СОФИЙСКИ УНИВЕРСИТЕТ „СВ. КЛИМЕНТ ОХРИДСКИ”
ФАКУЛТЕТ ПО МАТЕМАТИКА И ИНФОРМАТИКА
КАТЕДРА „ИНФОРМАЦИОННИ ТЕХНОЛОГИИ”

Дипломна работа

Приложение за Pocket PC, позволяващо сигурно съхранение на лични данни

Милена Радева Йорданова
специалност „Разпределени Системи и Мобилни Технологии”
факултетен № М-21474

Научен ръководител: доц. д-р Силвия Илиева

София, февруари 2007

Съдържание

1. Увод

Информацията придобива все по-централна роля в съвременното общество. Електронните комуникации и Интернет позволяват все повече дейности, свързани с ежедневието, да бъдат извършвани по всяко време и от всяко място. Тези дейности включват банкиране и използване на брокерски услуги, пазаруване, комуникации и много други. За използването на всички тези услуги отдалечено, обаче, се налага потребителят да помни най-разнообразна информация – от потребителски имена и пароли до номера на банкови сметки и кредитни карти – информация, чиято поверителност е от първостепенно значение. В повечето случаи тази информация е трудна за помнене, а количеството, което съвременният човек трябва да помни, нараства с всеки изминал ден.

Много хора вече притежават поне по едно мобилно устройство, което им помага в натовареното ежедневие – смарт-телефон, джобен компютър, лаптоп и т.н. Тъй като тези устройства са винаги с потребителите си, те биха били много удобни за съхранение на големите обеми лична информация, която всеки има нужда да носи със себе си. Остава обаче проблемът с поверителността на тази информация. Повечето такива устройства не предлагат защита на данните от неправомерен достъп при загубване или кражба на устройството; някои, дори и да криптират основната си памет при изключване, не предприемат никакви мерки за защита на информацията, намираща се на допълнителните карти памет. За защитата на тази информация, следователно, е необходимо използването на специализиран софтуер за съответното устройство, който да предлага сигурно съхранение и достъп до нея.

Целта на настоящата дипломна работа е да предложи решение на проблема за сигурното съхранение на лични данни в мобилни устройства. Решението, което предлагаме, е приложение, позволяващо криптографски сигурно съхранение на лични данни в Pocket PC и предлагащо лесен достъп до тях. Приложението е разработено на платформата .NET Compact Framework 2.0 и предлага на потребителя достъп чрез парола до съхранената информация. За да осигури нейната сигурност, приложението изисква използването на силни пароли, които могат да представляват леко неудобство на преносимо устройство с ограничени възможности за въвеждане. По-удобно решение би било използването на смарт-карти и биометрични техники за удостоверяване на идентичността на потребителя. За съжаление, такива механизми са все още слабо разпространени, но определено са насока за бъдещо развитие.

Избрахме Pocket PC, тъй като този тип устройства предлагат най-добър компромис между преносимост, налична памет, изчислителна мощ, необходима за криптирането на информацията, както и удобство при въвеждането на сравнително големи обеми от данни (за разлика от смарт-телефоните, при които наличната памет и особено механизмите за въвеждане на информация са силно ограничени).

Платформата .NET позволява лесно да се разработват надеждни приложения за различни операционни системи и типове хардуер – персонални компютри, Pocket PC, мобилни телефони и други. .NET Compact Framework е специална версия на платформата .NET, предназначена за мобилни устройства. Тя предлага голяма част от библиотеките, налични при пълната платформа, както и допълнителни библиотеки, предлагащи специализирани контроли за потребителския интерфейс, както и функционалност, налична само при специфични мобилни устройства (като

например показване и скриване на клавиатурата на екрана на Pocket PC). Използването на платформата .NET за разработката на предлаганото решение ще позволи при нужда лесното пренасяне на приложението и на други типове устройства, като за целта ще трябва да се подмени единствено потребителският интерфейс, тъй като той е единствената част от приложението, която е специфично написана за Pocket PC. Въпреки, че .NET Compact Framework в известна степен ни ограничава, тъй като не предлага абсолютно всяка функционалност, достъпна чрез операционната система на устройството, използването му все пак е един добър компромис, тъй като разработката и поддръжката се улесняват неимоверно, приложението е сигурно и надеждно.

1.1. Цел и задачи

Целта на тази дипломна работа е разработката на приложение за Pocket PC, позволяващо лесен и удобен достъп до съхранени в криптиран вид на лични данни, от които съвременния човек има нужда. Тези данни могат да бъдат най-разнообразни – от пароли и банкови сметки, до изображения и файлове в произволен формат.

За да постигнем тази цел ние си поставяме следните задачи:

- Анализ на данните, които потребителите биха желали да използват и които приложението трябва да предлага
- Анализ и избор на сигурни криптографски техники и алгоритми
- Оценка на възможностите, които биха улеснили потребителите при работа с приложението и проектирането на подходящ потребителски интерфейс, реализиращ тези възможности

- Оценка на начините за създаване на резервни копия на данните и избор на подходящ механизъм за създаване и възстановяване на данните от такива копия
- Създаване на подходяща архитектура, която да ни помогне да реализираме лесно и качествено приложението
- Тестване на приложението и отделните му компоненти, за осигуряване на високо качество и безпроблемна работа

1.2. Полза от реализацията

Разработеното приложение е незаменим помощник в ежедневието на съвременния човек. То предоставя на потребителите си свободата да разполагат винаги със своите важни данни без да се притесняват, че тяхната сигурност може да бъде компрометирана, тъй като тяхната защита е осигурена от доказали се в практиката криптографски методи и техники.

1.3. Структура на дипломната работа

Първата част на настоящата дипломна работа изследва проблемната област, свързана с предлаганото решение. Разглеждат се спецификите и особеностите на Pocket PC и разработката на приложения за тази платформа. Отделено е внимание на различни области от криптографията, свързани с разработката на решението – изследвани са различни алгоритми за криптиране, хеш алгоритми, както и методи за извличане на криптографски ключ от парола. Накрая са сравнени вече съществуващи решения на проблема, като са посочени възможностите, които предлагат, както и предимствата и недостатъците им.

„Проектиране на решението" се занимава с конкретния проблем по разработката на приложение за Pocket PC, позволяващо сигурно съхранение на лични данни. Изследвана е функционалността, която приложението трябва да предлага – необходими механизми за защита на данните, типове данни, които

трябва да се съхраняват, операции с тях, както и възможности, улесняващи работата на потребителя с програмата. Проектирани са потребителският интерфейс, както и архитектурата на приложението. Последната, освен с различните компоненти и взаимодействието между тях, се занимава и с проблема за отзивчив потребителски интерфейс.

В „Описание на реализацията" се разглежда конкретната имплементация на приложението. Описани са в детайли основните използвани механизми, компоненти и класове, комуникацията между тях, както и използваните алгоритми.

„Тестване на приложението" описва методите за проверка на програмния код и функционалността на приложението, които бяха употребени в процеса на разработка на решението.

„Изводи и възможности за усъвършенстване" изследва поуките, извлечени в процеса на разработка на настоящата дипломна работа, както и нерешените проблеми, с които се сблъскахме и които биха били подходящ обект за изследване при евентуалната разработка на следваща версия на приложението.

Завършваме с главата „Заключение", в където правим обзор над извършената работа, следвана от списък на използваните източници, както и приложения, включващи списък на фигурите.

2. Проблемна област

2.1. Специфика и особености на Pocket PC

Pocket PC са мобилни устройства, събиращи се в дланта на потребителя, и въпреки че възможностите на тези устройства се увеличават постоянно, те не могат да бъдат третирани като персонални компютри, както поради разликите в размера им, така и поради различните механизми за взаимодействие с потребителите.

Всяко Pocket PC е снабдено с екран, реагиращ на докосване (touch screen). Тъй като този екран е малък информацията, която може да се покаже на него е силно ограничена. За разлика от персоналните компютри, където потребителят си взаимодейства с програмите чрез използването на мишка и клавиатура, тук се използва специално „перо" (stylus), с което потребителят осъществява почти всички операции. Повечето такива устройства нямат хардуерни клавиатури и за въвеждането на текст се използва софтуерна клавиатура, чиито клавиши се натискат с вече споменатото перо. Следователно трябва да приемем, че всяко взаимодействие на потребителя с програмата ще се осъществява чрез използване на стандартното за всички устройства Pocket PC перо. Тъй като писането с него и софтуерната клавиатура, или дори с вградения модул за разпознаване на почерк не е лесно, трябва да се постараем потребителският интерфейс да изисква минимална работа с клавиатурата, командите да са удобно достъпни с най-много едно или две докосвания с перото.

Друго ограничение на Pocket PC е малкият обем работна памет. Всички устройства с операционна система по-ранна от Windows Mobile 5.0 съхраняват инсталираните от потребителите приложения и създадените файлове в тази памет. Устройствата имат и вградена постоянна памет, както и разширителни слотове

за флаш памет, на които може да се съхраняват допълнителни файлове. Оперативната памет обаче е малко и се използва както от операционната система, така и от всички вървящи приложения, които може да са много на брой тъй като при Pocket PC приложенията не се затварят, а само се минимизират. Устройството с което разполагаме за разработването на тази дипломна работа (HP iPAQ hx4700) разполага с 64 мегабайта вградена оперативна памет, от които обикновено са налични 6-7 мегабайта. Тъй като приложението трябва да може да криптира и съхранява големи файлове (примерно на разширителна карта с памет или на вградената постоянна памет), то трябва да може да прави това въпреки че тези файлове може да са по-големи от наличната оперативна памет.

Друго сериозно ограничение, с което приложението трябва да е съобразено, е бавният процесор на устройството. Използваните структури от данни и операциите, извършвани с тях, следва да са така подбрани, че да минимизират използването на изчислителни ресурси. По този начин приложението би било не само бързо, но и щадящо батерията на устройството.

Тъй като Pocket PC са мобилни устройства е лесно да бъдат, загубени. За това е добре а се предложи на потребителя удобен начин да съхрани резервно копие на своите данни, за да не бъдат те безвъзвратно загубени. Това също налага и нуждата от сигурното им криптиране, за да не може никой да получи достъп до тях, дори и да има на разположение файла. За да се направи резервно копие на данните е удобно да се използва вграденото средство за връзка с компютър – ActiveSync.

2.2. Криптография

Криптографията (от гръцки κρυπτός – таен, и γράφω – пиша) е наука, изучаваща математическите техники, свързани с различни аспекти на информационната сигурност като поверител-

ност, цялост и автентичност на данните (Menezes, van Oorschot, & Vanstone, 1996). За нуждите на настоящата дипломна работа се интересуваме предимно от поверителността на информацията. За целта криптографията ни предлага механизми, който позволяват да се скрие значението на съобщение, текст или друга информация от неоторизирани лица, като в същото време тази информация е достъпна за всеки, който притежава ключа за нейното отключване. Тези механизми включват алгоритми за криптиране на информацията с ключ, хеш алгоритми, както и алгоритми за образуване на криптографски ключове от пароли.

2.2.1. Алгоритми за криптиране

Алгоритмите за криптиране позволяват преобразуването на информация в нечетим вид по такъв начин, че тя да може да бъде извлечена обратно само от тези, за които е предназначена. За тази цел алгоритмите дефинират трансформации, чрез които информацията се криптира или декриптира. За преобразуване на информацията трансформациите изискват предоставянето на криптографски ключ. Използването на ключове премахва необходимостта от пазенето на трансформациите, прилагани от алгоритъма, в тайна. По този начин е необходимо да се пази в тайна единствено ключът за отключване на информацията, като при нужда може тя може просто да бъде криптирана наново с друг ключ. Пространството от възможни ключове обикновено е достатъчно голямо, за да затрудни силно простото налучкване (или намиране чрез пълно изчерпване) на правилния ключ, но само голямо пространство от възможни ключове не е достатъчно; необходимо е трансформациите, прилагани от алгоритмите за криптиране и декриптиране да са такива, че полученият криптиран текст да не се поддава лесно на криптографски анализ.

2.2.1.1. Симетрични и асиметрични алгоритми

Криптографските алгоритми могат да бъдат разделени на симетрични и асиметрични, в зависимост от това дали ключът за декриптиране на информацията е същият като ключа за нейното криптиране или не.

При симетричните алгоритми се използва един и същи ключ (или два тривиално зависещи един от друг ключа) както за криптиране, така и за декриптиране на информацията. Това позволява алгоритмите да са прости и бързи[1], но, при използването им за поверителни комуникации за всяка двойка комуникиращи страни трябва да се използва отделен ключ, което затруднява управлението на ключовете и при определени ситуации може да представлява проблем. Освен това не е възможно комуникиращите стани да си обменят ключове сигурно, освен ако вече не съществува сигурен канал за комуникация между тях.

Асиметричните алгоритми предлагат решение на горепосочените проблеми. За разлика от симетричните алгоритми, при които един и същи ключ се използва за криптиране и декриптиране и от двете страни, при асиметричните алгоритми всяка страна притежава двойка математически свързани ключове, наричани „публичен" и „частен", които са генерирани по такъв начин, че частният ключ да не може да бъде извлечен от публичния. Частният ключ се пази в тайна и е известен само на страната, на която принадлежи, а публичният се разпространява до всички, с които трябва да бъде осъществявана комуникация. Съобщение, криптирано с публичния ключ, може да бъде декриптирано единствено чрез съответния му частен ключ, и обратно – съдържание, криптирано чрез частния ключ, може да бъде декриптирано само чрез съответния му публичен ключ. Това свойс-

[1] В практиката симетричните алгоритми могат да бъдат стотици до хиляди пъти по-бързи от асиметричните (Wikipedia, 2007).

тво на двойката ключове позволява да се осигури едновременно както сигурността на информацията[2], така и нейната автентичност[3]; получавайки съобщение, криптирано с нашия публичен ключ и подписано с частния ключ на подателя, ние можем да сме сигурни, че това съобщение е не само защитено, но и че идва от този конкретен подател.

Тези два метода за криптиране могат да се използват и в комбинация – тъй като чрез използването на публични и частни ключове може лесно да се установи сигурен комуникационен канал, този канал може да се използва за обмяната на един-единствен ключ, който да бъде използван с по-бързото симетрично криптиране през остатъка от сесията (това е например подходът, използван при SSL и други комуникационни протоколи). За приложение като това, представляващо целта на настоящата дипломна работа, обаче, което има за цел единствено сигурно съхранение на данни, защитени с ключ (или парола на потребителя), и което няма нужда да обменя тези данни с други приложения, използването на асиметричен алгоритъм е излишно. В този случай бърз симетричен алгоритъм е идеалното решение.

2.2.1.2. Симетрични алгоритми

Симетричните алгоритми се делят на два класа – поточни и блокови – според метода, по който криптират данните. Блоковите алгоритми прилагат трансформации над групи (блокове) с константна дължина; за да се криптира дадена информация, тя трябва да бъде разбита на такива блокове (последният може да

[2] Чрез криптирането ѝ с публичния ключ на получателя, така че само той да може да го декриптира със своя частен ключ.

[3] Чрез криптиране на хеш на съобщението с частния ключ на подателя. Получателят, декриптирайки съобщението с частния си ключ, може да изчисли хеша наново, след което да използва публичния ключ на подателя, за да декриптира изпратения от него хеш. Ако двата хеша съвпадат, получателят може да е сигурен, че съобщението е изпратено именно от подателя (тъй като никой друг не би могъл да го е криптирал с неговия частен ключ), както и, че не е променено по пътя (тъй като хешът, идващ от подателя, отговаря на съдържанието на съобщението).

бъде допълнен с произволни данни до постигане на необходимата дължина, ако алгоритъмът го изисква). Поточните алгоритми, от своя страна, оперират над всеки символ индивидуално; можем да мислим за тях като за блокови алгоритми с дължина на блока един символ (Menezes, van Oorschot, & Vanstone, 1996). Всеки символ от входния поток при поточните алгоритми се комбинира, най често чрез използване на операцията „побитово или", със символ от специален ключов поток, състоящ се от произволни данни. Ключовият поток най-често се получава чрез криптографски-силен генератор на псевдослучайни числа (генератор на псевдослучайни числа, чиято генерирана поредица е изключително трудно да бъде предвидена и не се поддава лесно на криптографски анализ), който е инициализиран с начален помалък ключов поток – ключа за криптиране или декриптиране на информацията. Възможно е, разбира се, и цялото съдържание на ключовия поток да бъде използвано като ключ за декриптиране на информацията, както се прави, например, при механизма „one-time pad" – механизъм, който, ако ключовият поток е наистина произволен и се използва само веднъж, е теоретически непробиваем.

2.2.1.3. Инициализационни вектори

Тъй като криптирането е детерминистична операция, при криптиране на един и същи текст с един и същи ключ винаги се получава един и същи криптиран резултат. Това представлява практически проблем, тъй като това свойство на криптирането може да се използва за атакуване на алгоритъма чрез речник. Проблемът е още по-голям при поточните алгоритми, при които използването на един и същи ключ означава генериране на един и същи ключов поток. Тъй като при поточните алгоритми всеки отделен символ от входния текст е комбиниран със съответен символ от ключовия поток (най-често чрез операцията „изключ-

ващо или", какъвто е случаят при RC4), два входни текста, криптирани с един и същи ключ следователно са комбинирани с един и същи ключов поток. Чрез комбинирането на получените криптирани текстове (отново чрез операцията „изключващо или"), следователно, ключовият поток може да бъде елиминиран и в резултат да се получи комбинация от двата текста, които сравнително лесно могат да бъдат отделени един от друг (особено ако има някакво предварително знание за евентуалното им съдържание).

За разрешаването на този проблем се налага използването на т.нар. „инициализационни вектори". Инициализационните вектори са блокове данни с произволно съдържание, които се използват при инициализацията на алгоритъма с цел избягване на повторяемостта в криптирания текст при един и същи входен текст. Това означава, обаче, че за правилно декриптиране на информацията е нужно получателят също да знае какъв е използвания инициализационен вектор, за да може да инициализира алгоритъма правилно. Това обаче не е никакъв практически проблем, тъй като инициализационният вектор може да бъде съхранен или предаден заедно с криптирания текст, защото не е нужно да бъде пазен в тайна; единствената му цел е да осигури вариации в криптирания текст при един и същи вход, докато самата информация продължава да бъде защитена от ключа. Единственото изискване е един и същи инициализационен вектор никога да не бъде използван повече от веднъж с един и същи ключ.

2.2.1.4. Избор на алгоритъм

.NET Compact Framework предлага няколко симетрични алгоритъма за криптиране: DES, Triple DES, RC2, и Rijndael (AES). Освен тях, в практиката се използват и много други симетрични алгоритми като RC4, RC5, Blowfish, Twofish, IDEA и много други.

DES (Data Encryption Standard) е един от най-известните криптографски алгоритми, разработен още през 1975 г. от IBM. DES е блоков алгоритъм използващ ключове с дължина само 56 бита, чието пълно изчерпване е възможно за часове дори и със сравнително евтин хардуер (RSA Laboratories, 2007); през 1999 г. практическата му слабост е демонстрирана на DES Challenge III, организирано от RSA Laboratories, когато текст, криптиран чрез DES е декриптиран чрез пълно изчерпване за 22 часа и 15 минути.

Наследник на DES е по-силният Triple DES, който прилага DES три пъти, използвайки два или три ключа, по този начин увеличавайки ефективните битове на ключа до 80 и 112 съответно. Triple DES се смята за достатъчно сигурен, но и той, също като DES, е много бавен, особено при софтуерна реализация.

В днешно време DES и Triple DES стават все по-малко използвани. На тяхно място идва Rijndael (произнася се „рèйндàл") – блоков алгоритъм, разработен през 1998 г., който през 2001 г. печели конкурса за новия стандарт AES (Advanced Encryption Standard) на National Institute of Standards and Technology (NIST) на САЩ. Rijndael е около 6 пъти по-бърз от Triple DES при софтуерни реализации (Wikipedia, 2007) и поддържа ключове с размер между 128 и 256 бита, кратни на 32 (въпреки, че AES спецификацията позволява само три дължини – 128, 192 или 256 бита).

RC2 (RC от „Rivest Cipher", още известен като „Ron's Code") е блоков алгоритъм, спонсориран от Lotus и разработен от Рон Ривест през 1989 г. Алгоритъмът използва ключове с размер от 8 до 128 бита (със стъпка през 8 бита). Той е податлив на атака над близки ключове.

RC4, открояващ се със своята простота и бързодействие, е най-широко използваният поточен алгоритъм. Той се използва при SSL и WEP (Wired Equivalent Privacy); последният бе разбит

чрез използване на слабостите на RC4 (отново – податливост на атака над близки ключове).

Въз основа на направените наблюдения избрахме Rijndael, поради неговата сигурност и бързодействие, което е от още по-голямо значение при устройство с бавен процесор, каквото е Pocket PC. Трябва, обаче, да обърнем внимание, че имплементирахме приложението без каквито и да е специфични предположения по отношение на използвания алгоритъм. Промяната на използвания от приложението криптиращ алгоритъм може лесно да стане просто чрез избор на нов алгоритъм, наличен в .NET Compact Framework или имплементиран допълнително чрез наследяване на базовия клас SymmetricAlgorithm. Единственото, на което трябва да се обърне внимание, ако в бъдеща версия се наложи избор на нов алгоритъм е, че трябва да се запази възможността за отваряне на файлове, създадени с предишна версия на приложението, които все още използват стария алгоритъм.

2.2.2. Извличане на ключ от парола

Тъй като криптографските алгоритми изискват ключ, трябва да има начин потребителят да предостави този ключ на приложението, за да може да се осъществи процеса на криптиране и декриптиране. Би било много неудобно, обаче, ако потребителят трябваше да помни и въвежда двоичен ключ, освен ако не се използва смарт-карта или друг подобен механизъм. Както обсъдихме в увода, обаче, механизми като смарт-карти, биометрични системи и т.н. все още не са широко разпространени; най-удобният за момента механизъм за защита на данни и удостоверяване на самоличност, при масово разпространения хардуер, е въвеждането на парола. Паролата, обаче, сама по себе си не е директно използваема от криптографските функции; за да бъде това възможно, е необходимо тя да бъде сведена по някакъв начин до ключ, отговарящ на изискванията на съответния криптограф-

ски алгоритъм. Отделно, тъй като паролите в повечето случаи се избират от силно ограничен набор от възможности, свеждането на паролата до криптографски ключ дава възможност тя да се „подсили" допълнително. Този процес се нарича „извличане на ключ от парола" и по своето същество представлява прилагане на криптографска хеш функция върху паролата плюс блок данни със случайно съдържание, наречен „сол" („salt"); за солта може да се мисли като за индекс в голямото множество от възможни ключове, генерирани от паролата(RSA Laboratories, 1999). Целта на процеса е да ни даде силен ключ, като по този начин да направи директната атака над ключа неприложима. В процеса на извличане на ключа обикновено се правят голям брой итерации на хеш функцията, като по този начин значително се увеличава времето за проверка на дадена парола, силно намалявайки ефективността на атака чрез пълно изчерпване на възможните пароли.

PKCS (Public-Key Cryptography Standards) #5 v2.0 на RSA Laboratories дефинира два механизма за извличане на ключ от парола – PBKDF2 (Password-Based Key Derivation Function 2) и остарялата PBKDF1. PBKDF2 използва псевдослучайна функция, за да получи желания ключ от паролата. Тази функция може да е както криптографска хеш функция, така и криптираща функция или HMAC (Hash Message Authentication Code); ще се спрем на това подробно по-долу. PBKDF1 използва хеш функциите MD5 или SHA-1, като може да извлича ключове с ограничена дължина (дължината на използваната хеш функция) – 128 бита при MD5 или 160 бита при SHA-1, затова нейната употреба не е препоръчителна. PBKDF2 няма ограничение за дължината на изходния ключ и е възприетият стандарт за извличане на ключ от парола. Можем да опишем PBKDF2 по следния начин:

$$Derived\ key = PBKDF2(Password, Salt, Iterations, Key\ length)$$

Тъй като PBKDF2 не е налична в .NET Compact Framework, ние имплементирахме PBKDF2 алгоритъма описан в PKCS #5 v2.0 използвайки HMAC като псевдослучайна функция, за да може приложението да се възползва от него.

2.2.2.1. Хеш функции; избор на хеш функция

Хеш функциите съпоставят стойност с константна дължина на даден текст, съобщение или друга информация с произволна дължина, като тази стойност е винаги една и съща за дадено съобщение. Приложенията на хеш функциите са много – засичане и корекция на грешки при трансмисия или съхранение на данни; индексиране в хеш таблици; както и в криптографията – за осигуряване на целостта и автентичността на съобщения. Както ще видим по-долу, криптографските хеш функции могат да бъдат полезни и при извличането на ключ от парола чрез комбинирането на слабата парола и произволна сол в силен криптографски ключ; извличането на ключа е бавна операция, като по този начин се затруднява атака чрез пълно изчерпване на паролите.

За да бъде една хеш функция подходяща за криптографски цели, тя трябва да притежава следните важни свойства:

- да бъде трудно, имайки само хеш стойността, да намерим съобщение, което да образува тази хеш стойност;
- да бъде трудно, имайки съобщение, да намерим друго съобщение, което да има същата хеш стойност;
- да бъде трудно да се намерят каквито и да е две съобщения с една и съща хеш стойност (Menezes, van Oorschot, & Vanstone, 1996, стр. 322)

MD5 (Message-Digest Algorithm 5) е една от най-разпространените криптографски хеш функции. Нейният изход е 128-битова хеш стойност. Функцията обаче има слабост, която

позволява бързото намиране на колизии, затова вече не се счита за достатъчно сигурна.

SHA (Secure Hash Algorithm) е набор от криптографски хеш алгоритми, разработени от NSA (National Security Agency) и приети от NIST за държавен стандарт на САЩ. Съществуват пет различни SHA алгоритъма – SHA-1, SHA-224, SHA-256, SHA-384, както и SHA-512. SHA-1, възприет като заместник на MD5, генерира стойности с дължина 160 бита и е изключително широко използван; използва се от комерсиални приложения, протоколи като SSL/TLS, S/MIME и други. Широкото разпространение на SHA-1 го прави обект на сериозен криптографски анализ. В резултат са открити известни теоретични слабости в SHA-1, свързани с намирането на колизии в алгоритъма със сложност по-ниска от изискваната за пълно изчерпване. Тези теоретически резултати обаче за момента не представляват заплаха за приложения, използващи функцията за извличане на хеш стойности или ключове от пароли, тъй като простата възможност за откриване на колизии в хеш функцията не улеснява по никакъв начин намирането на оригиналните данни, от които хешът е извлечен (Wikipedia, 2007). SHA-256 и SHA-512 (генерирайки стойности с дължина 256 и 512 бита, съответни), както и техните съответни разновидности SHA-224 и SHA-384, са силно сходни и общо са известни като SHA-2; различават се единствено по детайли като начални стойности, ширина на данните, използвани от операциите, както и броя вътрешни итерации в алгоритъма. До момента не са известни слабости в SHA-2 функциите, но това може да се дължи на по-слабото внимание, което получават в сравнение с SHA-1.

Тъй като при реализацията на криптографски функции е много лесно да се допусне грешка, която да намали или тотално обезсмисли сигурността на програмата, винаги е за предпочита-

не, при възможност, да се използва готова и утвърдена в практиката имплементация на алгоритъма. Ето защо, за целите на настоящата дипломна работа, се спряхме на алгоритъма SHA-1. .NET Compact Framework предлага готови имплементации на MD5 и SHA-1, от които SHA-1 е безспорният фаворит. Както обсъдихме по-горе, SHA-1 е достатъчно сигурен и наложен в практиката алгоритъм, чиито чисто теоретични слабости не са важни за сценариите, за които го използваме. Нещо повече – тъй като употребата му в разработваното приложение е ограничена единствено до извличането на ключ от паролата, подмяната на използвания от приложението хеш алгоритъм е операция, еквивалентна на проста смяна на паролата от страна на потребителя; резултатът е просто ключ с различно съдържание. Както ще видим по-долу, това не изисква повторно криптиране на вече съхранените данни поради използването на отделен ключ за криптиране на данните.

2.2.2.2. *HMAC*

HMAC (Keyed-Hash Message Authentication Code) е код, извлечен от дадено съобщение, който може да се използва за проверка на целостта и автентичността на това съобщение. Този код се пресмята чрез комбинирането на криптографска хеш функция и таен ключ. Ако две страни искат да осъществят комуникация, в която да е сигурно, че съобщенията са автентични и непроменени, те си разменят таен ключ, който се използва за генерирането на HMAC, който се изпраща заедно със съобщението. При получаване на съобщението, другата страна използва същия ключ за генерирането на HMAC, и ако двете стойности съвпадат, то съобщението е автентично.

2.3. Съществуващи приложения

eWallet, от Ilium Software, е програма в няколко версии, както за персонален компютър, така и за Windows Mobile Pocket PC

и Windows Mobile Smartphone, предоставяща на потребителите възможността да съхраняват най-различни данни, от които имат нужда – от пароли и банкови сметки, до контакти и размери на дрехи. Тя предоставя шаблони за голямо разнообразие от данни, предлага синхронизация между различните версии и има вграден генератор на пароли. Потребителите имат избор дали да използват парола за достъп до данните си и дали те да са криптирани. eWallet използва RC4 с 256 бита ключ за криптиране на данните.

Password Master for Pocket PC, от DreameeSoft, е програма за Pocket PC, включваща и версия за персонален компютър, която улеснява потребителите при въвеждането на данни и служи за синхронизация. Тази програма предлага защита на данните със 128 битов ключ. Потребителите могат да създават множество файлове с данни, като данните във всеки файл могат да бъдат организирани в йерархична структура. Поддържаните типове данни включват кредитни карти и банково сметки, PIN кодове, акаунти за уеб сайтове, пароли, софтуерни ключове. Тя има и вграден генератор на пароли. Някои от удобствата при нейното ползване включват поддържането на преместваме на записи чрез влачене и пускане (drag and drop), създаването на шаблони за въвеждане на различни данни, поддръжката на икони, и възможността за сливане на различни файлове с данни.

visKeeper PPC – Password Wallet на SRF е програма за Pocket PC, позволяваща съхранението на множество типове данни в криптиран формат, чрез използването на 128 битово криптиране с алгоритъма Twofish. Потребителите могат да създават различни файлове за своите данни, а самите данни могат да организират в йерархична структура. Потребителите могат да създават шаблони за различни типове данни, дефинирайки имената на отделните текстови полета. Програмата позволява потребителят

да задава множество настройки за отделните записи и за общият и изглед, като избор на икони и цветове. Тази програма предлага на потребителите три начина да се защитят файла си – чрез текстова парола, чрез цифров код (подобно на PIN код) и чрез картинка, за която те са избрали няколко последователни точки, които трябва да се докоснат с перото на устройството, за да се осъществи достъп до файла.

Password Minder е програма за персонален компютър, която съхранява сигурно списък с паролите на потребителя. Тя изисква силна парола за достъп и също така подпомага потребителите в използването на сигурни пароли чрез вградения генератор на случайни пароли.

3. Проектиране на решението

3.1. Функционалност

3.1.1. Сигурност на данните

Основно изискване към приложението е да предлага високо ниво на сигурност на съхранените данни. Не бива да съществува възможност за неправомерен достъп до данните, дори и при загубването на устройството или откраднването по някакъв начин на файла с данните.

3.1.1.1. Криптографски и хеш алгоритми

За да осигурим поверителността на данните ще използваме множество криптографски техники, включващи криптиране на данните с ключ чрез сигурен алгоритъм, защита чрез парола и извлечения от нея ключ, HMAC.

Както посочихме в точка 2.2 – „Криптография”, избрахме симетричния криптографски алгоритъм Rijndael за криптиране на данните, функцията PBKDF2 за извличане на ключ от паролата на потребителя, както и HMAC като псевдослучайна функция, която PBKDF2 да използва. PBKDF2 и HMAC реализирахме сами, тъй като не са налични в .NET Compact Framework. Избрахме SHA-1 като криптографска хеш функция, която да използваме за реализацията на HMAC.

3.1.1.2. Защита на данните с парола

Потребителите ще се идентифицират пред приложението с парола, а не с ключ, тъй като използването на ключове е непрактично, ако не е налично устройство, работещо със смарт-карти. Тъй като паролите, които потребителите обикновено избират са сравнително прости и не предлагат голяма сигурност избрахме да помогнем на потребителите да повишат нивото на сигурност

чрез два механизма - осигуряване на минимална ентропия на паролата и извличане на по-силен ключ от паролата. Първата цел постигаме пресмятайки ентропията, която въведената парола предлага. За да може потребителят да използва тази парола за защита на своите данни, то тази парола трябва да има минимална ентропия от 64 бита. Както описахме в точка 2.2.2 – „Извличане на ключ от парола", за да можем да използваме паролата като ключ в криптографски алгоритъм се налага тя да бъде подсилена. Това постигаме чрез PBKDF2, която ни дава ключ с необходимата дължина и засилена ентропия.

За да можем да проверим дали въведената от потребителя парола е вярна пресмятаме верификационен код на предварително известно съобщение, използвайки ключа извлечен от паролата, което съхраняваме във файла. При опит от страна на потребителя да получи достъп до данните пресмятаме наново тази стойност с ключа, който съответства на въведената парола, и сравняваме резултата с вече съхранената стойност.

При криптиране на данните можем да подходим по два начина – да използваме ключа извлечен от паролата на потребителя или да използваме произволно генериран ключ, който в почти всички случаи ще има по-голяма ентропия.

Първият подход е по-лесен за реализиране, но при него операцията по смяна на паролата за достъп налага повторно криптиране на всички данни, но тъй като приложението може да съхранява голям обем данни това би било бавно.

При втория подход се използва ключ, произволно генериран от съответния алгоритъм за криптиране, който предлага максимална ентропия. За да може все пак да получим достъп до криптираните данни без потребителя да помни ключа се налага да го криптираме използвайки ключа, извлечен от паролата на потребителя и да съхраним криптирания вариант във файла. Така, ко-

гато потребителят се опита да осъществи достъп чрез паролата си, се декриптира ключа и ако паролата е била вярна с декриптирания ключ може да се осъществи достъп до съхранените данни. Така при смяна на паролата се налага да се криптираме отново единствено ключа.

3.1.2. Типове потребителски данни

Потенциално типовете данни, които потребителите могат да поискат да съхранят за много и най-различни. В процеса на разработка на тази дипломна работа анализирахме основния набор от данни, които всички потребители биха искали да имат. Също така приложението е така проектирано, че да може добавянето на нови типове данни в следващи версии да е безпроблемно. Тук ще опишем основните типове данни, които приложението ще предлага.

3.1.2.1. Текст / лична бележка

Това е най-простият тип данни, който позволява на потребителя да съхрани лична бележка по свой избор. Състои се от текст, който потребителят въвежда. Може да се използва и ако потребителят иска да съхрани някакви текстови данни за които не е наличен специализиран тип данни.

3.1.2.2. Парола

Паролата е един от най-очевидните типове данни, които приложението трябва да съхранява. Чрез този тип данни позволяваме на потребителя да въведе стойности за следните полета – кой изисква паролата, с какво име се идентифицира потребителя и каква е самата парола. Предлагаме възможността да се съхрани и малко описание за този запис под вид на бележка.

3.1.2.3. Парола за уеб сайт

Паролата за уеб сайт е просто специализиран тип парола. Тук са налични следните полета – име на уеб сайта, потребителско име, парола и допълнителни бележки.

3.1.2.4. Информация за e-mail account

Избрахме да създадем отделен тип данни за съхраняване на информация за електронна поща, защото въпреки че има много уеб-базирани услуги за електронна поща, за които гореописаният тип „Парола за уеб сайт" би бил достатъчен, съществуват и услуги за електронна поща, които потребителите ползват през e-mail клиентски програми като Microsoft® Office Outlook® или Windows® Mail, за което е нужно да се знае и допълнителна информация за адресите на сървърите, които се използват за получаване и изпращане на поща. Чрез този тип данни позволяваме на потребителя да съхрани своя e-mail адрес, парола, SMTP и POP3 сървър адреси, както и допълни бележки.

3.1.2.5. Банкова сметка

В съвременното ежедневие много финансови операции се извършват чрез използването на банкови услуги. Самата банкова сметка не е никаква тайна и ако искаме просто да я съхраним не е нужно да я криптираме. Но освен номера на банковата сметка, може да имаме нужда съхраним и друга информация като ПИН код или данни за електронно банкиране. Също така може да искаме да съхраним различни видове сметки – като пенсионни, инвестиционни, спестовни или кредитни. Чрез този тип данни потребителите могат да съхранят и друга важна информация, като телефон за връзка с финансовата институция, който да използват при нужда.

3.1.2.6. Банкова карта

Под банкова карта имаме предвид както дебитни, така и кредитни карти. Тези карти са все по-популярни и много хора

имат по няколко от тях. С тях освен номер на картата са свързани и ПИН код, верификационен код, информация за банката издател, телефон за връзка, който е полезен при загуба на картата, както и данни за електронно банкиране.

3.1.2.7. Мобилен телефон

Този тип данни позволява съхранението информация за мобилен телефон. Това включва както информация за апарата, като производител, модел и код за защита на телефона, така и информация за телефонния номер, ПИН код на картата, и информация за телефонния оператор.

3.1.2.8. Снимка

Това е един много полезен тип данни, позволяващ на потребителите да съхранят цяло изображение в криптиран вид. Предлага се възможността изображенията да се разглеждат от самото приложение, или да се прехвърлят в заден от потребителя файл в декриптиран вид.

3.1.2.9. Файл

Възможността за съхранение на произволен файл в криптиран вид е много полезна, тъй като позволява сигурното съхранение на произволна информация. Това е и една възможност, която програмите, които разгледахме в точка 2.3 - Съществуващи приложения, не притежават.

3.1.3. Операции с данните

Тук ще разгледаме операциите, които се извършват над определен предмет.

3.1.3.1. Въвеждане

За да може един предмет да съществува, той трябва първо да бъде създаден. Тъй като имаме различни типове данни, операцията по създаване трябва да попита потребителя какъв тип данни иска да създаде след което да му предложи удобен потре-

бителски интерфейс, чрез който потребителят да може да въведе данните, които иска да съхрани.

3.1.3.2. Преглед / Редакция

След като вече имаме съхранен даден предмет, той трябва да може да бъде разглеждан, за да може потребителят да прави справка, както и да бъде променян, за да може потребителят да отрази настъпили промени.

3.1.3.3. Триене

Друга важна операция е изтриване на предмет. Тя се използва, когато потребителят не желае повече да съхранява асоциираните данни.

3.1.4. Операции с файла

Отделно от разгледаните по-горе операции с данните, съществуват и няколко операции с файла с данни като цяло, които трябва да предлагаме на потребителя.

3.1.4.1. Създаване

За да работи приложението е нужно да се създаде файл, в който да се съхраняват данните на потребителя. Приложението също така може да работи с повече от един файл, което означава, че потребителят трябва да има възможността да създава файлове.

3.1.4.2. Отваряне

Тъй като потребителят може да има повече от един файл трябва да се предостави възможност за избор на файл, който да бъде отворен.

3.1.4.3. Свиване

Поради спецификата на файла за съхранение на данните, при извършване на операции по изтриване или промяна на предмети неговата дължина нараства, създавайки участъци, чие-

то съдържание не се използва от приложението. За да може потребителят да освободи това място се предлага команда за свиване на файла, чрез която незаетите блокове се освобождават.

3.1.4.4. Създаване на резервно копие

Както вече споменахме, е необходимо да има възможност за създаване на резервно копие на данните, от което те да могат да бъдат възстановени. За целта се създава файл, съдържащ всички съхранени данни. Този файл се криптира и защитава с избрана от потребителя парола.

3.1.5. Възможности, улесняващи работата с приложението

Ако просто показваме предметите като списък, ще е много трудно да се намери конкретен предмет, особено с нарастването на броя им. За да избегнем този проблем избрахме да имплементираме няколко техники, които да помогнат на потребителите при работата с приложението.

3.1.5.1. Организация на данните

Както при файловите системи е удобно да групираме различни файлове заедно в папки и под-папки, така и в нашето приложение ще е възможно да се създават папки, които да служат като организационен елемент. Всеки предмет ще е асоциран с определена папка, и ще може да бъде местен от папка в папка. Отново по аналогия с файловите системи ще има и основна папка. За да е лесно на потребителите да идентифицират различните предмети, всеки един предмет ще е свързан с име, което потребителят е избрал за него. Разбира се, нужно е потребителят да може да променя имената на папките и предметите.

Въвеждането на структура подобна файловата система, с възможности за преименуване, местене и създаване на нови пап-

ки е удобно, но също така е и концепция, с която потребителите са вече запознати и ще възприемат с лекота.

3.1.5.2. Икони

Друго неоценимо средство за улесняването на работата с приложението е използване на икони, за по-лесното идентифициране на предметите. Ако имаме просто име на предмет, може и да не се сетим какъв точно е той, но ако имаме икона, която да ни показва типа на този предмет, ние винаги ще знаем типа на предметите без да се налага да ги помним или да ги отворим, за да проверим тяхното съдържание. За това нашето приложение асоциира подходяща икона с всеки тип предмет и я показва до името му. Така винаги е ясно, че дадени предмети са папки, пароли или банкови сметки.

3.2. Дизайн на потребителски интерфейс

Целта, която си поставяме тук е да създадем прост и изчистен потребителски интерфейс, при който всяка операция да се извършва лесно, информацията да е поднесена удобно, без да има претрупване въпреки малкият размер на екрана. Именно поради малкия екран е наложително да показваме възможно най-малко информация наведнъж. Поради това трябва да обособим показваната информация в атомарни единици и да показваме само по една от тях наведнъж.

Друго важно свойство е консистентност на потребителския интерфейс. Трябва потребителят да може да очаква разположението на контролите и всички форми да са направени така, че да е ясно, че принадлежат на едно и също приложение.

Поради съществото на приложението някои от операциите могат да са бавни. Добра практика е докато върви дадена бавна операция да се подава периодично информация за прогреса на

операцията и, ако е възможно да се предостави начин за нейното прекратяване.

3.2.1. Основни сценарии

3.2.1.1. Стартиране на приложението

При стартиране на приложението ще улесняваме потребителя като му предлагаме да отвори последния използван файл. Ако приложението няма информация кой е бил последния използван файл, то ще предлага избор за създаване на нов или отваряне на вече съществуващ файл. Тези две операции всъщност са достъпни при всяко стартиране на приложението.

3.2.1.2. Създаване на файл

Създаването на нов файл за съхранение на данни е операция, включваща няколко стъпки. Най-напред трябва да се специфицира името и местоположението на файла във файловата система. Следва избор на силна парола за защита на данните. И накрая се избира ниво на защита, което всъщност влияе на броя итерации, които PBKDF2 ще извърши. Повече итерации предлагат по-висока сигурност, но също така водят до повече време за осъществяване на процеса по верификация на паролата при отваряне на файл. За да бъде потребителят информиран колко време ще отнема този процес ще предоставим средство за тестване, което потребителят може да използва, за да провери различни нива на сигурност преди да направи своя избор.

3.2.1.3. Отваряне на вече съществуващ файл

За отварянето на вече съществуващ файл е необходимо потребителят да предостави вярната парола за този файл. След въвеждането на паролата се осъществява процеса на верификация на паролата. Както описахме по-горе, от паролата се извлича ключ, с който се декриптира истинският ключ за данните. За да проверим дали паролата е вярна пресмятаме верификационен

код за предварително известно съобщение и го сравняваме с кода, който е бил пресметнат с оригиналния ключ. Ако двете стойности съвпадат, можем да отворим файла и потребителят да получи достъп до съхранените данни.

3.2.1.4. Създаване на резервно копие

Създаването на резервно копие копира всички предмети съхранени от приложението в един файл с последователен достъп, който при нужда може да бъде прочетен от приложението, за да се възстановят загубени данни.

3.2.1.5. Създаване на папка

Папката е организационен елемент. Нейната цел е да групира дадени предмети, за да може потребителя да ги намира полесно. Папките нямат ограничение за нивото на вложеност. За създаването на папка е нужно да се зададе само име.

3.2.1.6. Въвеждане на данни

Всеки един предмет се характеризира с име, тип, данни и папка, която го съдържа. За създаването на нов предмет потребителят трябва да зададе името и типа. Тъй като имаме различни типове данни, използваме информацията за типа им, за да покажем подходящ потребителски интерфейс за въвеждането им.

3.2.1.7. Редакция на данни

Както и при въвеждането на данните, типа определя как тези данни ще бъдат редактирани. Има типове данни, като изображения и файлове, които не позволяват редакция. За тях обаче е налична операцията по копирането им в зададен от потребителя файл в декриптиран вид. За всички останали типове редактирането е същото последната фаза на създаването.

3.2.1.8. Преименуване на папка / предмет

Потребителят лесно може да смени името на папка или предмет. Причините за преименуване може да са различни - допускане на правописна или семантична грешка в името, промяна на папката или предмета, или просто желание на потребителя да използва различно име. При всички случаи това е една необходима операция, чиято липса би затруднила потребителите.

3.2.1.9. Реорганизация на данните

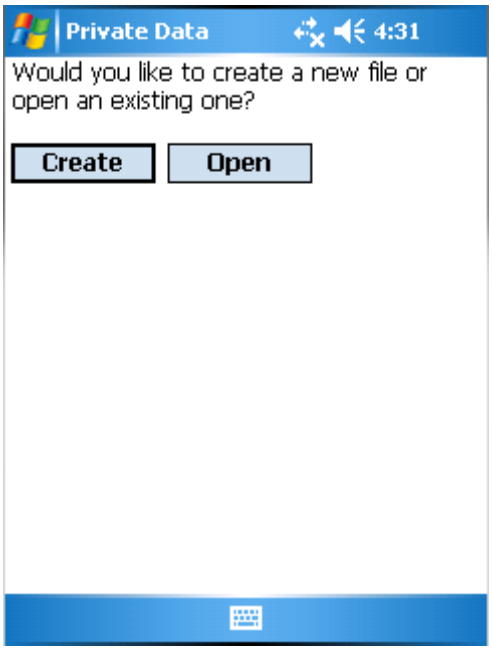
Както споменахме, потребителите могат да създават папки, които да им помагат да организират своите предмети. Също толкова важна е и възможността предметите да бъдат реорганизирани, тъй като не винаги е възможно да се прецени предварително кой е най-правилния начин за организация на данните. Дори е възможно, ако потребителят първоначално има само няколко предмета да няма нужда от тяхната организация, но с нарастването на броя на предметите да се затрудни работата с приложението и да се наложи реорганизация.

3.2.2. Екрани

3.2.2.1. Екран при първо стартиране

За да улесни потребителя приложението запомня последния файл в регистъра на устройството. Ако има такъв файл приложението го предлага на потребителя. Ако обаче приложението се стартира за пръв път, то не е запазило тази настройка. Това обаче не означава, че потребителят вече няма файл, който да иска да използва. Поради тази причина при първо стартиране на програмата на потребителя се предлага избор – да създаде нов файл или да посочи вече съществуващ файл, който да бъде използван (Фигура 1). Възможност за избор на друг файл, както и за създаване на нов файл се предлага, дори и вече да има съхранена

стойност за последно отворения файл в регистъра на устройството.

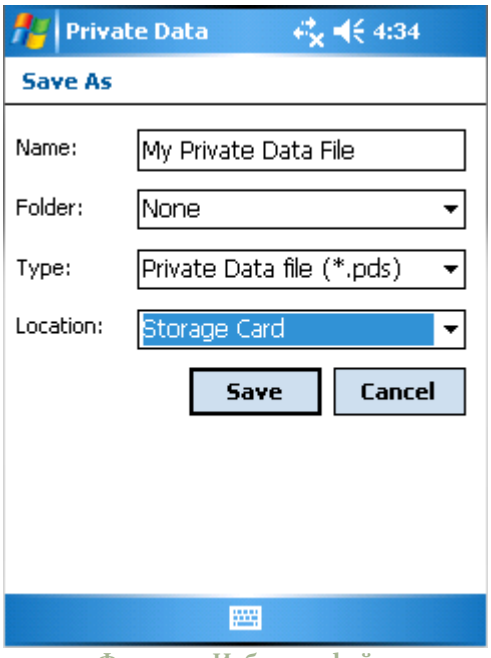


Фигура 1: Екран при първо стартиране

3.2.2.2. Създаване на нов файл

Създаването на нов файл е процес от три стъпки (Фигура 2) – избор на файл (Фигура 3), създаване на силна парола (Фигура 4) и избор на ниво на защита на паролата (Фигура 5). За първата стъпка на потребителя се показва стандартен диалог за съхранение на файл. Втората и третата стъпка ще разгледаме в следващата точка, тъй като те са възможни и самостоятелно, при операцията смяна на парола.

Фигура 2: Създаване на файл

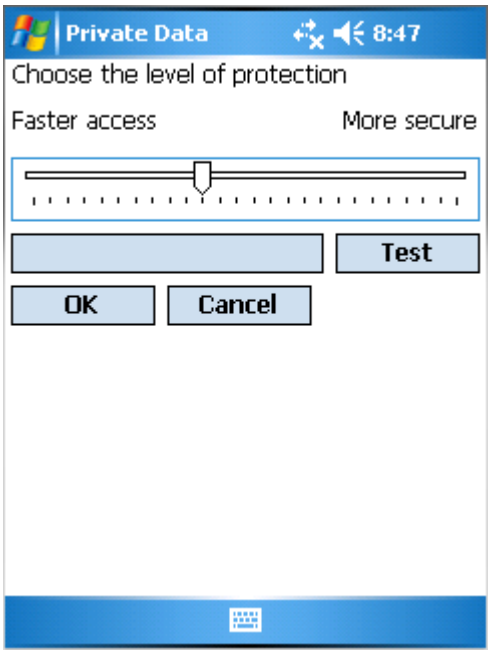


Фигура 3: Избор на файл

3.2.2.3. Създаване на парола

За създаването на нова парола, в случаите на смяна на паролата или създаване на нов файл, потребителят трябва да въведе силна парола в полето за парола и след това да въведе отново същата парола в полето за потвърждение (Фигура 4). Това е нужно, тъй като потребителят иначе може да допусне грешка и да е невъзможно да получи достъп до данните. Ако въведената парола е достатъчно силна (потребителят вижда това с помощта на индикатор на прогреса) и е една и съща и в двете полета, потребителят може да продължи. След създаването на нова парола потребителят избира и ниво на защитата на тази парола (Фигура 5), което се използва в алгоритъма за извличане на ключ от паролата и влияе на трудността на потенциална атака над паролата.

Фигура 4: Избор на парола



Фигура 5: Избор на ниво на сигурност

3.2.2.4. Оторизация

За да получи потребителя достъп до даден файл, е нужно да предостави паролата за неговото отключване и да изчака процеса на верификация (Фигура 6).

Фигура 6: Оторизация на потребителя

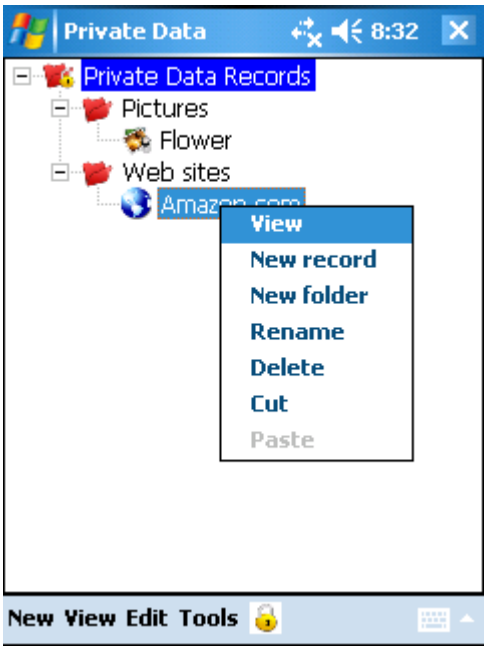
3.2.2.5. Основен екран

След като потребителят успешно е отключил файла се появява и основният екран на приложението. В него са видими всички съхранени предмети, като те са организирани по папките, които потребителят е създал (Фигура 7). За целта се използва дървовидна структура, по която потребителят може лесно да навигира. За да е по-лесна работата на потребителя с приложението до всяка папка или предмет има икона, указваща типа на предмета.

В този основен екран освен дървовидната структура са налични и менюта с команди. Те са два вида – основно и контекстно меню. В основното меню се предлагат всички команди, които потребителят може да извършва с предмет или с файла като цяло, докато в контекстното меню са достъпни командите, които може да се извикат за текущо избрания предмет. Това са команди за създаване на нови предмети и папки, преименуваме, местене, изтриване и разглеждане и редакция (Фигура 8). В основното меню Tools се предлагат команди за работа в файла – смяна

на парола, създаване на резервно копие, възстановяване на данни от резервно копие, свиване на файла.

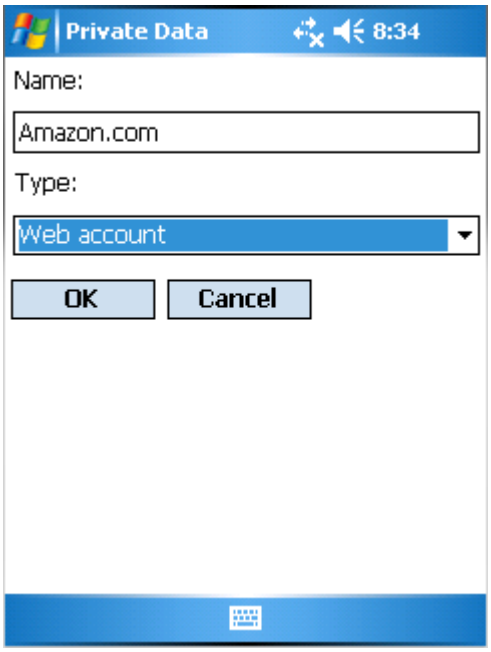
Фигура 7: Основен екран



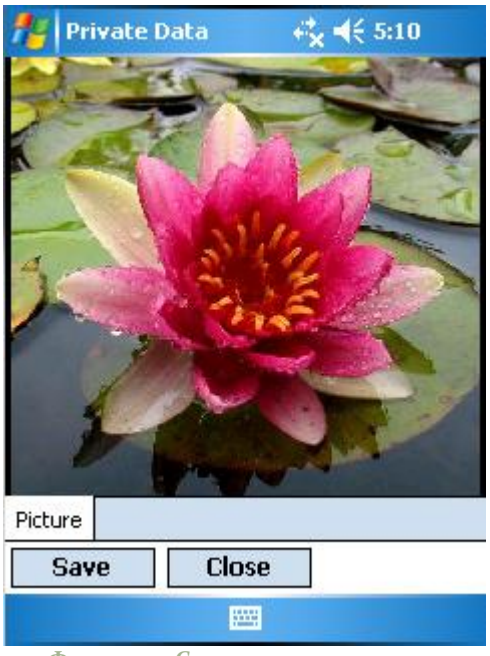
Фигура 8: Основен екран – контекстно меню

3.2.2.6. Създаване на нов предмет

Тъй като приложението работи с различни типове данни е необходимо то да знае желания тип преди да може да предостави на потребителя удобен начин въвеждането на данните. Това става по възможно най-лесният начин – чрез избиране от списък с наличните типове (Фигура 9 и Фигура 10). Потребителят трябва да зададе и име на предмета, с което да го идентифицира.



Фигура 9: Създаване на предмет



Фигура 10: Създаване на предмет

3.2.2.7. Редакция на данни

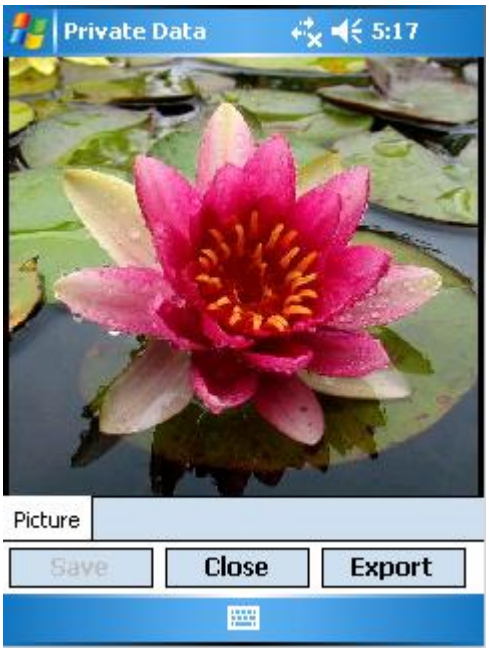
Повечето налични типове данни предлагат един и същ интерфейс както за създаване, така и за редакция и визуализация (Фигура 11). Има два типа данни, които обаче не допускат редакция - изображение (Фигура 12) и файл. Това е така, защото приложението има за цел съхранението им и не може да реализира методи за обработка на множество файлове с неизвестен формат.

Поради малката екранна площ на устройството различните полета за попълване са групирани по някакъв признак и са разпределени в различни страници (tab). Това е концепция силно използвана в множество програми за Pocket PC, с която потребителите са добре запознати. Тя помага за по-лесната работа с приложението, тъй като се избягва скролирането по екрана.

Друго улеснение за потребителя е възможността да избира от списък вместо да записва всеки път стойности, които са предвидими. Например при създаването на нов предмет от тип банкова карта, вида на картата може да се избере от списък с често използвани в практиката кредитни и дебитни карти като VISA, MasterCard и други. Това обаче не ограничава потребителя само

до предварително зададените в списъка стойности, тъй като винаги може да се въведе нова стойност.

Фигура 11: Редакция на предмет



Фигура 12: Редакция на предмет

3.2.2.8. Преименуване на предмет или папка

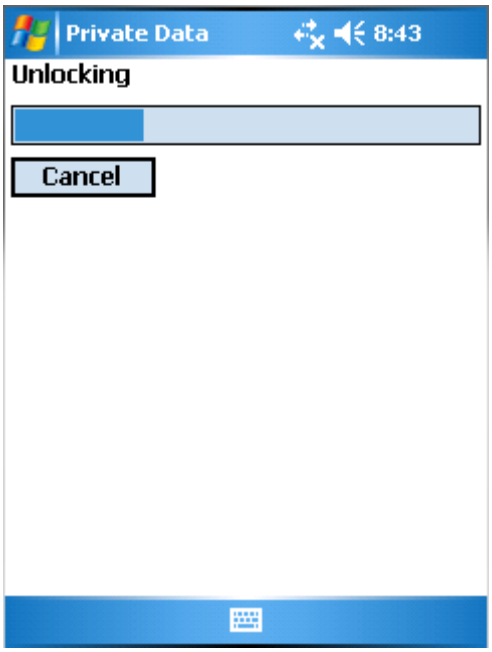
Преименуването на предмет или папка е лесно - потребителят просто въвежда новото име, като за улеснение в полето за въвеждане е изписано старото име, за да може потребителя да направи корекция, ако просто е допуснал малка грешка, без да се налага да изписва цялото име отново (Фигура 13).

Фигура 13: Преименуване на предмет

3.2.2.9. Прогрес (при бавна операция)

Тъй като някои от операциите извършвани от приложението са бавни е нужно да се покаже на потребителя до къде е стигнала операцията в своята работа и дори, ако е възможно да се предостави и начин потребителят да отмени операцията (Фигура 14). Показването на прогрес има няколко положителни страни – информира потребителя, че програма изпълнява някаква дейност и че е стигнала до определено ниво, предоставя възможност за отмяна на операцията, в случай че потребителят не иска да изчака нейното завършване, а от програмна гледна точка позволява да не се грижим за деактивирането на контроли, които потребителят не трябва да може да използва докато операцията върви. Това е така, защото диалога с прогреса е модален.

Нуждата от показване на информация за прогреса на операциите, предлагането на възможност за тяхното прекъсване и т.н., както и конкретната архитектура, позволяваща това, ще разгледаме подробно в точка 3.3.3 – „Отзивчив потребителски интерфейс”.

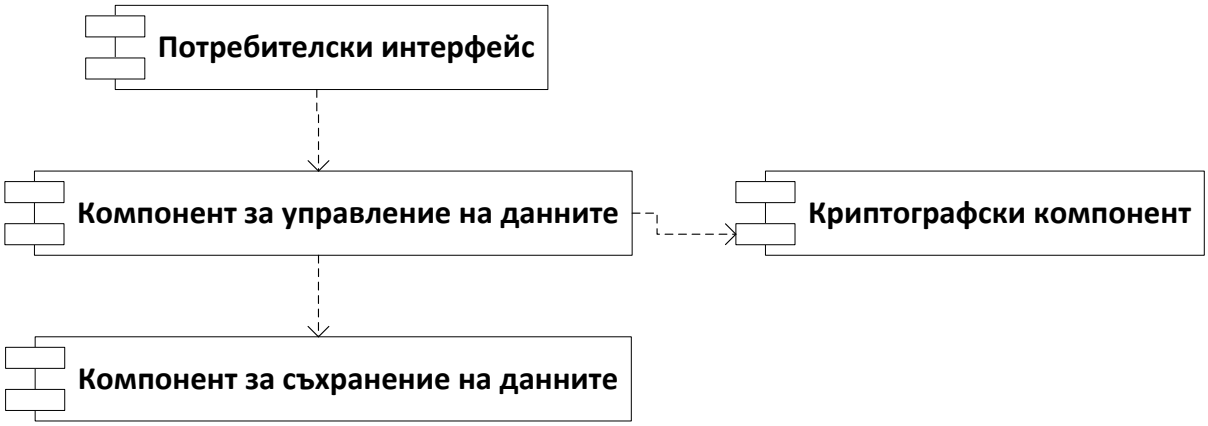


Фигура 14: Прогрес при бавна операция

3.3. Архитектура

3.3.1. Компоненти на приложението

Приложението може логически да бъде разделено на четири компонента, обособяващи отделните функционални блокове. Това са компонентът за съхранение на данните, компонентът за управление на данните, криптографският компонент, както и потребителският интерфейс (Фигура 15). Това разделение е не само логично, но и нужно, за да може приложението да има висококачествен програмен код и да бъде лесно за поддръжка. Освен това, чрез обособяване на различните функционалности в отделни модули е възможна тяхната лесна подмяна. Например, ако пожелаем да заменим потребителския интерфейс с уеб-базирана услуга (web service), това няма да наложи никаква промяна на другите компоненти.



Фигура 15: Компоненти на приложението

3.3.1.1. Компонент за съхранение на данните

Компонентът за съхранение на данните има грижата за съхранението на вече криптираните данни на приложението във файл и за извличането им обратно от него при поискване. Това е външен компонент за приложението, разработен от Стоян Йорданов като цел на неговата дипломна работа, така че в настоящата дипломна работа няма да се спираме подробно на него.

След съвместен анализ на функционалността, от която разработваното като цел на настоящата дипломна работа приложение има нужда, стигнахме до следния набор публични операции, които компонентът предлага:

- Създаване, отваряне и затваряне на файл с данни
- Свиване на файла с данни (операция, премахваща излишното свободно място от файла, което може да е останало там след изтриване или промяна на данни)
- Прочитане на списък със съхранените във файла предмети
- Четене и запис на допълнителни данни на приложението, които не са свързани с никой конкретен предмет (като например информация за използваната от приложението криптография, пароли и т.н.)

- Четене и запис на т.нар. „мета-данни” за предмет. Това са допълнителни данни, които приложението, което разработваме, може да съхрани отделно от самото съдържание на предмета, като например тип на съхранявания предмет, име, което да бъде показвано за него в потребителския интерфейс, и др. Ще се спрем по-подробно на мета-данните в точка 4.5.4 – „Metadata”.
- Четене на съдържанието на предмет
- Запис на мета-данни за предмет
- Запис на предмет (както мета-данни, така и съдържание)
- Изтриване на предмет

По-подробно описание на тези операции може да бъде намерено в дипломната работа на Стоян Йорданов (Йорданов, 2007).

3.3.1.2. Компонент за криптиране на данните

Компонентът за криптиране на данните предлага криптографската функционалност на приложението. Той предлага методи за криптиране и декриптиране на данни, както и за други важни операции като генериране на криптографски ключове, извличане на ключ от парола, както и проверка на парола. Той позволява да се постигне същинската цел на разработваното приложение – сигурност на данните.

3.3.1.3. Компонент за управление на данните

Компонентът за управление на данните реализира всички дейности с потребителските данни, които разработваното от нас приложение предлага. Той е същинската част от приложението, комуникирайки с всички останали компоненти. Всички потребителски данни минават през този компонент, като той се грижи да ги предаде на другите компоненти за криптиране, декриптиране, съхранение и т.н., на практика управлявайки всичко, което

се случва с данните на потребителя извън потребителския интерфейс.

Компонентът за управление на данните предлага набор от операции, позволяващи на потребителския интерфейс да се възползва от неговата функционалност. Те включват операции за създаване на нов файл за съхранение на данни, прочитане и запис на потребителски данни във файла, прочитане на списък със съхранените данни, преместване на предмет в друга папка, както и операции за създаване на резервно копие на данните и възстановяване на данни от резервно копие.

3.3.1.4. Потребителски интерфейс

Потребителският интерфейс предоставя на потребителя възможност да използва удобно и лесно предлаганата от приложението функционалност. Той е част от основната изпълнима част на разработваното решение, което, освен потребителския интерфейс, съдържа и функционалност за съхранение и извличане на настройките на приложението от системният регистър (registry).

3.3.2. Взаимодействие между компонентите

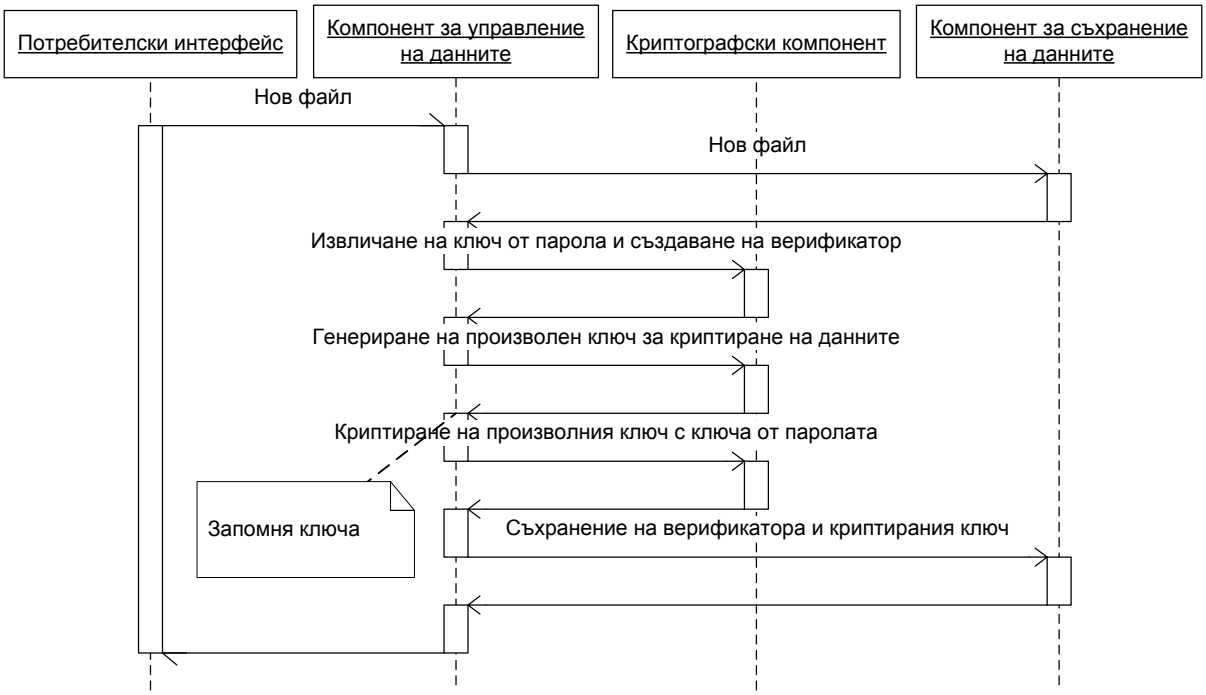
3.3.2.1. Оторизация

Фигура 16: Взаимодействие при оторизация

За да може потребителят да получи достъп до съхранените данни е нужно първо да се легитимира пред приложението (виж Фигура 16). За целта потребителският интерфейс предлага поле, в което потребителят да напише паролата за съответния файл. Потребителският интерфейс подава името на файла и въведената парола на компонента за управление на данните, който се извиква компонента за съхранение на данните да отвори файла. След това компонента за управление на данните се обръща към компонента за съхранение на данните, за да получи съхранената информация за криптографията. След това от тази информация и подадената парола се извлича ключа за криптиране на данните и се проверява дали полученият ключ е правилен. Ако е така, компонентът за управление на данните използва ключа, получен от паролата, за да декриптира основния ключ, който ще бъде из-

ползван за криптиране и декриптиране на данни от тук нататък, и уведомява основното приложение, че потребителят може да ползва този файл.

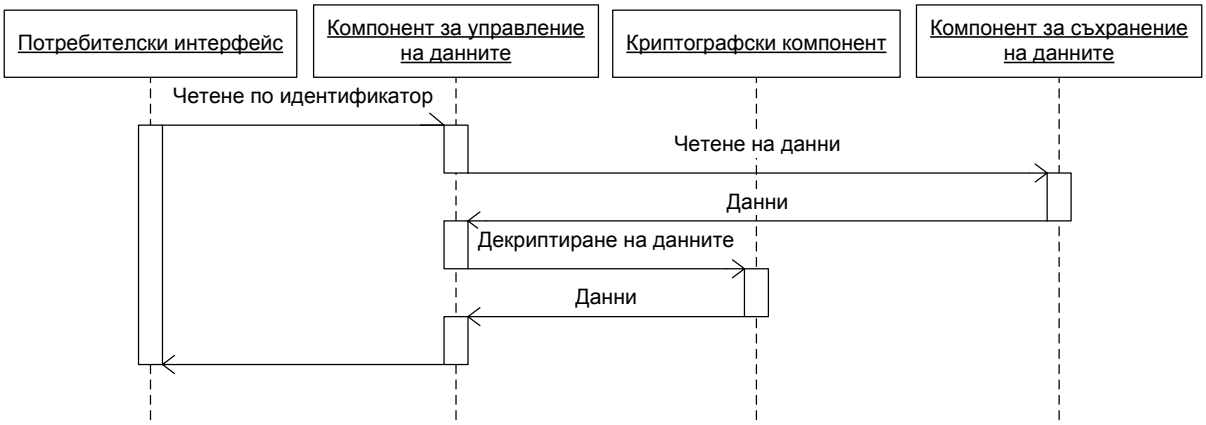
3.3.2.2. Създаване на нов файл



Фигура 17: Взаимодействие при създаване на файл

При създаването на нов файл потребителският интерфейс предлага на потребителя да си избере нов файл, които да използва за съхранение на своите данни, след което го подава на компонента за управление на данните, който извиква компонента за съхранение на данните да създаде новия файл (виж Фигура 17). Ако тази операция е успешна, компонентът за управление на данните създава информация, която е свързана с криптирането и декриптирането на данни и я подава на компонента за съхранение на данните, за да я съхрани в файла. След като вече новият файл е готов за употреба, компонентът за управление на данните може да приема заявки от потребителския интерфейс за работа с него.

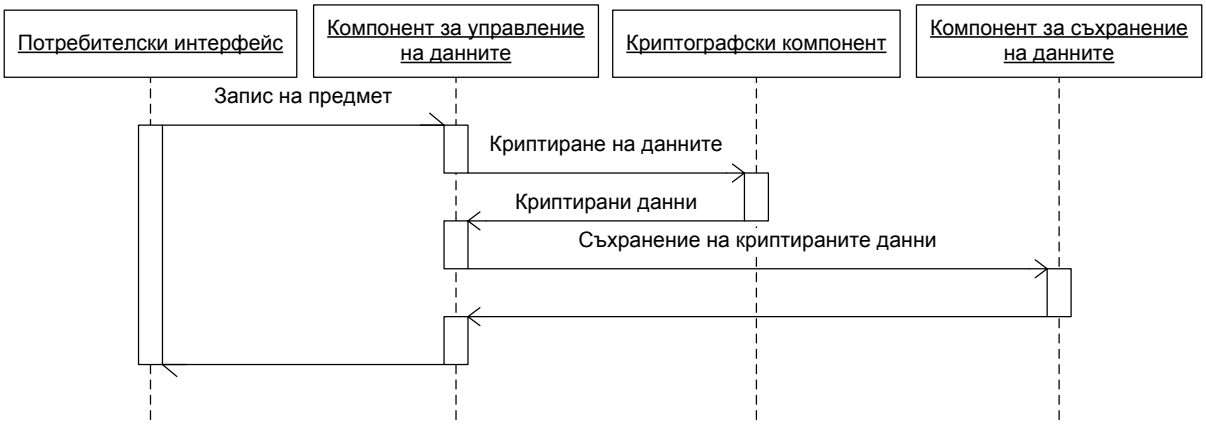
3.3.2.3. Четене на данни



Фигура 18: Взаимодействие при четене на данни

За да може потребителят да визуализира някакъв съхранен предмет е нужно той да бъде прочетен от криптирания файл (виж Фигура 18). За целта потребителят избира от потребителския интерфейс предмет, който иска да види. Потребителският интерфейс подава на компонента за управление на данните заявка за прочитането на този предмет и той от своя страна го поисква от компонента за съхранение на данните. Ако такъв предмет съществува, компонентът за управление на данните се обръща към компонента за криптиране и декриптиране на данните да декриптира получените от компонента за съхранение на данните данни. След това декриптираните данни се връщат на потребителския интерфейс, който ги показва на потребителя.

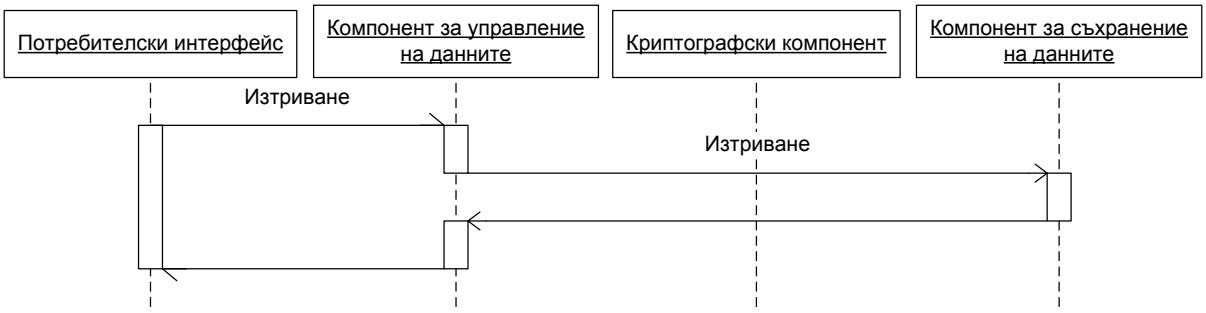
3.3.2.4. Запис на данни



Фигура 19: Взаимодействие при запис на данни

За да може потребителят да съхрани предмет във файла, потребителският интерфейс предоставя начин за въвеждане на желаната информация. След това подава тази информация на компонента за управление на данните, който се обръща към криптографския компонент, за да я криптира. Вече криптираните данни компонентът за управление на данните подава на компонента за съхранение на данните, за да бъдат записани във файла (виж Фигура 19).

3.3.2.5. Изтриване на данни

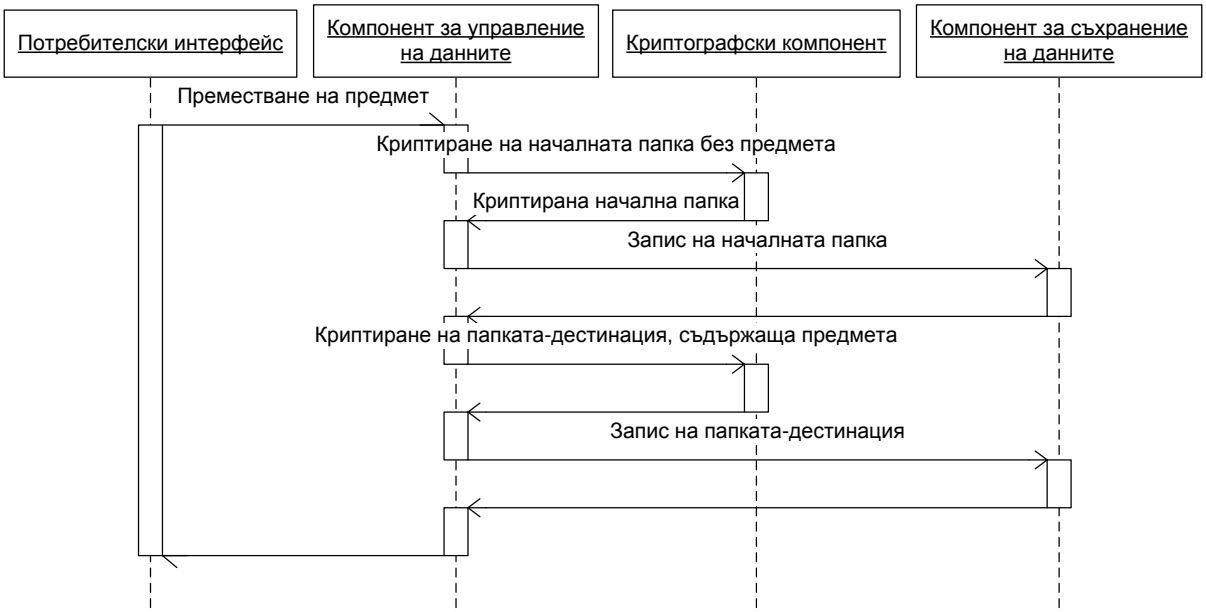


Фигура 20: Взаимодействие при изтриване на данни

След като потребителят избере предмет или папка, които да бъдат изтрити, потребителският интерфейс подава на компонента за управление на данните този предмет със заявка той да

бъде изтрит. Компонентът за управление на данните от своя страна се обръща към компонента за съхранение на данните да изтрие предмета, ако той съществува (виж Фигура 20). Ако потребителят е избрал да изтрие папка потребителския интерфейс прави заявка за изтриване на всички предмети и под-папки към компонента за управление на данните преди да изтрие избраната папка.

3.3.2.6. Реорганизация на данните



Фигура 21: Взаимодействие при реорганизация на данните

Когато потребителят желае да премести даден предмет от една папка в друга потребителският интерфейс се обръща към компонента за управление на данните, който променя желаната папка да съдържа този предмет и го премахва от папката, в която се е намирал до сега. След това промените папки се подават за съхранение във файла на компонента за съхранение на данните (Фигура 21).

3.3.3. Отзивчив потребителски интерфейс

Отзивчивостта на потребителския интерфейс на едно приложение е изключително важен фактор за приятната работа с него и за доброто му възприемане от потребителя. Дори и приложението да е бавно поради извършването на наистина сложни операции, запазването на отзивчивостта на потребителския интерфейс по време на операциите би могло да маскира факта, че те са бавни, или поне да показва на потребителя, че програмата не си губи времето, ами наистина извършва нещо, което си струва да бъде изчакано. Простото наличие на индикатор на прогреса (progress bar) и бързо опресняващи се и реагиращи прозорци може да направи иначе бавната програма да изглежда в очите на потребителя много по-бърза от програма, която изпълнява операциите си бързо, но чийто потребителски интерфейс „увисва", докато тя приключи работата, която извършва.

Начинът за постигане на отзивчив потребителски интерфейс е да не се извършва никаква (или поне не и сложна) работа в нишката на потребителския интерфейс. Всяка операция, която изисква повече от някакво минимално време за работа (около 30 милисекунди, достатъчно малко, че на практика да не бъде забелязано от потребителя), следва да бъде изпълнявано асинхронно в отделна нишка, като информация за нейния прогрес, както и резултатът от нейното изпълнение, бъдат получавани от потребителския интерфейс като събития, също както натискането на бутон или движение на мишката (Griffiths, 2003). По този начин нишката на потребителския интерфейс се освобождава и вместо да изпълнява бавната операция, тя е свободна да продължи да отговаря за изрисуването на контролите, опресняването на статуса, реакция на натискането на бутон за прекратяване на операцията и т.н. Когато операцията приключи, потребителският интерфейс може просто да реагира на събитието, уведомяващо го за нейното приключване.

За целите на настоящата дипломна работа, бихме искали разработваното от нас приложение да не увисва, защото това би било особено неприятно като се има предвид, че голяма част от предлаганата от него функционалност е бавна поради самото си естество (криптиране, декриптиране и съхранение на данни), особено на устройство с ограничени изчислителни ресурси, каквото е едно Pocket PC. Ето защо се налага да реализираме механизъм за асинхронно изпълнение на бавните операции, който да позволява визуализирането на прогреса на операцията (чрез използване на индикатор на прогреса), както и преждевременното ѝ прекратяване от страна на потребителя (чрез бутон за прекратяване на операцията).

3.3.3.1. Асинхронно изпълнение на бавните операции

Тъй като повечето, а не просто една или две, от изпълняваните от приложението операции са достатъчно бавни, за да е оправдано асинхронното им изпълнение, добре би било да имаме подходяща централна инфраструктура, позволяваща асинхронни операции и реализираща основната необходима функционалност за това, вместо да се опитваме да го правим поотделно за всяка от тях. Подобно на решението, описано в брой февруари/2003 на MSDN Magazine (Griffiths, 2003), можем да разглеждаме асинхронните операции като отделни класове в приложението. Това позволява реализирането на основната функционалност, свързана с изпълнението и управлението на асинхронните операции, като например стартиране на операцията асинхронно в отделна нишка, следене на прогреса ѝ, прекъсване на операцията, изчакването на края ѝ и т.н. да стане на едно-единствено място – в базовия клас на всички асинхронни операции. По този начин наследените класове трябва да реализират единствено конкретните бавни операции, които представят.

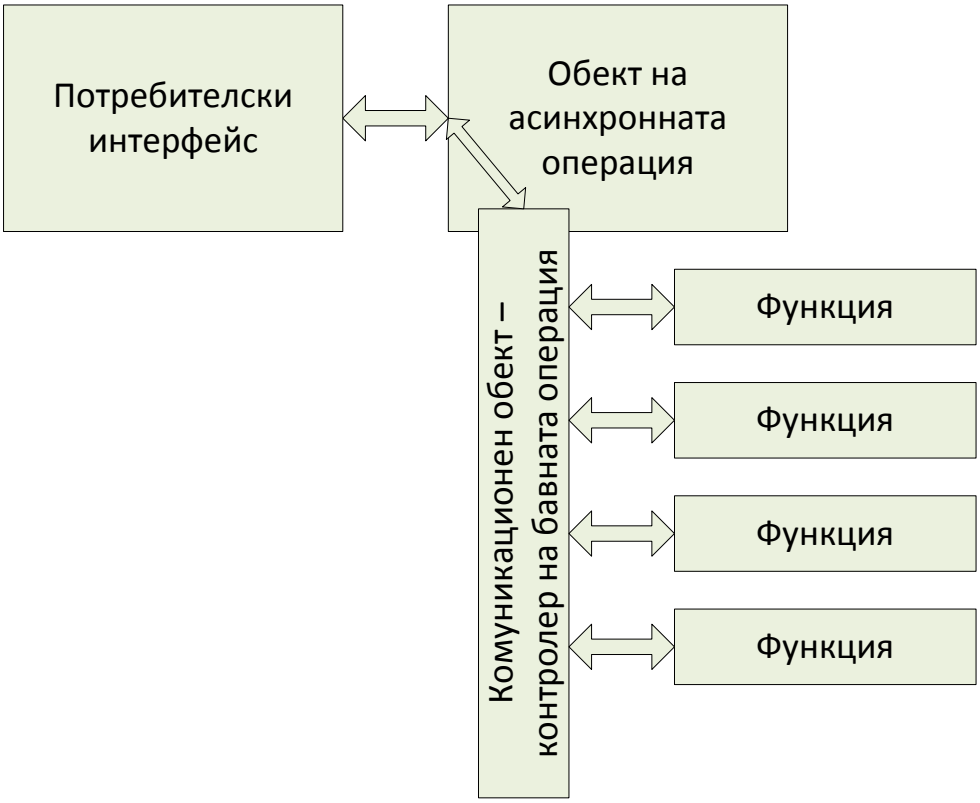
3.3.3.2. Комуникация между изпълняваната операция и потребителския интерфейс

За полезен и отзивчив потребителски интерфейс не е достатъчно просто да се освободи нишката му чрез асинхронно изпълнение на бавните операции; за целта е необходимо да се поддържа комуникация между него и изпълняваната операция по време на нейното изпълнение. Тази комуникация би позволила съобщаване на нивото на прогрес от страна на операцията, както и уведомяване на операцията в случай, че потребителят избере да я прекрати преждевременно.

Представянето на асинхронните операции като отделни класове, което описахме в предната точка, позволява да концентрираме комуникацията в една-единствена точка. Потребителският интерфейс би могъл да получава информация за статуса на операцията чрез събития от представящия я обект, както и да го уведоми в случай, че е необходимо преждевременното ѝ прекратяване. Самият обект, от своя страна, би могъл да поддържа комуникация със самия код, изпълняващ операцията, като по този начин служи като един вид ретранслатор между него и потребителския интерфейс.

За поддържане на комуникация между обекта, представящ асинхронната операция, и кода, който я изпълнява, можем да използваме помощен комуникационен обект – контролер на бавната операция. За целта този комуникационен обект се създава от обекта на асинхронната операция преди нейното започване и се предава надолу по веригата на изпълнение, когато тя бъде стартирана. По този начин всяка от функциите във веригата на изпълнение на операцията разполага с механизъм за комуникация с представящия я обект, а чрез него – и с потребителския интерфейс (Фигура 22). Функциите могат да се възползват от този механизъм, уведомявайки периодично комуникационния

обект за нивото на прогрес; той уведомява чрез събитие обекта на асинхронната операция, който от своя страна препредава това събитие към потребителския интерфейс. Функциите могат, също така, в подходящ момент от изпълнението си, когато операцията може да бъде прекъсната безопасно, да проверяват чрез комуникационния обект дали потребителят е изявил желание операцията да бъде прекратена (при такова желание от страна на потребителя потребителският интерфейс уведомява обекта на операцията, който установява съответния флаг в комуникационния обект), и ако това е така, да прекратяват изпълнението си.



Фигура 22: Комуникация между потребителския интерфейс и асинхронната операция

Важно е да се отбележи, че тъй като комуникационните обекти спомагат за комуникацията между различни нишки (нишката на потребителския интерфейс и нишката, изпълняваща асинхронната операция), методите им следва да бъдат обезопасени за изпълнение от няколко нишки (thread-safe) чрез използването на механизми за заключване на данните.

3.3.3.3. Изчисляване на нивото на прогреса

Отчитането на нивото на прогреса чрез комуникационен обект, предаван надолу по веригата на изпълнение на операцията, извежда на преден план интересният проблем за правилното му отчитане. Тъй като функциите, намиращи се дълбоко в сърцето на операцията, нямат ясна представа за това в каква фаза от нейното изпълнение се намират, те няма как да докладват правилно точно до къде е стигнала операцията с изпълнението си; всяка функция може да докладва само за собствения си прогрес.

Този проблем се решава, като си спомним, че всички функции, намиращи се по-високо във веригата на изпълнение на операцията, също докладват за нивото на собствения си прогрес. Възможно е, следователно, знаейки до къде всяка от функциите по веригата е стигнала, да се опитаме да извлечем информация за общия прогрес на операцията. Ако знаем, например, че най-външната функция на операцията има да изпълни пет стъпки, то можем да кажем, че всяка от тези пет стъпки отговаря на двадесет процента от изпълняваната операция. Ако втората от тези пет стъпки се състои в извикването на друга функция, имаща десет стъпки за изпълнение, то в такъв случай всяка една от тези десет стъпки отговаря на една десета от двадесетте процента, отделени на тази функция, или два процента от цялата операция.

Въпреки, че отделните стъпки най-вероятно отнемат различно време, което би правело отчитането на прогреса неравномерно спрямо изтеклото време, тестовете ни показват, че това решение се държи много добре в практиката. Индикаторът на прогреса не се движи равномерно, като вместо това избързва по време на бързите стъпки и забавя своя ход по време на по-бавните, но все пак това не е неприятно за окото и дава доста добра представа за степента на прогрес на операцията.

За реализирането на гореописаната логика трябва да се поддържат три различни съобщения за докладване на прогреса – съобщение, с което всяка функция, отчитаща прогреса си, съобщава, че започва работа и предава информация за броя на стъпките, които очаква да изпълни; съобщение, с което докладва, че вече е изпълнила определен брой от стъпките си; както и съобщение, с което функцията съобщава, че е изпълнила всичките си стъпки и че следващото съобщение за докладване на прогреса всъщност ще идва от родителската ѝ функция[4].

[4] Всъщност, теоретически, второто и третото съобщение могат да бъдат обединени. Докладвайки, че е изпълнила последната от стъпките си, функцията на практика докладва, че е изпълнила всичките си стъпки и че следващото съобщение ще бъде от родителската ѝ функция. От друга страна, наличието на изрично съобщение, което отчита този факт, прави нещата по-ясни и позволява кода на програмата да бъде по-устойчив на грешки, тъй като съобщенията са по-недвусмислени.

4. Описание на реализацията

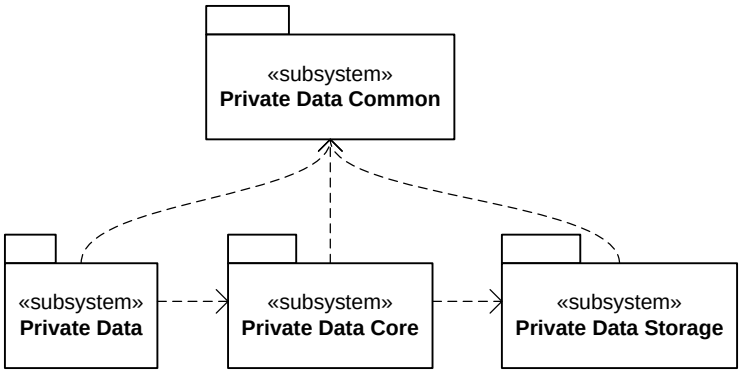
4.1. Основи

4.1.1. Комуникация чрез потоци

Тъй като оперативната памет на Pocket PC е силно ограничена, а разработваното приложение трябва да може да съхранява големи файлове, които може и да надвишават размера на оперативната памет, се налага да се реализира механизъм, чрез който данните да се обработват без да е необходимо да са заредени изцяло в паметта. Избрахме да реализираме този механизъм, чрез използването на потоци за предаване на данните между различните компоненти и методи на приложението. Потоците позволяват данните да се четат или пишат на по-малки блокове, с желания от нас размер, ефективно премахвайки проблема за обработката на файлове, по-големи от наличната оперативна памет. Криптографските алгоритми, които използваме извършват своите операции именно върху потоци, което прави използването на потоци за пренос на данните в приложението един логичен избор, избягващ излишното преобразуване на данните. Нещо повече – можем да създадем криптиращ или декриптиращ поток върху потока с данни, който искаме да обработим, което ще ни позволи да реализираме тези операции с използването на минимален обем оперативна памет и без излишни усилия от наша страна.

4.2. Физически пакети

Разработеното приложение се състои от четири физически пакета (Фигура 23).



Фигура 23: Физически пакети

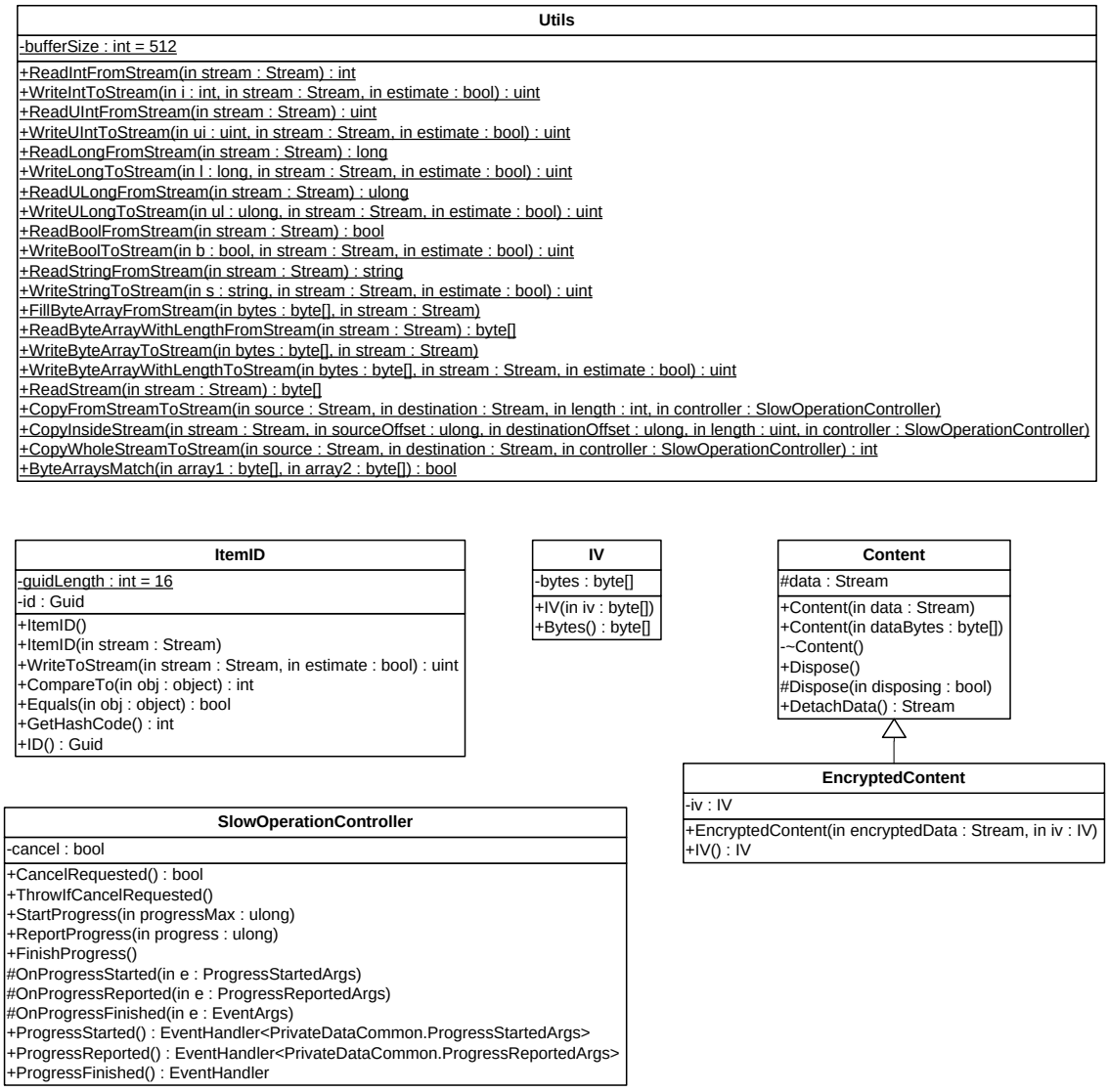
Private Data Common предлага обща функционалност, която се ползва от другите пакети. В класа Utils са реализирани редица помощни методи за работа с потоци, които използваме за лесното записване и четене на прости типове данни като int, long, string и други. Реализирани са също така и пет основни за приложението класа на които ще се спрем подробно по-долу.

Private Data Storage отговаря за съхранението на вече криптираните данни на приложението в файл, както и за тяхното прочитане от този файл. Този пакет е разработен като част от дипломната работа на Стоян Йорданов – „Компонент за съхранение на криптирани данни в Pocket PC". Този компонент ще разгледаме само в неговата публична част.

Private Data Core е сърцевината на разработеното приложение. Тук са реализирани класовете за криптографски операции, управление на данните, както и класовете представляващи различните типове потребителски данни.

Private Data е изпълнимият пакет на приложението. Той съдържа потребителския интерфейс с всички форми и диалози, както и класове реализиращи асинхронното изпълнение на операции, които потребителският интерфейс използва за изпълнението на бавни операции.

4.3. Private Data Common



Фигура 24: Класове на Private Data Common

4.3.1. Помощни методи – клас Utils

В класа Utils реализирахме множество методи, позволяващи лесната работа с потоци. Тъй като класовете в .NET, реализиращи работата с потоци предлагат единствено методи за писане и четене на байтове, а ние имаме нужда да оперираме с по-сложни типове е добре да изнесем тази функционалност на централно място. За това реализирахме методи за запис и прочитане от поток за следните типове данни – bool, int, uint, long, ulong, string,

byte array. Също така реализирахме и методи за копиране в поток.

4.3.2. ItemID

Класът ItemID капсулира представянето на идентификатора на потребителските данни. Този идентификатор е средството за разпознаване на отделните предмети и се използва във всички други пакети. Тъй като идентификаторът бива съхраняван във файла с данни, класът ItemID реализира методи за записването и прочитането си от файл.

4.3.3. IV

Класът IV реализира представянето на инициализационните вектори, които криптографският компонент използва за криптиране и декриптиране на данните, а компонентът за съхранение на данните съхранява заедно с всяка единица криптирана информация, за да може тя да бъде правилно декриптирана.

4.3.4. Content

Класът Content е основният клас представляващ съдържание в некриптиран вид. Той реализира комуникацията с потоци между отделните компоненти на приложението. Той съдържа в себе си поток, носещ съдържанието на потребителските данни и може да се конструира както от поток, така и от масив от байтове.

4.3.5. EncryptedContent

Класът EncryptedContent е наследник на Content, но тъй като той носи криптирани данни, освен наследения от Content поток, той съдържа и инициализационен вектор, с който данните в потока са криптирани. При криптиране на данни Content се преобразува в EncryptedContent, а при декриптиране процесът е обърнат.

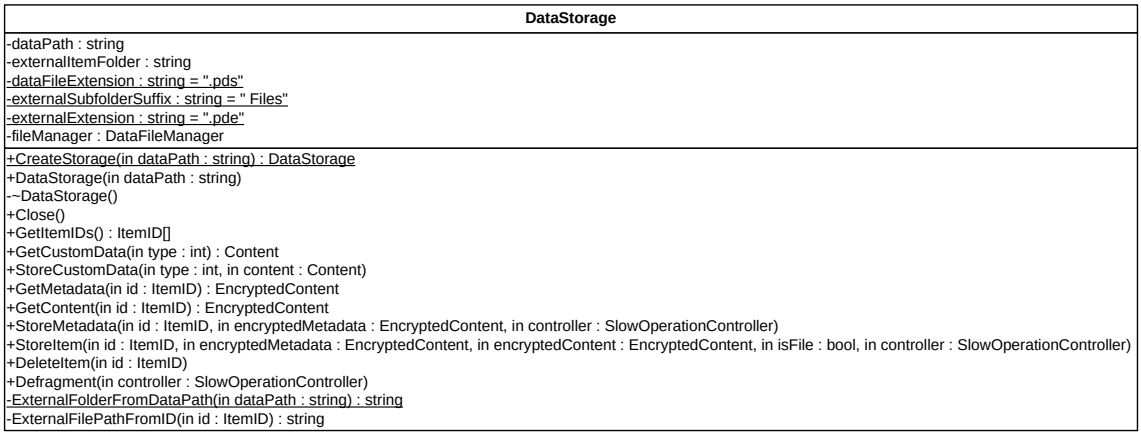
4.3.6. SlowOperationController

Класът SlowOperationController реализира механизма за комуникация между обект за асинхронна операция и изпълняващия я код, който описахме в точка 3.3.3.2 – „Комуникация между изпълняваната операция и потребителския интерфейс". Той представлява комуникационен канал предлагащ методи за комуникация от кода към обекта на операцията, както и от операцията към изпълнявания код.

Методите за съобщаване на прогрес StartProgress, ReportProgress и FinishProgress се изпълняват от кода на операцията, като при изпълнението им SlowOperationController хвърля събития ProgressStarted, ProgressReported и ProgressFinished. Както ще видим в 4.6.1 – „Асинхронни операции", за тези събития се абонира базовият клас за асинхронна операция – AsyncOperation.

SlowOperationController предоставя и механизъм, чрез който обектът на операцията може да съобщи на изпълняващия я код, че иска да бъде прекратена. Това се реализира чрез флаг, който операцията може да вдигне и който изпълняващият код проверява, когато прекратяването на операцията би било безопасно.

4.4. Private Data Storage



Фигура 25: Публичен интерфейс на Private Data Storage

Компонентът за съхранение на данните предлага необходимата на приложението функционалност във вид на класа DataStorage, чиито публични методи предлагат цялата функционалност, необходима за съхранение, извличане и работа с данните на разработваното от нас приложение (Фигура 25). Един обект от класа DataStorage е свързан с файл за съхранение на данни и извършва всички операции свързани с него.

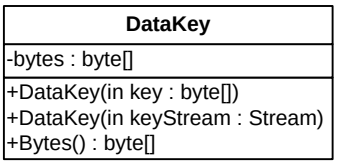
Класът DataStorage предлага следните публични методи:

- Статичен метод за създаване на обект от класа DataStorage, който създава нов файл за съхранение на данни
- Конструктор, който отваря съществуващ файл с данни
- Метод за извличане на идентификаторите на всички съхранени предмети
- Метод за съхранение на предмет
- Метод за съхранение на мета-данни на предмет
- Методи за извличане на мета-данните и съдържанието на предмет
- Метод за изтриване на предмет

- Методи за съхранение и извличане на допълнителни данни на приложението, които не са свързани с конкретен предмет (например криптографски заглавен блок)
- Метод за свиване на файла

4.5. Private Data Core

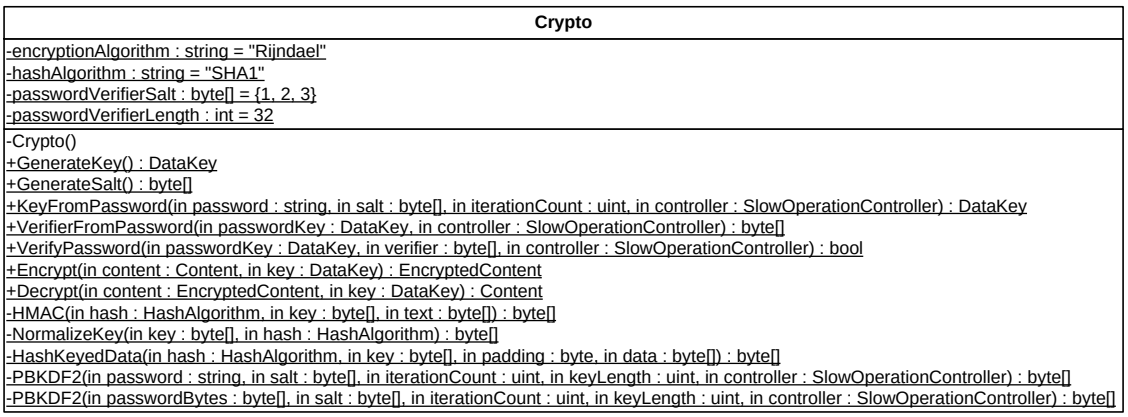
4.5.1. DataKey



Фигура 26: Клас DataKey

Класът DataKey представлява криптографски ключ, който класът Crypto използва за извършването на операциите по криптиране и декриптиране на данни. Самият клас DataKey не създава ключове – той е просто една обвивка над истинския ключ и може освен от масив с байтове (типа на генерираните от криптографските класове ключове) да бъде създаден и чрез поток.

4.5.2. Crypto



Фигура 27: Клас Crypto

Класът Crypto извършва всички криптографски операции. За операциите криптиране и декриптиране се създават криптиращи и декриптиращи потоци, като за да се получат желаните крипти-

рани или декриптирани данни, трябва да се осъществи просто четене от потока.

Освен операциите криптиране и декриптиране, класът Crypto предлага и методи за генериране на ключове и блокове със произволно съдържание (използвани за внасянето на случаен елемент при криптирането с един и същ ключ), извличане на криптографски ключ от парола чрез PBKDF2, и проверка на верността на парола.

За нуждите на приложението реализирахме PBKDF2 (Password-Based Key Derivation Function) следвайки алгоритъма описан в PKCS #5 v2.0, като реализирахме и HMAC като псевдослучайна функция. RFC 2140 дава следната дефиниция за HMAC: $HMAC = H(K\ XOR\ opad, H(K\ XOR\ ipad, text))$, където H е криптографска хеш функция (ние използваме SHA-1), K е ключ (в случая това е паролата), $text$ е обработваното съобщение (използваме солта), $ipad$ и $opad$ са константи, съдържащи n пъти байтовете $36h$ и $5Ch$ съответно, като n е дължината в байтове на блоковете, обработвани от хеш функцията (64 за SHA-1).

4.5.3. EncryptionHeader

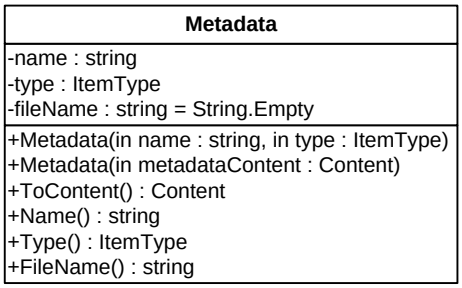
EncryptionHeader
-passwordSalt : byte[] -iterationCount : uint -passwordVerifier : byte[] -encryptedDataKey : byte[] -dataKeyIV : byte[]
+EncryptionHeader(in headerContent : Content) +EncryptionHeader(in password : string, in iterationCount : uint, in controller : SlowOperationController) +ToContent() : Content +GetDataKey(in password : string, in controller : SlowOperationController) : DataKey +ChangePassword(in oldPassword : string, in newPassword : string, in newIterationCount : uint, in controller : SlowOperationController) -EncryptDataKey(in dataKey : DataKey, in password : string, in iterationCount : uint, in controller : SlowOperationController)

Фигура 28: Клас EncryptionHeader

Класът EncryptionHeader представя информация, необходима за осъществяването на криптиране и декриптирана на данни. Този обект се записва във файла за съхранение на данните и се прочита от файла при неговото отваряне. Той съдържа информация, чрез която може да се провери верността на паролата, а

също и криптирания ключ, който е използван в този файл. Класът EncryptionHeader предлага метод за извличането на този ключ, но за да стане това, трябва да е подадена правилната парола за файла. Този клас се грижи и за операцията по смяна на паролата, тъй като промяната на паролата води до прекриптиране на ключа.

4.5.4. Metadata



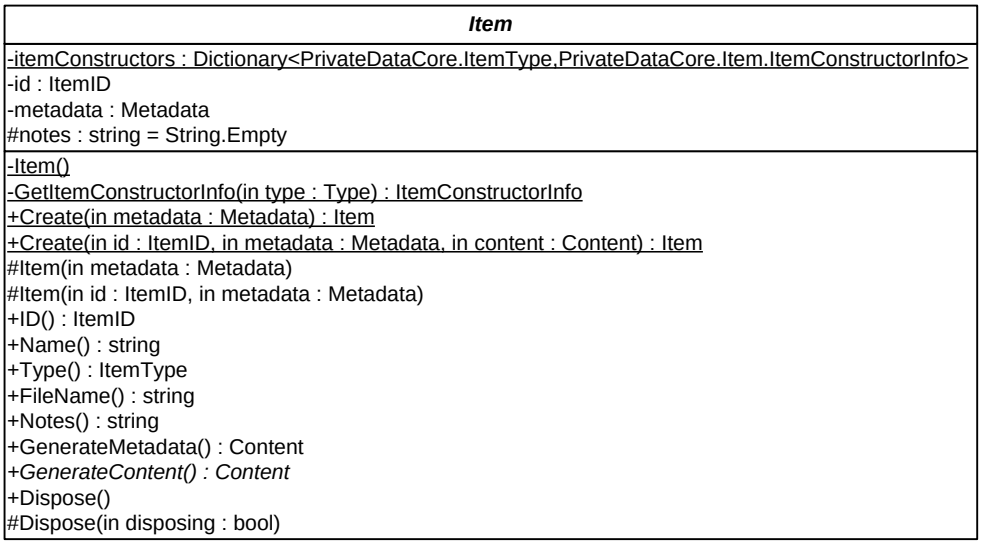
Фигура 29: Клас Metadata

За да изясним функцията на класа Metadata, първо ще се спрем на понятието „мета-данни”. Това са допълнителни данни към даден предмет, които не са част от неговото съдържание, но са нужни на приложението, за да може да работи с предмета.

Класът Metadata дефинира два вида мета-данни за даден предмет – име и тип. Името е необходимо, за да може потребителите да правят разлика между различните съхранени предмети, а типа се използва от потребителския интерфейс за предоставянето на подходящи контроли за редакция и визуализация.

Тъй като мета-данните се записват във файла, класът Metadata предоставя метод за конструирането си от Content обект, както и за конструирането на Content обект от себе си.

4.5.5. Item



Фигура 30: Клас Item

Класът Item е абстрактен клас, представляващ един предмет с потребителски данни. Той съдържа идентификатора и мета-данните за предмета. Понеже предметите биват записвани във файла (а както видяхме за целта се използва Content обект), Item предлага метод за конструиране на Content от мета-данните на предмета, както и абстрактен метод за конструиране на Content от същинските данни на предмета, който неговите наследници трябва да реализират, освен ако също не са абстрактни.

Тъй като имаме много различни типове предмети (наследници на Item) и искаме тяхното създаване да е лесно и безпроблемно, ще изнесем тази функционалност на едно централно място – в класа Item. За целта ще използваме статичен метод (този механизъм е известен като class factory), който създава предмет от правилния тип. Това ще е и единственият начин за създаване на обекти от тип Item.

4.5.6. ItemTypes

ItemTypes е общо название за класовете наследяващи Item. Тези класове отговарят на потребителските данни, които разгледахме в 3.1.2 – „Типове потребителски данни”.

Всички класове, които ще разгледаме по-долу съдържат по два частни конструктора, които са необходими за да може Item да създава техни обекти. Единият от тези конструктори се използва за създаване на нов предмет от този тип, за който още потребителят не е въвел данни, а другият – за създаването на вече запазен във файла предмет, който прочита същинското съдържание на предмета от Content обект.

Тези класове също така реализират и абстрактния метод за генериране на Content обект – GenarateContent от базовия клас Item.

4.5.6.1. BankAccount

Това е класът представящ банкова сметка. Той съдържа набор от текстови полета (string) указващи стойности като име на финансовата институция, вид и номер на сметката, ПИН код, телефон за поддръжка, информация за електронно банкиране.

4.5.6.2. CreditCard

Това е класът представящ кредитна или дебитна банкова карта. Той се състои от текстови и числови полета указващи име на финансовата институция, тип и номер на карта, име посочено на картата, срок на валидност, ПИН код и код за верификация на картата, телефон за поддръжка, както и информация за електронно банкиране.

4.5.6.3. GenericPassword

Този клас представя парола от произволен тип. Той съдържа полета указващи кой изисква паролата, с какво име се идентифицира потребителят, каква е самата парола.

4.5.6.4. WebAccount

WebAccount може да разглеждаме като специфична парола – този клас съдържа полета указващи името на уеб сайта, който изисква паролата, потребителско име и парола.

4.5.6.5. Email

Този клас съдържа полета указващи адрес на електронна поща, парола, както и адреси на POP3 и SMTP сървъри, които потребителят да използва от своя клиент за електронна поща (например Microsoft Office Outlook®).

4.5.6.6. CellPhone

Този клас представя информация за мобилния телефон на потребителя. Този клас съдържа полета описващи производителя и модела на телефона, ПИН код и код за защита на телефона, IMEI номер, телефон за поддръжка, както и информация за уеб сайта на телекомуникационната компания и потребителското име и паролата на потребителя за този сайт.

Всички гореописани класове съдържат и поле за допълнителни бележки към предмета, което потребителят може да използва за съхранението на произволна текстова информация.

4.5.6.7. Note

Това е класът представящ типа данни бележка. Той съдържа единствено текстово поле, което служи за съхраняване на бележка на потребителя.

Класовете Picture и FileItem ще разгледаме в следващата точка 4.5.7 – „Интерфейс IFileItem”.

4.5.7. Интерфейс IFileItem

Има още два класа, които наследяват класа Item – Picture и FileItem. Те обаче са по-особени, тъй като техните данни не са въведени от потребителя, а идват от някакъв файл и налагат по-специално обработване на данните. За да отразим тази разлика

реализирахме интерфейс IFileItem, който предлага методи Import и Export, които да осъществяват трансфер на данни между файла и обекта и обратно. Причината да реализираме IFileItem като интерфейс, а не като клас, е че в .NET не се поддържа множествено наследяване, а класовете, които наследяват IFileItem вече наследяват класа Item.

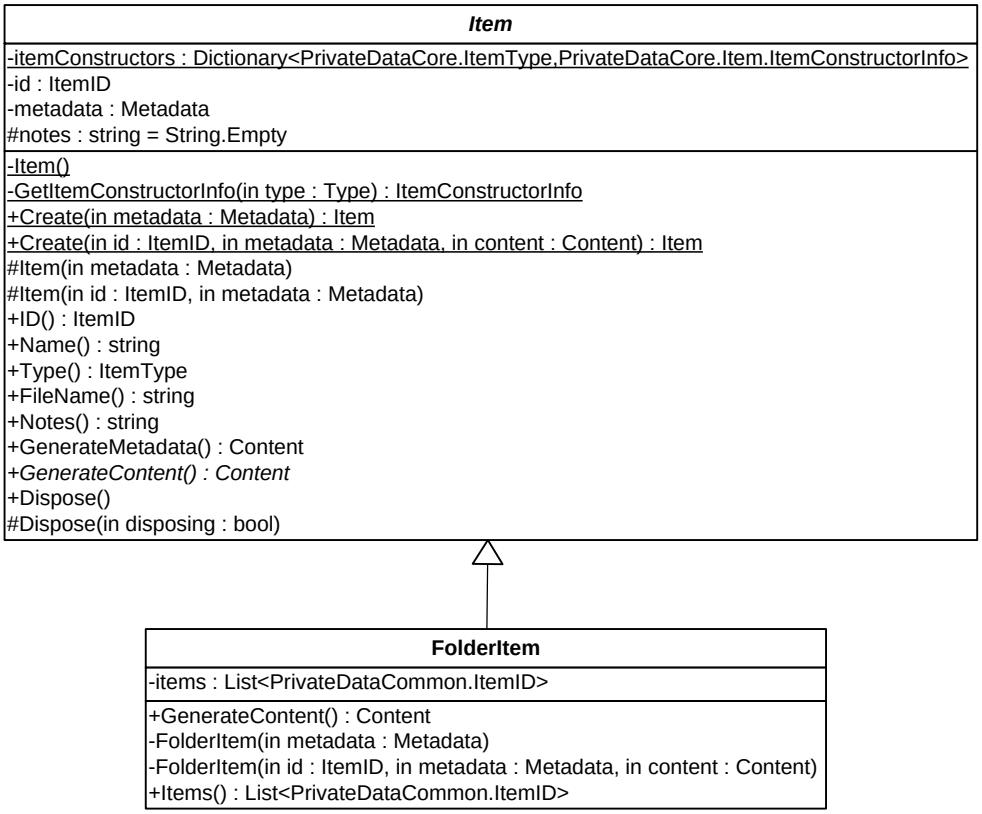
4.5.7.1. Picture

Това е клас представящ изображение. Той съдържа поток с данните за това изображение и предлага методи за взимането на данни от файл и записването на данни във файл.

4.5.7.2. FileItem

Това е клас представящ данни за произволен тип файл. Също като класа Picture той съдържа поток с данните на файла и предлага методите Import и Export.

4.5.8. FolderItem



Фигура 31: Клас FolderItem

FolderItem е по-специален предмет. Той всъщност не носи въведени от потребителя данни, а е представяне на организационната структура, изградена от потребителя. FolderItem реализира обекта „папка”, който разгледахме подробно в 3.1.5.1 – „Организация на данните”. Данните, които FolderItem съхранява са всъщност идентификаторите на всички предмети (включително и под-папки), които директно се съдържат в него. За него важат всички общи неща, които изброихме при Item.

4.5.9. ItemTree

Класът ItemTree поддържа в паметта дървовидната структура на предметите, съхранени от потребителя. Тази структура е дърво, а не списък, тъй като потребителят може да организира съхранените предмети в папки и под-папки. Тази структура се

опреснява при всяка промяна в организацията на данните като изтриване, добавяне или преместване на предмет в друга папка.

4.5.10. DataManager

DataManager
-key : DataKey -backupKey : DataKey -storage : DataStorage -itemTree : ItemTree -backupPassword : string
+CreateStorage(in dataPath : string, in password : string, in iterationCount : uint, in controller : SlowOperationController) +Login(in dataPath : string, in password : string, in controller : SlowOperationController) : bool +Close() +GetMetadata(in id : ItemID) : Metadata +GetItem(in id : ItemID) : Item -StoreMetadata(in id : ItemID, in metadata : Metadata, in controller : SlowOperationController) +StoreItem(in item : Item, in folder : ItemID, in controller : SlowOperationController) +DeleteItem(in id : ItemID, in controller : SlowOperationController) +RenameItem(in id : ItemID, in name : string, in controller : SlowOperationController) +GetFolderItems(in folder : ItemID) : ItemID[] +MoveItemToFolder(in item : ItemID, in folder : ItemID, in controller : SlowOperationController) +ChangePassword(in oldPassword : string, in newPassword : string, in newIterationCount : uint, in controller : SlowOperationController) +BackupData(in fileName : string, in password : string, in iterationCount : uint, in controller : SlowOperationController) +RestoreData(in fileName : string, in password : string, in controller : SlowOperationController) -backup_ReadItem(in sender : object, in e : ItemReadEventArgs) -backup_ReadCustomData(in sender : object, in e : CustomDataReadEventArgs) +Defragment(in controller : SlowOperationController) #OnItemTreeReloaded(in e : EventArgs) #OnItemStored(in e : ItemEventArgs) #OnItemDeleted(in e : ItemEventArgs) +TestLoginSpeed(in iterationCount : uint, in controller : SlowOperationController) +ItemStored() : EventHandler<PrivateDataCore.DataManager.ItemEventArgs> +ItemDeleted() : EventHandler<PrivateDataCore.DataManager.ItemEventArgs> +ItemTreeReloaded() : EventHandler<System.EventArgs>

Фигура 32: Клас DataManager

DataManager е класът, отговарящ за всички операции с данните. Той предлага цялата функционалност за работа с данните, от която потребителският интерфейс има нужда, като се обръща към класовете Crypto и DataStorage за извършване на операциите по криптиране и декриптиране, както и на всички операции свързани със съхранението на данните, съответно. Той предлага методи за:

- Създаване на нов файл за съхранение на данни;
- Отваряне на съществуващ файл (включва проверка на паролата на потребителя);
- Извличане на мета-данни и съдържание на предмет;
- Съхранение на предмет;

- Изтриване на предмет;
- Местене на предмет в папка;
- Смяна на паролата за защита на файла;
- Създаване на резервно копие на данните;
- Възстановяване на данни от резервно копие;
- Свиване на файла.

DataManager се грижи да криптира данните, преди да ги съхрани и да ги декриптира, преди да ги върне на потребителския интерфейс. Използват се събития, чрез които потребителският интерфейс се уведомява за настъпили промени като съхраняването и изтриването на предмет, както и за промяна на дървовидната структурата на предметите. DataManager също е отговорен за поддържане на дървовидната структура с предмети в актуално състояние, съответстващо на записаното във файла.

4.5.11. ArchiveFileManager

ArchiveFileManager е класът, който се грижи за записването на всички съхранени от потребителя данни в резервен файл, както и за възстановяването им от такъв файл. Този клас се използва от DataManager, при неговите операции за създаване на резервно копие и за възстановяване от резервно копие. ArchiveFileManager предоставя методи за записване на предмет и допълнителни данни, както и за прочитането на целия архив. ArchiveFileManager използва събития за да уведоми DataManager, за прочитането на предмет или допълнителни данни.

4.6. Private Data

4.6.1. Асинхронни операции

Както вече обсъдихме в точка 3.3.3.1 – „Асинхронно изпълнение на бавните операции”, за постигането на отзивчив потребителски интерфейс е нужно всяка бавна операция да бъде изпъл-

нявана асинхронно, в отделна нишка. За целта използваме йерархия от класове, в която всяка асинхронна операция е представена от отделен клас.

4.6.1.1. AsyncOperation

Базовият клас на йерархията от асинхронни операции е класът AsyncOperation, реализиращ основната функционалност, свързана с асинхронното изпълнение на операциите. Класът предлага абстрактен метод Execute, който наследените класове трябва да предефинират, и който представлява същинската бавна операция, която трябва да бъде изпълнена асинхронно. При изпълнението на предлагания от AsyncOperation публичен метод BeginExecute, базовият клас има грижата да изпълни метода Execute на наследения клас в нова нишка. При това на Execute се подава обсъденият в точка 3.3.3.2 – „Комуникация между изпълняваната операция и потребителския интерфейс" комуникационен обект, в лицето на SlowOperationController, който да бъде предаван надолу по веригата на изпълнение на операцията с цел подпомагане на комуникацията между обекта на операцията и изпълняващия я код. Освен това, базовият клас прихваща всички изключения, които могат да бъдат предизвикани по време на изпълнението на операцията, за да може да бъде установен резултатът от изпълнението ѝ (успех/неуспех/прекъсната операция) правилно при приключването на нейното изпълнение и потребителският интерфейс да бъде надлежно уведомен чрез събитие за този резултат.

Класът AsyncOperation освен това предлага събития, за които потребителският интерфейс може да се абонира, за да следи за завършването на операцията, както и за промяната на прогреса ѝ, което описахме в точка 3.3.3.3 – „Изчисляване на нивото на прогреса". Реализиран е и метод Wait за изчакване на операцията, в случай че потребителският интерфейс иска да изчака ней-

ното завършване преди да предприеме някакви други действия, както и метод CancelAndWait, който прекратява операцията и изчаква нейното завършване[5]. Свойствата Message и Cancellable, от своя страна, които трябва да бъдат предефинирани от всеки наследен клас, връщат информация за името на асинхронната операция, което да бъде показано в потребителския интерфейс, както и за това дали операцията може да бъде прекратена преждевременно безопасно или не.

Повечето от асинхронните операции, които сме създали просто извикват в метода си Execute метод на DataManager, който да извърши самите действия. Такива са операциите:

ArchiveOperation осъществява съхранението на данните на потребителя в резервно копие. За целта в метода си Execute, тя вика метода BackupData на обекта DataManager, който всъщност ще извърши работата асинхронно.

ChangePasswordOperation осъществява операцията по смяната на паролата за защита на файла. В метода си Execute тази операция вика метода ChangePassword на DataManger, който проверява дали въведената текуща парола на файла е вярна, и ако това е така пристъпва към смяната ѝ. Ако паролата е грешна, то потребителят няма право да я смени и операцията получава изключение, указващо това.

CompactFileOperation за свиване на файла;

CreateStorageOperation за създаването на нов файл с данни;

ExportItemOperation за записването във файл на предмет, наследяващ интерфейса IFileItem (като изображение и файл);

LoginOperation за проверка на паролата на потребителя и отключване на файла;

[5] Операцията не спира мигновено, тъй като тя може да прекрати изпълнението си само в момент, когато това е безопасно.

MoveItemToFolderOperation за преместването на предмет (или цяла папка) в папка;

RenameItemOperation за преименуване на предмет;

RestoreOperation за възстановяване на данни от резервно копие;

StoreItemOperation за съхраняване на предмет във файла с данни;

TestLoginSpeedOperation за тестване на скоростта за отключване на файл при избор на нова парола;

Ще разгледаме по-подробно операцията *DeleteItemOperation*, която извършва малко по-сложни. Тъй като потребителят може да изтрие не само предмет, а и цяла папка с предмети и под-папки се налага операцията да може да изтрие множество предмети. За целта при своето създаване тя изготвя структура стек с всички предмети, намиращи се под избрания за изтриване предмет. Ако този предмет не е папка, то в стека има само един елемент. Чрез използването на стек предметите се подреждат така, че да бъдат изтривани от долу на горе, като по този начин една папка бива изтрита, чак когато се изтрият всичките и предмети и под-папки. При извикването на метода Execute операцията се грижи за отчитането на прогреса (знаейки колко предмета трябва да бъдат изтрити и колко вече са били изтрити) и извиква DataManager да изтрие всеки следващ елемент в стека до тяхното изчерпване.

4.6.1.2. AsyncOperationProgressTracker

Класът AsyncOperationProgressTracker служи за отчитане на прогреса на асинхронна операция, като може да работи с вложеност на операциите, както описахме в точка 3.3.3.3 – „Изчисляване на нивото на прогреса”. Този клас приема обект от тип ProgressBar (контрол от WindowsForms за изрисуване на

прогрес), на който трябва да се показва прогреса на операцията. Тъй като операцията е на различна нишка спрямо този обект, AsyncOperationProgressTracker се грижи да извика опресняването на контрола за прогреса на правилната нишка с метода BeginInvoke, който всички контроли имат.

4.6.1.3. AsyncOperationUI

Класът AsyncOperationUI се грижи за показването на диалог (ProgressDialog) с прогреса на изпълняваната бавна операция (диалогът използва AsyncOperationProgressTracker за отчитането на прогреса) и бутон за нейното прекратяване, ако операцията го позволява. Обектите на този клас са свързани със асинхронна операция, контрол, изпълняващ операцията и методи за обработка на събитията, предоставени от този контрол. AsyncOperationUI получава събитията на операцията и ги предава на предоставените от контрола методи за тяхната обработка.

4.6.2. Settings

Класът Settings предоставя функционалност за съхранение на различни настройки на приложението перманентно по прозрачен начин, така, че останалата част от приложението да може да ги съхранява и извлича бързо и лесно. Тази функционалност се предлага във вид на свойства за четене и писане, по едно за всяка възможна настройка, която може да бъде съхранена. За конкретната реализация избрахме съхранението на тези настройки да става в системният регистър (registry).

Класът предлага свойството DataPath, позволяващо на приложението да съхранява и извлича пътя към последния отворен файл с данни. Това позволява той да бъде отворен по подразбиране при следващото стартиране на програмата.

4.6.3. Program (основна програма)

Класът Program съдържа входната точка на програмата. Той се опитва да създаде обект DataManager, който е необходим за работата на приложението. За да можем да използваме този обект, той трябва да бъде свързан с файл със съхранените от потребителя данни. Ако такъв файл не съществува, потребителя може да го създаде.

Както описахме в предната точка „Settings", налична е функционалност, позволяваща на приложението да запише в системния регистър пътя до последния използван от него файл. При старт на приложението, то се опитва да прочете тази стойност. Ако тя не съществува, на потребителя се предоставя избор (чрез диалога OpenOrCreateDialog, описан по-долу) да създаде нов файл или да отвори вече съществуващ такъв. Ако потребителят избере да създаде нов файл (чрез CreateStorageDialog), след неговото създаване пътят му се запомня в системния регистър. Ако ли пък потребителят посочи вече съществуващ файл и успее успешно да го отключи (чрез LoginDialog), се запомня пътят към него.

От друга страна, ако при отваряне на приложението се окаже, че в системния регистър вече съществува път към последно отворения файл, то приложението направо показва диалога за отключването му (LoginDialog).

При успешно създаване на нов файл или при отключване на вече съществуващ файл, потребителят бива препратен към основния екран на приложението, а при неуспех приложението прекратява своята работа.

4.6.4. Форми и диалози

Основното приложение съдържа и класове, реализиращи различните форми и диалози, разгледани в точка 3.2.2 – „Екрани". Тук ще ги разгледаме накратко, посочвайки техни осо-

бености, както и по-интересни моменти от имплементацията на тези от тях, които реализират някаква по-сложна логика.

4.6.4.1. OpenOrCreateDialog

OpenOrCreateDialog реализира екрана описан в 3.2.2.1 – „Екран при първо стартиране". Този диалог се показва на потребителя, когато приложението няма информация за последния отворен от потребителя файл. Той предлага избор на потребителя дали да създаде нов файл за съхранение на данните си, или да отвори вече съществуващ файл, в случай че такъв файл съществува.

4.6.4.2. CreateStorageDialog

CreateStorageDialog реализира екрана описан в 3.2.2.2 – „Създаване на нов файл". Той описва стъпките, през които потребителя трябва да мине, за да създаде файла – избор на име на файла, въвеждане на силна парола и избор на ниво на защита. За първата стъпка CreateStorageDialog показва на потребителя стандартен диалог за съхранение на файл. След като потребителят избере къде и под какво име да се съхрани новия файл, CreateStorageDialog показва NewPassworDialog за избор на парола за защита на файла, който ще разгледаме след малко. Ако той върне успех, потребителят трябва да избере и ниво на защита чрез SecurityLevelDialog (също разгледан по-долу). Ако и този диалог върне успех, CreateStorageDailog стартира асинхронна операция CreateStorageOperation за създаване на новия файл, подавайки ѝ като параметри избраните от потребителя име на файл, парола и ниво на защита. Чрез AsyncOperationUI обект, свързан с асинхронната операция и методи за обработка на събития на операцията, на потребителя се показва диалог с прогреса на операцията, а при нейното завършване се уведомява приложението за изхода и.

4.6.4.3. LoginDialog

LoginDialog реализира екрана описан в 3.2.2.4 – „Оторизация”. За неговото създаване трябва да се специфицира името на файла, който ще бъде отключван. Диалогът показва това име на потребителя и предоставя поле, където да бъде въведена паролата за този файл. При натискане на бутона OK, диалогът създава асинхронна операция LoginOperation с параметри името на файла и въведената парола, и я стартира, за да провери дали така въведената парола е вярна. Ако това е така, този диалог връща успех, а в противен случай уведомява потребителя, че въведената парола не е вярна. Този диалог предлага и възможността да се създаде нов файл за съхранение на данни, както и да се избере друг съществуващ файл, който да бъде отворен.

4.6.4.4. NewPassworDialog

NewPasswordDialog реализира първата част от екрана описан в 3.2.2.3 – „Създаване на парола” – избор на сигурна парола. Този диалог се използва при два случая – създаване на нов файл или смяна на паролата за защита на вече съществуващ и текущо отворен файл. И в двата случая потребителят трябва да въведе силна парола с определен брой битове минимална ентропия. За да знае потребителят дали въведената парола е достатъчно силна, диалогът пресмята нейната ентропия и показва индикатор на прогреса (progress bar), показващ нейната сила. Потребителят трябва да въведе тази парола втори път, за да се предотврати възможността за допускане на грешка при въвеждането ѝ. Само ако двете полета съвпадат и въведената парола е достатъчно силна потребителят може да продължи. При случая на смяна на паролата, диалогът предоставя още едно поле, в което потребителят въвежда текущата парола.

4.6.4.5. SecurityLevelDialog

SecurityLevelDialog реализира втората част от екрана описан в 3.2.2.3 – „Създаване на парола" – избор на ниво на защита. Той предоставя на потребителя плъзгач (slider), за избор на това ниво. Тъй като от избраната стойност зависи скоростта на отваряне на файла, на потребителя се предоставя възможност да тества тази скорост чрез бутона Test. Чрез индикатор на прогреса потребителя вижда колко време ще отнема процеса по отключване на файла.

4.6.4.6. ItemListForm

ItemListForm реализира екрана описан в 3.2.2.5 – „Основен екран". Тази форма визуализира съхранените от потребителя предмети в дървовидна структура, която се поддържа синхронизирана с файла, чрез абониране за събития за промяна на данните на класа за управление на данните (DataManager). Формата предоставя менюта за извършване на различни операции с данните и файла.

Създаване на нов предмет или папка- за създаването на нов предмет с потребителски данни или на нова папка, на потребителя се показва специализиран диалог – CreateItemDialog, който ще разгледаме по-долу.

Редакция и визуализация на предмет – Когато даден предмет се визуализира, той може да бъде и редактиран, без да е необходимо да се предприемат някакви допълнителни действия, стига обаче предметът да позволява редакция. За да визуализира избран от потребителя предмет основната форма първо поисква този предмет от класа за управление на данните (DataManager) и след като го получи, го подава на диалога за визуализация и редакция (EditItemDialog), който ще разгледаме след малко.

Преименуване на предмет – Когато потребителят поиска да смени името на даден предмет, основната форма му показва диа-

лог за редакция на името на предмета (RenameItemDialog), който за удобство на потребителя изписва текущото име на предмета. След като потребителят въведе ново име за предмета, формата стартира асинхронна операция (RenameItemOperation), която да отрази промяната във файла с данните. При завършване на операцията, формата получава събитие, че предметът е бил променен и презарежда неговото име.

Изтриване на предмет – Потребителят може да изтрие както отделен предмет, така и цяла папка, което изтрива всички предмети и под-папки в нея. Основната форма иска от потребителя потвърждение и ако го получи стартира асинхронна операция (DeleteItemOperation), която да изтрие предмета или папката и всичко, съдържащо се в нея. Чрез събитията на компонента за управление на данните, за които формата е абонирана, тя опреснява дървото на предметите, премахвайки тези, които са били изтрити.

Реорганизация – Потребителят може да реорганизира съхранените предмети, като ги мести в нови папки. Възможно е и преместването на цяла папка. Това се реализира с познатия за потребителите механизъм на изрязване и вмъкване (cut и paste). След като потребителят избере къде да вмъкне предмета или папката, формата стартира асинхронна операция (MoveItemToFolderOperation), която осъществява преместването. Отново формата бива уведомена чрез събития за настъпилите промени и опреснява, за да ги отрази.

Смяна на паролата за достъп – Ако потребителят поиска да смени паролата за достъп до файла, формата показва диалога за нова парола (NewPassworDialog), в който потребителят въвежда текущата парола и избира нова и диалога за избор на ниво на защита (SecurityLevelDialog). Ако потребителят не отмени операцията, формата стартира асинхронна операция (ChangePasswor-

dOperation), която проверява дали паролата е вярна и ако е така я подменя с нововъведената.

Свиване на файла – След извършването на множество операции като изтриване и промяна на предмети, размерът на файла нараства. Потребителят може да свие файла, така, че той да заема точно толкова място, колкото му е необходимо. За целта се изпълнява асинхронна операция (CompactFileOperation), която да свие файла.

Създаване на резервно копие – Създаването на резервно копие на данните се състои от няколко стъпки – избор на файл, където да се запишат данните, избор на парола (по вече описания начин) и изпълнение на асинхронна операция, която записва всички данни в избрания файл.

Възстановяване на данните от резервно копие – За да се възстановят данните от резервно копие, потребителят избира файл, съдържащ резервното копие, предоставя паролата за достъп до този файл и, ако тя е вярна, се стартира асинхронна операция, която прочита всички данни и ги съхранява във файла на потребителя.

4.6.4.7. CreateItemDialog

CreateItemDialog реализира екрана описан в 3.2.2.6 – „Създаване на нов предмет”. Този диалог може да се използва както за създаването на нов предмет, така и за създаването на нова папка. Този диалог предоставя поле за въвеждане на име, с което потребителят да идентифицира предмета или папката, и поле за избор на тип за новия предмет, което не се показва при създаване на папка. Ако потребителят е избрал да създава папка, то всички необходими данни са налични и диалогът пристъпва към съхраняването на папката, като за целта стартира асинхронна операция (StoreItemOperation) след което се затваря. В случая на създаване на нов предмет, се конструира обект от искания тип

и се показва диалог за редакция (EditItemDialog), съдържащ подходящите контроли за редакция на този тип. Ако данните на предмета трябва да дойдат от файл, какъвто е случаят с типовете изображение и файл, то на потребителя се показва стандартен диалог за избор на файл.

4.6.4.8. EditItemDialog

EditItemDialog реализира екрана описан в 3.2.2.7 – „Редакция на данни”. Той се използва както за редакция и визуализация на данни, така и при създаването на нов предмет, както видяхме в описанието на CreateItemDialog. Всъщност тук е мястото където се осъществява съхранението на предметите. Този диалог разполага с област за визуализация на избрания предмет, която се запълва с различни контроли, в зависимост от типа на обекта (тези контроли сме описали в 4.6.5 – „Контроли за редакция на данни”) и бутони за съхранение на предмета (Save), затваряне на диалога без съхранение (Close) и в случая на предмет, чиито данни идват от файл, бутон за записване на данните обратно във файл (Export). Когато потребителят натисне бутона за съхранение, формата пита контрола за редакция, дали предметът има нужда да се съхрани и ако това е така стартира асинхронна операция (StoreItemOperation), която записва предмета във файла на потребителя. Ако потребителят избере да запише съдържанието на предмет, представляващ изображение или файл, в некриптиран вид, се стартира асинхронна операция, която да извърши това.

4.6.4.9. RenameItemDialog

RenameItemDialog реализира екрана описан в 3.2.2.8 – „Преименуване на предмет или папка”. Този диалог се показва от основната форма, когато потребителят поиска да смени името на даден предмет. Той предлага поле за въвеждане на новото име и бутони за приемане и отказ на промените.

4.6.4.10. BackupPasswordDialog

BackupPasswordDialog използваме за въвеждане на паролата за файл с резервно копие на данните. Той указва името на файла и предоставя поле за въвеждане на парола. Този диалог не извършва никаква работа и единствено предлага свойство, съдържащо въведената от потребителя парола.

4.6.4.11. ProgressDialog

ProgressDialog реализира екрана описан в 3.2.2.9 – „Прогрес (при бавна операция)". Той приема в конструктора си асинхронна операция, чието изпълнение да следи. Показва на потребителя индикатор на прогреса, който се управлява чрез обект AsyncOperationProgressTracker. Диалогът предлага и бутон за прекратяване на операцията, който обаче е активен само за операции които поддържат тази функционалност.

4.6.5. Контроли за редакция на данни

Както описахме по-горе, диалогът EditItemDialog разполага с контрол за редакция на предмета, който е различен за различните типове предмети. Тук ще опишем тези контроли по отделно, като първо разгледаме и интерфейса, който те имплементират.

4.6.5.1. IEditItemControl

IEditItemControl е интерфейс, който всички контроли за редакция на предмети трябва да имплементират. Той предлага само булев метод Save, чиято цел е да определи, дали редактираният предмет трябва да бъде съхранен.

4.6.5.2. Контроли

В приложението сме дефинирали следните контроли за редакция на данни, като за всеки тип данни, дефинирани в 4.5.6 – „ItemTypes" и 4.5.7 – „Интерфейс IFileItem", съответства един контрол за редакция:

- BankAccountControl

- CreditCardControl
- GenericPasswordControl
- WebAccountControl
- EmailControl
- NoteControl
- CellPhoneControl
- PictureControl
- FileControl

Въпреки, че имаме различни контроли за различните типове данни, те имат много общи елементи. Както вече споменахме всички те реализират булевия метод Save, чрез който преценяват дали предмета, който се редактира трябва да се съхрани. Например, ако потребителят не е променил нито едно от полетата, предметът няма да бъде съхранен, дори и да е натиснат бутонът за съхранение. За да знаят контролите дали предметът е променен, те прихващат събитията за промяна на текста на отделните си полета и ако някое от тях бъде променено, те си отбелязват това. При контролите за изображение и файл, където редакция не е възможна, методът Save връща истина само, ако предметът е бил току що създаден и още не е съхранен.

5. Тестване на приложението

5.1. Тестване на основната функционалност

За да осигурим високото качество на приложението, разработихме инфраструктура за автоматичното изпълнение на тестове на основната функционалност. Това осъществихме чрез приложение за Pocket PC, стартиращо класове извършващи тестове на основните функционални модули. Тестовата функционалност съобщава не само дали даден тест е завършил успешно или неуспешно, но и уведомява тестващото приложение за текущо извършваната операция.

За тестването на криптографската функционалност, особено за разработената от нас HMAC, сме създали автоматизирани тестове, които следят за правилното изпълнение. За HMAC сме реализирали тест, който изпълнява функцията над всички тестови вектори описани в RFC 2202 (Cheng & Glenn, 1997), и сме изпълнили тези тестове не само с използваната от нас криптографска хеш функция SHA-1, а и с MD5.

5.2. Тестване на потребителския интерфейс

Потребителския интерфейс тествахме използвайки емулатор на Windows Mobile 5.0, както и реално устройство с Pocket PC 2003. Тествахме всички форми и диалози, както и всички потребителски сценарии като добавяне на предмет от всеки отделен тип, редакция на тези предмети с промяна на най-различни техни полета, преместване на предмет и цяла папка с множество предмети и под-папки, изтриване на отделни предмети и цели папки, създаване на резервни копия и възстановяване от тях, свиване на файл и други. Тестовете бяха извършени с реалистично количество данни, но трябва да споменем, че компонентът за съхранение на данните е проектиран и реализиран така, че да

позволява бързо съхранение и извличане на данни и при много големи обеми от тях. Този компонент също беше надлежно тестван с редица автоматизирани тестове за натоварване и производителност.

5.3. Тестване за използваемост

За да проверим използваемостта на разработеното приложение, то беше използвано за съхранение на реални данни от реални хора. Приложението се показа удобно за ежедневна употреба, а множеството типове данни и простият потребителски интерфейс спомагат за приятната работа с него. Въпреки старанието ни да постигнем наистина удобен потребителски интерфейс, бяхме ограничени от използването на .NET Compact Framework, тъй като контролите, които той предлага не са толкова мощни, колкото при MFC за Pocket PC. Забелязахме, че дървовидната структура в основната форма страда в своето удобство, именно поради недостатък на контрола, който я реализира.

6. Изводи и възможности за усъвършенстване

Както видяхме в точка 2.3 – „Съществуващи приложения", съществуват и други решения, които адресират разглеждания проблем за сигурно съхранение на лични данни в мобилно устройство като Pocket PC. Повечето от тях са професионално разработени комерсиални продукти, минали през множество версии. Спрямо нашето приложение, те предлагат повече удобства на потребителите, включително и възможности за различни настройки, с които приложението да придобие по-индивидуален вид. Много от тях позволяват на потребителите си да дефинират шаблони с нови типове данни, да асоциират различни икони за всеки предмет поотделно, както и да извършват всички дейности с приложението удобно от своя персонален компютър, и просто да синхронизират данните си на устройството.

Въпреки, че разработеното от нас приложение не предлага всички тези улеснения за потребителите, то има своята стойност в сигурните алгоритми за защита на данните, както и във възможността за съхранение и експорт на файлове и изображения – възможност, която нито едно от разгледаните вече съществуващи приложения не предоставя. При разработката на приложението една от основните цели беше да направим работата с него лесна и удобна, и до голяма степен това е така. Приложението е така проектирано и реализирано, че да позволява лесна разширяемост и бъдещи промени.

Разбира се, като всяко приложение в своята първа версия, то не е перфектно. Има редица подобрения, които бихме могли да реализираме при следващи версии на приложението, като например предоставянето на алтернативни методи за проверка на идентичността на потребителя (биометрични сензори и цифро-

ви сертификати) и възможност за по-добро управление на данните, като сливане на файлове на приложението, експорт и импорт на отделни предмети. Като цяло обаче, успяхме да постигнем целите които си поставихме при разработката на приложението.

7. Заключение

Защитата на информацията се превръща във все по-важен проблем в съвременното информационно общество. Нуждата всеки да носи различни данни със себе си, без същевременно да ги излага на риск от попадане в чужди ръце, се увеличава с всеки изминал ден. Преносимите устройства като Pocket PC, смартфони и мобилни компютри стимулират пренасянето на лична информация, но проблемът за тяхната защита все още е налице.

Съществуват редица приложения, адресиращи проблема за сигурното носене на лични данни в преносими устройства. Разработеното като цел на настоящата дипломна работа решение, обаче, въпреки, че не е перфектно, не е просто едно от тях, а се диференцира чрез използването на силни криптографски алгоритми, стимулиране на потребителя да мисли за защитата на данните си (като например използването на силна парола), и с предлагането на възможност за съхранение на цели файлове и изображения.

В процеса на разработка избрахме подходящи криптографски алгоритми, реализирайки сами липсваща в .NET Compact Framework критична криптографска функционалност. Сблъскахме се и с проблема за отзивчив потребителски интерфейс, който решихме чрез използването на асинхронни операции и механизъм за двупосочна комуникация между потребителския интерфейс и нишките на изпълняваните операции.

Разработеното като цел на настоящата дипломна работа приложение решава важен практически проблем. Гъвкавата му и разширяема архитектура и добре написаният му код позволяват лесно да бъдат добавени нови възможности, които да го направят още по-атрактивно за потребителите му.

Използвана литература

Cheng, P.-C., & Glenn, R. (септември 1997 г.). *RFC 2202 - Test Cases for HMAC-MD5 and HMAC-SHA-1.* Изтеглено на 6 октомври 2006 г. от Internet Engineering Task Force: http://tools.ietf.org/html/rfc2202

Griffiths, I. (2003). Give Your .NET-based Application a Fast and Responsive UI with Multiple Threads. *MSDN Magazine , 18* (2).

Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1996). *Handbook of Applied Cryptography* (Fifth Printing изд.). CRC Press.

RSA Laboratories. (25 март 1999 г.). *PKCS #5 v2.0: Password-Based Cryptography Standard.* Изтеглено на 6 октомври 2006 г. от RSA Laboratories: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs5v2-0.pdf

RSA Laboratories. (2007). *The RSA Laboratories Secret-Key Challenge*. Изтеглено на 30 януари 2007 г. от RSA Laboratories: http://www.rsasecurity.com/rsalabs/node.asp?id=2100

Wikipedia. (2007). *SHA hash functions*. Изтеглено на 30 януари 2007 г. от Wikipedia: http://en.wikipedia.org/wiki/SHA_hash_functions

Wikipedia. (2007). *Symmetric-Key Algorithm*. Изтеглено на 22 януари 2007 г. от Wikipedia: http://en.wikipedia.org/wiki/Symmetric-key_algorithm

Wikipedia. (2007). *Triple DES*. Изтеглено на 29 януари 2007 г. от Wikipedia: http://en.wikipedia.org/wiki/Triple_DES

Йорданов, С. Й. (2007). *Компонент за съхранение на криптирани данни в Pocket PC.* София: Софийски Университет "Св. Климент Охридски".

Приложение 1 – Списък на фигурите

Приложение 2

Изходният код на разработеното приложение е приложен към настоящия текст под формата на компакт диск.