**Технически университет - София**

**Филиал Пловдив**

**Проект**

**по РСКК и ПИ**

**на тема: Предаване на шифрована информация чрез**

**REST - базирана услуга. Управление на хранлища и ключове чрез уеб базирано приложение**

**На:** Драгомир Ивелинов Тодоров

**Група:** 41a

**Специалност:** КСТ

**Фак. №:** 609813

Гр. Пловдив Проверил:.........................

Дата 20.01.2015 г. / гл. ас. д-р М. Шопов/

# Задание

Да се реализира система за криптиране с публични и частни ключове (PKI).

Системата да се състои от клиент, сървър и хранилище на публичните ключове.

Хранилището поддържа списък с публични ключове и търсене в него по зададен ключ.

Клиентът извлича публичния ключ на сървъра и му изпраща криптирано съобщение.

Сървъра декриптира съобщението, извлича публичния ключ на клиента и връща съобщението (ехо сървър).

Функциите на хранилището и сървъра да бъдат реализирани посредством REST интерфейс.

За курсовия проект по ПИ да се реализира уеб приложение, което позволява управление на хранилището - добавяне и изтриване на ключове, както и уеб интерфейс за клиента, с възможност за избор на повече от едно и измежду повече от едно хранилища. При реализирането да се използват HTML, CSS, JavaScript, AJAX.

# Теоретична обосновка

* 1. **Криптиране на информация**

Криптирането на информация се изучава от науката криптография. Това е наука за принципите, средствата и методите за преобразуване на данни с оглед укриване на тяхната семантика, предотвратяване на неоторизирано ползване или на тяхната незабележима промяна от трети неоторизирани лица. Тези принципи, средства и методи се полват широко от съвременната информатика за осигуряване на информационна сигурност (англ. information security), вкл. поверителност (confidentiality), цялост на данни (data integrity), невъзможност да отрича (non-repudiation) и гаранция за автентичност (authenticity).

В началото криптографията е изучавала начините за шифриране (и дешифриране) на информация или обратимо преобразуване на прав (нешифриран) текст на база таен алгоритъм и/или ключ в кодиран текст (шифрограма). Тази, т.нар. традиционна криптография, обхваща само раздел симетрично криптиране, в който шифрирането и дешифрирането се извършва с ползване на един и същ секретен ключ. Освен този раздел съвременната криптография включва и раздел асиметрично криптиране, както и системите за електронен подпис, хеширане.

За целите на конкретния проект ще се разучи в детайли асинхронната криптография, при която за **кодиране се използва публичен ключ**, а за разкодиране друг – личен ключ.

* 1. **Криптография с публичен ключ**

Криптографията с публичен ключ е особено важна за осигуряване на сигурното предаване на данни в [Интернет](http://bg.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82). За разлика от метода със симетрични ключове, тук ключовете са два и се генерират едновременно по определен алгоритъм, като всеки желаещ да обменя криптирани данни, трябва да има своя собствена уникална двойка от публичен и частен ключове:

* **публичен ключ** — използва се за криптиране на данните и е общодостъпен за всеки, които иска да изпрати информация на притежателя му. Той не може да декриптира данните.
* **частен ключ** — достъпен е само за притежателя си. Използва се за декриптиране на криптирани с публичния ключ данни.

Следователно когато искате да комуникирате защитено, трябва да генерирате два ключа – криптиращ и декриптиращ. Те не могат да бъдат извлечени един от друг (освен с груба сила), затова трябва да ги пазите и двата в паметта (или написани на хартия за по-лесно разбиране).

нешифрован\_текст ---> криптиращ\_ключ ( нешифрован\_текст ) ---> шифрован\_текст ---> декриптиращ\_ключ ( шифрован\_текст ) ---> нешифрован\_текст

Ключовете са взаимно еквивалентни – това работи дори в обратната посока (т.е. и декриптиращият ключ също може да се използва за криптиране, като тогава другият ще служи за декодиране):

нешифрован\_текст ---> декриптиращ\_ключ (нешифрован\_текст) ---> шифрован\_текст ---> криптиращ\_ключ (шифрован\_текст) ---> нешифрован\_текст

Такива криптографски алгоритми се наричат асиметрични заради тяхната асиметрична природа.

Обикновено процесът на обмяна на ключовете с другата страна протича така:

* генерирам ключовете
* криптиращият ключ публично (т.е. без абсолютно никакъв страх от подслушване) се транспортира до моя приятел
* декриптиращият ключ остава скрит при мен
* когато приятелят ми изпрати съобщение, той го криптира с *публичния* криптиращ ключ
* дори всички да знаят *публичния криптиращ ключ*, никой не може да декриптира съобщението, защото то може да бъде декриптирано само чрез моя *частен декриптиращ ключ*

Голямото значение на криптографията с публичен ключ произтича от липсата на необходимост от предварително разпределяне или обмяна на ключове между комуникиращите страни. Това прави възможно предлагането на редица онлайн услуги като електронни разплащания, сигурен обмен на данни и др.

Поради по-голямата изчислителна интензивност, необходима за реализиране на алгоритмите за криптиране с публичен ключ, понякога методът се прилага за кратък комуникационен обмен, при който двете страни обменят ключове за продължаване на по-нататъшната комуникация чрез криптиране със симетрични ключове.

Времето за криптиране на дадено съобщение расте експоненциално с неговата дължина. Казано просто, това означава, че ако текст с дадена дължина изисква 10 секунди за кодиране/декодиране, то обработката на четири пъти по-дълго съобщение ще отнеме 16 пъти повече време, т.е. 160 секунди.

В реалния живот заради скоростта на криптиране тези алгоритми са неизползваеми при съобщения, по-дълги от 20 байта. Както можете да си представите, това е много кратък текст (например предното изречение е дълго 115 байта). От друга страна, криптографията със секретен ключ (тази от предишната глава) е бърза и е приложима без значение от дължината на съобщението.

* 1. **REST**

За целите на текущия проект използваме PHP Micro-framework Slim, чрез който

реализираме REST- базирана уеб услуга. Използва се имено този framework поради гъвкавостта му и възможността за бърза редакция на съществуващи функции на REST api-то.

Услугата REST представлява разпределителна системна рамка, базирана на уеб протоколи и технологии. Архитектурният модел Rest включва взаимодействията между сървър и клиент, осъществени по време на трансфера на данни.

1. **Обяснителна записка** 
   1. **Server / Storages**

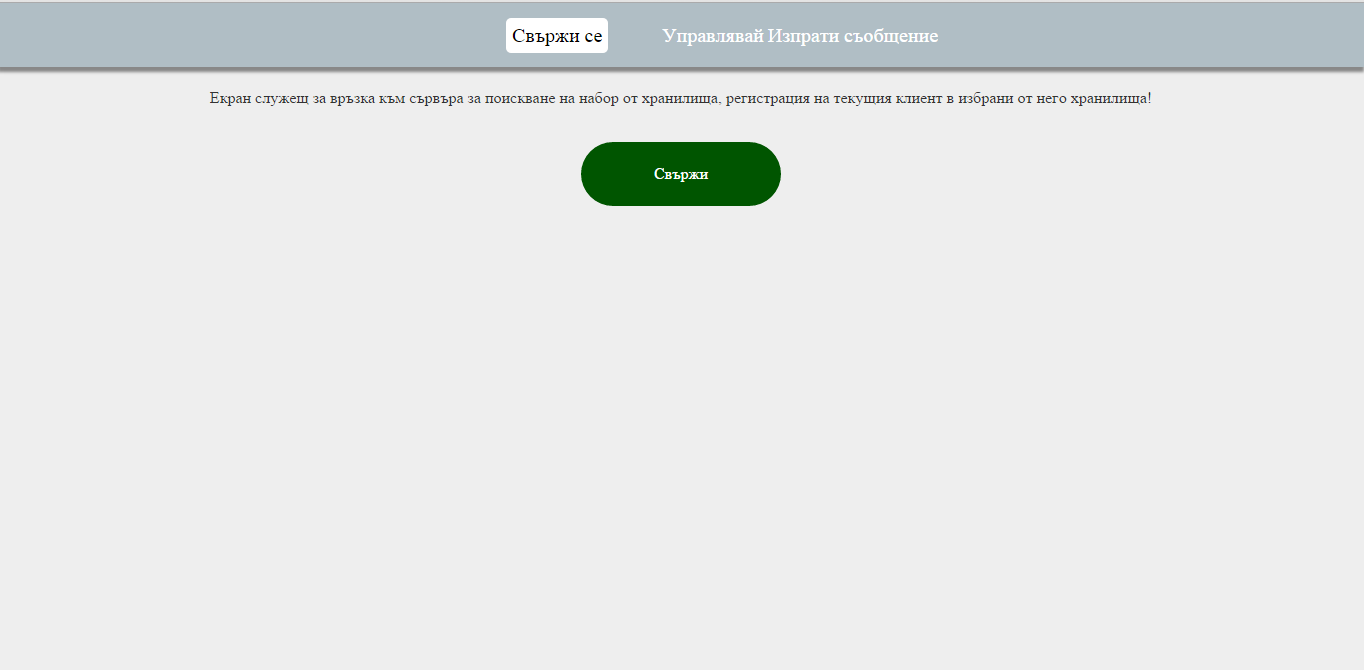
Проектирани са:

* **Сървърна част със вградено „хранлище“;**
* **Самостоятелно хранилище;**
  + 1. **Сървърната част** предоставя следните основни функционалности, проектирани като REST-базирана услуга:
* **/server/key [GET] -** позволява извличане на публичния ключ на сървъра
* **/server/storages [GET] –** получават се адресите на хранилищата (за по-голяма модуларност се предоставя и сървърното хранилище
* **/server [POST] ( encryptedmessage , ip, [array of storage id's] ) –** получаваме следните параметри форматирани в JSON формат. Извършва се следния алгоритъм:
* Функцията извлича IP адрес-а на клиента и намира съответстващ PublicKey, който отговаря на клиента.
* При намиране на ключ той се проверява за валидност и чрез него се разкриптира информацията **encryptedmessage**
* При обхождане на хранилищата и не намиране на ключ се връща error code: 1
* При обхождане на хранилищата и намиране на двойка несъответстващи ключове се връща error code: 2
* При намиране на ключ, но той не е валиден public key се връща error code: 3
* При валиден ключ се криптира информацията и тя се връща на клиента.
  + 1. **Хранилището** е опростен вариант използвайки БД за съхранение на IP адреси и публични ключове на клиентите. То предоставя следните функционалности:
* **/storage/register [POST] ( ipaddress , publickey ) –** получава информация от клиента, форматирана в JSON формат, за запис в БД
* **/storage/clients [GET] –** извлича информация за всички клиенти в хранилището
* **/storage/:ipaddress [GET] –** извлича информация за клиент със съответен **ipaddress**
* **/storage/update [PUT] ( ipaddress , publickey ) –** обновява информация на клиент със съответен ipaddress
* **/storage/:ipaddress [DELETE] –** изтрива клиент със съответен **ipaddress**
  1. **Клиент**

За реализиране на клиентската част са използвани езиците html5, css, jQuery/JavaScript и php за криптиране и разкриптиране на съобщения, поради неналичие на сигурни javascript-базирани библиотеки за работа с openssl.

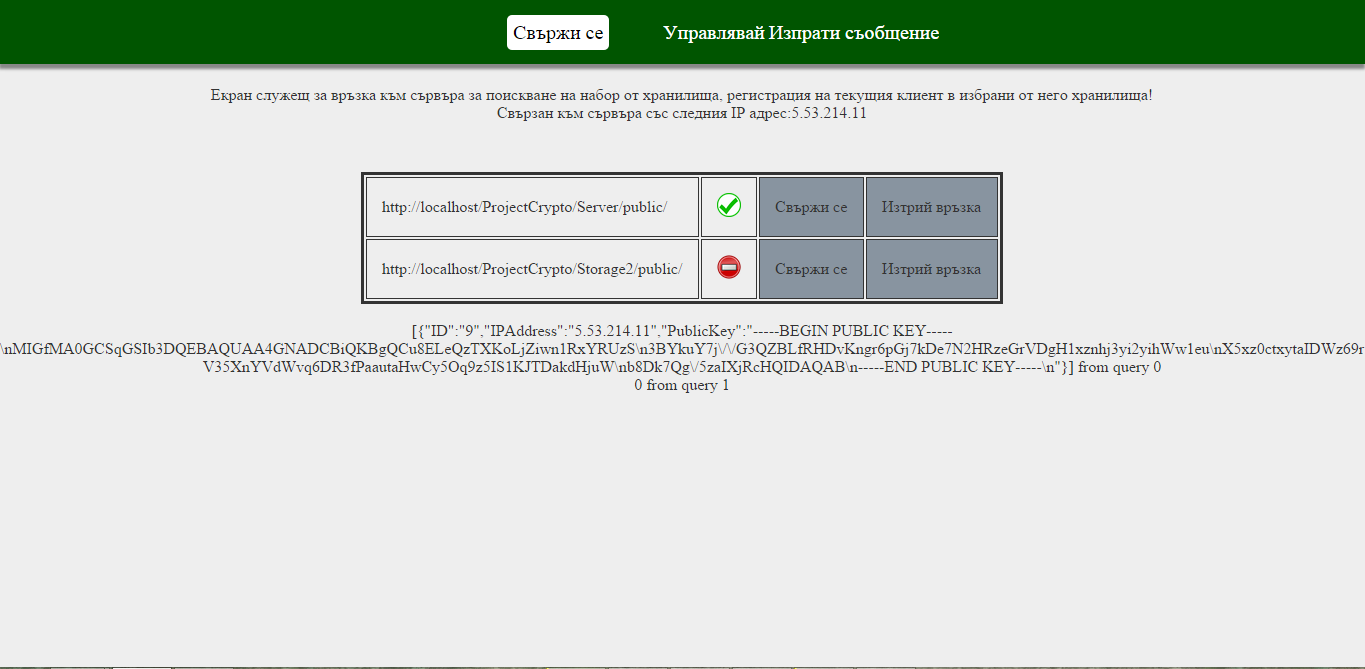
Имаме наличие на следните основни файлове в проекта, всеки от тях изпълняващ следните основни роли:

* index.html – описващ базовото скеле и някои статични елементи на интерфейса;
* base.css – стилизира статичните и динамично генерираните обекти;
* script/application.js – основен програмен файл, описващ логиката на приложението. Алгоритъма извършващ се вътре ще бъде обяснен по-надолу;
* php/…php – тези файлове се използват за свръзка с openssl функциите за кодиране и разкодиране на съобщение на база на съответен ключ.
  + 1. **Начален екран**



Началния екран е опростен до минимум, като загатва за възможностите на потребителя, но не предоставя такива, докато потребителя не се е свързал със сървъра. При свръзка със сървъра се проверява дали потребителя присъства във всички storage елементи и се изобразява на следващия екран.

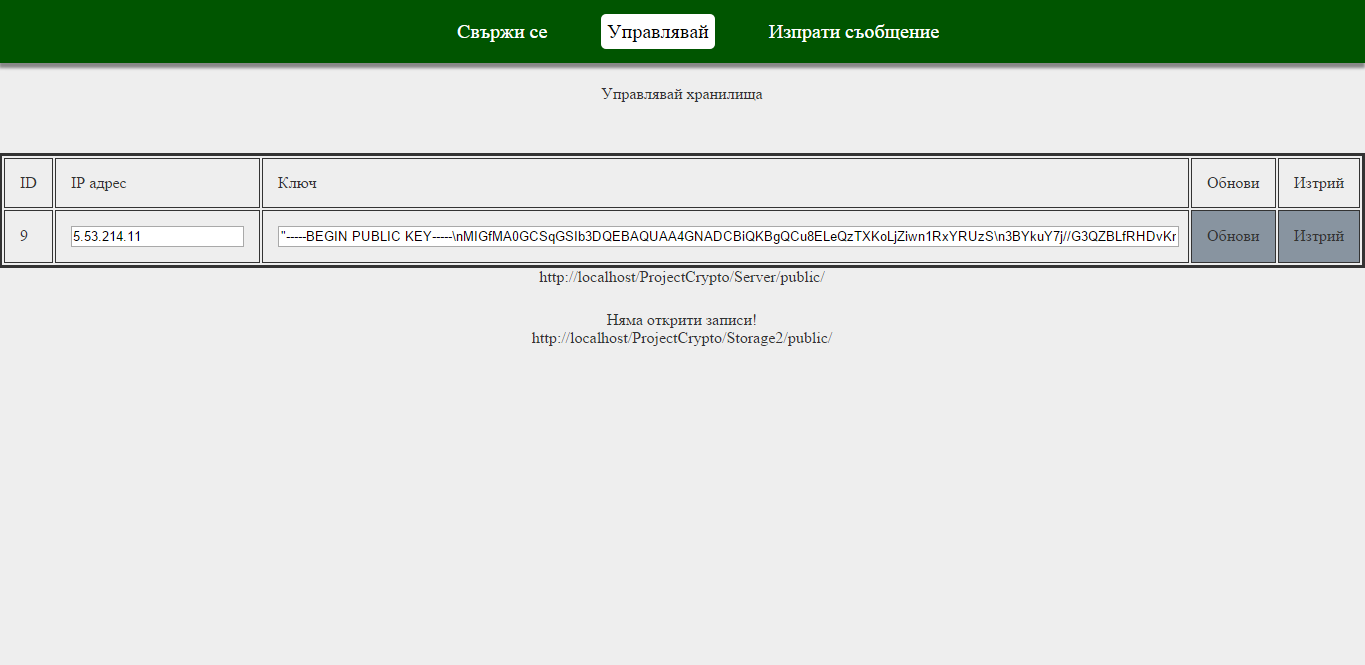
* + 1. **Свързан потребител**



След клик на бутона свързване се прави **GET** заявка към услугата <http://jsonip.com/> за определяне на IP адреса на клиента. След това се отправя **GET** заявка към **/server/storages** за определяне на адресите storage елементите. Поетапно се извършват брой заявки (в случая 2), към всеки един елемент с **GET** заявка към **/storage/:ipaddress.** След извършване на тези стъпки се попълва информацията по-горе. При клик на **Свържи се** или **Изтрий връзка** се осъществява съответно **/storage/register [POST] ( ipaddress , publickey )** и **/storage/:ipaddress [DELETE]** като не се правят допълнителни запитвания за достъп (като API-key), поради целта на проекта.

В тази стъпка (свързване) се извършва и генерация на ключове, чрез заявка **GET** към **php/generate.key.pair.php.** Функцията връща JSON форматиран обект със 2 ключа – частния и публичния. За момента от страна на клиента ги съхраняваме като localStorage елементи. Ако те присъстват при следващото зареждане на софтуера, вместо генериране на нови, те се извличат от localStorage. Това позволява използването на единствен публичен ключ за изпращане на съобщения на клиента, вместо всеки път да се налага той да се регистрира в PKI.

* + 1. **Екран за управление на Storages**



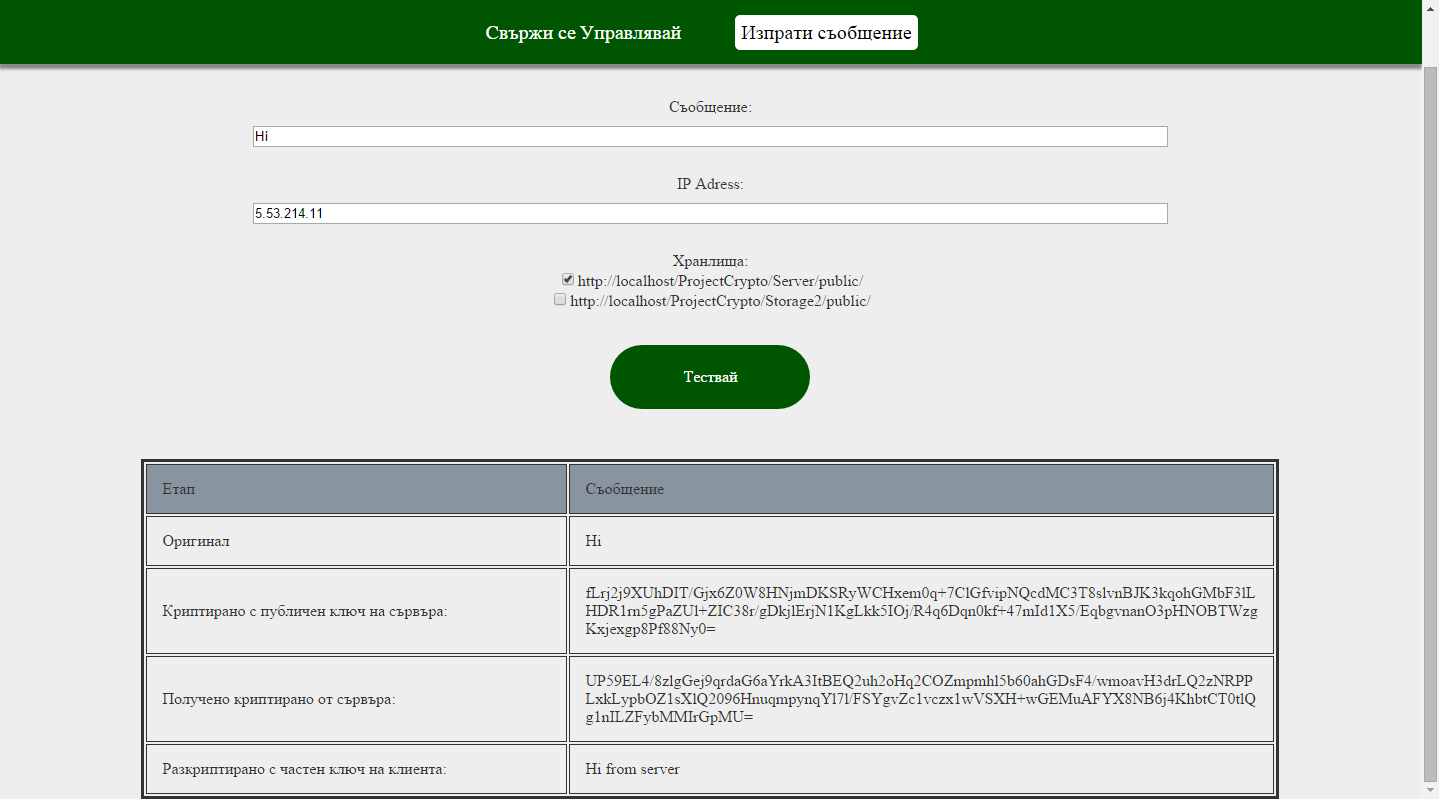
Този екран служи за преглед на всички елементи в съответен storage елемент и за тяхното управление. Това става чрез двата бутона **Обнови** и **Изтрий**, вземайки данните за съответен елемент и извършвайки както следва:

* **Обнови** - **/storage/update [PUT] ( ipaddress , publickey )**
* **Изтрий - /storage/:ipaddress [DELETE]**

За прегледност, при редакция се осветява променения елемент в списъка. Тези елементи се генерират на база на **/storage/clients [GET]** входната точка в съответен storage елемент. При не наличие на елементи се извежда съобщение обосноваващо това.

* + 1. **Екран за тестване на услугата**

След редакция на елементи и при свързване на потребител със услугата, той може да въведе съобщение което искаме да го пратим защитено на сървъра и да получим отговор от него **(echo server)**. Цялостния резултат е както следва:



Екрана ни дава възможност за избор на това кой storage елемент да се търси ключа на клиента. За тестови цели ни се дава възможност и за задаване на IPAddress.

Както можем да проследим в таблицата, оригиналното съобщение се криптира използвайки **php/public.encode.php**, чрез публичния ключ на сървъра който извличаме чрез **/server/key [GET]**.

След това се изпраща заявка към **/server [POST] ( encryptedmessage , ip, [array of storage id's] )**, съдържаща както е и описано: кодираното съобщение, IP адреса на клиента и масив със избраните storage елементи, всичко в JSON формат. След обработка на сървъра и получаване на успешен отговор използваме php-функцията **php/private.decode.php** за разкриптиране на елемента. Всички стрингове се визуализират в таблицата за лесен преглед.

В случай че не получим валиден резултат извеждаме съответна грешка на потребителя.