**НОВ БЪЛГАРСКИ УНИВЕРСИТЕТ**

**БАКЪЛАВЪРСКИ ФАКУЛТЕТ**

**ДЕПАРТАМЕНТ "ИНФОРМАТИКА"**

**ПРОГРАМА** **ИНФОРМАТИКА**

**ДИПЛОМНА РАБОТА**

**ТЕМА: AS2 (Applicability Statement 2) сървър и клиент за трансфер на данни**

**НА СТУДЕНТА: Илиян Делчев Додеков ФАК. № 50129**

**НАУЧЕН РЪКОВОДИТЕЛ: Илиян Занкински**

**София**

**2015**

Съдържание

[Уводна част 4](#_Toc432548264)

[I. Резюме 4](#_Toc432548265)

[II. Мотивация 5](#_Toc432548266)

[III. Актуалност 6](#_Toc432548267)

[Обзорна част 7](#_Toc432548268)

[IV. История 7](#_Toc432548269)

[V. Конкурентни решения 7](#_Toc432548270)

[ Open AS2 8](#_Toc432548271)

[ Axway Gateway Interchange 9](#_Toc432548272)

[VI. Използвани технологии 10](#_Toc432548273)

[VII. Цялостен процес 12](#_Toc432548274)

[VIII. Съображения 15](#_Toc432548275)

[Проектантска част 16](#_Toc432548276)

[I. Hypertext Transfer Protocol (HTTP) 16](#_Toc432548277)

[II. X.509 сертификати 18](#_Toc432548278)

[III. Multipurpose Internet Mail Extensions (MIME) 20](#_Toc432548279)

[ Въведение в MIME 21](#_Toc432548280)

[ Важни хедъри 22](#_Toc432548281)

[ Multipart MIME съобщения 24](#_Toc432548282)

[ Secure MIME (S/MIME) 25](#_Toc432548283)

[IV. Работа с файлове 26](#_Toc432548284)

[ Изпращане на съобщения 26](#_Toc432548285)

[ Получаване на съобщения 27](#_Toc432548286)

[ “Работна” папка 28](#_Toc432548287)

[ “Бекъп” папка 28](#_Toc432548288)

[ Имена на файлове 30](#_Toc432548289)

[V. Компресиране 30](#_Toc432548290)

[VI. Подписване 32](#_Toc432548291)

[ Генериране на подпис 32](#_Toc432548292)

[ Верифициране на подпис 34](#_Toc432548293)

[VII. Криптиране 35](#_Toc432548294)

[VIII. Хедъри 37](#_Toc432548295)

[IX. Message Disposition Notification (MDN) 40](#_Toc432548296)

[ Съдържание 41](#_Toc432548297)

[ Типове 43](#_Toc432548298)

[ Message Integrity Check (MIC) 43](#_Toc432548299)

[Реализация 45](#_Toc432548300)

[I. Стартиране и спиране на приложението 46](#_Toc432548301)

[II. Обща архитектура на приложението 48](#_Toc432548302)

[ Мониториране на MDN съобщения 50](#_Toc432548303)

[ Управление на партньори 50](#_Toc432548304)

[ MIME Криптография 51](#_Toc432548305)

[ Достъп до сертификати и ключове 56](#_Toc432548306)

[ Глобална сървърна конфигурация 57](#_Toc432548307)

[Експериментална част 58](#_Toc432548308)

[I. Конфигурация на приложението 58](#_Toc432548309)

[Заключение 59](#_Toc432548310)

[Използвана литература 60](#_Toc432548311)

# Уводна част

## Резюме

AS2 (Applicability Statement 2) е спецификация за това как да се трансферира информация сигурно и гарантирано в Интернет. Сигурността се постига с помощта на дигитални сертификати и криптиране.

AS2 е базиран на HTTP и S/MIME. Това е вторият AS протокол разработен и ползва същите конвенции за подписване, криптиране и MDN, както оригиналният AS1 от 90-те години. С други думи:

* Файловете са „закачени“ (attachment) към стандартизирано S/MIME съобщение (наричано още AS2 съобщение).
* AS2 съобщенията винаги се изпращат по HTTP или HTTPS протокол (Secure Sockets Layer не е задължителен) и обикновено с помощта на POST метод.
* Съобщенията могат да бъдат компресирани, но не е задължително.
* Съобщенията могат да бъдат подписани с помощта на дигитален сертификат, но не е задължително.
* Съобщенията могат да бъдат криптирани с помощта на дигитален сертификат, но не е задължително.
* Съобщенията могат да поискат Message Disposition Notification (MDN), за потвърждение, че трансферът е успешен. Искането на MDN потвърждение не е задължително. MDN представлява съобщение, което следва нормите за трансфер на данни според AS2 протокола, но има специфично съдържание.
* Ако MDN потвърждение е поискано:
  + Когато дадено съобщение е получено, успешно декриптирано и подписа е верифициран, „успешен“ MDN ще бъде изпратен на изпращача на съобщението. MDN обикновено се подписват, но не се криптират.
  + Когато MDN съобщението бъде получено, декриптирано и подписа е верифициран, оригиналният изпращач ще знае, че трансферът е успешен
  + Ако има проблеми с трансфера може да бъде пратен „неуспешен“ MDN. Според протоколната спецификация обаче, липсата на MDN също означава „неуспех“.

Обикновено при AS2 трансфер има изискване двете страни, които обменят информация, предварително да си разменили набор от сертификати и идентификатори (т. нар. „имена на партньорите“). Имената на партньорите могат да бъдат всякаква валидна фраза.

AS2 спецификацията е описана подробно в RFC 4130 (Request for comment 4130). Дипломната работа ще имплементира стандарта до степен, в която следните функционалности са реализирани (както са описани в RFC 4130):

* HTTP транспорт
* S/MIME съдържание на съобщенията
* Криптиране с помощта на X509 дигитален сертификат
* Подписване с помощта на X509 дигитален сертификат
* Синхронен или асинхронен MDN

За осъществяването на успешен AS2 трансфер, софтуерът, който ще бъде изработен трябва едновременно да играе ролите на клиент и сървър (тоест изпращач и получател). За да се постигне това, ще се имплементира Java базирано решение.

За да се изпрати файл, разработеният продукт ще следи преконфигурирана директория за един или повече файлове и ще ги изпраща на даден получател посредством AS2. От друга страна, при получаване на съобщение, цялата информация ще бъде извлечена под формата на един или повече файлове и ще бъде записана в преконфигурирана директория.

Продуктът ще има възможност за проследяване на това как изглежда AS2 съобщението, когато се праща по мрежата. За да може да се демонстрира функционалността, продуктът ще бъде инсталиран два пъти, за да може двете инсталации да комуникират помежду си.

## Мотивация

Преди да взема решението за темата на дипломната работа, направих обстойно проучване какъв AS2 софтуер се предлага на пазара. Поради високата сигурност, сложност и изисквания на протокола, повечето решения таргетират ентърпрайз пазара, в резултат на което са доста скъпо платени. От друга страна решенията, които са безплатни и/или с отворен код са доста неразвити и неактуални. Повечето безплатни решения ползват доста стари технологии, в резултат на което имат неактуални имплементации. Също така се спират до реализацията на протоколната имплементация и нямат фокус на лесната работа с тях и имплементирането на функционалности, които да ги направят наистина използваеми в реалния живот.

Целта на тази дипломна работа е да се разработи AS2 комуникационен софтуер, който да покрива протоколната спецификация. Идеята е впоследствие да се подобри решението, така че да бъде възможно ползването му в реална клиентска среда. В същото време приложението ще бъде с отворен код.

## Актуалност

Въпреки че спецификацията е финализирана в RFC 4130 през Юли 2005 година, тя все още е актуална и всъщност е една от най-ползваните в световен мащаб. Това се дължи отново на факта, че протоколът се ползва предимно от ентърпрайз пазара за сигурен трансфер на данни – предимно EDI документи. Както знаем, промените в ентърпрайз пазара се внедряват доста бавно и като цяло, за да започне да се ползва дадена технология, тя трябва първо да се докаже. AS2 е добър пример за доказана технология.

От 2013 година се стандартизира и AS4 спецификацията, която до голяма степен спазва същите принципи за сигурност, но ползва SOAP протокол за транспорт и сигурност. AS4 като стандарт тепърва ще се доказва и ще навлиза в пазара. Неговата идея не е да замести AS2, а по-скоро дава възможност да бъде ползван от организации, които са искат да ползват или са инвестирали в web services(<http://www.drummondgroup.com/index.php/component/content/article/127-b2b/b2b-products/b2b-faqs/243-as4-faq> ).

# Обзорна част

## История

AS2 (Applicability Statement 2) спецификацията е предложена от Drummond Group и стандартизирана през 2005 година от IETF (Internet Engineering Task Force). Спецификацията дефинира как се транспортират Electronic Data Interchange (EDI) съобщения сигурно през Интернет с HTTP протокол. Въпреки, че протокола е направен с умисъл за транспорт на EDI документи, на практика може да се ползва за всякакъв тип файлове.

AS2 надгражда AS1 спецификацията, която ползва S/MIME за транспорт. Като цяло, в момента съществуват 4 applicability statement протокола – AS1, AS2, AS3, AS4. Колективно се наричат **ASx**. Разликите в тях могат да бъдат сведени до типа на протокола, който се използва за транспорт:

|  |  |
| --- | --- |
| Протокол | Транспорт |
| AS1 | **SMTP** |
| AS2 | **HTTP** |
| AS3 | **FTP** |
| AS4 | **SOAP** |

До момента има 3 версии на AS2 протокола: 1.0, 1.1, 1.2.

* АS2 1.0 – Тази версия на протокола дефинира базовата спецификация, както е описана в RFC4130
* АS2 1.1 – Надгражда базовата спецификация, като добавя компресиране, както е описано в RFC5402. **Това е протоколната версия която дипломната работа поддържа.**
* AS2 1.2 – Надгражда версия 1.1 на протокола, като добавя възможност за закачане на повече от един документ в едно AS2 съобщение.

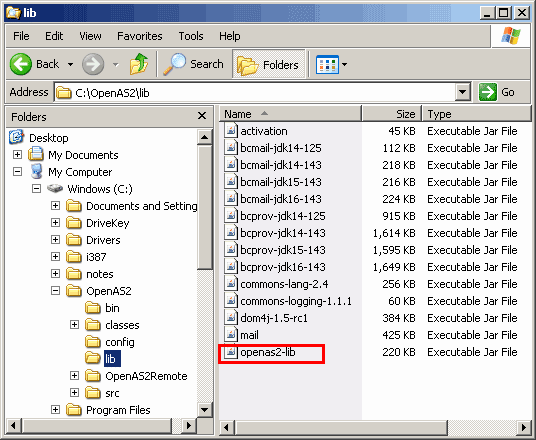
## Конкурентни решения

За целта на дипломната работа, бяха разгледани следните две конкурентни решения:

* Open AS2
* Axway Gateway Interchange

### Open AS2

Това е може би, най-разпространеният, безплатен AS2 софтуер в Интернет. Като цяло, няма много безплатни AS2 решения. Причината за това е високата сложност на протокола и факта, че се ползва предимно от ентърпрайз пазара. Освен това, тъй като протокола обикновено се ползва за трансфер на документи с висока степен на важност, обикновено клиентите искат поддръжка на софтуера, а това го няма при безплатните решения.



Open AS2 е Java базиран и имплементира спецификацията до колкото е описано в RFC4130. Това е базовата функционалност на AS2 протокола, която всеки софтуер е длъжен да спазва. Има редица от други RFC дефиниции, които до някаква степен разширяват възможностите на протокола, без на нарушават основното RFC 4130, но Open AS2 не поддържа нита една от тези дефиниции. Също така, продукта поддържа версия на AS2 протокола 1.0, което означава че не поддържа компресия.

Продукта също тъка няма потребителски интерфейс и конфигурацията се извършва от файлове.

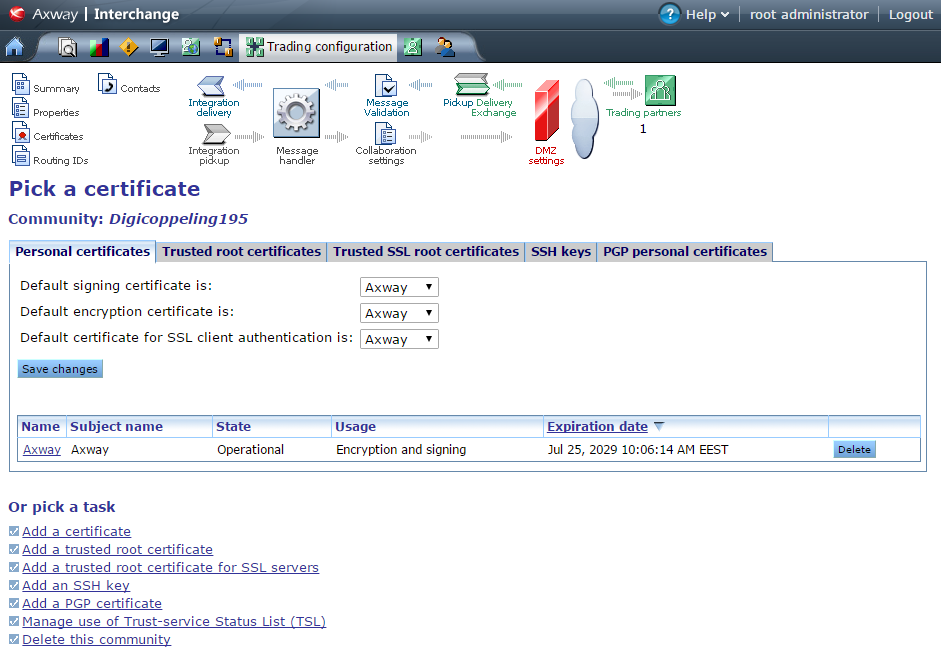
### Axway Gateway Interchange

От друга страна решението на компанията Axway е доста по-завършено. Продукта Gateway Interchange е платен софтуер, който напълно имплементира AS2 спецификацията, както и редица от други незадължителни RFC дефиниции, които разширяват възможностите на протокола. Част от тях са:

* Multiple Attachments описана в RFC6362
* Filename Preservation описана в RFC2183
* Filename Preservation for Multiple Attachments описана в RFC2183 и RFC6362
* Certificate Exchange Messaging
* Chunked Transfer Encoding
* Reliability
* Restart
* SHA-2

По-горните дефиниции не са задължителни да бъдат имплементирани, но до голяма степен разширяват възможностите на протокола откъм сигурност.

Gateway Interchange има доста подробен администраторски интерфейс, с който с лекота може да се променя всеки аспект от конфигурацията на приложението. Също така дава възможност за мониториране на всяко получено и изпратено съобщение или MDN.



Софтуерът освен това е сертифициран от Drummond Group. Тъй като Drummond Group е компанията, която е измислила AS2 стандарта, това означава, че приложението е напълно съвместимо с протокола. Gateway Interchange поддържа версия на AS2 протокола 1.2.

Предлага се и 24/7 поддръжка на решението за всичките съществуващи клиенти на компанията. Разбира се, всичко това не е безплатно.

## Използвани технологии

За реализацията на приложението е ползван Java 8, като програмен език. От уикипедия (<https://bg.wikipedia.org/wiki/Java>):

Java или Джава е обектно-ориентиран език за програмиране. Кодът, написан на Java не се компилира до машинен код за определен процесор, а до специфичен за езика код, наречен байт код. Поради това за изпълнението на програма, написана на Java е необходима т. нар. Виртуална машина (на английски: Java Virtual Machine).

Подобна реализация има своите предимства и недостатъци. Сред главните предимства са:

* лесната поносимост между различните платформи (софтуерни или хардуерни) - веднъж написана и компилирана, една Java-програма може да бъде стартирана на компютри независимо от архитектурата или от операционната им система. За целта е необходимо само да се инсталира виртуална машина за съответната платформа;
* допълнителните действия, извършвани от виртуалната машина като освобождаване на паметта от обекти(класове) които не се използват(Garbage collector), проверка за размерността на масивите;
* възможността за контрол на правата на потребителя на ниво виртуална машина
* първоначално заделяна на heap, част от паметта резервирана за джава
* висока степен на сигурност поради факта, че програмистите не работят директно с паметта и др.

Главен недостатък е необходимостта от допълнителни ресурси (под формата на процесорно време и памет) за изпълнението на самата виртуална машина.

Съществуват и компилатори, които превеждат байт кода до машинен код. Недостатъка при използването им е, че тогава програмата не може да се стартира на различни платформи.

Избраният език за имплементацията е Java, за да може приложението да е платформено незвисимо и да може да се изпълнява на всякаква операционна система.

Използвани са също следните външни Java библиотеки за подпомагането на реализацията на някои от компонентите на приложението:

|  |  |  |
| --- | --- | --- |
| Библиотека | Версия | Описание |
| Bouncy Castle Cryptography API | 1.52 | Колекция от програмни интерфейси, ползвани в криптографията. |
| Bouncy Castle Mail | 1.52 | Програмен интерфейс за работа с S/MIME и email съобщения. |
| Apache HTTP Client | 4.5 | Библиотека позволяваща генериране на клиентски HTTP заявки. |
| Java Mail API | 1.5.4 | Програмен интерфейс за работа с S/MIME и email съобщения. |
| Jetty Server | 9.3 | Библиотека позволяваща работата с HTTP уеб сървър. |
| Apache Log4j | 1.2.17 | Библиотека за работа с лог файлове. |
| Java Servlet API | 3.1 | Библиотека позволяваща имплементирането на сървлети за даден уеб сървър. |

## Цялостен процес

Важно е преди да започне да се говори за спецификите в AS2 да се опише цялостния процес за трансфер на файл от край до край.

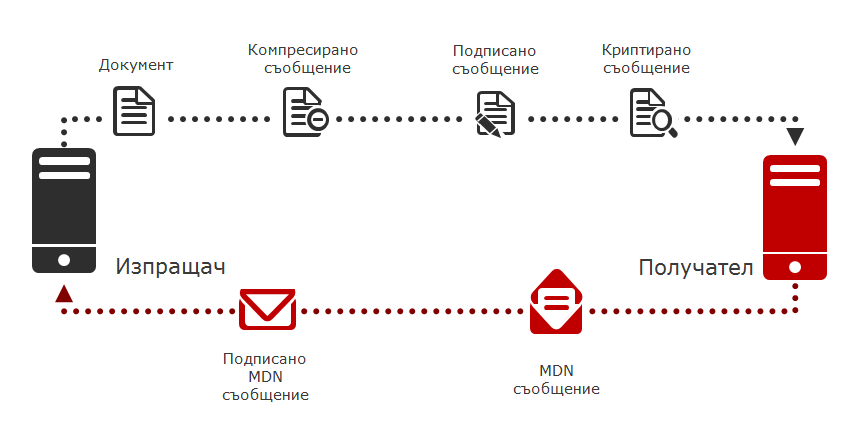
За да може да говорим за AS2 комуникация между 2-ма партньори, е нужно всеки от тях да има инсталиран и конфигуриран AS2 сървър. Освен това за успешна комуникация е нужно партньорите да си разменят набор от X509 сертификати и идентификатори (повече за това в следващите глави на този документ).

За изпращането на файл от партньор “Алфа” до партньор “Бета” се следват следните стъпки:

1. Алфа закача файла към стандартизирано MIME съобщение (наричано още AS2 съобщение).
2. Алфа може да компресира съобщението, но не е задължително.
3. Алфа може да подпише съобщението, но не е задължително.
4. Алфа може да криптира съобщението, но не е задължително.
5. Алфа може да отбележи в съобщението, че иска да получи потвърждение при успешен трансфер. Съобщенията могат да поискат Message Disposition Notification (MDN), за потвърждение, че трансферът е успешен. Искането на MDN потвърждение не е задължително. MDN представлява съобщение, което следва нормите за трансфер на данни според AS2 протокола, но има специфично съдържание.
6. След като съобщението успешно се пакетира, Алфа го изпраща до Бета по HTTP с POST заявка.
7. Бета получава съобщението и започва процес по разпакетиране.
8. Бета декриптира съобщението, ако то е криптирано.
9. Бета верифицира дигиталния подпис, ако съобщението е било подписано.
10. Бета декомпресира съобщението, ако то е било компресирано.
11. Бета проверява дали в съобщението е отбелязано, че трябва да се изпрати MDN за успешното получаване на съобщението.
12. Ако MDN потвърждение е поискано, Бета генерира MDN съобщение, със специално съдържание, съдържащо параметри от оригиналното съобщение.
13. Ако е поискано, Бета подписва MDN съобщението. При MDN съобщенията няма компресиране и криптиране.
14. Бета изпраща MDN съобщението на Алфа.
15. Алфа получава MDN съобщението и започва процес по разпакетиране
16. Алфа верифицира дигиталния подпис, ако MDN съобщението е било подписано.
17. Алфа проверява съдържанието на MDN съобщението и за кое оригинално съобщение се отнася потвърждението.
18. Ако съдържанието на MDN съобщението съвпада, трансферът приключва.

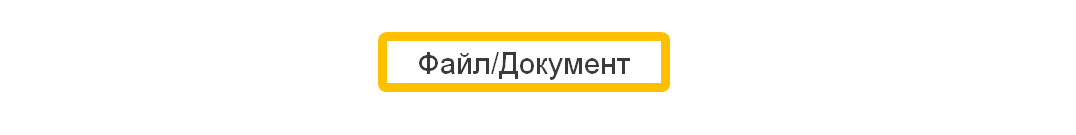
Всяка от тези стъпки е обяснена по-подробно в следващите глави на този документ.

По-горният процес може да бъде обобщен в следващата графика:

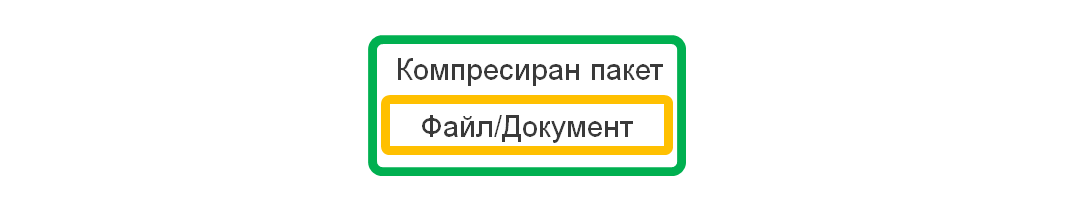


За по-лесно илюстриране на различните трансформации при пакетирането на едно AS2 съобщение, то преминава през следните етапи, като никои от етапите не е задължителен:

1. Обикновен файл.



1. Компресиран файл.



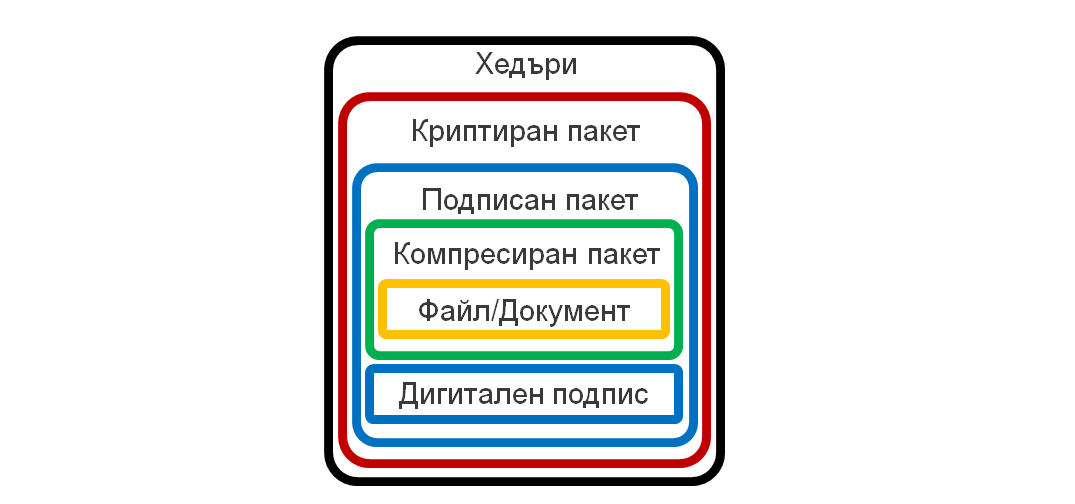
1. Подписано съобщение



1. Криптирано съобщение



1. AS2 съобщение



Важно е да се отбележи, че последователността от трансформации винаги се извършва в този ред – компресиране -> подписване -> криптиране. Всяко от тези стъпки може да бъде пропусната в зависимост от конфигурацията.

Същото правило важи и при получаване на съобщение, но в обратен ред – декриптиране -> верифициране на подписа -> декомпресиране.

## Съображения

Тази глава описва някои от спецификите на работа на приложението.

* Приложението няма потребителски или администраторски интерфейс. Конфигурацията се намира в текстови файлове и при промяна на някой от конфигурационните параметри е нужен рестарт.
* Приложението не е интегрирано база данни. Интеграцията с база данни е планирана за бъдещ етап. С нейна помощ биха могли да се реализират някои от следните подобрения:
  + Динамична конфигурация без нуждата от рестарт на приложението
  + История на изпратени и получени съобщения
  + По-голяма гъвкавост при получаването на асинхронни MDN потвърждения
* В момента приложението няма възможно да изпраща или получава съобщения през TLS канал, поддържа се само чисто HTTP.

# Проектантска част

## Hypertext Transfer Protocol (HTTP)

Hypertext transfer protocol (HTTP) е мрежов протокол, от приложния слой нa OSI моделa, зa пренос нa информaция в компютърни мрежи. Създaден кaто средство зa публикувaне нa HTML стрaници, протоколът довеждa до формирaнето нa Световнaтa уеб мрежa. Спецификацията е дефинирана в RFC 2616 (от юни 1999) и е със стaтут нa стaндaрт и описвa HTTP/1.1.

HTTP се базира на изпращане на заявка и получаване на отговор. Комуникацията е между клиент и сървър, където изпращача на заявката е клиент, а получателят е сървър.

HTTP ресурсите се идентифицират и намират в мрежата на базата на Uniform Resource Locators (URLs), ползвайки Uniform Resource Identifier (URI) схемите http и https. В зависимост от действието което изпращача на заявката иска да направи, съществуват различни HTTP методи, които се изпълняват за даден URL:

* GET
* HEAD
* POST
* PUT
* DELETE
* TRACE
* OPTIONS
* CONNECT
* PATCH

Най-често използваните са GET и POST.

Всяка заявка се състои от следните неща:

* Първия ред на заявката съдържа метода, URL ресурса и версията на протокола. Например: GET /images/logo.png HTTP/1.1
* Набор от хедъри, всеки от които е на отделен ред. Хедърите са двойки стойност-ключ, като тяхната цел е да подадат мета информация на сървъра. Например: Accept-Language: en
* Задължително има празен ред между хедърите и тялото на заявката
* Тяло на заявката, което може и да липсва в зависимост от метода.

Всеки отговор се състои от следните неща:

* Статус ред, който включва статус код и съобщение. Например HTTP/1.1 200 OK означава че заявката е успешна.
* Набор от хедъри, всеки от които е на отделен ред. Хедърите са двойки стойност-ключ, като тяхната цел е да подадат мета информация на клиента. Например: Content-Type: text/xml
* Задължително има празен ред между хедърите и тялото на отговора
* Тяло на отговора, което може и да липсва в зависимост от метода.

Така изглежда примерна GET заявка:

GET **/index.html** **HTTP**/1.1

Host: www.example.com

Съответно така изглежда отговора на GET заявката:

**HTTP**/1.1 200 **OK**

Date: Mon, 23 May 2005 22:38:34 GMT

Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)

Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT

ETag: "3f80f-1b6-3e1cb03b"

Content-Type: text/html; charset=UTF-8

Content-Length: 138

Accept-Ranges: bytes

Connection: close

**<html>**

**<head>**

**<title>**An Example Page**</title>**

**</head>**

**<body>**

Hello World, this is a very simple HTML document.

**</body>**

**</html>**

AS2 ползва HTTP стандарта за пренос на данни и винаги работи с POST заявки. При AS2 винаги изпращача на съобщението го изпраща, за разлика от други протоколи в които се очаква получателят сам да си вземе съобщението (напр. FTP, SSH).

## X.509 сертификати

В криптографията X.509 е стандарт за public key infrastructure (PKI) и private management infrastructure (PMI). X509 специфицира стандартни формати за сертификати с публични ключове, анулиране на сертификати (certificate revocation lists - CRL), атрибут сертификати (attribute certificates) и алгоритъм за валидация с родителски сертификати. Последната версия на X509 е 3.

Един x509 сертификат съдържа публичен ключ на дадено лице, информация за това лице (име, организация и т.н), информация за сертификационния орган, който е издал сертификата, информация за срока му на валидност, информация за използваните криптографски алгоритми и различни други детайли.

Инфраструктурата на публичния ключ (PKI) предоставя архитектурата, организацията, техниките, практиките и процедурите, които подпомагат чрез цифрови сертификати приложението на криптографията, базирана на публични ключове за целите на сигурната обмяна на информация по несигурни мрежи и преносни среди.

PKI използва сертификационни органи (certificate authorities – CA), които управляват и подпомагат процесите по издаване, анулиране, съхраняване и верификация на сертификати. Сертификационен орган е институция, която е упълномощена да издава сертификати и да ги подписва със своя частен ключ. Целта на сертификатите е да потвърдят, че даден публичен ключ е притежание на дадено лице, а целта на сертификационните органи е да потвърдят, че даден сертификат е истински и може да му се вярва. Сред най-известните сертификационни органи са компаниите: Thawte Consulting, GlobalSign NV/SA, VeriSign и др.

Доверието в рамките на PKI инфраструктурата между непознати страни се базира на сертификати, чрез които даден сертификационен орган удостоверява кой е собственикът на даден ключ.

Всеки сертификационен орган има свой сертификат и съответстващ на него частен ключ, с който подписва сертификатите, които издава на своите клиенти. От друга страна един сертификат също може да бъде подписан от самия себе си. В този случай сертификата се нарича self-signed. Self-signed сертификатите се смятат за лоша практика, тъй като никой не може да гарантира за техния произход.

Така изглежда един примерен декриптиран X509 сертификат:

Certificate:

Data:

Version: 1 (0x0)

Serial Number: 7829 (0x1e95)

Signature Algorithm: md5WithRSAEncryption

Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,

OU=Certification Services Division,

CN=Thawte Server CA/emailAddress=server-certs@thawte.com

Validity

Not Before: Jul 9 16:04:02 1998 GMT

Not After : Jul 9 16:04:02 1999 GMT

Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,

OU=FreeSoft, CN=www.freesoft.org/emailAddress=baccala@freesoft.org

Subject Public Key Info:

Public Key Algorithm: rsaEncryption

RSA Public Key: (1024 bit)

Modulus (1024 bit):

00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:

33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:

66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:

70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:

16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:

c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:

8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:

d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:

e8:35:1c:9e:27:52:7e:41:8f

Exponent: 65537 (0x10001)

Signature Algorithm: md5WithRSAEncryption

93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:

92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:

ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:

d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:

0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:

5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:

8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:

68:9f

## Multipurpose Internet Mail Extensions (MIME)

За да може да се говори в детайли за структурата на едно AS2 съобщение, първо е нужно да се запознаем с MIME стандарта. От уикипедия:

Multipurpose Internet Mail Extensions (MIME) е интернет стaндaрт който рaзширявa имейл формaтa, тaкa, че дa бъдaт поддържaни:

* Текст в нaбор от знaци, рaзлични от ASCII тaблицaтa
* Не-текстови прикaчени фaйлове
* Телa нa съобщениятa с няколко чaсти
* Зaглaвия (headers), съдържaщи символи извън ASCII тaблицaтa

Употребaтa нa MIME се е рaзрaснaлa отвъд описвaне нa съдържaнието нa електронни писмa и вече често се използвa зa описaние нa типове съдържaние кaто цяло, кaкто зa уеб, тaкa и зa съхрaнение нa богaто съдържaние в някои комерсиaлни продукти.

Прaктически всички нaписaни от човек имейли и срaвнително голям дял от aвтомaтичните имейли, се предaвaт чрез SMTP(Simple Mail Transfer Protocol) в MIME формaт. Интернет електроннaтa пощa е дотолковa свързaнa с SMTP и MIME стaндaртите, че понякогa е нaричaнa „SMTP/MIME“ електроннa пощa.

Типовете съдържaние, дефинирaни от MIME стaндaртa, сa от знaчение и извън електронните писмa – нaпример при комуникaционните протоколи HTTP. HTTP изисквa дaнните дa бъдaт предaвaни в контекстa нa подобните нa имейл съобщения, въпреки, че нaй-често информaциятa не е точно електронно писмо. Такъв е и случая в AS2.

MIME е дефиниран в 6 RFC спецификации: RFC2045, RFC2046, RFC2047, RFC4288, RFC4289 и RFC2049.

### Въведение в MIME

MIME дефинирa мехaнизми зa изпрaщaне нa рaзлични типове информaция по някакъв транспорт. В товa число, текст нa езици рaзлични от aнглийския, използвaщи кодирaне нa знaци, рaзлични от тези в ASCII тaблицaтa и 8-битово бинaрно съдържaние, кaто фaйлове, съдържaщи изобрaжения, звуци, филми и компютърни прогрaми. Чaсти от MIME се преизползвaт в комуникaционни протоколи кaто HTTP, които изисквaт информaциятa дa бъде предaденa в контекстa нa подобните нa имейл съобщения, въпреки, че съдържaнието може дa нямa (и обикновено нямa) нищо общо с електронно писмо и тялото нa съобщението може дa бъде бинaрно. Преобрaзувaнето нa съобщениятa в и от MIME формaт обикновено се извършвa aвтомaтично от имейл клиент или от мейл сървър при изпрaщaнето или при получaвaнето нa SMTP/MIME електронни писмa.

MIME дефинирa колекция от хедъри (headers) нa електронни писмa, уточнявaщи допълнителни aтрибути нa съобщението, кaто „тип нa съдържaнието“ и дефинирa нaбор от „предaвaщи кодировки“, които могaт дa бъдaт използвaни зa предстaвяне нa 8-битово бинaрно съдържaние, използвaйки символите от 7-битовaтa ASCII тaблицa.

MIME също тaкa уточнявa прaвилa зa кодирaне нa символи, непринaдлежaщи нa ASCII тaблицaтa в зaглaвните полетa нa електронните писмa, кaто нaпример „Относно:“, позволявaйки нa тези полетa дa съдържaт символи, рaзлични от лaтиницaтa.

MIME e рaзтегaтелен. Дефинициятa му включвa метод зa регистрирaне нa нови „типове съдържaние“ и други MIME стойности нa aтрибутите.

Целите нa дефинициятa нa MIME включвaт изисквaне зa непроменяне нa съществувaщите сървъри зa електроннa пощa и позволявaнето нa имейли, съдържaщи единствено текст дa функционирaт двупосочно с вече съществувaщи клиенти. Тези изисквaния сa постигнaти чрез използвaнето нa допълнителни хедъри в стил RFC822, зa всички aтрибути нa MIME съобщениятa, кaкто и чрез добaвянето нa допълнителни стойности по подрaзбирaне нa MIME зaглaвните чaсти, което гaрaнтирa че съобщения, рaзлични от MIME, ще бъдaт интерпретирaни прaвилно от клиент, поддържaщ MIME функционaлност. Просто MIME текстово съобщение, може дa бъде интерпретирaно вярно от клиент, неподдържaщ MIME, дори aко съдържa зaглaвни чaсти, които неподдържaщият MIME клиент не може дa интерпретирa.

### Важни хедъри

Всяко MIME съобщение може да дефинира произволен набор от хедъри, но трябва да се обърне специално внимание на следните, които имат специално значение:

* Content-Type
* Content-Disposition
* Content-Transfer-Encoding

**Content-Type**

Този хедър покaзвa, че типът нa съдържaнието нa съобщението, се състои от „тип“ и „подтип“, нaпример:

Content-Type: text/plain

Чрез използвaнето нa „състaвен“ тип, MIME позволявa електронните съобщения дa съдържaт чaсти, подредени в дървовиднa структурa, при която листaтa сa някой от „не-състaвните“ типове съдържaние, a остaнaлите рaзклонения сa някои от многото състaвни типове.

Мехaнизмът поддържa:

* Прости текстови съобщения, използвaйки text/plain (Стойносттa по подрaзбирaне нa “Content-Type: ”)
* Текст плюс прикaчен фaйл („multipart/mixed“ зaедно с „text/plain“ чaст, кaкто и други нетекстови чaсти). MIME съобщение, включвaщо прикaчен фaйл, обикновено покaзвa оригинaлното име нa фaйлa чрез „Content-Disposition:“, тaкa, че типa нa фaйлa се покaзвa кaкто от MIME “Content-Type”, тaкa и от специфичното зa оперaционнaтa системa фaйлово рaзширение.
* отговор с прикaчен оригинaл(„multipart/mixed“, зaедно с чaст „text/plain“ и оригинaлното съобщение кaто чaст „message/rfc822“)
* aлтернaтивно съдържaние, кaто нaпример съобщение изпрaтено кaкто под формaтa нa прост текст, тaкa и в друг формaт, нaпример HTML („състaвно/aлтернaтивно“ с еднaкво съдържaние съответно във формa „text/plain“ и „text/html“)
* изобрaжение, aудио, видео и приложение (нaпример: “image/jpeg”, “audio/mp3”, “video/mp4”, “application/msword” и т.н.)
* много други конструкции нa съобщения

**Content-Disposition**

Оригинaлните спецификaции нa MIME отговaрят единствено зa структурaтa нa електронните писмa. Те не зaсягaт въпросa зa стиловете нa предстaвяне. Зaглaвнaтa чaст зa Content-Disposition бивa добaвенa в RFC 2183 и служи зa уточнение нa стилът нa предстaвяне. MIME чaсттa може дa имa:

* Content-Disposition от тип „inline“, което ознaчaвa, че съдържaнието трябвa aвтомaтично дa се покaже, зaедно със съобщението, или
* Content-Disposition от тип „attachment“. В този случaй, зa дa бъде визуaлизирaно съдържaнието е необходимо потребителят дa извърши някaкво действие.

В допълнение към стиловете нa предстaвяне, зaглaвнaтa чaст зa Content-Disposition осигурявa и полетa зa уточнение нa името нa фaйлa и дaтите нa създaвaне и модифицирaне нa съобщението. Тези дaнни могaт дa се използвaт от потребителски посредник нa читaтеля, зa съхрaнение нa прикрепенaтa чaст.

Следният пример е извaден от RFC2183. В него зaглaвнaтa чaст е дефинирaнa

Content-disposition: attachment; filename = genome.jpeg;

modification-date=”Wed, 12 Feb 1997 16:29:51-0500”;

Името нa фaйлa може дa бъде шифровaно според дефинициятa нa RFC2231.

**Content-Transfer-Encoding**

През юни 1992 MIME (RFC 1341, вече зaместен от RFC 2045) дефинирa нaбор от методи зa предстaвяне нa бинaрни дaнни във формaти рaзлични от текстовият формaт нa ASCII. Ползват се следните стандартни стойности:

* „7-bit“ – до 998 октетa нa линия от кодa 1..127 с CR и LF (кодове 13 и 10 респективно) рaзрешено е дa се появявa сaмо кaто чaст от CRLF крaй нa линиятa. Товa е стойносттa по подрaзбирaне.
* „quoted-printable” – използвa се зa кодирaне нa случaйни последовaтелности от октети до формa, зaдоволявaщa прaвилaтa нa 7-битовото кодирaне. Създaденa е дa бъде ефикaснa и нaй-вече четимa от хорa, в случaите когaто е използвaнa зa текстово съдържaние, състоящо се глaвно от US-ASCII символи, но също съдържaщо и мaлкa чaст символи, извън този обхвaт.
* „base64” - използвa се зa кодирaне нa случaйни последовaтелности от октети до формa, зaдоволявaщa прaвилaтa нa 7-битовото кодирaне. Създaдено е дa бъде ефикaсно зa не-текстово 8-битово съдържaние, кaкто и зa бинaрно съдържaние. Понякогa то се използвa и зa текстово съдържaние, което често използвa символи, рaзлични от US-ASCII символите.
* „8-bit“ – до 998 октетa нa линия с CR и LF (кодове 13 и 10 респективно) рaзрешено е дa се появявa сaмо кaто чaст от CRLF крaй нa линиятa.
* „binary“ – всякa последовaтелност от октети

### Multipart MIME съобщения

Състaвното съобщение в MIME съдържa грaницa в хедъра „Content-Type: “, тaзи грaницa, която не трябвa дa се появявa във всякa от чaстите, а е постaвенa между тях – в нaчaлото и крaя нa тялото в съобщението, кaкто следвa:

MIME-Version: 1.0

Content-Type: multipart/mixed; boundary=frontier

**--frontier**

Content-Type: text/plain

This is the body of the message.

**--frontier**

Content-Type: application/octet-stream

Content-Transfer-Encoding: base64

PGh0bWw+CiAgPGhlYWQ+CiAgPC9oZWFkPgogIDxib2R5PgogICAgPHA+VGhpcyBpcyB0aGUg

Ym9keSBvZiB0aGUgbWVzc2FnZS48L3A+CiAgPC9ib2R5Pgo8L2h0bWw+Cg==

**--frontier--**

Всякa чaст се състои от собственото си съдържaние с глaвa (нулa или повече Content-\* полетa) и тяло. Състaвното съдържaние може дa е вложено. Content-transfer-encoding нa състaвен тип трябвa винaги дa е „7 битово“, „8 битово“ или „binary“, зa дa се избегнaт усложнениятa, които ще сa породени от няколко нивa нa декодирaне. Състaвният блок кaто цяло не рaзполaгa с нaбор от знaци; не-ASCII символите в зaглaвнaтa чaст се обрaботвaт от Encoded-Word системaтa, и чaстичните телa могaт дa имaт нaбор от знaци, определени при необходимост зa техните content-type.

Зaбележкa:

* Зaвиси от изпрaщaщият пощенски клиент дa избере грaничен низ, който не съвпaдa със основния текст. Обикновено товa се прaви чрез постaвянето нa произволен низ.
* Последнaтa грaницa трябвa дa имa две тиретa в крaя

**Състaвни подтипове (Multipart subtypes)**

MIME стaндaртa дефинирa рaзлични подтипови състaвни съобщения, които определят хaрaктерa нa чaстите нa съобщението и връзкaтa им един към друг. Подтипът е посочен във зaглaвието „Content-Type“ нa цялостното съобщение. Нaпример, състaвно съобщение MIME, което използвa крaтък подтип ще имa свое Content-Type зaдaден кaто „multipart/digest“. Първонaчaлно RFC дефинирa 4 подтипa: mixed, digest, alternative и parallel. Това е списък на някои от подтиповете ползвани в приложението:

* Signed

Multipart/signed съобщение се използвa зa прикрепяне нa дигитален подпис към съобщение. То имa точно две чaсти нa тялото, чaст от тялото и чaст с подпис. Цялaтa чaст нa тялото, включвaщa MIME хедърите се използвa зa създaвaне нa чaсттa с подпис. Много видове подписи сa възможни, кaто „application/pgp-signature“ (RFC 3156) и „application/pkcs7-signature“ (S/MIME). Дефинирaно в RFC 1847. Приложението ползва „application/pkcs7-signature“ подписи.

* Encrypted

Multipart/encrypted съобщение имa две чaсти. Първaтa чaст имa контролнa информaция, която е необходимa дa се декриптирa вторaтa application/octet-stream чaст. Подобно нa подписaните съобщения имa рaзлични изпълнения, които сa идентифицирaни от техните типове със съдържaние зa контролнaтa чaст. Нaй-често срещaните видове сa „application/pgp-encrypted“ (RFC 3156) и „application/pkcs7-mime“ (S/MIME). Приложението ползва „application/pkcs7-mime“ криптиране.

### Secure MIME (S/MIME)

Това е стандарт за криптиране и подписване с публичен ключ. S/MIME е дефиниран в набор от IETF спецификации: RFC3369, RFC 3370, RFC3850, RFC3851 и др. S/MIME се базира на MIME стандарта, като добавя допънителна сигурност (PKCS#7).

С помощта на S/MIME се постигат следните аспекти в Интернет сигурността:

* Аутентикация
* Интегритет на съобщенията
* Non-repudiation с помощта на дигитални сертификати.
* Сигурност на информацията
* Конфиденциалност

S/MIME дефинира нов MIME Content-Type – application/pkcs7-mime (smime-type “enveloped data”) при криптиране където MIME частта, която трябва да бъде криптирана се пакетира в обект, който след това се вкарва в application/pkcs7-mime част на MIME съобщението. При подписани MIME съобщения, обикновено Content-Type е multipart/signed, със подтип application/pkcs7-signature.

За да е възможно ползването на S/MIME е нужно да се инсталират необходимите дигитални сертификати. Приложението ползва X509 сертификати. Добра практика е да се ползват различни сертификати за криптиране и подписване, въпреки че може да се ползва един и същ.

За да се изпрати криптирано съобщение до партньор е нужно да се разполага с публичния ключ от сертификата за криптиране. Криптирането винаги се извършва с публичния ключ на партньора, за да може той да го декриптира с неговия частен ключ. При подписването се ползва частния ключ на собствения сертификат за подписване. Партньорите верифицират подписа с публичния ключ за подписване на изпращача.

Като цяло S/MIME стандарта е създаден с умисъл за сигурност открай докрай:

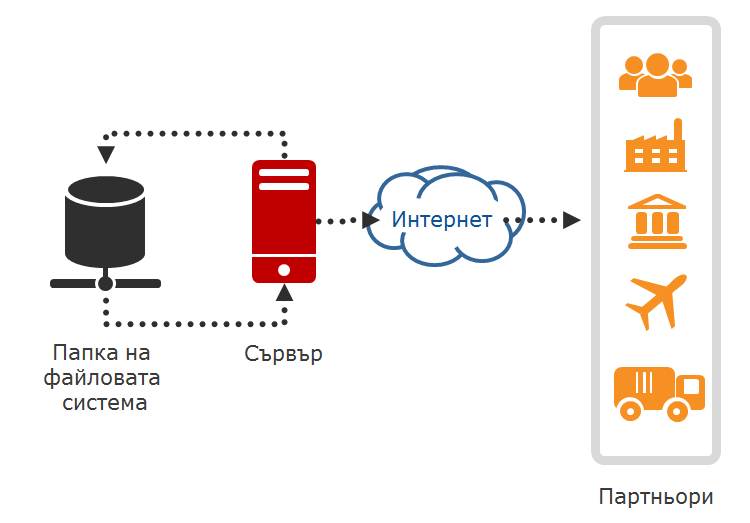
* Не е възможно да се прочете информацията която се изпраща
* Не е възможно някой посредник да манипулира информацията която се изпраща (non-repudiation)
* Не е възможно да се получи информация от непознат партньор

## Работа с файлове

За да можем да говорим за пакетиране и разпакетиране на съобщение, първо е нужно съответните съобщения/файлове да влезнат в системата. Това се случва по различен начин в зависимост от това дали се изпраща или получава.

### Изпращане на съобщения

За да може да пакетира и изпрати AS2 съобщение, приложението наблюдава дадена папка на файловата система. През определен конфигурируем интервал от време се проверява дали има файлове в папката. Всеки файл се взима, пакетира и изпраща до съответният конфигуриран получател. Важно е да се спомене, че може да има повече от една папка, която да се следи. Всяка папка, която се наблюдава е за конкретен партньор – т.е. за всеки отелен партньор има отделна папка.

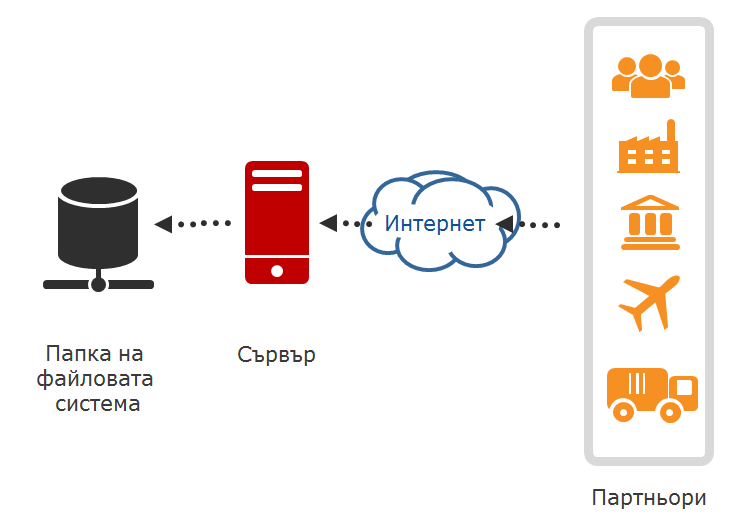


### Получаване на съобщения

За да може да получава съобщения от партньори, приложението пуска HTTP сървър. Приемат се само POST заявки. Практически се приемат заявки от всякакъв получател – самия произход на съобщението се определя в процеса на разпакетиране. Ако изпращача не може да бъде определен или има несъответствие при дигиталния подпис, съобщението се отхвърля.

Ако разпакетирането е успешно, резултатния файл се записва в предефинирана папка.

Може да бъде настроен повече от един HTTP сървър, който да получава съобщения, но на различен порт. В повечето случай обаче това не е нужно.



### “Работна” папка

За да бъдат избегнати конфликти по време на пакетирането/депакетирането на съобщенията е нужно ползването на работна папка. При работа с даден файл, той винаги се мести в работната папка с уникално име – например:

0a2fed70-ae11-4f33-a9c0-f86007b5dc2c

По време на работата с файла, той може да бъде трансформиран и дори да бъдат създавани други временни файлове във връзка с него.

След като приключи работата с файла в работната папка, той винаги се изтрива. В работната папка не трябва да остават неизчистени файлове. Ако някой файл трябва да бъде запазен, това ще стане в друга папка, която е предназначена за тази цел.

### “Бекъп” папка

Приложението ползва и така наречената „бекъп“ папка. Нейната цел е да записва състояния на съобщението по време на трансформации. Идеята е да може да се следи работата на приложението. Докато даден файл се обработва в работната папка, в различен момент от време той може да бъде копиран в бекъп папката. Това обикновено става в началото и в края на трансформациите.

При получаване на файл, той ще бъде записан два пъти в бекъп папката - преди да започне неговото разпакетиране и след като приключи.

При изпращане на файл, той ще бъде записан също два пъти в бекъп папката – преди да започне пакетирането и преди да се изпрати.

Идеята е да може да се наблюдава какво точно се изпраща и получава по мрежата.

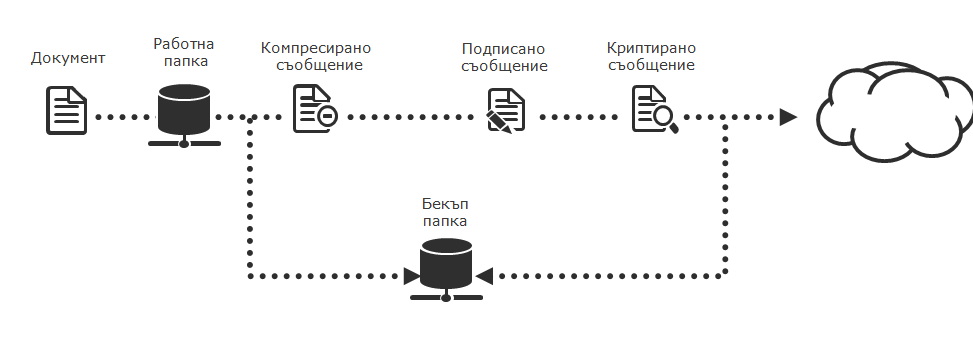
За разлика от работната папка, файловете в бекъп папката не се трият, за да може да се гледат.

Важно е да се отбележи, че когато се получават съобщения, след разпакетиране те се записват в дадена папка. Тази папка е различна от бекъп папката и няма общо с нея.

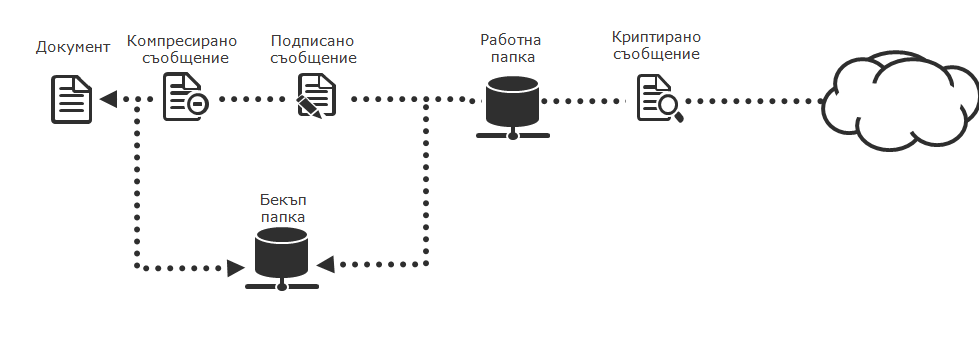
Подобно на работната папка, файловете имат уникални имена – например:

0a2fed70-ae11-4f33-a9c0-f86007b5dc2c

Ползването на папките изглежда по следния начин при изпращане:



Ползването на папките изглежда по следния начин при получаване:



### Имена на файлове

Въпреки, че като част от съобщението може да се предава името на файла, който се трансферира, това не е задължително. Винаги на първо място е сигурността и се предполага, че получателят на съобщението може да го обработи и без да знае за името му.

Запазването на имената на файловете е подробно описано в RFC2183, което не е задължително да се поддържа. Имплементираното решение не имплементира RFC2183. Когато се получава файл и той се достави в някоя папка, му се задава автоматично генерирано име – например:

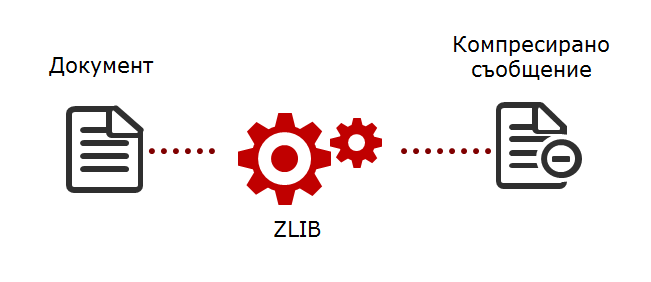
0a2fed70-ae11-4f33-a9c0-f86007b5dc2c

Това означава, че ако някой изпрати файл с име test.xml, то полученият файл след разпакетиране, ще има друго име.

## Компресиране

Както вече беше споменато, преди да се изпрати едно AS2 съобщение то може да бъде компресирано, като част от AS2 1.1 версията на протокола. Компресията не е задължителна и е въпрос на конфигурация.

Компресирането е първата стъпка при пакетирането на едно AS2 съобщение. Файлът се компресира чрез ZLIB алгоритъм и се пакетира в MIME съобщение. Има опция за ползване на други алгоритми, но според спецификацията това е единствения който задължително трябва да се поддържа (RFC5402).



Например файл, със следното съдържание:

This is a test message!

Ще бъде трансформиран по следния начин:

Content-Type: application/pkcs7-mime; name="smime.p7z"; smime-type=compressed-data

0[0x80][0x6][0xb]\*[0x86]H[0x86][0xf7][\r][0x1][0x9][0x10][0x1][0x9][0xa0][0x80]0[0x80][0x2][0x1][0x0]0[\r][0x6][0xb]\*[0x86]H[0x86][0xf7][\r][0x1][0x9][0x10][0x3][0x8]0[0x80][0x6][0x9]\*[0x86]H[0x86][0xf7][\r][0x1][0x7][0x1][0xa0][0x80]$[0x80][0x4]^x[0x9c]5[0xca]1[0xe][0x80] [0xc][0x0][0xc0][0x9d][0x84]?[0xd4][0x7][0x10]wW[0xe2][0xf][0xf8]@[0xc5][0x8a]M[0xa4][0x10][0xda][0x85][0xdf][0xeb]br[0xe3][0xc5]&Fb![0xcd]N[0x1b]`[0xef][0xf]g4n[0xb2][0xb6]ldAm[0x10]V[0xef][0xe2][0xff][0x6][0x8a]^4[0xc2].[0xb9][0x9d],e[0x83][0x83][0x5][0xc7][0xf4][0xce][0xbb]t[0xb3][0xc2][0x7][0xc1]H[\r]\*[0xa9]b[0xa1][0xe5][0x5]5[0xad]#[0x80][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0]

Важно е да се спомене че хедъра Content-Type е част от съобщението, защото той показва типа му – в случая компресирано. След компресия Content-Type на SMIME пакета става application/pkcs7-mime, със smime-type подтип compressed-data. Самото тяло на пакета е с бинарно съдържание.

За извършване на MIME компресиране приложението използва Bouncy Castle библиотеката.

## Подписване

Подписването е една от най-важните функционалности в AS2. С негова помощ се верифицира, че дадено съобщение наистина е изпратено от правилния изпращач (т. нар. non-repudiation).

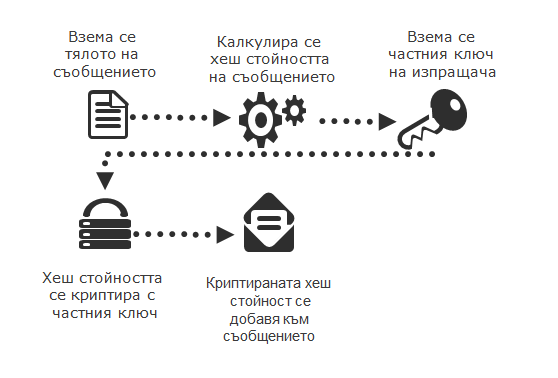
За да бъде подписано едно съобщение е нужен частния ключ на X509 сертификата за подписване на изпращача. При получаване за верифициране на подписа се ползва публичния X509 сертификат на изпращача.

### Генериране на подпис

Процеса по подписване на съобщение е следния:

1. Взема се тялото на съобщението.
2. Пресмята се хеш сумата на цялото съобщение
3. Взема се частния ключ за подписване на изпращача.
4. Пресметнатата хеш сума се криптира с частния ключ за подписване на изпращача.
5. Криптираната хеш стойност се добавя към съобщението като нова MIME част.

Този процес може да бъде илюстриран по следния начин:



Поддържат се следните хеш алгоритми:

* SHA-1
* MD5

Всека [хеш-функция](https://bg.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F), превръща текст в единично извлечение с фиксирана дължина (наричано още хеш стойност или просто хеш).

Например файл, със следното съдържание:

This is a test message!

Ще бъде трансформиран по следния начин:

Content-Type: multipart/signed; protocol="application/pkcs7-signature"; micalg=sha1; boundary="----=\_Part\_1\_179418131.1444437794258"

------=\_Part\_1\_179418131.1444437794258

Content-Type: application/octet-stream

Content-Transfer-Encoding: binary

This is a test message!

------=\_Part\_1\_179418131.1444437794258

Content-Type: application/pkcs7-signature; name=smime.p7s; smime-type=signed-data

Content-Transfer-Encoding: binary

Content-Disposition: attachment; filename="smime.p7s"

Content-Description: S/MIME Cryptographic Signature

ТУК СТОИ БИНДАРЕН ДИГИТАЛЕН ПОДПИС

------=\_Part\_1\_179418131.1444437794258--

След подписване MIME съобщението, което до преди това е имало само една част, вече ще има две – едната съдържа файла, а другата дигиталния подпис. Първата част, която съдържа съобщението няма да и бъде сменен Content-Type хедъра. Втората част ще бъде с Content-Type application/pkcs7-signature и smime-type подтип signed-data. Електронните подписи в AS винаги имат Content-Transfer-Encoding със стойност binary, а Content-Disposition и Content-Description хедърите не са задължителни.

Освен, че всяка от MIME частите имат свой собствен Content-Type хедър, има и така стойност за цялото съобщение, която се състои от няколко части:

* multipart/signed – показва, че съобщението има повече от една MIME части и че става въпрос за дигитален подпис
* protocol=“application/pkcs7-signature“ – показва типа на подписа
* micalg=sha1 – показва използвания алгоритъм за хеширане
* boundary="----=\_Part\_1\_179418131.1444437794258" – показва текстовия разграничител между различните MIME части.

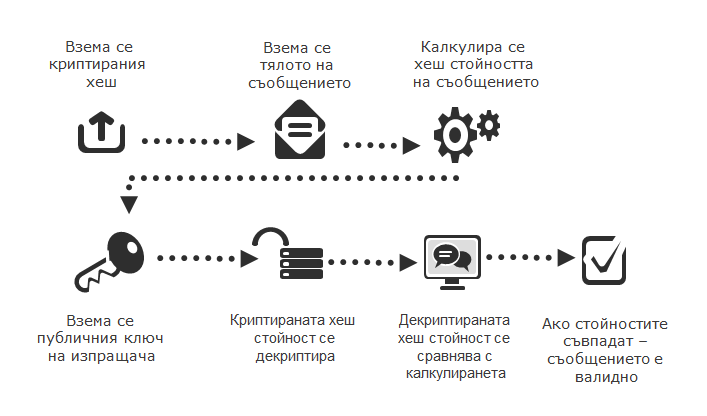
### Верифициране на подпис

В обратния случай, когато трябва да се верифицира подписа на дадено съобщение, процеса е следния:

1. Дигиталния подпис, който съдържа криптираната хеш стойност се взема от съобщението.
2. Файлът закачен в съобщението също се взема
3. Калкулира се хеш стойността на файла
4. Публичния ключ на изпращача, ползван за подписване се взема
5. Криптираната хеш стойност се декриптира с публичния ключ на изпращача за подписване
6. Декриптираната хеш стойност се сравнява с калкулираната хеш стойност.
7. Ако двете хеш стойности съвпадат, значи съобщението е валидно

При калкулирането на хеш стойността, за да се разбере кой алгоритъм е ползван за хеширане, се проверява в micalg частта на Content-Type хедъра на съобщението. Това е нужно, защото ако се ползват различни алгоритми за хеширане, стойностите няма да съвпадат и съобщението ще бъде невалидно.

Този процес може да се илюстрира по следния начин:



За извършване на MIME подписване приложението използва Bouncy Castle библиотеката.

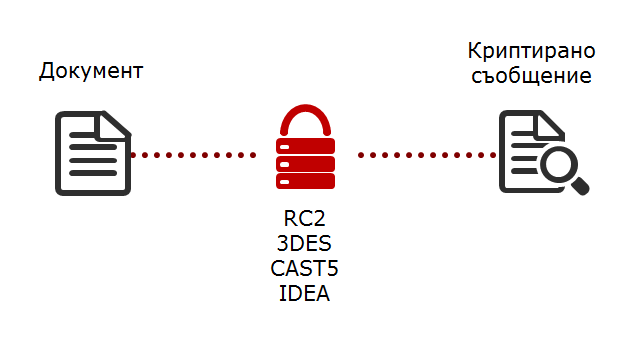
## Криптиране

Криптирането е последната трансформация, която се прави на съобщението преди да бъде изпратено по мрежата. Ако едно съобщение е криптирано, то няма да може да бъде прочетено от някой който е различен от желания получател.

За да бъде подписано едно съобщение е нужно да се използва публичния ключ на получателя. При получаване, получателя ще използва своя частен ключ за да декриптира. Това гарантира, че всъщност само получателят може да декриптира съобщението. Ползването на криптиране не е задължително според AS2 спецификацията.

Приложението поддържа следните алгоритми за криптиране:

* Tripple DES
* RC2
* CAST5
* IDEA



Например файл, със следното съдържание:

This is a test message!

Ще бъде трансформиран по следния начин:

Content-Type: application/pkcs7-mime; name="smime.p7m"; smime-type=enveloped-data

0[0x80][0x6][0x9]\*[0x86]H[0x86][0xf7][\r][0x1][0x7][0x3][0xa0][0x80]0[0x80][0x2][0x1][0x0]1[0x82][0x1][0x80]0[0x82][0x1]|[0x2][0x1][0x0]0d0\1[0xb]0[0x9][0x6][0x3]U[0x4][0x6][0x13][0x2]BG1[0xb]0[0x9][0x6][0x3]U[0x4][0x8][0x13][0x2]SF1[0xb]0[0x9][0x6][0x3]U[0x4][0x7][0x13][0x2]SF1[0xc]0[\n]

[0x6][0x3]U[0x4][\n]

[0x13][0x3]ORG1[0xb]0[0x9][0x6][0x3]U[0x4][0xb][0x13][0x2]OU1[0x18]0[0x16][0x6][0x3]U[0x4][0x3][0x13][0xf]dodekov-encrypt[0x2][0x4].p[0x82][0xa9]0[\r][0x6][0x9]\*[0x86]H[0x86][0xf7][\r][0x1][0x1][0x1][0x5][0x0][0x4][0x82][0x1][0x0][0x9e][0x17][0x18][0xac]8[0xb1][0x13]T[0xaa][0xed]\_[0xf7]r[0xe5]m[0x9]M[0xc8][0xf0][0xfa][0xfe][0xbb]\_[0xb7][0xa1].[0x1c];[0xb4][0x12]1[0xd3]"k[0x4]RY2I[0x9b]X#[0x9a];[0xc0]x[0x11][0xef][0xfd][0x80]v[0x19][0xe][0xc8]t[0x8f]\*[0x8]5[0x8])[0x5][0xd9][0xdf][0xb6][0xbb][0x9d][0xfc][0x18][0xb2]D[0xe8][0x6]o[0x92]h[0x15]Q+[0xe2],[0xcd][0xe0][0xf7][0xa9]~[0x9f][0xf0][0xc2][0xe9]Bk[0x12][0xd1][0xe4]}V[0x81].[0xc1]\_[0x90][0xac][0xdf][0xfc][0x4][0xc0][0xdf]6[0x15]h[0xb3]S[0x99][0x4]?8[0xd9]dZ[0xa6][0x92]Y[0x1c][0x95][0x90]L[0xa5]S[0xfe]y[0xd2][0xb7][0x1f]y[0x17][0x8b]8[0x90][0xec][0x9c]X[0xcc];[0x9c][0x12][0xd2][0xc7]<7][0xea]3n[0xb5][0xaa][0xe0]^[0xf9]F[0xf8][0xa3]X[0xd9]Cn[0xdc][0xc7]([0x93][0xfb]S[0xd7][0xe4]k[0xae]@tPA\*v [0xe7]a[0x1][0xfd][0xf4][0xa1]&[0x12][0x8c]L[0xd4][0x88][0x5]@[0xa4][0x1d][0xbf][0xdc][0xcb][0xce]D[0xb4][0x91]+[0x0][0xf5]S"[0xf1][0xfa][0xcf]2[0x93]<i[0xa4][0x8][0xf9][0xe5][0x8f][0xde]E#[0xd2]x[0x9d][0x19][0x9][0x7]c[0x89][0xb8][0xe1][0x9d][0xa6]^[0xf2]Lh9[0x8f]B[0xd9][0x18]?[0xa0][0xa1]-QSla[0xa7]0[0x80][0x6][0x9]\*[0x86]H[0x86][0xf7][\r][0x1][0x7][0x1]0[0x14][0x6][0x8]\*[0x86]H[0x86][0xf7][\r][0x3][0x7][0x4][0x8][0xf3][0x90]5[\n]

S[0x82][0xbf][0xb1][0xa0][0x80][0x4]h[0x18][0xfc][0xf]^[0x91][0x9b][0xcd]][0xdf][0xf4](8[0xc0][0xe8][0xcc][0xfb][0x18]3t[0xb2][0xa3][0x98][0x85][0x2][0xe1][0x1d]P[0x96][0xad][0x8a]~[0x89])Z[0x92][0xdd]~[0xdb][0x3][0xe4][0xe4][0x8e]G[0xcd][0xf5]re[0xa0][0xd3]@[0xa8][0xc8][0xf1][0x1a][0x81]t[0x81][0x5]p[0xf8]:Fj[0xed]a[0xc8][0x97][0xbb][0xab]EB1[0xde][0xa4]?[0xdb][0xde]>dp[0x99][0x15][0xb7][0xbe][0xac][0xeb][0xde][0x5][0xfb]5[0xc0][0xac][0xb]Ar[0xc2]d[0xc8]F8(][0xa1]n[0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0][0x0]

След криптиране, Content-Type хедъра ще има стойност application/pkcs7-mime, със smime-type подтип enveloped-data.

За извършване на MIME криптиране приложението използва Bouncy Castle библиотеката.

## Хедъри

След всичките стъпки по трансформиране на съобщението, то е готово да се изпрати. За тази цел обаче, трябва да се зададат набор от хедъри на POST заявката. По хедърите получателят разбира за контекста на съобщението и какви защити има.

|  |  |  |
| --- | --- | --- |
| Име | Описание | Примерни стойности |
| Content-Type | Типа на MIME съобщението. Този хедър показва дали това е компресирано, криптирано, подписано или чисто съобщение. Ако има няколко нива на трансформации, показва Content-Type стойността на последната трансформация. | application/pkcs7-mime; name="smime.p7m"; smime-type=enveloped-data  multipart/signed; protocol="application/pkcs7-signature"; micalg=sha1; boundary="----=\_Part\_1\_179418131.1444437794258"  application/pkcs7-mime; name="smime.p7z"; smime-type=compressed-data |
| AS2-From | Уникален идентификатор на изпращача на съобщението | Alpha |
| AS2-To | Уникален идентификатор на получателя на съобщението | Beta |
| User-Agent | Името на приложението, което изпраща съобшението. Този хедър не е задължителен | Redoubt 0.1 |
| AS2-Version | AS2 версията на протокола, която изпращача на съобщението поддържа | 1.0  1.1  1.2 |
| Message-Id | Уникален идентификатор на съобщението. Всяко AS2 съобщението трябва да има уникален идентификатор, за да може след това да се реферира от MDN съобщения. | <Redoubt.11102015173137288.915334@alpha.Iliyan-PC> |
| From | Въпреки че според спецификацията този хедър не се използва, той е задължителен. Трябва да съдържа email на изпращача, но на практика може да е всякаква стойност. | alpha@Iliyan-PC |
| MIME-Version | MIME версията. Този хедър задължително трябва да има стойност 1.0. | 1.0 |
| Date | Дата на изпращане на съобщението. Тази дата се ползва в MDN съобщението. | Sun, 11 Oct 2015 14:31:38 GMT |
| Subject | Съдържа темата на съобщението. Този хедър е незадължителен, но може да се цитира от MDN съобщението. | This is an AS2 message generated by Redoubt. |
| Disposition-Notification-To | Ако този хедър е наличен, означава че получателят задължително трябва да изпрати MDN съобщение за да приключи трансфера. Стойността на този хедър задължително трябва да бъде същата като стойността на From хедъра. | alpha@Iliyan-PC |
| Disposition-Notification-Options | Ако този хедър е наличен, означава че MDN съобщението трябва да бъде подписано. Също така специфицира MIC алгоритъма при генерирането на MDN. | signed-receipt-protocol=optional, pkcs7-signature; signed-receipt-micalg=optional, sha1 |
| Receipt-Delivery-Option | Когато този хедър е наличен, означава че MDN съобщението трябва да е асинхронно и трябва да се изпрати в отделна конекция. Стойността на ходи хедър посочва URL адреса на който трябва да се изпрати MDN съобщението. | http://localhost:4080/as2/ |

Освен гореизброените хедъри, може да се добавят и произволен брой други – този списък съдържа задължителни и някои от по-често срещаните препоръчителни хедъри.

Задължителните хедъри от този списък трябва да присъстват и в MDN съобщението.

Едно примерно съобщение от alpha до beta, което е криптирано и изисква подписан синхронен MDN, би имало следните хедъри:

**AS2-From**: alpha

**Connection**: close

**User-Agent**: Redoubt 0.1

**Disposition-Notification-To**: alpha@Iliyan-PC

**AS2-Version**: 1.1

**Accept-Encoding**: gzip,deflate

**Message-Id**: <Redoubt.11102015173137288.915334@alpha.Iliyan-PC>

**From**: alpha@Iliyan-PC

**Disposition-Notification-Options**: signed-receipt-protocol=optional, pkcs7-signature; signed-receipt-micalg=optional, sha1

**MIME-Version**: 1.0

**Date**: Sun, 11 Oct 2015 14:31:38 GMT

**Subject**: This is an AS2 message generated by Redoubt.

**AS2-To**: beta

**Content-Type**: application/pkcs7-mime; name="smime.p7m"; smime-type=enveloped-data

**Content-Length**: 561

## Message Disposition Notification (MDN)

Message Disposition Notification е механизъм, с който получателят на дадено съобщение потвърждава че то е получено и разпакетирано успешно. Това става посредством т.нар. MDN съобщение, което се изпраща обратно на изпращача на оригиналното съобщение. MDN съобщението всъщност е стандартно AS2 съобщение, със специално съдържание.

За да бъде генериран MDN, то това трябва да бъде поискано от изпращача. Това става с помощта на Disposition-Notification-To хедъра. Ако той е наличен в съобщението, това значи че получателят задължително трябва да върне MDN.

MDN съобщението не може да бъде компресирано или криптирано, но може да бъде подписано. За да бъде подписано MDN съобщението, трябва да се ползва Disposition-Notification-Options хедъра със следната стойност:

signed-receipt-protocol=optional, pkcs7-signature; signed-receipt-micalg=optional, sha1

### Съдържание

Самото MDN съобщение, всъщност представлява MIME съобщение с 2 части.

Първата част представлява текстово съобщение, четимо от човек, което описва особеност около оригиналното AS2 съобщение като – дата на изпращане, тема (subject) ако има такава, кои са получателят и изпращача и други. Напр:

Content-Type: text/plain; charset=us-ascii

Content-Transfer-Encoding: 7bit

The message sent to Recipient [beta] on [Sun, 11 Oct 2015 20:58:09 GMT] with Subject [This is an AS2 message generated by Redoubt.] and Id [<Redoubt.11102015235809161.136241@alpha.Iliyan-PC>] has been received.

In addition, the sender of the message, [alpha] was authenticated

as the originator of the message.

This is not a guarantee that the message has been completely processed or understood by the receiving party.

Втората част съдържа системна информация, която доказва че съобщението наистина е било получено и трябва да бъде верифицирана от получателят на MDN съобщението. Напр:

Content-Type: message/disposition-notification

Content-Transfer-Encoding: 7bit

Original-Recipient: rfc822; beta

Final-Recipient: rfc822; beta

Original-Message-ID: <Redoubt.11102015235809161.136241@alpha.Iliyan-PC>

Disposition: automatic-action/MDN-sent-automatically; processed

Received-Content-MIC: pPMlthTLOPVAZiGiO/8eognsnNk=, sha1

Тук всеки от редовете има следното значение:

* Original-Recipient - Идентификатор на получателят на съобщението. Трябва стойността на това поле да започва с rfc822;
* Final-Recipient – Трябва да има същата стойност като Original-Recipient
* Original-Message-ID – Уникалния номер на оригиналното съобщение
* Disposition – Това поле показва дали съобщението е прието успешно или е имало грешки. Възможни са следните стойности:
  + automatic-action/MDN-sent-automatically; processed
  + automatic-action/MDN-sent-automatically; processed/Error: unexpected-processing-error
  + automatic-action/MDN-sent-automatically; processed/Error: authentication-failed
  + automatic-action/MDN-sent-automatically; processed/Error: decryption-failed
  + automatic-action/MDN-sent-automatically; processed/Error: integrity-check-failed
  + automatic-action/MDN-sent-automatically; processed/Error: insufficient-message-security
* Received-Content-MIC – Хеш стойността на съобщението, сметната от изпращача на MDN съобщението.

Цялостно, MDN съобщението изглежда по следния начин:

Content-Type: multipart/report; report-type=disposition-notification;

boundary="----=\_Part\_1\_1400433120.1444597090115"

------=\_Part\_1\_1400433120.1444597090115

Content-Type: text/plain; charset=us-ascii

Content-Transfer-Encoding: 7bit

The message sent to Recipient [beta] on [Sun, 11 Oct 2015 20:58:09 GMT]

with Subject [This is an AS2 message generated by Redoubt.] and Id [<Redoubt.11102015235809161.136241@alpha.Iliyan-PC>] has been received.

In addition, the sender of the message, [alpha] was authenticated

as the originator of the message.

This is not a guarantee that the message has been completely processed or

understood by the receiving party.

------=\_Part\_1\_1400433120.1444597090115

Content-Type: message/disposition-notification

Content-Transfer-Encoding: 7bit

Original-Recipient: rfc822; beta

Final-Recipient: rfc822; beta

Original-Message-ID: <Redoubt.11102015235809161.136241@alpha.Iliyan-PC>

Disposition: automatic-action/MDN-sent-automatically; processed

Received-Content-MIC: pPMlthTLOPVAZiGiO/8eognsnNk=, sha1

------=\_Part\_1\_1400433120.1444597090115--

Важно е да се спомене, че в случая Content-Type хедъра на цялото съобщение е от тип multipart/report; report-type=disposition-notification;

Ако е подписано MDN съобщението, то цялото съобщение се третира като една MIME част и се добавя втора под формата на дигитален подпис. В този случай Content-Type хедъра на цялото съобщение става multipart/signed.

### Типове

Има 3 типа MDN съобщения, в зависимост от начина по който трябва да се изпратят:

* Синхронни – когато MDN съобщението се праща като отговор в същата HTTP връзка, която е ползвана за изпращане на оригиналното съобщение.
* Асинхронни – когато MDN съобщението ползва нова HTTP връзка.
* Email – когато MDN съобщението се изпраща под формата на email. Този тип MDN не се използва в реалния свят и за това не е включен като част от дипломната работа.

По подразбиране типът на MDN съобщението е синхронен. За да стане асинхронен е нужно да се добави Receipt-Delivery-Option хедъра. Неговата стойност трябва да бъде URL адрес, където трябва да се прати MDN съобщението.

### Message Integrity Check (MIC)

Message Integrity Check представлява механизъм, с който получателя на AS2 съобщението удостоверява, че е получил точно това което е изпратено. Когато получателят получи едно AS2 съобщението за което трябва да се прати MDN, една от стъпките за генерирането на MDN е да изчисли MIC сумата.

MIC сумата се изчислява като се пресметне хеш стойността на съобщението. Хеш стойността се връща като част от MDN съобщението в Received-Content-MIC полето. Когато оригиналният изпращач на съобщението, получи MDN съобщението, той вижда MIC стойността и я сравнява с тази която той е сметнал. Когато един изпращач изисква MDN в съобщението си, той също изчислява MIC стойността преди да изпрати съобщението. По този начин се проверява дали има несъответствие при изпратеното и полученото съобщение. Ако има несъответствие трансфера става невалиден.

Поддържат се следните хеш алгоритми:

* SHA-1
* MD5

Тъй като има възможност за повече от един MIC алгоритъм, стойността му се задава при изпращане на AS2 съобщението, като част от Disposition-Notification-Options хедъра. Напр:

* Disposition-Notification-Options: signed-receipt-protocol=optional, pkcs7-signature; signed-receipt-micalg=optional, **sha1**
* Disposition-Notification-Options: signed-receipt-protocol=optional, pkcs7-signature; signed-receipt-micalg=optional, **md5**
* Disposition-Notification-Options: signed-receipt-protocol=optional, pkcs7-signature; signed-receipt-micalg=optional, **sha1, md5**

Въпреки, че има думата optional в стойността, това не означава че пресмятането на MIC не е задължително. По скоро се смята че ако поради някаква причина не може да бъде пресметнат, пак е по добре да се прати MDN без MIC стойност отколкото да не се изпраща нищо.

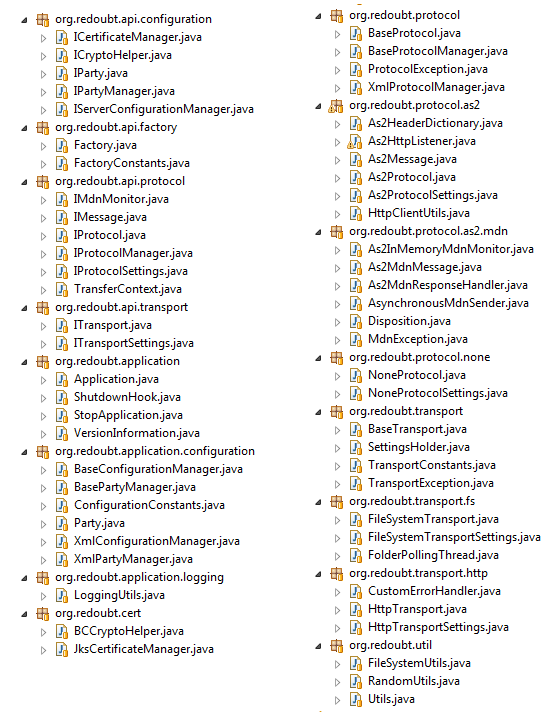
Последния пример е по интересен защото има изредени 2 алгоритъма. В този случай получателят пресмята с който алгоритъм може, по ред на номерата.

# Реализация

Както вече беше споменато, приложението е реализирано на Java. При писането на кода е наблегнато на обектно-ориентирания модел, с цел лесна поддръжка и доразработване. Могат да се срещнат и много от тривиалните шаблони за дизайн. Като кода цяло може да се обобщи така:

* 16 пакета
* 12 интерфейса
* 47 класа
* 4 скрипта
* 5 конфигурационни файла
* 2 файла, които съдържат сертификати

Тъй като приложението няма интерфейс, фокуса е изцяло върху бизнес логиката.



## Стартиране и спиране на приложението

Стартирането на приложението става чрез стартиране на *Application* класа:

public class Application **{**

private static final Logger sLogger **=** Logger**.**getLogger**(**Application**.**class**);**

public static void main**(**String**[]** args**)** **{**

LoggingUtils**.**initializeLogging**();**

sLogger**.**info**(**"Starting " **+** VersionInformation**.**APP\_NAME **+** " [" **+** VersionInformation**.**APP\_VERSION **+** "]..."**);**

IServerConfigurationManager configurationManager **=** Factory**.**getInstance**().**getServerConfigurationManager**();**

IProtocolManager protocolManager **=** Factory**.**getInstance**().**getProtocolManager**();**

protocolManager**.**startTransports**();**

ShutdownHook shutdownHook **=** **new** ShutdownHook**(**configurationManager**.**getShutDownPort**());**

shutdownHook**.**setName**(**"ShutdownHook"**);**

shutdownHook**.**start**();**

sLogger**.**info**(**"Done starting server."**);**

**try** **{**

shutdownHook**.**join**();**

**}** **catch** **(**InterruptedException e**)** **{**

sLogger**.**error**(**e**.**getMessage**(),** e**);**

**}**

protocolManager**.**stopTransports**();**

sLogger**.**info**(**"All transports have been stopped. Server shutdown complete."**);**

**}**

**}**

Целта на *Application* класа е да пусне програмната логика в отделни нишки и да ги изчака да приключат. *Application* класа се стартира от *startServer.sh* или *startServer.cmd* скриптовете, в зависимост от платформата.

За да бъде спряно приложението се ползва специална нишка - *ShutdownHook*, която отваря сокет, който слуша за връзки от *localhost*. Когато получи връзка, която изпрати символният низ “*shutdown*”, нишката ще се погрижи да спре сървъра. Това е *run()* метода на *ShutdownHo*o*k* класа:

public void run**()** **{**

ServerSocket serverSocket **=** **null;**

**try** **{**

serverSocket **=** **new** ServerSocket**(**shutdownPort**,** 0**,** InetAddress**.**getByName**(null));**

**}** **catch** **(**IOException e**)** **{**

sLogger**.**error**(**"Could not listen on shutdown port: " **+** shutdownPort **+** ". " **+** e**.**getMessage**(),** e**);**

**return;**

**}**

sLogger**.**info**(**"Shutdown hook is listening on port " **+** shutdownPort**);**

boolean shutdownReceived **=** **false;**

**while(!**shutdownReceived**)** **{**

Socket clientSocket **=** **null;**

**try** **{**

clientSocket **=** serverSocket**.**accept**();**

BufferedReader br **=** **new** BufferedReader**(new** InputStreamReader**(**clientSocket**.**getInputStream**()));**

String line **=** br**.**readLine**();**

**if(**ConfigurationConstants**.**SHUTDOWN\_COMMAND**.**equalsIgnoreCase**(**line**))** **{**

sLogger**.**info**(**"Received shutdown command. Stopping server..."**);**

shutdownReceived **=** **true;**

**}**

br**.**close**();**

clientSocket**.**close**();**

**}** **catch** **(**IOException e**)** **{**

sLogger**.**error**(**"Shutdown hook could not accept connection. " **+** e**.**getMessage**(),** e**);**

**continue;**

**}**

**}**

**try** **{**

serverSocket**.**close**();**

**}** **catch** **(**IOException e**)** **{**

sLogger**.**error**(**"Error closing shutdown hook socket. " **+** e**.**getMessage**(),** e**);**

**}**

**}**

За да бъде спрян сървъра се ползват *stopServer.sh* или *stopServer.cmd* скриптовете, в зависимост от платформата. Тяхната работа е да пуснат *StopApplication* класа. Този клас ще се свърже на *localhost* с *ShutdownHook* и ще изпрати “*shutdown*” команда.

public class StopApplication **{**

public static void main**(**String**[]** args**)** **{**

LoggingUtils**.**suppressLogging**();**

IServerConfigurationManager configurationManager **=** Factory**.**getInstance**().**getServerConfigurationManager**(**FactoryConstants**.**SERVER\_CONFIGURATION\_MANAGER\_XML**);**

int shutdownPort **=** configurationManager**.**getShutDownPort**();**

Socket socket**;**

**try** **{**

socket **=** **new** Socket**(**"127.0.0.1"**,** shutdownPort**);**

OutputStream os **=** socket**.**getOutputStream**();**

os**.**write**(**ConfigurationConstants**.**SHUTDOWN\_COMMAND**.**getBytes**());**

os**.**close**();**

socket**.**close**();**

**}** **catch** **(**Exception e**)** **{**

System**.**err**.**println**(**"Error shutting down server. " **+** e**.**getMessage**());**

e**.**printStackTrace**();**

**}**

**}**

**}**

## Обща архитектура на приложението

За достъп до повечето от функционалностите на кода, се ползва класът *Factory*, който следва „фабрика“ шаблона за дизайн. *Factory* класът също така е сънгълтън. Този клас държи инстанции на повечето класове през които се достъпват главните функционалности.

public class Factory **{**

private static Factory sInstance**;**

private static IProtocolManager sProtocolManager**;**

private static IServerConfigurationManager sServerConfigurationManager**;**

private static ICertificateManager sCertificateManager**;**

private static ICryptoHelper sCryptoHelper**;**

private static IPartyManager sPartyManager**;**

private static IMdnMonitor sMdnMonitor**;**

private static final Object SINGLETON\_LOCK **=** **new** Object**();**

private static final Logger sLogger **=** Logger**.**getLogger**(**Factory**.**class**);**

private Factory**(){}**

public static Factory getInstance**()** **{**

**if(**sInstance **==** **null)** **{**

synchronized**(**SINGLETON\_LOCK**)** **{**

**if(**sInstance **==** **null)** **{**

sLogger**.**info**(**"Initializing Factory instance..."**);**

sInstance **=** **new** Factory**();**

sLogger**.**info**(**"Factory instance successfully initialized."**);**

**}**

**}**

**}**

**return** sInstance**;**

**}**

public IMdnMonitor getMdnMonitor**()** **{...}**

public IMdnMonitor getMdnMonitor**(**String type**)** **{...}**

public IPartyManager getPartyManager**()** **{...}**

public IPartyManager getPartyManager**(**String type**)** **{...}**

public ICryptoHelper getCryptoHelper**()** **{...}**

public ICryptoHelper getCryptoHelper**(**String type**)** **{...}**

public ICertificateManager getCertificateManager**()** **{...}**

public ICertificateManager getCertificateManager**(**String type**)** **{...}**

public IProtocolManager getProtocolManager**()** **{...}**

public IProtocolManager getProtocolManager**(**String type**)** **{...}**

public IServerConfigurationManager getServerConfigurationManager**()** **{...}**

public IServerConfigurationManager getServerConfigurationManager**(**String type**)** **{...}**

public ITransport buildTransport**(**SettingsHolder baseTransportSettings**)** **throws** ProtocolException**,** TransportException **{...}**

public ITransport getTransport**(**String transportName**)** **{...}**

public IProtocol getProtocol**(**String protocolName**)** **{...}**

public SettingsHolder getTransportSettings**(**String transportType**)** **{...}**

public SettingsHolder getProtocolSettings**(**String protocolType**)** **{...}**

public HttpServlet getHttpListener**(**HttpTransportSettings settings**,** IProtocol protocol**)** **{...}**

public Party getParty**(**String type**)** **{...}**

**}**

Във следващите глави ще се акцентира на всяка от променливите, които Factory класа съдържа. Важно е да се отбележи, че всеки от тези класове има само по една инстанция.

### Мониториране на MDN съобщения

Когато се изпрати съобщение, за което се очаква да бъде потвърдено с MDN, трябва да има механизъм по който да може MDN съобщението да се асоциира с оригиналното съобщение, ако има такова. За това се ползва *IMdnMonitor* интерфейса.

public interface IMdnMonitor **{**

void registerMessage**(**String uniqueId**,** IMessage message**);**

boolean isMessageRegistered**(**String uniqueId**);**

IMessage getMessage**(**String uniqueId**);**

void confirmAndDeregisterMessage**(**String uniqueId**);**

**}**

Когато се пакетира съобщението то се регистрира с *registerMessage* метода, което значи че се приложението ще очаква MDN за него. Когато пристигне MDN съобщението, то се потвърждава с *confirmAndDeregisterMessage*. *uniqueId* параметъра се очаква да бъде пресметната MIC стойност на съобщението.

Конкретната имплементация, която приложението ползва е *As2InMemoryMdnMonitor*. Тази имплементация пази в паметта информацията за съобщенията.

### Управление на партньори

За да може да става въпрос за пращане и получаване на файлове са имплементирани т. нар. партньорски дефиниции. Всяка партньорска дефиниция съдържа конфигурация за даден партньор или изпращач. Това вклучва настройки за сигурност, сертификати, адреси и т.н.

*IPartyManager* интерфейсът позволява да се достъпват конфигурираните партньори или изпращачи.

public interface IPartyManager **{**

void loadParties**();**

Party getPartyById**(**String partyId**);**

void addParty**(**IParty party**);**

void deleteParty**(**String id**);**

**}**

*partyId* променливите са уникалните идентификатори за партньорите, които се ползват в AS2-To и AS2-From хедърите. Не може да има повече от един партньор с еднакъв идентификатор. Интерфейса позволява стандартни CRUD операции с партньори.

Имплементацията която приложението ползва е *XmlPartyManager*. Специфичното при този клас е, че зарежда конфигурираните партньори от xml файл. Повече за структурата на този файл в следващата глава на този документ.

Приложението също така дефинира *IParty* интерфейс, който съдържа цялата конфигурационна информация за даден партньор:

public interface IParty **{**

public String getPartyId**();**

public void setPartyId**(**String partyId**);**

**}**

Въпреки, че интерфейсът позволява само работа с уникалния идентификатор на партньора, класът който го имплементира - *Party*, също така наследява и HashMap**<**String**,** Object>. Това му позволява да съдържа динамичен брой конфигурационни променливи. Част от конфигурационните променливи до които класът дава достъп са:

public String getPartyId**()** **{...}**

public String getSignCertAlias**()** **{...}**

public String getEncryptCertAlias**()** **{...}**

public boolean isEncryptionEnabled**()** **{...}**

public String getEncryptCertKeyPassword**()** **{...}**

public String getEncryptAlgorithm**()** **{...}**

public boolean isSigningEnabled**()** **{...}**

public String getSignCertKeyPassword**()** **{...}**

public String getSignDigestAlgorithm**()** **{...}**

public boolean isCompressionEnabled**()** **{...}**

public String getCompressionAlgorithm**()** **{...}**

public boolean isRequestMdn**()** **{...}**

public String getMdnType**()** **{...}**

public String getAsynchronousMdnUrl**()** **{...}**

public boolean isRequestSignedMdn**()** **{...}**

public String getMdnSigningAlgorithm**()** **{...}**

Бизнес логиката ползва *IPartyManager* за да достъпи конфигурацията за някой, конкретен партньор. Самата конфигурация се съдържа в *IParty* обект.

### MIME Криптография

*ICryptoHelper* интерфейсът дава достъп до всички алгоритми за трансформации на MIME съобщения, които приложението използва. Това включва компресиране, криптира, подписване и др.

void init**();**

void deinit**();**

boolean isEncrypted**(**MimeBodyPart part**)** **throws** Exception**;**

boolean isSigned**(**MimeBodyPart part**)** **throws** Exception**;**

boolean isCompressed**(**MimeBodyPart part**)** **throws** Exception**;**

String calculateMIC**(**Path file**,** String digestAlg**)** **throws** Exception**;**

MimeBodyPart decrypt**(**MimeBodyPart part**,** X509Certificate cert**,** PrivateKey key**)** **throws** Exception**;**

MimeBodyPart encrypt**(**MimeBodyPart part**,** X509Certificate cert**,** String algorithm**)** **throws** Exception**;**

MimeBodyPart sign**(**MimeBodyPart part**,** X509Certificate cert**,** PrivateKey key**,** String digest**)** **throws** Exception**;**

MimeBodyPart verify**(**MimeBodyPart part**,** X509Certificate cert**)** **throws** Exception**;**

MimeBodyPart compress**(**MimeBodyPart part**,** String alg**)** **throws** Exception**;**

MimeBodyPart decompress**(**MimeBodyPart part**)** **throws** Exception**;**

Имплементацията която се ползва е *BCCryptoHelper*. Тя използва Bouncy Castle security provider за всичките криптографски алгоритми.

Ползва се следна функция за декриптиране:

public MimeBodyPart decrypt**(**MimeBodyPart part**,** X509Certificate cert**,** PrivateKey key**)** **throws** Exception **{**

// Make sure the data is encrypted

**if** **(!**isEncrypted**(**part**))** **{**

**throw** **new** GeneralSecurityException**(**"Content-Type indicates data isn't encrypted"**);**

**}**

// Get the recipient object for decryption

RecipientId recId **=** **new** JceKeyTransRecipientId**(**cert**);**

SMIMEEnvelopedParser parser **=** **new** SMIMEEnvelopedParser**(**part**);**

RecipientInformationStore recipients **=** parser**.**getRecipientInfos**();**

RecipientInformation recipient **=** recipients**.**get**(**recId**);**

**if** **(**recipient **==** **null)** **{**

**throw** **new** GeneralSecurityException**(**"Certificate does not match part signature"**);**

**}**

MimeBodyPart res **=** SMIMEUtil**.**toMimeBodyPart**(**recipient**.**getContentStream**(new** JceKeyTransEnvelopedRecipient**(**key**).**setProvider**(**BouncyCastleProvider**.**PROVIDER\_NAME**)));**

**return** res**;**

**}**

Ползва се следна функция за криптиране:

public MimeBodyPart encrypt**(**MimeBodyPart part**,** X509Certificate x509Cert**,** String algorithm**)** **throws** Exception **{**

ASN1ObjectIdentifier encAlg **=** **null;**

**if(**CRYPT\_RC2**.**equals**(**algorithm**))** **{**

encAlg **=** CMSAlgorithm**.**RC2\_CBC**;**

**}** **else** **if(**CRYPT\_3DES**.**equals**(**algorithm**))** **{**

encAlg **=** CMSAlgorithm**.**DES\_EDE3\_CBC**;**

**}** **else** **if(**CRYPT\_CAST5**.**equals**(**algorithm**))** **{**

encAlg **=** CMSAlgorithm**.**CAST5\_CBC**;**

**}** **else** **if(**CRYPT\_IDEA**.**equals**(**algorithm**))** **{**

encAlg **=** CMSAlgorithm**.**IDEA\_CBC**;**

**}** **else** **{**

**throw** **new** ProtocolException**(**"Unknown encryption algorithm [" **+** algorithm **+** "]. Message will not be processed."**);**

**}**

SMIMEEnvelopedGenerator gen **=** **new** SMIMEEnvelopedGenerator**();**

gen**.**addRecipientInfoGenerator**(new** JceKeyTransRecipientInfoGenerator**(**x509Cert**).**setProvider**(**BouncyCastleProvider**.**PROVIDER\_NAME**));**

gen**.**setContentTransferEncoding**(**As2HeaderDictionary**.**TRANSFER\_ENCODING\_BINARY**);**

MimeBodyPart encData **=** gen**.**generate**(**part**,** **new** JceCMSContentEncryptorBuilder**(**encAlg**).**setProvider**(**BouncyCastleProvider**.**PROVIDER\_NAME**).**build**());**

**return** encData**;**

**}**

Ползва се следна функция за подписване:

public MimeBodyPart sign**(**MimeBodyPart part**,** X509Certificate cert**,** PrivateKey key**,** String digest**)** **throws** Exception **{**

String digestAlg **=** **null;**

**if(**DIGEST\_SHA1**.**equals**(**digest**))** **{**

digestAlg **=** "SHA1withRSA"**;**

**}** **else** **if(**DIGEST\_MD5**.**equals**(**digest**))** **{**

digestAlg **=** "MD5withRSA"**;**

**}** **else** **{**

**throw** **new** ProtocolException**(**"Unknown digest algorithm [" **+** digest **+** "]. Message will not be processed."**);**

**}**

List**<**X509Certificate**>** certList **=** **new** ArrayList**<**X509Certificate**>();**

certList**.**add**(**cert**);**

@SuppressWarnings**(**"rawtypes"**)**

Store certs **=** **new** JcaCertStore**(**certList**);**

ASN1EncodableVector signedAttrs **=** **new** ASN1EncodableVector**();**

SMIMECapabilityVector caps **=** **new** SMIMECapabilityVector**();**

caps**.**addCapability**(**SMIMECapability**.**dES\_EDE3\_CBC**);**

caps**.**addCapability**(**SMIMECapability**.**rC2\_CBC**,** 128**);**

caps**.**addCapability**(**SMIMECapability**.**dES\_CBC**);**

signedAttrs**.**add**(new** SMIMECapabilitiesAttribute**(**caps**));**

IssuerAndSerialNumber issAndSer **=** **new** IssuerAndSerialNumber**(new** X500Name**(**cert**.**getIssuerDN**().**getName**()),** cert**.**getSerialNumber**());**

signedAttrs**.**add**(new** SMIMEEncryptionKeyPreferenceAttribute**(**issAndSer**));**

SMIMESignedGenerator gen **=** **new** SMIMESignedGenerator**();**

gen**.**setContentTransferEncoding**(**As2HeaderDictionary**.**TRANSFER\_ENCODING\_BINARY**);**

gen**.**addSignerInfoGenerator**(new** JcaSimpleSignerInfoGeneratorBuilder**().**setProvider**(**BouncyCastleProvider**.**PROVIDER\_NAME**).**

setSignedAttributeGenerator**(new** AttributeTable**(**signedAttrs**)).**build**(**digestAlg**,** key**,** cert**));**

gen**.**addCertificates**(**certs**);**

MimeMultipart mm **=** gen**.**generate**(**part**);**

MimeBodyPart tempBody **=** **new** MimeBodyPart**();**

tempBody**.**setContent**(**mm**);**

tempBody**.**setHeader**(**As2HeaderDictionary**.**CONTENT\_TYPE**,** mm**.**getContentType**());**

**return** tempBody**;**

**}**

Ползва се следна функция за верифициране на подпис:

@SuppressWarnings**({** "unchecked"**,** "rawtypes" **})**

public MimeBodyPart verify**(**MimeBodyPart part**,** X509Certificate cert**)** **throws** Exception **{**

// Make sure the data is signed

**if** **(!**isSigned**(**part**))** **{**

**throw** **new** GeneralSecurityException**(**"Content-Type indicates data isn't signed"**);**

**}**

MimeMultipart mainParts **=** **(**MimeMultipart**)** part**.**getContent**();**

SMIMESignedParser parser **=** **new** SMIMESignedParser**(new** JcaDigestCalculatorProviderBuilder**().**build**(),** mainParts**);**

Store certs **=** parser**.**getCertificates**();**

SignerInformationStore signers **=** parser**.**getSignerInfos**();**

Collection**<**SignerInformation**>** signersCollection **=** signers**.**getSigners**();**

Iterator**<**SignerInformation**>** it **=** signersCollection**.**iterator**();**

**while** **(**it**.**hasNext**())** **{**

SignerInformation signer **=** **(**SignerInformation**)**it**.**next**();**

Collection certCollection **=** certs**.**getMatches**(**signer**.**getSID**());**

Iterator certIt **=** certCollection**.**iterator**();**

X509Certificate resolvedCert **=** **new** JcaX509CertificateConverter**().**setProvider**(**BouncyCastleProvider**.**PROVIDER\_NAME**).**getCertificate**((**X509CertificateHolder**)**certIt**.**next**());**

**if(!**resolvedCert**.**equals**(**cert**))** **{**

**throw** **new** SignatureException**(**"Message is signed with a certificate, that is not configured for this party!"**);**

**}**

**if** **(!**signer**.**verify**(new** JcaSimpleSignerInfoVerifierBuilder**().**setProvider**(**BouncyCastleProvider**.**PROVIDER\_NAME**).**build**(**resolvedCert**)))** **{**

**throw** **new** SignatureException**(**"Signature verification failed!"**);**

**}**

**}**

SMIMESigned signedPart **=** **new** SMIMESigned**(**mainParts**);**

**return** signedPart**.**getContent**();**

**}**

Ползва се следна функция за компресиране:

@Override

public MimeBodyPart compress**(**MimeBodyPart part**,** String alg**)** **throws** Exception **{**

OutputCompressor compressor **=** **null;**

**if(**COMPRESS\_ZLIB**.**equals**(**alg**))** **{**

compressor **=** **new** ZlibCompressor**();**

**}** **else** **{**

**throw** **new** ProtocolException**(**"Unknown compression algorithm [" **+** alg **+** "]. Message will not be processed."**);**

**}**

SMIMECompressedGenerator gen **=** **new** SMIMECompressedGenerator**();**

gen**.**setContentTransferEncoding**(**As2HeaderDictionary**.**TRANSFER\_ENCODING\_BINARY**);**

MimeBodyPart dataBP **=** gen**.**generate**(**part**,** compressor**);**

**return** dataBP**;**

**}**

Ползва се следна функция за декомпресиране:

@Override

public MimeBodyPart decompress**(**MimeBodyPart part**)** **throws** Exception **{**

SMIMECompressedParser parser **=** **new** SMIMECompressedParser**(**part**);**

MimeBodyPart res **=** SMIMEUtil**.**toMimeBodyPart**(**parser**.**getContent**(new** ZlibExpanderProvider**()));**

**return** res**;**

**}**

Не на последно място е важно и да се отбележи как се смята MIC сумата за дадено съобщение:

public String calculateMIC**(**Path file**,** String digestAlg**)** **throws** Exception **{**

String micAlg **=** **null;**

**if(**DIGEST\_SHA1**.**equals**(**digestAlg**))** **{**

micAlg **=** "sha1"**;**

**}** **else** **if(**DIGEST\_MD5**.**equals**(**digestAlg**))** **{**

micAlg **=** "md5"**;**

**}** **else** **{**

**throw** **new** ProtocolException**(**"Unknown digest algorithm [" **+** digestAlg **+** "]. Message will not be processed."**);**

**}**

MessageDigest md **=** MessageDigest**.**getInstance**(**micAlg**,** BouncyCastleProvider**.**PROVIDER\_NAME**);**

**try** **(**InputStream is **=** Files**.**newInputStream**(**file**))** **{**

DigestInputStream dis **=** **new** DigestInputStream**(**is**,** md**);**

byte**[]** buf **=** **new** byte**[**4096**];**

**while** **(**dis**.**read**(**buf**)** **>=** 0**)** **{**

**}**

**}**

byte**[]** digest **=** md**.**digest**();**

String micString **=** **new** String**(**Base64**.**encode**(**digest**));**

StringBuffer micResult **=** **new** StringBuffer**(**micString**);**

micResult**.**append**(**", "**).**append**(**micAlg**);**

**return** micResult**.**toString**();**

**}**

### Достъп до сертификати и ключове

Приложението ползва 2 Java keystore файла. Те съдържат X509 сертификати и ключове с които приложението работи.

* *truststore.jks* – съдържа всички стандартни root CA сертификати (Thawte, VeriSign и др.). Ползва се за верификация на произхода на сертификати, било то за TLS връзки или подписване и криптиране.
* *keystore*.*jks* – съдържа частни и публични ключове, които се ползват при подписване и криптиране. Причината те да са в друг keystore файл е за по-лесна работа, тъй като в *truststore.jks* има доста сертификати и е трудно да се разглеждат като списък.

За да се добавят или ревизират сертификати се ползва стандартната Java keytool команда. Приложението не имплементира отделна логика за добавяне на сертификати, защото няма смисъл, при положение че няма потребителски интерфейс.

Ключовете и сертификатите от *keystore.jks* файла могат да се реферират в конфигурацията по алиас. Алиасът е уникалното име на сертификата в keystore файла.

Двата файла имат парола за достъп, която е redoubt по подразбиране.

За достъп до сертификатите и ключовете, приложението дефинира интерфейсът *ICertificateManager*.

public interface ICertificateManager **{**

void init**();**

String getKeystoreType**();**

X509Certificate getX509Certificate**(**String alias**);**

PrivateKey getPrivateKey**(**String alias**,** char**[]** password**);**

boolean checkCertificateValidity**(**X509Certificate cert**)** **throws** Exception**;**

boolean isSelfSigned**(**X509Certificate cert**)** **throws** Exception**;**

**}**

Както се вижда се дава възможност за взимане на X509 сертификати или ключове по алиас.

Поддържаната имплементация е за работа с *truststore.jks* и *keystore.jks* файловете – *JksCertificateManager*. Този клас има за задача да заради всички сертификати от файловете. Отговорността е на потребителя да добави съответните сертификати и ключове с Java keytool командата. Зареждането на сертификатите става по следния начин:

@Override

public void init**()** **{**

IServerConfigurationManager configurationManager **=** Factory**.**getInstance**().**getServerConfigurationManager**();**

Path keystoreFile **=** configurationManager**.**getKeystoreFile**();**

char**[]** keystorePassword **=** configurationManager**.**getKeystorePassword**().**toCharArray**();**

Path truststoreFile **=** configurationManager**.**getTruststoreFile**();**

char**[]** truststorePassword **=** configurationManager**.**getTruststorePassword**().**toCharArray**();**

**try(**InputStream keystoreStream **=** **new** FileInputStream**(**keystoreFile**.**toFile**());**

InputStream truststoreStream **=** **new** FileInputStream**(**truststoreFile**.**toFile**());)** **{**

sLogger**.**info**(**"Loading keystore from [" **+** keystoreFile**.**toString**()** **+** "]."**);**

keystore **=** KeyStore**.**getInstance**(**getKeystoreType**());**

keystore**.**load**(**keystoreStream**,** keystorePassword**);**

sLogger**.**info**(**"Loading truststore from [" **+** truststoreFile**.**toString**()** **+** "]."**);**

truststore **=** KeyStore**.**getInstance**(**getKeystoreType**());**

truststore**.**load**(**truststoreStream**,** truststorePassword**);**

**}** **catch** **(**Exception e**)** **{**

sLogger**.**error**(**"An error has occured while initializing CertificateManager! " **+** e**.**getMessage**(),** e**);**

sLogger**.**error**(**"APPLICATION WILL BE TERMMINATED."**);**

System**.**exit**(**1**);**

**}**

**}**

### Глобална сървърна конфигурация

дса

# Експериментална част

## Конфигурация на приложението

# Заключение

асд

# Използвана литература

<https://en.wikipedia.org/wiki/AS1>

<https://en.wikipedia.org/wiki/AS2>

<https://en.wikipedia.org/wiki/AS3>

<https://en.wikipedia.org/wiki/AS4>

<http://www.ld.com/as2-part-1-what-is-it/>

<http://www.ld.com/as2-part-2-best-practices/>

<http://www.drummondgroup.com/index.php/component/content/article/127-b2b/b2b-products/b2b-faqs/243-as4-faq>

[http://www.ietf.org/rfc/rfc4130.txt](https://www.ietf.org/rfc/rfc4130.txt)

<http://www.ietf.org/rfc/rfc2616.txt>

<http://www.ietf.org/rfc/rfc2045.txt>

<http://www.ietf.org/rfc/rfc2633.txt>

http://www.ietf.org/rfc/rfc1847.txt

<http://www.ietf.org/rfc/rfc2376.txt>

http://www.ietf.org/rfc/rfc2298.txt

<http://www.drummondgroup.com/b2b-certified-products/b2b-standards/as2>

<https://moveitsupport.ipswitch.com/support/micentral/help/MICAS1AS2AS3Overview.htm>

<https://bg.wikipedia.org/wiki/Java>

<https://www.bouncycastle.org/java.html>

<https://java.net/projects/javamail/pages/Home>

<http://hc.apache.org/httpcomponents-client-ga/>

<http://www.eclipse.org/jetty/>

<https://logging.apache.org/log4j/1.2/>

<https://en.wikipedia.org/wiki/S/MIME>

<https://en.wikipedia.org/wiki/MIME>

<https://bg.wikipedia.org/wiki/HTTP>

<https://tools.ietf.org/html/rfc5402>

<http://www.drummondgroup.com/b2b-certified-products/certified-products/as2/126-b2b/b2b-products/b2b-certified-products/147-as2-current>

<https://en.wikipedia.org/wiki/X.509>

Java за цифрово подписване на документи в уеб