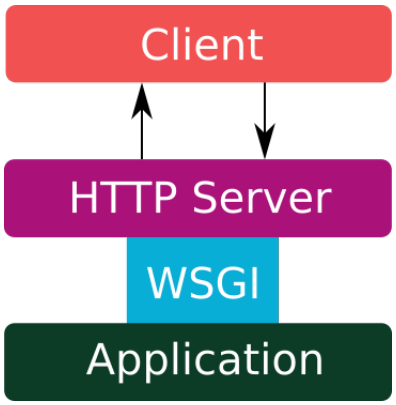


Figure 6.1: Client - Server Architecture using WSGI Interface

of operation and its details. Response can also contain one optional key, `data`, containing custom data payload specific to each resource. Format of request varies, depending on requested resource. Thanks to standardized message syntax, both client and server can be developed independently, as far as they support common communication protocol.

6.1 Web Server Application

Server side is implemented in Python programming language and built on Django framework [8]. **PostgreSQL** [54] database is used for holding application's relational data. PostgreSQL is an open-source relational database management system, not controlled by any organization and implementing most of the functionality of SQL standard while supporting many custom extensions.

6.1.1 Django

Django is an Open Source, multi-platform framework written in Python, focused to ease development of web sites. Django is implemented as a Model-view-controller (MVC) application, consisting of three essential components: model, view and template. Simplified schema of these components and their communication is depicted on Figure 6.2.

Model in Django is an Object-relational mapper (ORM), serving as a source of data for application. Models serves as a proxy object for access to database and every operation on database is thus performed via this model. Every model is a Python class that is mapped to a single database table, where each attribute represents a database field (column).

View handles most of the application's custom logic, and in terms of MVC it plays role of a controller. It describes the data that gets presented to the user, but not how they are presented. View is basically a Python callback function for particular URL, specifying action to be performed when user makes a request to given URL. Requests are forwarded to correct view by URL dispatcher, which has access to predefined list that binds URL to a view.

Every view in application is subclassed from Django's `TemplateView`[2], containing list of allowed HTTP methods for this view and definition of python method to handle the request. In general, following actions are performed, after view receives a request from dispatcher:

1. check user permissions for requested action;
2. check parameters sent in request;
3. perform requested operation; and
4. form a response and send it to the user.

For better illustration of how forwarding an HTTP request in Django works, two code listings are present. Listing 6.1 contains excerpt from Django's dispatcher config. Selected entry matches any request made to `'resources/document/version/'` and forwards it to

[2] `TemplateView` is not only view class available, but serves best for application's needs. Moreover, it is also possible to define view as a function – function based view.

`as_view` method of `DocumentVersionHandler` class, which according to type of request internally calls another method, i.e. `get` or `post`.

Listing 6.2 contains definition of `post` method, handling any HTTP POST request dispatched to `DocumentVersionHandler` view. In following text, numbers in parenthesis denotes line numbers in Listing 6.2. By inheriting from `LoginRequiredMixin` (1), it is ensured that only authenticated user can access this method. Variable `http_method_names` (3) specifies allowed methods for this view. Other HTTP methods will return HTTP Not Allowed response. `transaction.atomic` (5) decorator ensures, that either all database operations called within a method will be successfully performed, or none of them. After pulling data from request, authorization check is performed (13) to ensure that user has permissions to modify this document, which is subsequently saved to database. JSON encoded response is returned every time, on both success and failure, with correct HTTP status set (18, 37, 43).

```
1 urlpatterns = [
2   ...
3   url(r'^resources/document/version/$',
4   views.DocumentVersionHandler.as_view(),
5   name='res_documentversion'),
6   ...
7 ]
```

Listing 6.1: URL dispatcher, forwarding document version upload and download request to correct view.

```
class DocumentVersionHandler(LoginRequiredMixin, TemplateView):
"""Handle operations over custom document version."""
http_method_names = ['get', 'post']

  @transaction.atomic
  def post(self, request):
  """Process new file from the user."""
    try:
      document_id = int(request.POST.pop('document_id')[0])
      iv = request.POST.pop('iv')[0]
      data = request.POST.pop('data')[0]
      size = request.POST.pop('size')[0]
      if not request.user.can_modify(document_id):
        response = {
          'status': 'Fail',
          'message': "You don't have permissions to add new document",
        }
        return JsonResponse(response, status_code=403)
      dc = DocumentContainer.objects.get(id=document_id)
      current_version = dc.current_version().version
      dv = DocumentVersion(
        document=dc,
        user=request.user,
        content=data,
        iv=iv,
        version=current_version+1,
        size=size)
      dv.save()
      response = {
        'status': 'OK',
        'message': '',
        'data': {
          'version': dv.version,
          'size': dv.size
        }
      }
      return JsonResponse(response)
    except IndexError:
      response = {
        'status': 'Fail',
        'message': 'Incorrect parameters.'
      }
      return JsonResponse(response, status_code=400)
```

Listing 6.2: Example of view class handling document upload.

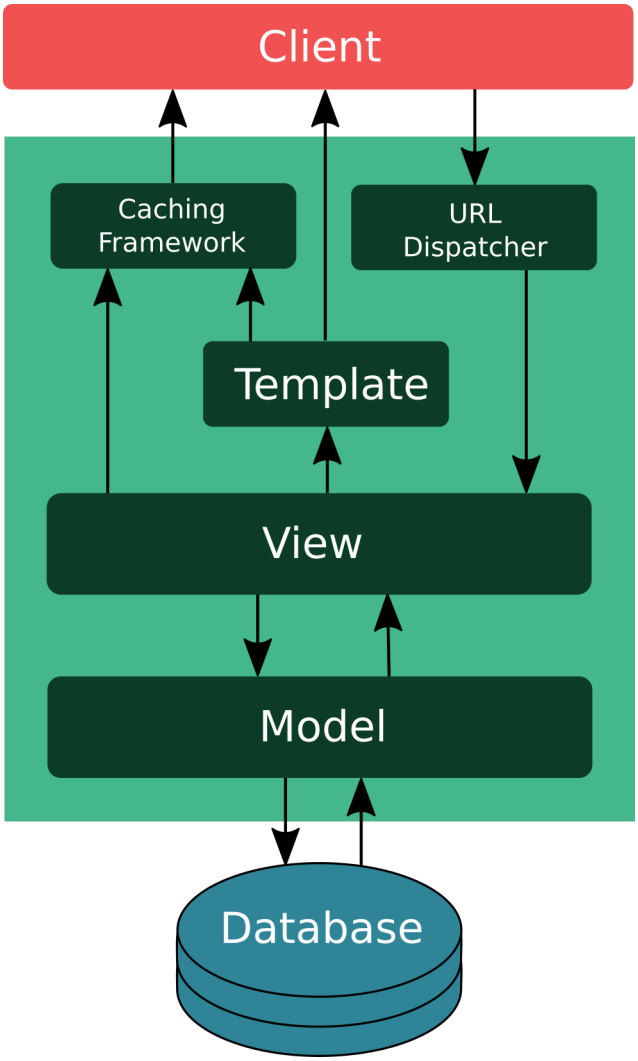


Figure 6.2: Simplified Model of Django Framework

Template specifies the form of data presented to the user and in terms of MVC, template roughly stands for view. Template is composed of static parts (typically HTML), and parts that specify, how dynamic content will be rendered to the user – filters. Filters allows us to use conditional statements, loops, substitutions, and many other directives, and thus move logic responsible for data presentation to the template.

6.1.2 Database Model

Database contains 12 custom tables and 7 tables implicitly created by Django. Every custom model, except `User`, is subclassed from Django's `Model` class, allowing to make database queries from model. Database schema in form of an entity relationship diagram is depicted on the Figure 6.3 and description of individual models/tables and their fields is as follows:

User model represents single user in the system. Every user is uniquely identified by `ID`, `email` or `phone`, where value of last two fields can change over time. Other

mandatory attributes are `name`, `publicKey` and `encryptedPrivateKey`. Private key is encrypted using secret password, and if key is forgotten, new certificate must be issued to the user and previous certificate revoked. Last two fields are optional `address` field and automatically generated `dateCreated` key containing date and time of user registration. **User** model is subclass of Django's `AbstractBaseUser` class. This inheritance enables us to specify custom properties for user and at the same time makes it possible to conveniently use default Django's authentication system.

Organization holds information about single organization in system. It is uniquely identified by `ID`, `email` or `phone`. `dateCreated` and `address` fields have same properties as in User model.

Document represents abstract view of document in the system. Every document is identified by `ID` and attached to `organization` by foreign key relation. Other mandatory fields are `path` and `name` specifying full path to document in tree and document's name respectively[3] `status` specifies current state of document – In Progress, Rejected or Approved. `lockOwner`, `lockedFrom` and `lockedTo` keeps information about who, since and till when holds lock on the document, if any. `archivedFlag` and `dateArchived` indicates, whether document has been moved to archive and when. Model also contains field `dueDate`, specifying deadline for document to be approved and optional `description` field.

DocumentVersion holds information about concrete version of document and is uniquely identified by its `ID` field. Every document must be referenced from at least one documentVersion through `document` field as a foreign key. Model also contains reference to user, who created the version and thus identifies owner of the document (via owner of first document's version). `versionNumber` holds document's version number, with first starting at one and is incremented by one for every next version. `documentContent` field keeps document as base64[4] string encoding BER[5] file. For indicating document's validity (data can be e.g. malformed or irrelevant), `validityFlag` is present. Actual size of raw document (i.e. not base64 encoded, nor encrypted) is stored in `size` field and finally in `initializationVector` is stored initialization vector necessary for document decryption.

DefaultUserPermissions defines many-to-many relationship between user and organization, and every relation in the model is uniquely identified by their combination (referenced via `user` and `organization` fields). Model holds permissions for user in related organization in fields `superuser` and `canInvite`, which define whether user has administrator rights and whether can invite new user to join related organization in the system. Model also holds default document permissions for user, which can be overwritten for each document in DocumentUserPermissions table. These permissions are `canView`, `canSign`, `canComment`, `canModify`, and `canRemove`, where name indicates action to be taken.

DocumentUserPermissions defines many-to-many relationship between user and document, and holds user's permissions on related document. It is uniquely identified by combination of `user` and `document` fields, referencing user and document. In

[3] Currently, path processing is not implemented, so all documents are stored in „same directory".
[4] base64 is encoding schema for representing binary data as an ASCII string.
[5] Basic Encoding Rules, BER is set of rules for representing ASN.1 objects as strings of ones and zeros.

`encryptedDocumentKey` field is stored symmetric key encrypted by user's public key, necessary for decrypting document.

Certificate model holds list of user's certificates, where only one at time can be valid. Relation in model is identified by unique `ID` field. Owner of certificate is referenced by `user` field and validity of certificate is indicated by `validityFlag`. Certificate itself is stored in `certificate` field, as a base64 string encoding BER file.

Signature model holds detached user's signature for given version of document. `user` and `documentVersion` fields references signed user and version of document, and their combination is unique (i.e. user can not sign same document more than once). Model also holds date and time when signature was made in `timestamp` field. `signature` field holds signature as base64 string encoding BER file.

MandatorySignator defines users, whose signatures are mandatory for approval of related document. User and document are referenced by fields `user` and `document`, together making up a private key.

Comment contains all comments added by arbitrary user to specific version of document. Relation in model is uniquely identifined by `ID`. `user` and `documentVersion` fields reference comment's owner and commented documentVersion, `dateCreated` contains date and time of posting comment, and in `comment` is stored comment's content.

Action serves as basic structure for keeping track of actions performed on a document. It is uniquely identified by its `ID` field and in `user` and `documentVersion` are referenced related action's originator and version of document accessed. In `actionPerformed` is stored what action has been taken on document and in `datePerformed` is stored when it was performed.

Invitation field stores necessary data for user to register in the system. Invitation is identified by unique `ID` and in `organization` field references organization into which user is invited. `name` specifies name of the invitee, `email` stores email address used to which invitation is sent, and `phone` stores phone number to which secret code is sent. `expiration` specifies when will invitation expire, if user will not register into the system. `uuid` holds 32 characters long unique identifier used to compose registration link, and `secretCode` stores code, that must be entered upon accessing registration page. Model also contains fields, that specify default user permissions in system, later stored in DefaultUserPermissions table – `superUser`, `canInvite`, `canView`, `canComment`, `canSign`, `canModify`, `canRemove`. Invitation is automatically deleted on expiration or after successful registration.

Implicit models holds additional data that Django operates on. These are users' and groups' permissions specifying access model, list of active sessions, list of migrations that were applied to the database, admin log with information about administrators' actions, list of tables with details, and possibly other information as well.

6.2 Client Application

Client side application logic is implemented in JavaScript language. For cryptographic functionality, WebCrypto API [56] is used, currently available in all major desktop web browsers for GNU/Linux, Microsoft Windows and OS X.

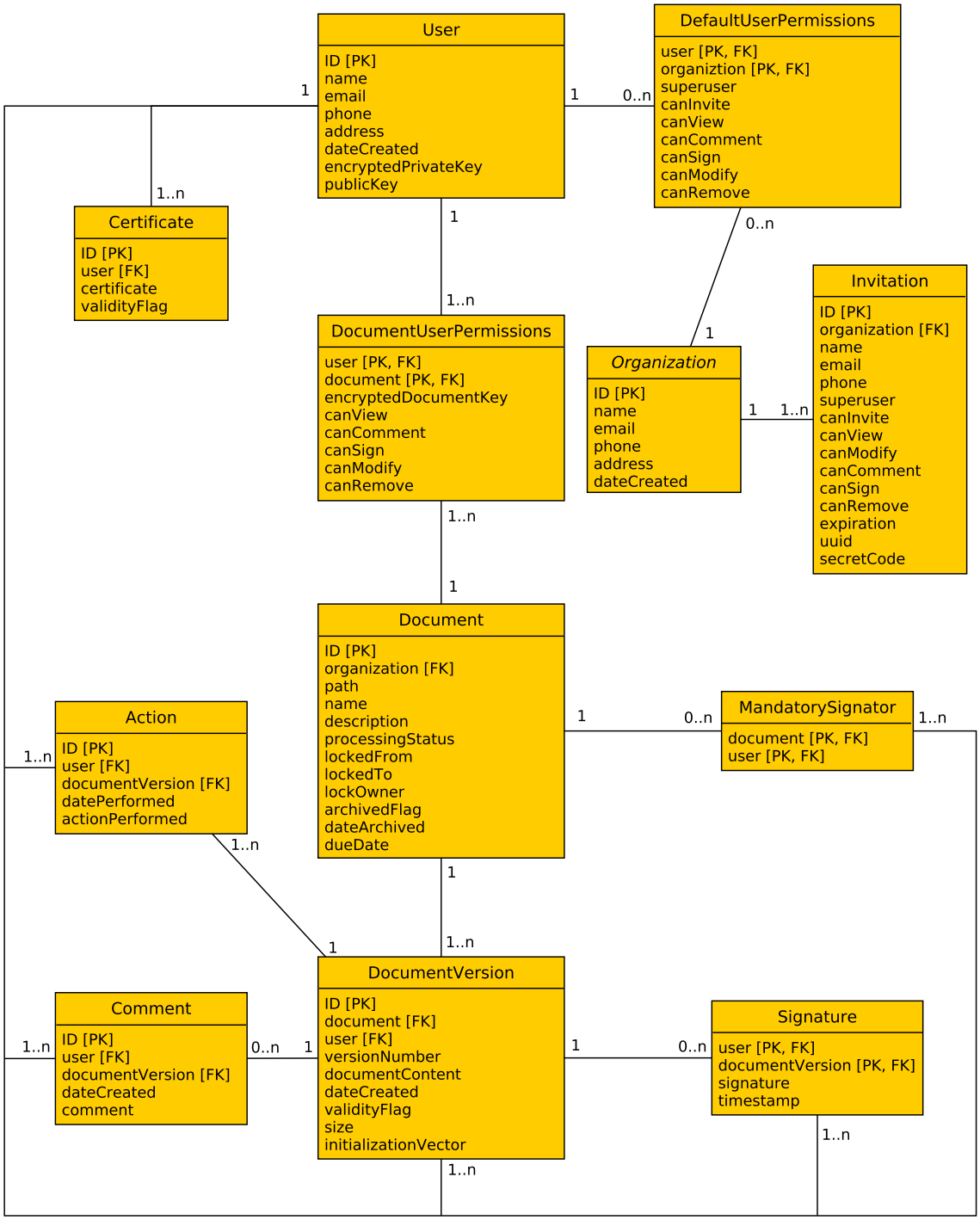


Figure 6.3: Entity Relationship Model of the System

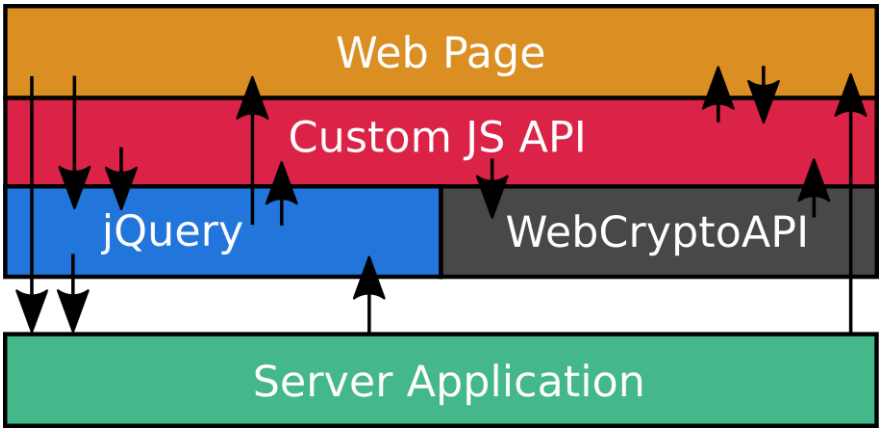


Figure 6.4: Communication between JavaScript libraries and the server.

The Web Cryptography API defines a low-level interface for interacting with cryptographic key material that is managed or exposed by user agents. The API itself is agnostic of the underlying implementation of key storage, but provides a common set of interfaces that allow web applications to perform operations such as signature generation and verification, hashing and verification, encryption and decryption, without requiring access to the raw keying material.

For operations related to Public Key Infrastructure, PKI.js [22] library is used. PKI.js is an open-source pure JavaScript library build on WebCrypto API supporting wide range of current cryptographic standards. Cryptographic structures described in Chapter 5 are represented as JavaScript objects using ASN.1 notation. Abstract Syntax Notation One (ASN.1) is a method of specifying abstract objects in Open Systems Interconnection (OSI) architecture. It is flexible notation that allows one to define a variety data types, from simple types such as integers and bit strings to structured types such as sets and sequences, as well as complex types defined in terms of others [29].

Another JS library utilized in project is jQuery [28], used mainly for handling asynchronous requests to the server and for most of the Document Object Model (DOM) manipulations. Flow of communication between JavaScript libraries, web page, and server is depicted in Figure 6.4.

Web application provides user with simple interface, allowing him to register in the system, basic document and document version manipulations, sign documents, verify signatures, review collaboration history and also comparison of textual documents. When user tries to access document, private key for document decryption is downloaded from the server and user is prompted to enter password to decrypt it. On success, private key, together with certificate, is stored as a session storage variable. Session storage lasts for as long as the browser is open and survives over page reloads and restores. Opening a page in a new tab or window will cause a new session to be initiated, which differs from how session cookies work.

Chapter 7

Security Analysis

Security is essential property of the system, especially when dealing with highly sensitive data. This chapter reviews application in terms of RMIAS model and analyzes protection against most common attacks aimed to web applications. As a main source of information on attacks and protection against them was used Open Web Application Security Project (OWASP) [43], which is worldwide not-for-profit charitable organization focused on improving the security of a software.

7.1 RMIAS Analysis

During application development, two stages of IS life cycle has been identified: first one includes application & protocol design and second is implementation.

Since RMIAS is aimed primarily to information systems deployed within a company, not every aspect of RMIAS is applicable to developed application. Some restrictions were thus applied:

- all the information that system handles is in electronic form;
- sensitivity of all documents is considered as arbitrary;
- location of all information is set to partially controlled, since we assume its deployment in the cloud and no direct way of exporting document from system (e.g sending as an email) exists; and
- security countermeasures are restricted only to technical ones.

In both stages (design and implementation), each of five possible states of information has been considered, and technical countermeasures has been analyzed for every security goal. Table 7.1 summarizes countermeasures that has been taken to achieve security goals in the system. Some goals are however not applicable to a certain category of information, e.g. it doesn't make sense to try to achieve privacy in process of creating an information in the system. ISPD document in real deployment would include much broader range of security policies.

7.2 Attack Vectors

Following list of analyzed attacks is created from WhiteHat's 2015 Website Security Statistics Report [57] and OWASP's top 10 list, identifying some of the most critical risks facing

Information State	**Security Goal**	**Applied Countermeasure**
creation	confidentiality	authorization, end-to-end encryption
	integrity	end-to-end encryption
	non-repudiation	user authentication, PKI
	availability	database controls
	privacy	not applicable
	auditability	server logs, logging user actions
	accountability	logging user actions
	authenticity & trustworthiness	user identity verification
processing	confidentiality	end-to-end encryption
	integrity	end-to-end encryption
	non-repudiation	user authentication, PKI
	availability	database controls
	privacy	not applicable
	auditability	server logs, user actions logging
	accountability	logging user actions
	authenticity & trustworthiness	user identity verification
storage	confidentiality	end-to-end encryption, database access protection
	integrity	end-to-end encryption, database controls
	non-repudiation	user authentication, PKI
	availability	database controls
	privacy	not applicable
	auditability	server logs, user actions logging
	accountability	user actions logging
	authenticity & trustworthiness	user identity verification
transmission	confidentiality	end-to-end encryption, encrypted connection
	integrity	end-to-end encryption, encrypted connection
	non-repudiation	not applicable
	availability	not applicable
	privacy	not applicable
	auditability	server logs, user actions logging
	accountability	user actions logging
	authenticity & trustworthiness	not applicable
destruction	confidentiality	end-to-end encryption, database controls
	integrity	not applicable
	non-repudiation	user actions logging
	availability	not applicable
	privacy	not applicable
	auditability	server log, user actions logging
	accountability	user actions logging
	authenticity & trustworthiness	user identity verification

Table 7.1: Simplified ISPD of the system, created using RMIAS method, where only technical countermeasures are applied.

organizations.

Insuffiecent Transport Layer Protection. This type of attack typically exploits missing or improperly configured Transport Layer Security (TLS), or vulnerabilities discovered in TLS cryptographic libraries, such as OpenSSL [19].

Successful protection against this attack lies in requiring secured connection for all traffic to/from the server, proper TLS configuration, and most recent version of critical software with latest bug fixes deployed. It is however not possible to protect against unknown or non-disclosed bugs.

Information Leakage. An information leak occurs when system data or debugging information leaves the program through an output stream or logging function, revealing potentially sensitive information, or information that can help adversary with further exploration of the system. Sensitive information can leak from the server in several ways, some most common are through data queries, error messages, transmission of sensitive data and unhandled exception on the server.

To avoid this type of attack, we must ensure that all error messages that leave the server are parsed with any sensitive information removed and that all exceptions are correctly handled. Django largely helps with this task by providing default templates when error or exception occurs on the server.

Sensitive Data Exposure. Most security flaws in this category raises from not encrypting sensitive data.

All communication between server and client is encrypted, moreover, potentially most sensitive data, documents, are encrypted on client side, so even server provider doesn't have access to them.

Insufficient Authorization. This category of attacks exploit missing or insufficient check of permissions. Attack is possible, if unathorized external user can access or even modify content intended only for authorized users, by crafting a custom request, modifying request's parameters, or possibly in some other way. Similarly, regular system user could access or modify content that he isn't authorized to.

In application, authorization checks are performed on both, client and server side. On client side, protection is achieved only by not displaying controls/objects (security by obscurity), while on server side, actual check of user's permissions is performed for every request accessing some protected resource. No attack to the system from this category is known.

Injection Flaws. Injection flaws occur when an application sends malicious data to an interpreter, which subsequently executes them. Among injection flaws, SQL injection is most prevalent type of attack, where malicious user is able to execute arbitrary SQL code on a database. This can result in records being deleted, modified, or in data leakage.

Application communicates with database solely over Django's ORM and doesn't contain any custom queries. Since underlying database driver escapes all the queries made over ORM, no SQL injection should be possible, as no vulnerability of this kind is known in Django's ORM. Another code interpreter on server, Python, does not execute any code provided by client, so no injection is possible in this case as well.

Cross Site Scripting. Cross Site Scripting (XSS) attacks allow a user to inject client side scripts into the browsers of other users. This is usually achieved by storing the malicious scripts in the database where it will be retrieved and displayed to other users (persistent XSS), or by getting users to click a link which will cause the attacker's JavaScript to be executed by the user's browser (reflected XSS). However, XSS attacks can originate from any untrusted source of data, such as cookies or Web services, whenever the data is not sufficiently sanitized before including in a page.

Since Django provides protection against XSS by escaping specific characters in its templating mechanism, it provides protection against most attacks of this type. Moreover, application provides only the one entry point, the registration of user, where data can be submitted to the server by outsider.

For regular system user, there is more ways he can post malicious data to the server. All data, before presented back to a user, is somehow processed though, making attack more difficult. Imagine situation, when an attacker discovers vulnerability in PKI.js, allowing him to append malicious code to the signature, that would execute on every verification of this signature, without user noticing it. Attacker then could, for example, craft apparently valid signature containing malicious code, that would send him private key of every user who verifies this signature. To eliminate this type of attack, additional data check should be performed on the server, e.g signature verification.

Session Fixation. The session fixation attack belongs into a class of session hijacking attacks, which steals the established session between the client and the Web Server after the user logs in. In session fixation, attacker first establishes valid session to a server and then tricks user to login to server using this session. As with most other attacks, server must contain vulnerability which allows it.

Django provides full support for anonymous sessions, and currently there is no know session related vulnerability that would make session fixation attack possible. Sophisticated attacker could however hijack user's session by exploiting some other system's vulnerability, i.e. by XSS attack.

Brute Force. A brute force attack can manifest itself in many different ways, but primarily consists in an attacker configuring predetermined values, making requests to a server using those values, and then analyzing the response.

In implemented system, brute force attack could be used to guess user name and password to enter the system, this would be however too computationally expensive and can be easily avoided by incurring some minimum amount of time between consecutive login attempts. Brute force attack can also used to discover hidden pages on website or for testing various request parameters. This attack is not applicable, since most of the Djano views require login to access it. Only exceptions are index, login and registration page. The last one, registration, is protected against posting data from untrusted sources, as described in Chapter 5.

Content Spoofing. Content spoofing, also referred to as content injection or virtual defacement, is an attack targeting a user made possible by an injection of vulnerability in a web application. This type of attack similar to XSS attack, but instead of executing malicious code on client side, uses other techniques to modify user's page.

Since similar protection as against XSS can be applied, application doesn't seem to be vulnerable to this type of attack.

Cross Site Request Forgery. Cross-Site Request Forgery (CSRF) is an attack that forces an end user to execute unwanted actions on a web application in which they're currently authenticated, or in other words, submitting a malicious request to the server.

Django provides protection against most types of CSRF attacks, by setting a CSRF cookie to a random value, that other sites doesn't have access to. New CSRF cookie is generated each time user logs in to the application. This ensures that only data that have originated from trusted domains can be used to post data to the server, since cookie is present in each query. CSRF cookie does not have to be present in so called safe methods, which doesn't make any modification to the server, such as `GET`, `HEAD`, `OPTIONS` and `TRACE`.

URL Redirector Abuse. URL Redirector Abuse exploits redirecting functionality of web application, where incoming request is redirected to an alternate resource. If alternate resource is not checked, malicious link can be forged redirecting to attacker's website.

Application is not vulnerable to this type of attack, since it doesn't provide automatic redirection to any external site.

Predictable Resource Location. This attack exploits easy to guess names of resources for gaining access to them. This is similar to previously described brute force attack with same solution.

Insufficient Authentication. Attacks in this category mostly stems from weak passwords used or transmitting clear-text passwords over insecure channel.

To avoid this type of attack, system requires user to use strong password for login to the system as well as for private key encryption. Two factor authentication will be supported in case of production deployment.

Directory Indexing. A directory listing provides an attacker with the complete index of all the resources located inside the server root directory. Although allowing visitor to list files in web server's root is not direct security risk, it's good practice to disable it, if listing is not necessary for usage of application.

Abuse of Functionality. It should be ensured, that no software functionality can be abused to perform a function not intended by the developer. Abuse of functionality is strongly dependent on the design and implementation for application functions and features. The best way to mitigate potential functionality abuses, is to maintain set of tests stretching as much functionality of application as possible. Application currently contains limited set of unit tests, not covering most of the functionality, and needs to be extended.

As set of application's features is limited, there is currently no known bug allowing abuse of functionality.

Insufficient Password Recovery. Weak password recovery processes allow stronger password authentication schemes to be bypassed, proper password recovery technique is thus essential. Multiple options thus should be available to recovery a user password, but none of them sending new password to the user directly.

Currently, application does not support login password recovery at all. This feature is however necessary for production deployment. Recovery of the password for access to private key is not, and will never be supported, since no mechanism exists to achieve this

task. In case user forgets password to his private key, he must generate new key pair and request a new certificate.

Other attacks. Whole new category of attacks stems from vulnerabilities that are specific to cloud infrastructure and must be considered when deploying system on th cloud. Some of most common vulnerabilities in cloud are broken authentication into virtual machine or web console, insecure API, threat of virtual machine escape, and other dangers, that are direct consequences of sharing common resources.

Another attack, not directed to web application but to client's machine is also possible. If attacker is regular user of the system, he can post document containing malicious code, that is executed after opening it on client. This type of attack can only be prevented by sufficient protection of client's computer, e.g. keeping software up to date, enabled virus and malware protection, etc.

Conclusion. Although Django provides extensive set of tools to improve application's security, it is not bulletproof and may also contain critical vulnerabilities. Therefore it is necessary to stay alert and maintain custom security policies as well.

Chapter 8

Future Work

This chapter discusses set of additional features that could find their way into application, and changes or improvements on the implementation side that should be considered before further project development.

8.1 Additional Features

Multiple extensions that would increase user experience and ease application usage are possible. Some of most requested ones are described below.

Built in Text Editor and Templates Support. Built-in text editor would certainly make sense for creating document drafts and would enable user to edit document inside a window of a web browser, thus saving some time for the user. Similarly, templates could relieve a user from repetitive task of a new document creation, since documents very often follow the same or very similar structure.

However, the main disadvantage of these solutions is that built-in editor (embedded by template system) would necessarily lack a lot of functionality from a fully featured MS office software and that users often refuse to learn how to utilize another text editor.

Documents Comparison. With growing number of documents that user needs to keep track of, a lot of time could be saved if it would be possible to view changes from a previous versions of document inside a web browser, immediately sign or comment them and thus avoid downloading and comparing them manually[1]

Drawback of this approach resides in a fact, that existing Office documents would have to be converted to a markup language before comparing, and no tool can guarantee 100% formatting similarity of documents before and after conversion.

Document Changes Tracking. Collaboration on a document brings a lot of confusion about who changed what, and when. This feature would enable a user to select a line or block of a text, and find originator of every change that has been made to that selection (similar to git-blame functionality).

As with previous features, support of mark up language would be necessary, to properly implement this functionality.

[1] Although, software like MS Office or Libre Office provide „visual diff“ functionality, necessity of downloading files still remains.

PDF Export. Similarly, support for direct export to PDF format for better portability would be of frequent use cases, but we face the same problem here as in the case of comparison using markup language. Reasonable compromise would be creation of a PDF document automatically and keep it together with original document and its signatures.

Authentication Based on Biometric ID. Password can be leaked, mobile can be stolen but biometrics stay with us in any case. Despite this fact, no current technology for personal use is safe and reliable enough to provide adequate security for a biometric authentication. Therefore, biometric authentication can be used for additional level of security together with common authentication techniques in PKI [49].

Integration with Existing Services. Application uses custom authentication system and certificate management. This enforces user to maintain yet another online „identity“. Integration with existing tools and services, could make system more appealing for users. OpenID [41] for example, allows user to use single set of user credentials to access multiple websites, or YubiKey [61] that allows authentication using one-time password. Allowing users to use their own certificate, could also be convenient for some of them.

Client as a Browser Extension or Stand-alone application. Despite its growing popularity, still many concerns exists about security of in-browser JavaScript cryptography [46, 3]. By providing client application as a signed browser extension, application's security would increase, since only verified code would be executed in user's browser. Another way how to deliver executable code to client, is by providing stand-alone, installable application, that would run entirely out of browser. Both approaches would thus require more effort on client side, possibly discouraging users from application usage.

8.2 Implementation Changes

While proceeding in implementation, some decisions have shown up to be better than selected ones. In case of the further development, following options should be considered, in order to achieve better application performance, security and usability.

Storing Versions as a Document Differences. Currently, system stores every version of a document as a whole file, even if minimal or no change was made since previous version. In this way, storage requirements on the server can be enormous, if dealing with multiple versions of „big“ files. Storing new version as a difference from previous one, could significantly reduce storage usage. On the other side, it would bring notable overhead to the client, which would have to download first document version and all subsequent differences, and merge them all to get the current version.

As a reasonable compromise seems to save each new version as a difference from first one, store each n-th version as a whole file, or store whole file only when major change in document was made.

Improved Document Storage. Documents are currently stored as a files encoded as base64 strings in database. This brings overhead on client side – in encoding to, resp. decoding from base64 – as well as in increased network traffic and required storage on the server side, since base64 encoded file is approximately 33% bigger than the original

one. Moreover, storing big files in database is not recommended, since it can degrade performance of whole database.

Better solution would be to store files in the filesystem, where relational database would only hold path to the file. Another, more scalable solution is to use some NoSQL database[2]. This approach however, increases complexity of whole system and thus potentially decreases its security.

Signature Time-stamping by Server. When signing document, current time, according to the client computer is attached to the signature. This gives space to the user, to claim different signing time on the document, than is real signing time. Signature time-stamping by server would solve this problem, by confirming that user signed specific version of a document at specific time. Sequential time-stamping (as provided by DocVerify), would also increase security of whole system.

Extensibility over Plugins. In many cases, only portion of functionality of the system is required. Allowing administrator to select certain set of features, would provide great flexibility for custom deployments of application. To ease extensibility of the application, it would be necessary to redesign it in a way, that only minimal functionality would be enabled by default(e.g. basic document and user management), and everything else would require specific plugin. Moreover, by splitting logic of the application into multiple modules, accessible over strictly specified interface, its development and maintenance would also get easier.

[2]NoSQL is type of database, where data have simpler structure than traditional relation tables, allowing better scalability and flexibility.

Chapter 9

Conclusion

Goal of this master thesis was to design cryptographic protocol for secure management and approval of document versions, and implement this protocol as a client-server application.

As a first step to achieve this goal, it was necessary to explore and review existing systems with similar functionality, for assessing viability and competitiveness of a new system. Next step was to gain understanding of security principles required for further development, followed by forming application's requirements and underlying communication protocol. After that, prototype of system was developed, and application was discussed in security context according to current standards.

Outcome of this is work is the communication protocol describing sequence and content of messages exchanged between client and server, database model used for storing relational data, and client-server application, built above this model and implementing designed communication protocol. Throughout whole development process, emphasis was put to fundamental aspects of information security, using RMIAS model as a conceptual framework for assessing security threats and developing appropriate countermeasures. Thanks to utilization of widely used frameworks and libraries, and following security standards and recommendations, application is well protected against the most common types of attacks on the internet.

System is implemented as a web application, allowing invited user to register into the system, upload and download arbitrary version of a document, sign a document, review document details and collaboration history, or even compare different versions of same document. End-to-end document encryption and optional two-factor authentication are also present among features. Each component is implemented using freely available open source tools and libraries. Since system will be also released under a public license, anyone will be able to join its development and contribute to the code.

Application is currently not suitable for production deployment, since it lacks some necessary functionality, e.g. graphical user or permissions management, more user friendly interface, and also thorough testing. System can be however used as a solid base for further development and later could certainly find its way into many groups or organizations, where effective collaboration on documents is necessary.

Bibliography

[1] C. Adams, P. Cain, D. Pinkas, and R. Zuccherato. Internet x.509 public key infrastructure time-stamp protocol. RFC, RFC Editor, 2001.

[2] Archevos Corporation. Experdocs. http://www.experdocs.com. [Accessed: 2015-12-29].

[3] Tony Arcieri. What's wrong with in-browser cryptography? https://tonyarcieri.com/whats-wrong-with-webcrypto. [Accessed: 2016-05-21].

[4] BlackBerry. WatchDox. https://www.watchdox.com. [Accessed: 2016-01-31].

[5] K. P. Bosworth and N. Tedeschi. Public key infrastructures — the next generation. *BT Technology Journal*, 19(3):44–59, 2001.

[6] Y. Cherdantseva and J. Hilton. A Reference Model of Information Assurance & Security. 2013.

[7] Y. Cherdantseva and J. Hilton. Information security and information assurance. the discussion about the meaning, scope and goals, 2013.

[8] Django Software Foundation. Django. https://www.djangoproject.com. [Accessed: 2016-05-10].

[9] DocuSign Inc. DocuSign. https://www.docusign.com. [Accessed: 2015-12-29].

[10] DocuXplorer. DocuXplorer. http://www.closingtable.com. [Accessed: 2015-12-29].

[11] E-Sign UK Ltd. E-Sign. http://e-sign.co.uk. [Accessed: 2016-01-25].

[12] Effacts. Effacts. http://www.effacts.com. [Accessed: 2016-01-31].

[13] European Parliament, Council of The European Union. Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community framework for electronic signatures. http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:31999L0093, 1999. [Accessed: 2016-01-11].

[14] Fluix. Fluix. https://fluix.io. [Accessed: 2015-12-30].

[15] International Organization for Standardization. Information technology—Security techniques—Information security management systems—Requirements. ISO 27001:2013, 2013. [Accessed: 2016-01-01].

[16] International Organization for Standardization. Information technology – Open Systems Interconnection – The Directory. ISO 9594:9598, 2014. [Accessed: 2016-05-09].

[17] International Organization for Standardization. Long term signature profiles for CMS Advanced Electronic Signatures. ISO 14533-1:2014, 2014. [Accessed: 2016-05-15].

[18] Apache Software Foundation. Subversion. https://subversion.apache.org. [Accessed: 2016-01-10].

[19] OpenSSL Software Foundation. OpenSSL Vulnerabilities. https://www.openssl.org/news/vulnerabilities.html. [Accessed: 2016-05-11].

[20] S. Frankel, R. Glenn, NIST, and S. Kelly. The AES-CBC Cipher Algorithm and Its Use with IPsec. https://tools.ietf.org/html/rfc3602, 2003. [Accessed: 2016-05-11].

[21] Free Software Foundation. Concurrent Versions System. https://savannah.nongnu.org/projects/cvs. [Accessed: 2016-01-10].

[22] GlobalSign. PKI.js. http://pkijs.org/. [Accessed: 2016-05-15].

[23] Google. Google Docs. https://www.google.com/docs/about. [Accessed: 2016-01-09].

[24] Google. Protecting data for the long term with forward secrecy. https://googleonlinesecurity.blogspot.sk/2011/11/protecting-data-for-long-term-with.html. [Accessed: 2015-12-31].

[25] grecs. Information assurance versus information security. https://www.novainfosec.com/2011/08/30/information-assurance-versus-information-security. [Accessed: 2016-01-01].

[26] R. Housley. Cryptographic Message Syntax (CMS). https://tools.ietf.org/html/rfc5652, 2009. [Accessed: 2016-05-14].

[27] iClosing.com, Inc. Closing table. http://www.closingtable.com. [Accessed: 2015-12-29].

[28] jQuery Foundation. jQuery. https://jquery.com/. [Accessed: 2016-05-15].

[29] Burton S. Kaliski Jr. A Layman's Guide to a Subset of ASN.1, BER, and DER. http://luca.ntop.org/Teaching/Appunti/asn1.html. [Accessed: 2016-05-15].

[30] B. Kaliski. PKCS 5: Password-Based Cryptography Specification, Version 2.0. Technical report, RFC Editor, 2000. [Accessed: 2016-05-11].

[31] B. Kaliski. Public-Key Cryptography Standards (PKCS) 8: Private-Key Information Syntax Specification Version 1.2. https://tools.ietf.org/html/rfc5208, 2008.

[32] M. Mackall. Mercurial. https://www.mercurial-scm.org. [Accessed: 2016-01-10].

[33] J. Marchesini and S. Smith. Modeling public key infrastructures in the real world. In *Public Key Infrastructure*, pages 118–134. Springer Berlin Heidelberg, 2005.

[34] Microsoft Corporation. Advancing our encryption and transparency efforts. http://blogs.microsoft.com/on-the-issues/2014/07/01/advancing-our-encryption-and-transparency-efforts. [Accessed: 2015-12-31].

[35] Microsoft Corporation. Office 365. https://products.office.com/en/products. [Accessed: 2016-01-09].

[36] Microsoft Corporation. Team Foundation Server. https://savannah.nongnu.org/projects/cvs. [Accessed: 2016-01-10].

[37] K. Moriarty. PKCS 12: Personal Information Exchange Syntax v1.1. https://tools.ietf.org/html/rfc7292, 2014.

[38] National Conference of Commissioners on Uniform State Laws. Uniform Electronic Transactions Act. http://www.uniformlaws.org/shared/docs/electronic%20transactions/ueta_final_99.pdf. [Accessed: 2016-01-11].

[39] National Institute of Standards and Technology. FIPS PUB 186-4: Digital Signature Standard (DSS). http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdfss, 2013. [Accessed: 2016-05-08].

[40] M. Nystrom and B. Kaliski. PKCS 10: Certification Request Syntax Specification, Version 1.7. https://tools.ietf.org/html/rfc2986, 2000.

[41] OpenID Foundation. OpenID. https://openid.net. [Accessed: 2016-05-20].

[42] S. Otte. Version control systems, 2009.

[43] OWASP. OWASP. https://www.owasp.org. [Accessed: 2016-05-11].

[44] Donn B. Parker. *Fighting Computer Crime: A New Framework for Protecting Information*. John Wiley & Sons, Inc., 1998.

[45] Bart Preneel. Cryptographic hash functions. *European Transactions on Telecommunications*, 1994.

[46] Thomas Ptacek. Javascript cryptography considered harmful. https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2011/august/javascript-cryptography-considered-harmful/. [Accessed: 2016-05-21].

[47] RSA Laboratories. PKCS 1 v2.2: RSA Cryptography Standard . Technical report, EMC Corporation, 2012. [Accessed: 2016-05-11].

[48] J. Salowey, A. Choudhury, D. McGrew, and Inc. Cisco Systems. AES Galois Counter Mode (GCM) Cipher Suites for TLS. https://tools.ietf.org/html/rfc5288, 2008. [Accessed: 2016-05-11].

[49] W. Scheirer, B. Bishop, and T. Boult. Beyond PKI: The biocryptographic key infrastructure. In *Information Forensics and Security (WIFS), 2010 IEEE International Workshop on*, pages 1–6, 2010.

[50] Bruce Schneier. *Applied Cryptography (2Nd Ed.): Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., 1995.

[51] Security Protocols Open Repository. Security Protocols Open Repository. http://www.lsv.ens-cachan.fr/Software/spore/format.html. [Accessed: 2016-05-18].

[52] Senate and House of Representatives of the United States of America. Electronic Signatures in Global and National Commerce Act (Public Law 106–229, June 30, 2000). https://www.gpo.gov/fdsys/pkg/PLAW-106publ229/pdf/PLAW-106publ229.pdf. [Accessed: 2016-01-11].

[53] TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU. Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks. Itu-t Recommendation, International Telecommunication Union, 2014. [Accessed: 2016-05-11].

[54] The PostgreSQL Global Development Group. PostgreSQL. http://www.postgresql.org. [Accessed: 2016-05-10].

[55] L. Torvalds. Git. https://git-scm.com. [Accessed: 2016-01-10].

[56] Web Cryptography Working Group. Web Cryptography API. https://www.w3.org/TR/WebCryptoAPI/, 2014. [Accessed: 2016-05-14].

[57] WhiteHat Security. Website Security Statistics Report. https://info.whitehatsec.com/rs/whitehatsecurity/images/2015-Stats-Report.pdf. [Accessed: 2016-05-15].

[58] M. Whitman and H. Mattord. *Principles of Information Security*. Cengage Learning, 2011.

[59] xDTM Standard Association. Transaction Management Standard. http://www.xdtm.org, 2016. [Accessed: 2016-01-11].

[60] Yozons, Inc. Open eSignForms. http://open.esignforms.com. [Accessed: 2016-01-23].

[61] Yubico. YubiKey. https://www.yubico.com. [Accessed: 2016-05-20].

Appendices

List of Appendices

Appendix A

CD Content

- `technical_report.pdf` – technical report as a PDF (this document)
- `technical_report.zip` – source code of technical report in LaTeXformat
- `application.zip` – source code of application, including `README` document

Appendix B

Application Screenshots

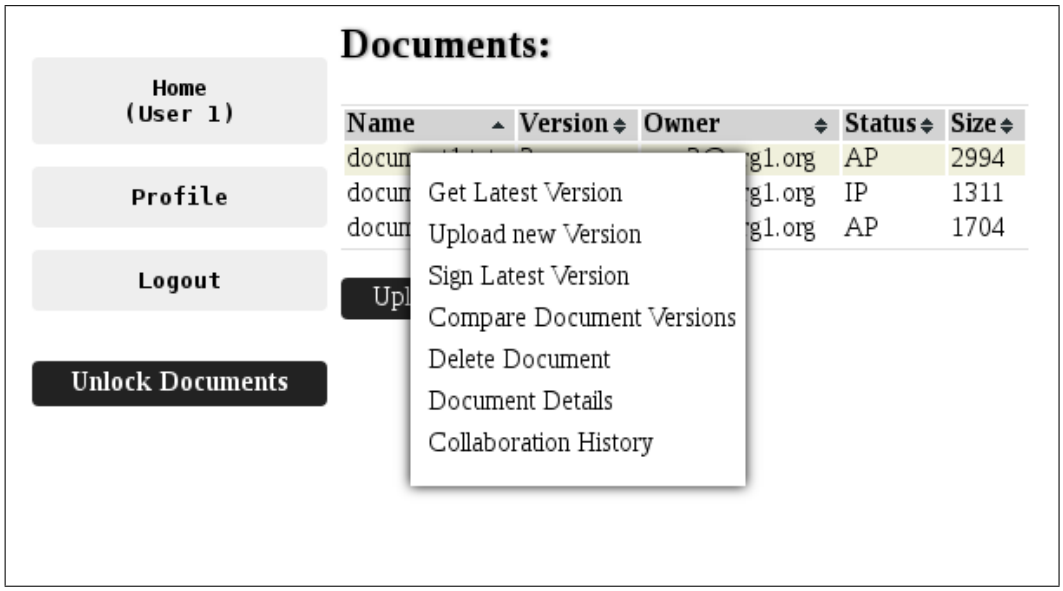


Figure B.1: Document browser.

Figure B.2: Invitation form.

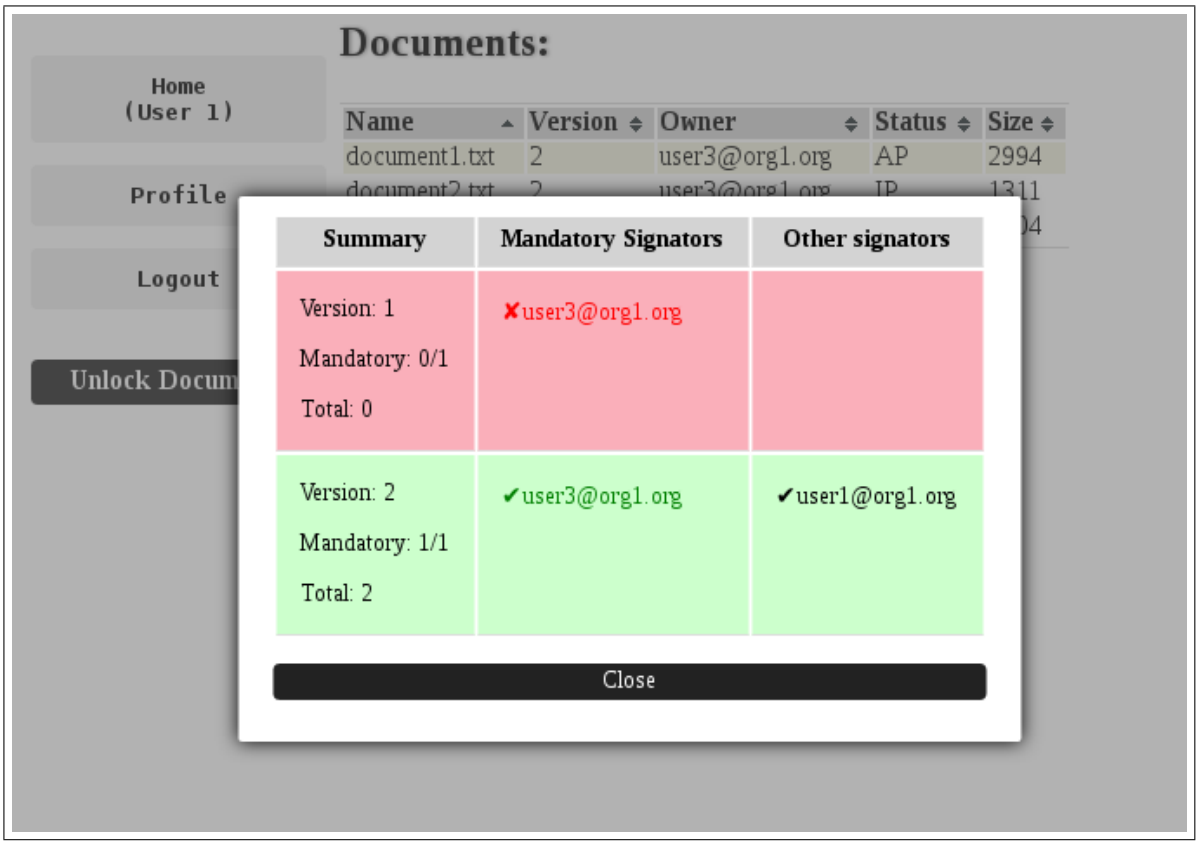


Figure B.3: Collaboration history example.