VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

BEZPEČNOSTNÍ APLIKACE NA PLATFORMĚ APPLE IOS

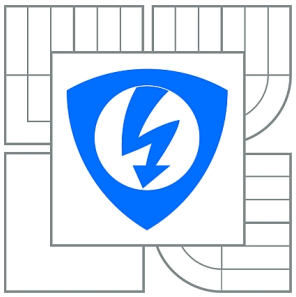
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MICHAL KOLÁŘ
AUTHOR

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

DEPARTMENT OF TELECOMMUNICATIONS

BEZPEČNOSTNÍ APLIKACE NA PLATFORMĚ APPLE IOS

SECURITY APPLICATIONS ON THE APPLE IOS PLATFORM

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

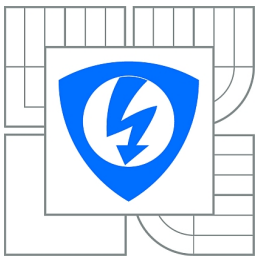
AUTOR PRÁCE — MICHAL KOLÁŘ

AUTHOR

VEDOUCÍ PRÁCE — Ing. JAN HAJNÝ, Ph.D.

SUPERVISOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Michal Kolář	***ID:***	125484
Ročník:	3	***Akademický rok:***	2012/2013

NÁZEV TÉMATU:

Bezpečnostní aplikace na platformě Apple iOS

POKYNY PRO VYPRACOVÁNÍ:

Téma je zaměřeno na využití operačního systému Apple iOS pro vývoj bezpečnostních aplikací. Student v rámci svého projektu nastuduje architekturu operačního systému iOS a zaměří se na podporu aplikací pro bezpečnou autentizaci uživatelů. Konkrétním zaměřením bude využití dvojrozměrných kódů pro přenos dat v kryptografických protokolech. Výstupem projektu bude implementace autentizační aplikace na platformě iOS. Důraz bude kladen na funkčnost aplikace a ověření její mobilní a serverové části.

DOPORUČENÁ LITERATURA:

[1] STALLINGS, William. Cryptography and Network Security: Principles and Practice (5th Edition). USA : Prentice Hall, 2010. 744 s. ISBN 0136097049.

[2] IOS Security. [online]. 2012 [cit. 2012-10-09]. Dostupné z: http://images.apple.com/ipad/business/docs/iOS_Security_May12.pdf.

Termín zadání:	11.2.2013	***Termín odevzdání:***	5.6.2013

Vedoucí práce: Ing. Jan Hajný, Ph.D.
Konzultanti bakalářské práce:

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Tato práce se zabývá zabezpečením operačního systému iOS. Popisuje mechanismy, které ručí za bezpečnost celého systému souborů i soukromých souborů uživatele. Dále popisuje mechanismy pro autentizaci uživatele a pro ověřování důvěryhodnosti third-party aplikací. V praktické části popisuje instalaci vývojového prostředí Xcode a seznamuje čtenáře s jeho pracovním prostředí. Praktická část je zaměřena na tvorbu programu, který bude implementovat krytografický protokol. Zahrnuje také práci s dalšími přídavnými knihovnami, které budou potřebné pro vytvoření aplikace. Výstupem programu bude generování QR kód, který bude přenášet všechny požadované hodnoty kryptografického protokoluu.

KLÍČOVÁ SLOVA

iOS, Xcode, SHA-1 hash, QR kód, knihovna GMP, ZbarSDK, ZXing

ABSTRACT

This document deals with the security of iOS operating system. It describes applications which ensure the safety of the entire file system and private files of the user. It further describes mechanisms for authentication of the user and for credibility verification of third- party applications. The practical part is devoted to the installation of the development environment Xcode and introduces the reader to its work environment. The practical part is focused to create an application which will implement a security protocol. Also include work with another additional libraries which will be needed to create an application. Output application will be generated QR code with all required values of cryptographic protocol.

KEYWORDS

iOS, Xcode, SHA-1 hash, QR code, library GMP, ZbarSKD, ZXing

KOLÁŘ, Michal *Bezpečenostní aplikace na platformě Apple iOS*: bakalářská práce. místo: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2012. 57 s. Vedoucí práce byl Ing. Jan Hajný, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Bezpečenostní aplikace na platformě Apple iOS“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

místo ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Ing. Janu Hajnému, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Také bych chtěl velmi poděkovat rodičům, kteří mi umožnili studovat a neustále mě podporovali.

místo ....

(podpis autora)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této bakalářské práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

místo . . . .

(podpis autora)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OBSAH

SEZNAM OBRÁZKŮ

1 ÚVOD

Systém iOS je operační systém navržený společností Apple. Je svojí architekturou podobný MAC OS, který se používá pro počítače Macbooky, iMaacy i servery. Jedná se o systém UNIXového typu, což znamená že je multitaskingový a určen pro více uživatelů. iOS je v podstatě zjednodušený MAC OS, ale je přidáno dotykové ovládání pro displeje. Operační systém iOS se využívá pro zařízení iPhone, iPod touch a iPad. Obsluhuje řízení hardwaru a obsahuje aplikace pro jeho ovládání. Některé jsou již implicitně předinstalované jako například Safari nebo Mail. Pomocí různých nástrojů přímo od Apple lze vyvíjet vlastní aplikace. Toto rozhraní se jmenuje iOS Software Development Kit (SDK) a obsahuje frameworky, které usnadňují vývoj konkrétních aplikací. Podpora je i pro vytváření webových aplikací například přes HTML nebo JavaScript. Nativní aplikace a nástroje jsou neustále k dispozici ve všech režimech, ale webové apliace vyžadují přístup k internetu, protože běží přes Safari. Všechny aplikace je možné synchronizovat s PC pomocí iTunes.

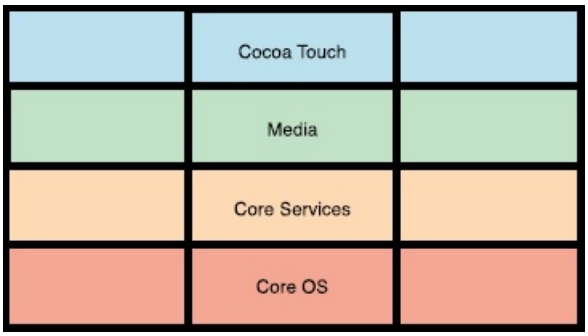
1.1 Historie iOS

První mobilní software byl představen 9. ledna 2007 na konferenci MacWorld a byl primárně určen pro iPhone. První iPhone se začal prodávat v červnu téhož roku. Na tu dobu to byl velmi pokročilý stroj, který odstartoval vývoj těchto mobilních technologií. Na rozdíl od konkurence měl skvělý design, velký dobře fungující displej a velké množství užitečných funkcí. Vývojářské prostředí bylo k systému vydáno až v lednu 2008.

Verze 1.0 si prošla brzy prvními zlepšeními. Zvýšilo se zabezpečení prohlížeče Safari, byla přidána možnost vlastního vyzvánění a vylepšená synchronizace map. Od veze 1.1 fungoval systém iOS i na zařízení iPod touch. Další aktualizace opravovaly menší systémové bugy a přinášely další nové funkce. Společně s příchodem iPhone 3G byla vydána verze 2.0, která obsahovala nové jazyky, App Store, vylepšila práci s kontakty, poštou i kalendářem. Verze 3.0 přinesla kopírování a vkládání MMS, optimalizaci Youtube klienta a automatické připojování na Wi-fi. Dále zrychlila bootování systému a od verze 3.2 přišla podpora i pro iPad. Od verze 4.0 přišel plnohodnotný multitasking, snadnější přepínaní mezi aplikacemi, vylepšený zámek a iPad lze ovládat ze zamčeného zařízení. Velká změna přišla s verzí 5.0. Například pro ukládání dat na PC už není nutnost se spojovat přes kabel, úložiště iCloud, synchronizace přes Wi-Fi, Oznamovací centrum, iMessage a novou verzí webového prohlížeče s podporou záložek na iPadu. Od verze 6.0 jsou dostupné zcela nové mapy, integrovaný Facebook, sdílení fotografií nebo VIP seznam pro e-maily.

2 ARCHITEKTURA IOS

Celá architektura iOS je složená z několika hlavních vrstev. Tyto vrstvy jsou Cocoa Touch, Media, Core Services a Core OS, jak je zobrazeno na obrázku 2.1. Nejvyšší úroveň zprostředkovává fungování mezi hardwarem a aplikacemi. K tomu jsou využita jistá definovaná rozhraní, která zároveň usnadňují psaní aplikací, protože mohou pracovat na zařízeních s rozdílnými hardwarovými vlastnostmi. Většina těchto rozhraní je dostupná ve speciálních balíčkcích, nazvaných frameworky. Ty obsahují knihovny, hlavičkové soubory, pomocné aplikace, apod. Je jich velké množství a pro jejich použití je nutné je implementovat do projektu k dalším zdrojovým souborům. Využívají se především v prostředí Xcode. Toto vývojové prostředí slouží k vytváření a ladění aplikací pro systém iOS. Vytvořené aplikace lze rovnou testovat, protože Xcode obsahuje i simulátor systému iOS. Kromě toho obsahuje i odkazy na programovací příručky a další zdroje, pro kvalitní programování. Nižší úrovně pak obsahují služby, na kterých jsou aplikace závislé.[4]



Obr. 2.1: Vrstvy systému iOS

2.1 Vrstva Cocoa Touch Layer

Tato vrstva obsahuje důležité frameworky pro vytváření aplikací pro systém iOS. Vrstva obsahuje High-Level funkce a frameworky, které ulehčují práci při navrhování uživatelských rozhraní a aplikací. Například Auto Layout zlepšuje pružnost systému využívaného k rozvržení prvků uživatelského rozhraní a Storyboards slouží přímo k návrhu uživatelských rozhraní, kde je možné zadefinovat například přechody uživatelského rozhraní. Velmi důležitá je funkce Multitasking, která při stisku tlačítka HOME pomáhá hladce přesunout aplikace do pozadí, místo jejich ukončení. Toho často využívá systém kvůli zachování dobré životnosti baterie, a sám přesouvá aplikace do pozadí. Tam jsou uložené v paměti, takže je lze rychle obnovit. Existují i aplikace, které pracují i po přesunutí do pozadí.[4]

Vrstva také obsahuje fukce pro notifikace. Jsou to Apple Push Notification Service, které upozorňují uživatele na nové informace o aplikacích, a fukce Local Notification, které doplňují předešlou fuknci o generování oznámení místně a nemusejí se spoléhat na externí server. Tyto místní notifikace pak mohou používat i aplikace přesunuté na pozadí.[4]

Cocoa Touch Layer také zprostředkovává rozpoznávání gest, zajišťuje komunikaci s blízkými zařízeními pomocí Bluetooth, umožňuje připojování externích zobrazovacích zařízení a podává informace o jejich obrazovce.[4]

Kromě těchto funkcí obsahuje tato vrstva i frameworky pro vytváření nových a editaci starých kontaktů, rozhraní pro editaci a práci s kalendářem. Umí pracovat s mapovými podklady a umožňuje je i integrovat do aplikací. Řídí všechny klíčové grafické události a všechny aplikace v systému iOS, a poskytuje podporu pro vytváření a spravování front e-mailových zpráv.[4]

Nejdůležitější frameworky Cocoa Touch Layeru:

- Address Book UI Framework
- Event Kit UI Framework
- Game Kit Framework
- Map Kit Framewok
- Messsage UI Framework

2.2 Vrstva Media Layer

Tato vrstva se stará především o vizuální stránku - grafiku, audio a video technologie. Kvalitní grafické zpracování je důležité pro všechny aplikace a nejjednodušší způsob je využít předredenderované obrazy k vytvoření aplikace spolu se standardními pohledy. Pokud chce uživatel použít složitější grafiku, existují nástroje pro tento účel. Technologie zajišťující audio stránku systému iOS, umožňují využívat zvuk nejvyšší kvality a ve stejné kvalitě ho i nahrávat. Vrstva zahrnuje také používání vibrací a nabízí několik možností přehrávání a zaznamenávání audio obsahu. Pro přehrávání videosouborů nebo pro streamování nabízí tato vrstva několik aplikací a způsobů, umožňuje také video zaznamenávat a následně i streamovat. O streamování se stará technologie AirPlay. Pro audio i video funkce jsou navrženy frameworky, které celou práci usnadňují a mají různé úrovně rozhraní. [4]

Příklady frameworků:

- Assets Library Framework
- AV Foundation Framework
- Core Audio Core Graphics Framework
- Media Player Framework
- OpenGL ES Framework
- GLKit Framework
- Quartz Core Framework

2.3 Vrstva Core Services Layer

Zajišťuje základní systémové služby, které využívají všechny ostatní aplikace. Pro centrální ukládání a sdílení dat slouží iCloud Storage. To zajišťuje uživatelovi přístup ke svým uloženým dokumentům z jakéhokoliv iOS zařízení. iCloud umožňuje snadné prohlížení, tvorbu a editaci dokumentů z libovolného zařízení bez nutnosti neustálé synchronizace. Takto uložené dokumenty jsou navíc v bezpečí i při ztrátě zařízení. Většina nově vytvořených aplikací je navržena k ukládání dat pomocí iCloud. Vrstva podporuje sdílení uživatelových souborů přes iTunes. Adresář určený pro sdílené soubory je neustále k dispozici. Neumoňuje ovšem sdílení souborů mezi ostatními aplikacemi na stejném přístroji. Aplikace, které pracují s citlivými daty, mohou využívat předdefinovaných šifrovacích metod. Pokud aplikace vyhodnotí některý soubor jako chráněný, iOS uloží soubor v zašifrovaném formátu, kdy po zamčení přístroje je jeho obsah nepřístupný.[4]

Vrstva také zajišťuje podporu SQL a XML. SQL databáze je možné vkládat do aplikací bez nutnosti přístupu k vzdálenému serveru a přímo z vaší aplikace umožňuje vytvořit lokální databázi. XML poskytuje podporu pro načítání těchto dokumentů. Veškerá podpora je řízena speciální knihovnou, pro zápis nebo čtení libovolných dat ze souboru.[4]

2.3.1 Core Services Support

Poskytuje jednotné přihlášení pro podporované uživatelské účty, což znamená že některé aplikace již nevyžadují zadání přihlašovacích údajů a zjednodušuje celou správu vašeho účtu. Zajišťuje přístup ke kontaktům uložených v uživatelově zařízení a lze pomocí speciálnícho rozhraní databázi editovat. Má také rozhraní pro přístup k účtům sociálních sítí jako je Twitter nebo Facebook. Výhoda je, že pro všechny účty využívá jednotné přihlášení. V případě účtů, které se starají o finanční transakce, zpracovává jejich žádosti a zajišťuje bezpečnost celého přenosu. Platba se provádí přes iTunesStore účet.[4]

Vrstva se stará o práci se síťovými protokoly díky vysoce výkonnému rozhraní CFNetwork. Ulehčuje komunikaci a práci s FTP a http servery. Poskytuje také rozhraní k určování síťových konfigurací, např. Wifi. Vrstva se stará o datové modely, především v aplikacích, kde je jejich datový model už velice strukturovaný. Umožňuje využít Xcode pro vybudování schématu datového modelu.[4]

Poskytuje informace o umístění s použitím Core Location Framework, který využívá zabudovanou GPS, mobilní nebo wifi rádia a určí tak přesnou lokaci zařízení. Zjišťování zeměpisné délky a šířky lze pomocí frameworku využívat i ve svých aplikacích. Dále zpracovává pohyb a otáčení přístroje a tyto data umožňuje zpracovávat a uchovávat.[4]

Nejdůležitější frameworky

- Accounts Framework
- CFNetwork Framework
- Core Data Framework
- Social Framewok
- Store Kit Framework
- System Configuration Framework

2.4 Vrstva Core OS Layer

Tato vrstva obsahuje většinu low-level funkcí a frameworky, na kterých jsou založeny téměř všechny ostatní technologie. Zahrnuje rozhraní pro provádění DSP výpočtů a počítání celé lineární algebry, které je optimalizováno pro všechny dostupné konfigurace iOS zařízení. Poskytuje podporu pro kominukaci s přídavnými doplňky, které jsou připojeny k zařízení pomocí Bluetooth nebo přímo přes kabel. Jedná se zejména o sluchátka a nabíječku, která slouží i k přenosu dat na PC.[4]

Tento oddíl zahrnuje veškeré ovladače a low-level UNIX rozhraní pro OS. Jádro se stará o všechny aspekty operačního systému, takže spravuje celý souborový systém, stará se o síť, alokuje virtuální paměť systému a zajišťuje meziprocesové komunikace. Pro zvýšení bezpečnosti systému je přístup do samotného jádra a ovladačů velmi omezen. Přístup je jen pomocí speciální sady systémových aplikací. Kromě vestavěných bezpečnostních prvků obsahuje další bezpečnostní rámce, které se využívají pro zabezpečení dat aplikací. Tato rozhraní slouží pro správu certifikátů, veřejných či soukromých klíčů a důvěryhodnostní politiky. Probíhá zde i generování kryptografických pseudonáhodných čísel.[4]

Nejdůležitější frameworky

- Accelerate Framework
- Security Framework
- External Accessory Framework

3 ZABEZPEČENÍ SYSTÉMU IOS

iPhone, iPad i iPod touch jsou navrženy s několika typy vrstev zabezpečení. High-level prostředky umožňují bezpečný přístup k osobním a citlivým informacím, a tak zabraňují neoprávněnému přístupu a zneužití a pomáhají likvidovat útoky, které se chtějí k citlivým datovým souborům dostat. Low-level prostředky obsahují funkce pro ochranu proti spamu v podobě malware, a dalším virům.[3]

Zařízení se systémem iOS poskytují precizní bezpečnostní technologie a funkce pro maximální ochranu dat. Zároveň ale minimálně omezují uživatele při běžné práci. Přístroje jsou navrženy tak, že mnoho nastavení zabezpečení je už nastaveno implicitně a některé funkce, jako je šifrování dat, není možné ani konfigurovat, takže je uživatel nemůže omylem zakázat. Model zabezpečení systému iOS umožňuje používání third-party aplikací a jejich synchronizaci při nesnížené ochraně ostatních informací. Tento bezpečnostní model je vylepšován s každou verzí systému. Prostředky pro zabezpečení jsou rozděleny do několika logických sekcí. Každá zajišťuje zabezpečení pro jiný typ dat nebo provozu, či ověřování uživatele.[3]

3.1 Systémová architektura

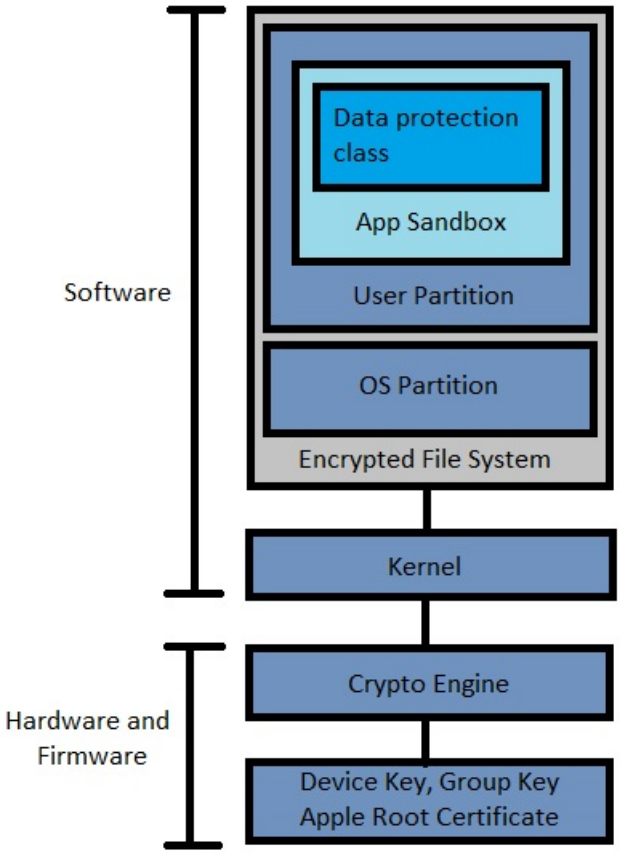
Zařízení se systémem iOS jsou navržena tak, že dochází k těsné integraci hardwaru a softwaru, jak je zobrazeno na obrázku 3.1. To umožňuje využívání ověřovacích činností ve všech vrstvách daného zařízení. Jako první se spouští boot-up proces, který ověřuje základní položky, jako je UID přístroje, a postupně dochází ke spouštění dalších aplikací. Každý krok procesu je analyzován a prověřen, jestli je důvěryhodný. Pokud je už systém v provozu, o bezpečnost a důvěryhodnost se stará XNU a jádro systému iOS. To vyhodnocuje bezpečnostní složky za běhu systému a je implicitně důvěryhodné pro všechny vyšší úrovně, funkce i aplikace.[3]

Systémová architektura obsahuje několik funkcí pro ochranu dat při bootování zařízení, stará se o aktualizace softwaru, správu aplikací a jejich přístup k souborům. Jednotlivé funkce rozeberu podrobněji.

3.1.1 Secure Boot Chain

Jakmile dojde k zapnutí zařízení, automaticky se spustí boot-up proces. Ten obsahuje komponenty, které jsou kryptograficky podepsané Applem, a probíhají řetězcem důvěry. Tento řetězec obsahuje základní zavaděče, jádro i s jeho rozšířením, a další firmware. Jako první z položek je kontrolován Boot ROM kód. Tento kód je zapsaný v read-only paměti a nedá se změnit, neboť je vytvořen během výroby čipu a je nastavený jako implicitně důvěryhodný. Obsahuje Apple Root CA veřejný klíč,

který se využívá k ověřování LLB, což je první krok v ověřovacím řetězci. Postupně probíhá jeden krok za druhým a každý je ověřován, jestli je podepsán Applem. Po dokončení se spustí další fáze, která ověří a následně spustí jádro systému iOS. Tato fáze se jmenuje iBoot a zajišťuje, že během bootovacího procesu není manipulováno s nejnižšími úrovněmi softwaru, a také, že systém iOS lze spustit jen na důvěryhodných zařízeních. Jakmile jeden krok ze zaváděcího procesu nelze provést nebo ověřit, dojde k zastavení zapínání přístroje a bude zobrazena výzva pro připojení k iTunes. Pokud není Boot ROM schopen ověřit nebo načíst LLB, vstoupí zařízení do DFU módu a přes iTunes dojde k obnovení továrního nastavení.[3]



Obr. 3.1: Bezpečnostní architektura systému iOS

3.1.2 System Software Aktualizace

Protože se neustále objevují nové bezpečností problémy, vydává Apple pravidelně aktualizace softwaru, které poskytují záplaty pro tyto problémy. Aktualizace jsou vydávány vždy pro všechna podporovaná zařízení současně a oznámení o nich dostávají uživatelé přes iTunes. Přes iTunes lze aktualizace přímo stáhnout a nainstalovat. Během aktualizace se iTunes připojí k Apple serveru a pošle mu kryptografický seznam

pro všechny části instalace aktualizace. Server tento seznam zkontroluje a porovná ho s verzemi aktualizací, které jsou již k dispozici. Vyhovující aktualizace přidá do konečné sady, kterou pošle zpátky na zařízení a přiřadí ji povolení. Toto povolení umožní nainstalovat aktualizaci a je tím zajištěno, že aktualizace je schválená a ověřená Applem. Výše popsaný boot-up proces zajišťuje, že jen Applem podepsaný software může být na zařízení nainstalován. Postupně ověří, jestli podpis opravdu pochází od Apple, a zda obsahuje ECID povolení. Navíc také zabraňuje instalování starých verzí, které nemají nejnovější aktualizace a mohou obsahovat bezpečností chyby, a znemožňuje i kopírovaní starších verzí systému na zařízení. [3]

3.1.3 App Code Signing

Pro zajištění bezpečnosti musí všechny aplikace obsahovat spustitelný kód, ověřený Applem, a third-party aplikace používají k ověření důvěryhodnosti certifikát vydaný Applem. Tím je zajištěno že všechny pocházejí ze schváleného zdroje. Toto ověřování proběhne ihned po úspěšném spuštění jádra a systému iOS, ten potom ověřené aplikace a procesy spustí. iOS neumožňuje instalovat a spouštět potenciálně nebezpečné aplikace stažené z webových stránek nebo spouštět kód, který není důvěryhodný.[3]

Certifikát k ověřování pravosti aplikací pro systém iOS dostávají vývojáři po registraci u Apple a připojují se tak k vývojářskému programu. Jakmile je každý jednotlivec prověřen, dostane certifikát, který umožňuje vytvořené aplikace předkládat v App Store. Takové aplikace jsou i přesto ověřovány a testovány Applem pro zajištění nejvyšší možné bezpečnosti.[3]

Firmy mají možnost vytvářet speciální aplikace, které budou využity jen v mezích jejich podniku. Tyto aplikace se nazývají in-house a jsou distribuovatelné zaměstnancům. Jednotlivé firmy jsou ověřovány a schvalovány Applem. Po ověření jejich způsobilosti každý podnik získá svoji vlastní in-house aplikaci pro provoz zařízení. Každý uživatel poté dostane svůj zajišťovací profil, díky němuž může in-house aplikace spustit.[3]

3.1.4 Runtime Process Security

I aplikace, která je ověřena, že pochází z důvěryhodného zdroje, prochází dalším zabezpečením, které vynucuje systém iOS. Systém se vždy snaží zajistit, aby se aplikace vzájemně nemohly ohrozit, nebo aby neohrozily celý systém. Proto všechny third-party aplikace mají omezený přístup k uloženým souborům a mají velmi omezené možnosti v provádění změn na zařízení. Aplikace tak nemohou shromažďovat data jiných aplikací nebo je nějakým způsobem měnit. Pro každou aplikaci je zvlášť

vytvořen při instalaci specifický adresář, kam aplikace ukládá veškeré svoje soubory. Pokud vyžaduje přístup k informacím, které jsou mimo její adresář, musí využít služby poskytované systémem právě pro tyto příležitosti.[3]

Systémové adresáře a zdroje jsou také chráněny, přičemž většina systému běží jako neprivilegovaná. Celý oddíl systému iOS, kromě několika nástrojů, je připojen jen pouze pro čtení a znemožňuje tím jakékoliv změny v systémových souborech. Pokud chtějí third-party aplikace přistupovat k citlivým informacím, například o uživateli, musí využít deklarované nároky. Nárok je tvořen párem klíče a hodnoty, které jsou podepsané v app a umožňují tak ověření faktorů jako je ID uživatele za běhu zařízení. Tyto nároky jsou využívány především systémovými aplikacemi, které provádějí privilegované operace. Tyto privilegované operace by jinak vyžadovaly spustit procesy root, což by celý systém zpomalovalo. Tyto, i další činnosti, mohou aplikace provádět na pozadí prostřednictvím systému API. Díky tomu může aplikace pracovat bez omezení výkonu a neomezuje ani výrazně životnost baterie. Aplikace, které jsou implicitně nainstalované na zařízení, i všechny aplikace vytvořené ve vývojovém prostředí Xcode pro systém iOS, mají automaticky zapnutou podporu ASLR, která zabraňuje zahlcení prostoru paměti chybami. Jako další ochrana se používá funkce ARM's Execute Never, která označuje jednotlivé paměťové stránky jako non-executable. Tím je zabráněno jejich neúmyslnému odstranění.[3]

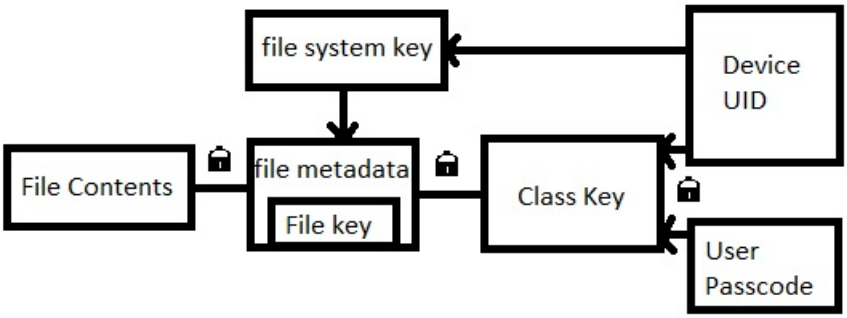
3.2 Kryptografie a ochrana dat

V případě, že by selhaly nebo byly narušeny počáteční bezpečnostní prvky struktury jako je Secure boot chain nebo Runtime proces, obsahuje systém iOS další bezpečnostní prvky k ochraně uživatelských dat. Jedná se především o šifrování souborů a zabezpečování pomocí různých typů klíčů. Tyto úkony umožňují integrované vrstvy v hardwarových a softwarových technologiích.[3]

Celá kryptografie a další ochrana dat je rozdělena na hardwarovou a softwarovou úroveň. Kryptografické operace jsou většinou velice složité a náročné na výpočet. To vede ke snižování výkonu zařízení i životnosti baterie. Pro mobilní zařízení jsou rychlost a energetická účinnost velmi důležité parametry. Kryptografické operace je tedy nutné používat a provádět správně a efektivně. Každé zařízení obsahuje svoje jedinečné ID (UID), které je reprezentováno 256-bitovým AES klíčem, který je vygenerován přímo při výrobě čipu. UID umožňuje data kryptograficky vázat přímo na konkrétní zařízení. Pokud by tedy byl paměťový modul přemístěný do jiného zařízení, soubory zůstanou nepřístupné, neboť by se při dešifraci neshodovalo UID přístroje, který soubor takto zašifroval.[3]

Stejný postup platí i pro GID, což je identifikační číslo celé skupiny přístrojů.

Jsou vypáleny do silikonu, aby se zabránilo jejich možnému zneužití nebo obejití. Tyto ID není možné přímo zjistit pomocí nějakého softwaru nebo firmwaru, zobrazují se pouze až výsledky šifrování nebo dešifrování, které provádějí. Kromě UID a GID klíčů, jsou ostatní klíče pro šifrování vytvářeny pomocí generátoru náhodných čísel (RNG). Jakmile vytvořený klíč už není používaný, je důležité, aby byl správně odstraněn, aby nemohlo dojít k jeho zneužití. Systém iOS proto obsahuje funkci speciálně zaměřenou ne bezpečné mazaní tohoto typu dat. Kromě hardwarového šifrování, využívá systém iOS k ochraně dat další technologie, které uvedu níže.[3]



Obr. 3.2: Schéma architektury kódování

3.2.1 File Data Protection

Tato technologie byla navržena speciálně pro mobilní zařízení. Ty se mohou kdekoliv připojovat k internetu, provádět telefonní hovory, pracovat s textovými zprávami nebo e-maily. Bylo potřeba vytvořit technologii, která by zajišťovala, aby zařízení mohla reagovat na všechny tyto události, aniž by musela dešifrovat citlivá data nebo stahovat nové informace, i přes zamčení přístroje. Data jsou podle citlivosti a možnosti přístupu rozdělena do jednotlivých tříd a každá třída má svoje zabezpečení. Přístup je určen podle generovaných klíčů a z nich je poté sestavena celá hierarchie.[3]

Data Protection vytváří nový 256-bitový klíč pro každý nově vytvořený oddíl dat a poté jej předává AES, který pomoci tohoto klíče soubor zašifruje. Takto zašifrovaný klíč je pak zabalen s několika dalšími klíči, podle toho za jakých okolností může být soubor otevřen a uložen do souborových metadat. Pokud je takový soubor otevřen, dojde k rozbalení jeho metadat a ta jsou následně dešifrována klíčem systému souborů, který najde příslušný klíč k dané třídě souboru. Metadata všech souborů v systému iOS jsou zašifrována náhodným klíčem, který je vygenerován při první

instalaci systému na zařízení. Tento klíč se nepoužívá pro zabezpečení důvěryhodných dat, protože je uložen přímo na zařízení a mohlo by tak dojít k jeho zneužití. Je ovšem navržen tak, aby jej mohl uživatel snadno a rychle vymazat a pokud se tak stane, všechny soubory se stanou kryptograficky nepřístupné. Obsah každého souboru je šifrován klíčem, který je vnořen do třídy klíčů, které jsou chráněny pomocí UID a pro některé třídy i uživatelským heslem. Takto chráněný klíč je uložen v metadatech souboru, ty jsou pak šifrovány dalším klíčem systému souborů. Taková hierarchie, jak je zobrazena na obrázku 3.2, poskytuje požadovanou flexibilitu i výkon. [3]

3.2.2 Passcodes

Při zapnutí zařízení je uživatel vyzván k zadání minimálně čtyřčíselného kódu. Jakmile je kód korektně vložen, povolí se automaticky i ochrana dat. Systém iOS podporuje širokou škálu alfanumerických hesel, neomezených maximální délkou. Kromě odblokování zařízení, poskytuje přístupový kód i entropii pro šifrovací klíče, které nejsou uloženy na zařízení. Kromě bezpečnostních hesel obsahuje systém iOS i technologii, která zabraňuje prolomení kódu k zařízení pomocí tzv. hrubé síly. Každý pokus o přístup k zařízení, který skončí neúspěchem, prodlouží další pokus. Počet iterací je nastaven tak, že jeden pokus trvá přibližně 80 milisekund, a poté se stupňuje. Díky této technologii by prolomení hesla o šesti alfanumerických znacích trvalo více než pět let, navíc uživatel si může zvolit maximální možný počet neúspěšných pokusů. Pokud je limit překročen, přístroj bude automaticky vymazán. [3]

3.2.3 Classes

Každému nově vytvořenému souboru v systému iOS je automaticky přidělena příslušná třída. Tyto třídy obsahují různé politiky k určení přístupu k souboru.[3]

Complete protection - klíč třídy je chráněn klíčem, který je odvozen od uživatelského přístupového kódu a UID zařízení. Po zamčení přístroje jsou všechna data v této třídě nepřístupná, dokud uživatel nezadá heslo. Tato ochrana je implicitní, například pro zprávy, přílohu nebo obrázky.[3]
Protected Unless Open - některé soubory potřebují mít možnost zapisovat, i když je přístroj zamknutý. Například stahování nových e-mailů. Aby to bylo možné, Data Protection vytvoří pár veřejný/soukromý klíč. Poté je sdílený tajný klíč je vypočítán pomocí soukromého klíče a pomocí klíče třídy, jejíž soukromý klíč je chráněn uživatelským kódem a UID. Tento sdílený klíč chrání vytvořený pár. Tento sdílený klíč a

pár veřejného a soukromého klíče je po ukončení práce se souborem vymazán.[3]
Protected Unitl First User Authentication - tato třída se chová stejným způsobem jako Complete protection, ale nemaže klíč třídy z paměti po uzamčení přístroje.[3]
No Protection - tato třída je chráněna pouze UID. Je to výchozí třída pro všechny soubory. Všechny klíče potřebné k dešifraci souborů jsou uloženy na zařízení, takže mohou být vzdáleně smazány.[3]

3.2.4 Keychain Data Protection a Keybags

Keychain je implementován jako SQLite databáze uložená v sobourovém systému, která poskytuje bezpečný způsob pro ukládání klíčů nebo přihlašovacích údajů. Předměty typu Keychain mohou být sdíleny mezi aplikacemi od stejného vývojáře. Aby to bylo možné, thrid-party aplikace musí využít přístupovou skupinu s prefixem, který je vynucený prostřednictvím podepisování kódů a profilů. Tento prefix je jim přidělen programem Developer iOS. Data typu Keychain jsou chráněna pomocí struktury třídy, která je podobná té souborové. Tyto třídy se chovají stejně jako ty pro ochranu dat, ale k šifrování a zabezpečení používají jiné klíče a jsou součástí rozhraní API. Navíc je každá Keychain třída chráněna pomocí UID, takže nemůže dojít k zneužití zálohy na jiném zařízení.[3]

Všechny Keychain třídy a jejich klíče, ale i všechny ostatní klíče souborů, jsou spravovány pomocí Keybags. Ty jsou rozděleny do několika hlavních typů:

System KeyBag obsahuje zabalené třídy klíčů, které se používají v běžném provozu. Jejich součástí je například přístupový kód.
Backup KeyBag je vytvořen, po zašifrování zálohy pomocí iTunes, která se pak uloží na počítač. Tento nový keybag je vytvořen z nové sady klíčů, pomocí kterých mohou být data opět dešifrována. Keybag je chráněný heslem nastaveným v iTunes.
Escrow KeyBag se používá pro synchronizaci iTunes a Mobile Device Management(MDM). To umožňuje zálohovat a synchronizovat bez nutnosti zadávání hesla.
iCloud Backup KeyBag je podobný Backup KeyBag, ale všechny třídy v tomto Keybag jsou asymetrické, takže iCloud zálohování lze provádět na pozadí.

3.3 Zabezpečení sítě

Protože uživatelé mobilních zařízení potřebují přistupovat k firemní síti odkudkoliv, je nutné, aby tento přístup a ochrana dat při přenosu, byl co nejbezpečnější. Kromě

opatření k ochraně dat na zařízeních se systémem iOS, které poskytuje Apple, existuje celá řada dalších zabezpečení a opatření, které mohou organizace podniknout k zajištění bezpečné výměně informací. Systém iOS podporuje standardní síťové protokoly a nejnovější standardy pro zabezpečení Wi-Fi a mobilních síťových připojení. To umožňuje bezpečné povolování a šifrování komunikace. Systém iOS, narozdíl od jiných platforem, nepotřebuje k ochraně portů firewall, a to ani při otevřené komunikaci. Je toho dosaženo omezením poslechu na portech, takže může odstranit zbytečné síťové nástroje jako je Telnet nebo webový server. Komunikace pomocí iMessage, FaceTime a push Aplle Notification Server je plně šifrována.[3]

Podpory Secure Socket Layer (SSL) a Transport Layer Security (TSL) využívají Safari, Kalendář, Mail a další internetové aplikace. Tyto služby zajišťují šifrovaný komunikační kanál a další síťové služby pro bezpečnou komunikaci.[3]

Zařízení se systémem iOS umí pracovat s VPN servery, které podporují určité protokoly, jako například Juniper Networks, Cisco, Aruba Networks, SonicWALL nebo PPTP, a ověřovací metody. Tyto zabezpečené virtuální sítě vyžadují obvykle jen minimální konfiguraci pro práci se systémem.[3]

Jako další standardní protokoly podporuje systém iOS Wifi a to včetně WPA2 Enterprise, který zajišťuje ověřený přístup k podnikovým bezdrátovým sítím. WPA2 Enterprise používá 128-bitové AES šifrování, které zajišťuje ochranu dat během přenosu. Podpora standardu 802.1X je samozřejmostí a iOS zařízení se tak mohou připojovat k nejrůznějším veřejným i domácím sítím.

[3] Samozřejmostí je i podpora staršího Bluetooth. Jeho technologie pracuje co nejefektivněji, ale nezpřístupňuje osobní informace.

4 VÝVOJ APLIKACÍ PRO SYSTÉM IOS

Pro vývoj aplikací pro iOS je nejprve nutné se zaregistrovat u Apple jako vývojář. Tato registrace je zdarma. Získáte tak přístup ke všemu, co je třeba na vývoj aplikací pro systém iOS i OS X. Přístup je i k SDK a vývojovému prostředí Xcode. Vše se stahuje od Apple zdarma. Xcode pracuje jen v prostředí Macového systému, takže je nutné mít vyhovující hardware. Pokud chcete svou napsanou aplikaci dostat do svého zařízení, musí se tak učinit nejdříve přes App Store, a to je zpoplatněno. Jednotlivec nyní platí 99\$ za rok.

Vývoj aplikace pro iOS je nutné posuzovat z několika směrů. V prvním případě jde o výhody a nevýhody jazyka Objective C, v druhém případě se jedná o Cocoa framework, a pak o samotné vývojové prostředí Xcode. To je velmi individuální k posouzení, každému vyhovuje něco jiného.

O jazyce Objective C je hodně literatury a velké množství zdrojů na internetu. Byl vyvinut na počátku osmdesátých let a vycházel ze staršího Smalltalk. Ten byl ale velice náročný a nebyl kompatibilní s jazykem C. Na rozdíl od C++ se v té době Objective C nestalo velmi populární. To se začalo měnit, až když si ho vybrala firma NeXT jako vývojový jazyk pro svou pracovní stanici. Je to jazyk vymykající se běžnému standardu. Jeho syntaxe je založená na posílání zpráv objektům, což znamená užívání hranatých závorek. Výsledkem je pak kód, který pro neznalého může být velmi těžké přečíst. Jedna z výhod jazyka Objective C je, že jakémukoliv objektu lze poslat jakoukoliv zprávu (zavolat nějakou metodu) bez ohledu na to, jestli ji tento objekt podporuje. To v jiných verzích jazyka C není možné. Další z výhod jsou protokoly a kategorie. Další jistá výhoda je beztypovost, která velmi usnadňuje programování. Jako největší nedostatek je pro začátečníky poměrně složitá syntaxe. Poté Objective C obsahuje velké množství jistých pravidel, které je třeba si zapamatovat a oproti C++ neobsahuje některé záležitosti, které byly dříve považovány za samozřejmost. To je do značné míry způsobené tím, že jde o jazyk vyvíjený jednou firmou a používaný na minoritní platformě. Stejně jako ostatní jazyky používá Objective C proměnné. To jsou jména, která odkazují na určité části paměti v počítači a během programu lze jejich obsah kdykoliv změnit. Středník, jako v C nebo C++, slouží jako ukončení příkazu. Například přiřazení nějakého obsahu do proměnné. Čísla rozděluje na dva základní typy. Integer, který je pro celá čísla, a Float pro desetinná čísla.

Skoro každý začátečník, který zkusí v Cocoa programovat, si bude pochvalovat elegantní rozhraní a počet nástrojů k dispozici. Například v levém rohu jsou předvedeny možnosti tlačítek a Cocoa vám umožňuje vybrat z široké škály jejich graficky odlišných podob. Pod tlačítky jsou editační boxy a následují další komponenty na zobrazení cesty v adresářové struktuře a volby barev. Další nabídky obsahují check

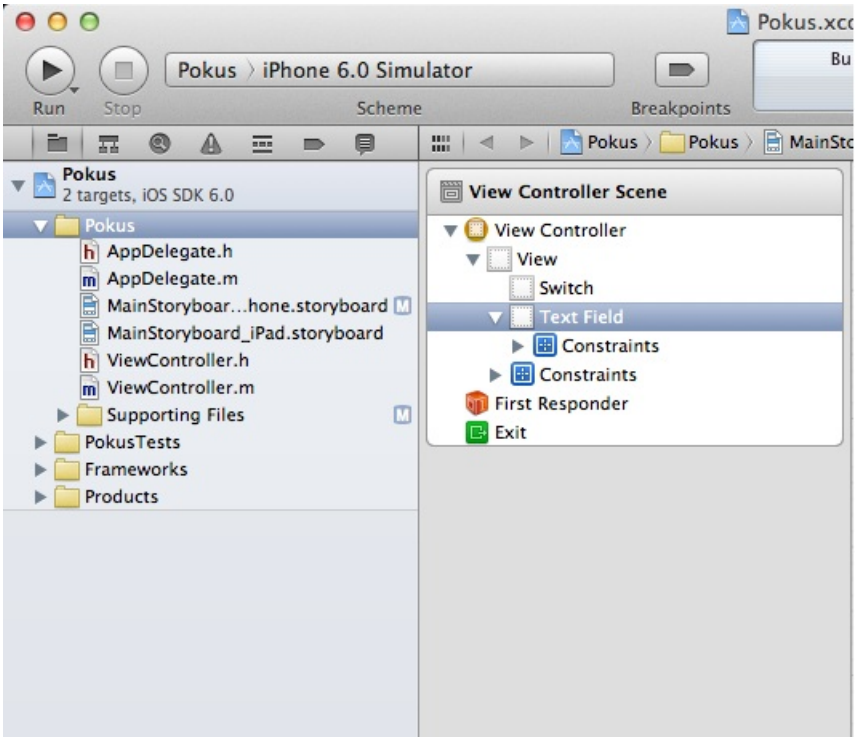
a radio butony, slidery či indikátory. Cocoa nabízí k dispozici téměř celý textový editor, který zvládá formátování, kontrolu pravopisu a další standardní věci pro editory textu. V Cocoa bohužel neexistuje komponenta typu ListBox a CheckedBox, což je velká škoda a vývojář to musí nahrazovat Comboboxy. K dispozici je velké množství komponent jako je Toolbar, komponenty pro rozdělování plochy okna nebo pro separátní pohledy. Cocoa je velmi svázaná se svým frameworkem CoreData, který ale není příliš vhodný pro zpracování velkého množství dat. Spolupracovat s profesionálními SQL servery je tak velmi obtížné.

Pro programování v Xcode je napsáno velké množství literatury, ale pro laiky to i tak nebude lehké. První problém může být, že programování v Xcode je ze značné části vizuální. To znamená, že napojujete "bindings"nebo "outlets", propojujete akce jednotlivých komponentů a lze takto napsat celou aplikaci bez napsání jediného řádku kódu. Xcode 4 je zpětně kompatibilní s Xcode 3, takže starší projekty lze bez problémů otevřít. Prostředí Xcode obsahuje navigátor (obrázek 4.1), ve kterém jsou zobrazeny jednotlivé položky projektu. Navigátor nabízí služby, jako je libovolné přesouvání souborů mezi skupinami, a možnost nové skupiny zakládat. IDE potom při sestavování programu skupiny ignoruje, slouží jen jako prostředek pro vývojářovu přehlednost. Dále pak do kontextové nabídky navigátoru lze vkládat nové objekty, dvojitým klepnutím na název můžeme objekt nebo skupinu přejmenovat. Pomocí Delete lze objekty odstraňovat, a to buď jen jeho záznam nebo celý objekt. Poklepáním na vybraný objekt se nám zobrazí jeho možnosti v editoru. V levé části příkazové lišty jsou umístěna tlačítka play a stop. Tlačítko play program sestaví a spustí a tlačítko stop běžící program zastaví. Lze si samozřejmě nastavit Breakpointy. Pokud si zvolíme schéma a platformu, lze vybrat jednu z pěti akcí, kterou má Xcode provést.

Run - sestaví cíl a spustí zvolený testovací program. To je asi nejběžnější, program se automaticky spustí pod debuggerem a lze jej vyzkoušet.
Test - sestaví speciální testovací třídy a spustí je.
Profile - sestaví cíl a spustí je pod profilovací aplikací Instruments, jež dokáže analyzovat jeho efektivitu, spotřebu paměti apod.
Analyze - spustí speciální analyzér, který se pokusí vyhledat a označit sémantické chyby. Ovšem jen výjimečně nalezne chybu, která by nebyla hned vidět, naopak mívá tendenci za podezřelé označit spoustu bezchybných konstrukcí.
Archive - Služba sestaví instalační balík obsahující aplikaci podepsanou adekvátním certifikátem pro přímou distribuci na testovací zařízení nebo pro odeslání na AppStore.

View Controller Scene panel zobzrazuje jednotlivé prvky přesunuté na pracovní

prostředí a jejich možnosti. Po kliknutí pravým tlačítkem na příslušný objekt se zobrazí rolovací menu, kde je možné provést různá nastavení daného objektu.



Obr. 4.1: Navigátor a View Controller

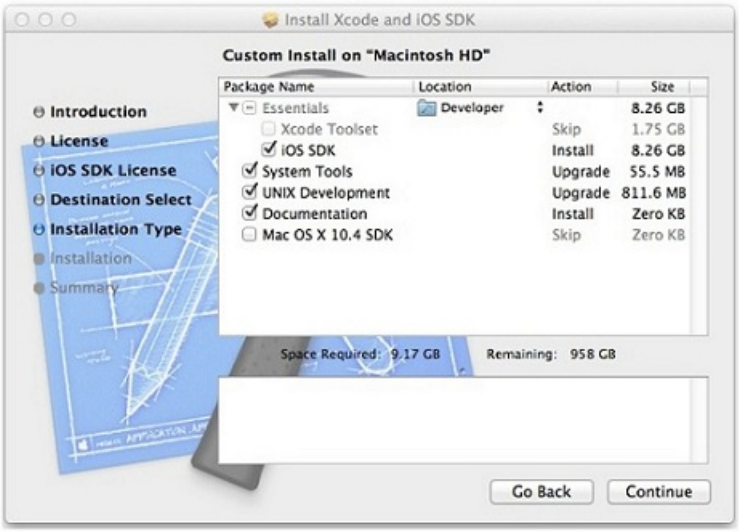
Pravá strana Xcode obsahuje panely (obrázek 4.4) pro nastavení dalších vlastností objektů a objekty samotné k přetažení na pracovní plochu. Pracovní plocha představuje display příslušného zařízení. Pokud na ni přetáhneme příslušné prvky, vidíme je přesně tak, jak budou vypadat po sestavení aplikace a vyzkoušení na daném zařízení.

4.1 Instalace Xcode

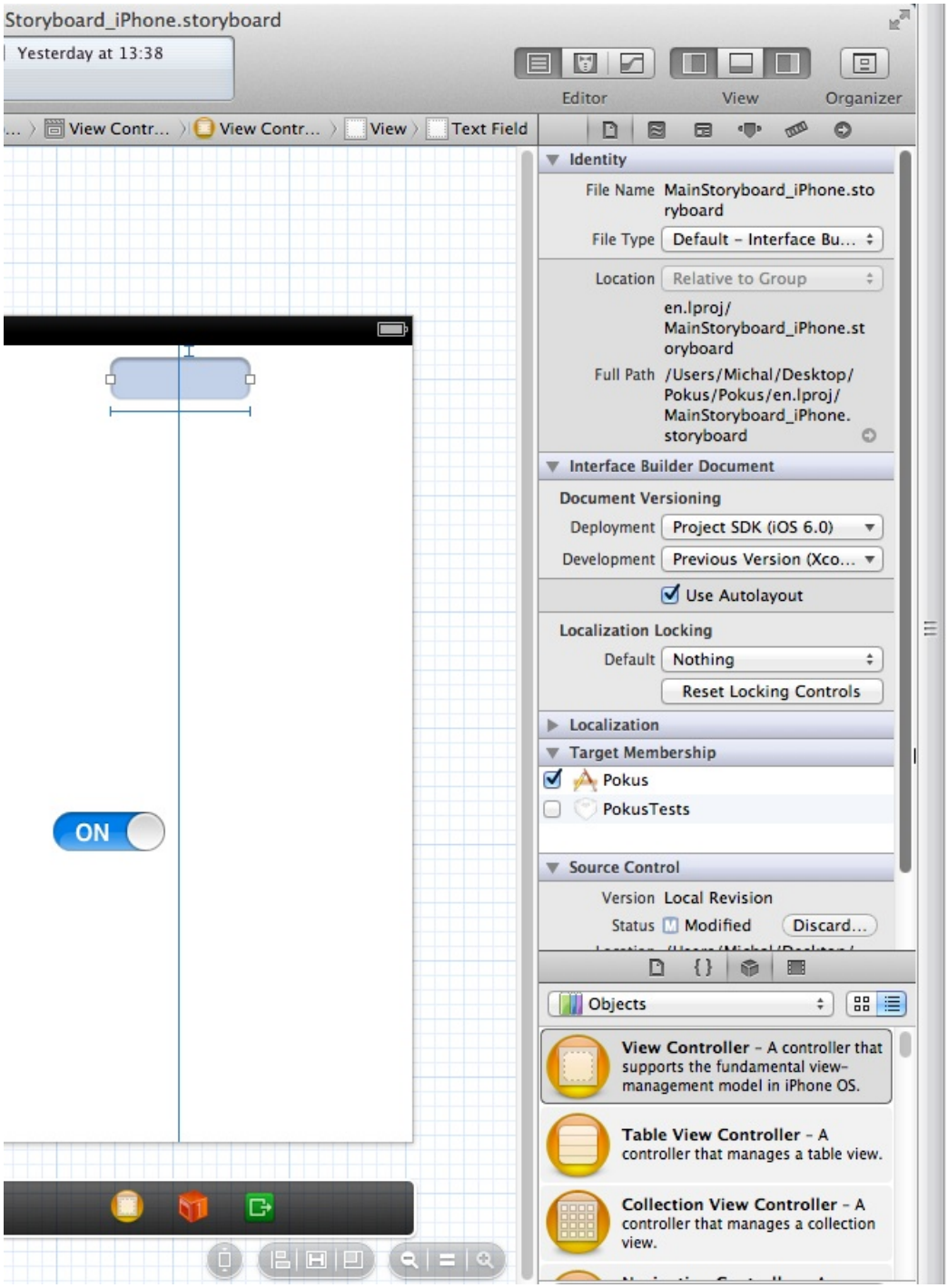
K instalaci Xcode jsou třeba instalační binární soubory. Ty jsou buďto dodávány s počítači Mac, nebo je lze stáhnout přes web. Instalace přes web je zdlouhavější a dost záleží na internetovém připojení. Po spuštění aplikace Install Xcode.app se spustí standardní instalační interface, který obsahuje obvyklé věci, jako je licenční smlouva, umístění instalace apod. Pokud byla nainstalovaná starší verze, nedojde k její aktualizaci, ale instalátor ji pouze přejmenuje a novou verzi nainstaluje kompletně celou. Pokud instalujete aplikaci z disku nebo DVD, instalátor vám přímo nabídne možnost výběru balíčků, které se mají nainstalovat (obrázek 4.3).

Instalace v závislosti na výkonu zařízení trvá přibližně 30 minut. Během instalace máte možnost instalovat různé části Xcode. Základní balíček je třeba vždy, ale další nástroje, podpora pro UNIX a dokumentace Mac OS X 10.4, je volitelná. Doporučuji instalaci dokumentace a dalších nástrojů. Zejména další nástroje mají široké možnosti využití. Jakmile Xcode dokončí svou instalaci, je okamžitě k dispozici. Kromě nejdůležitějších souborů budou nainstalovány i další, které využijete až v pokročilejším programování.

Obr. 4.2: Instalační prostřední Xcode



Obr. 4.3: Výběr balíčků k instalaci

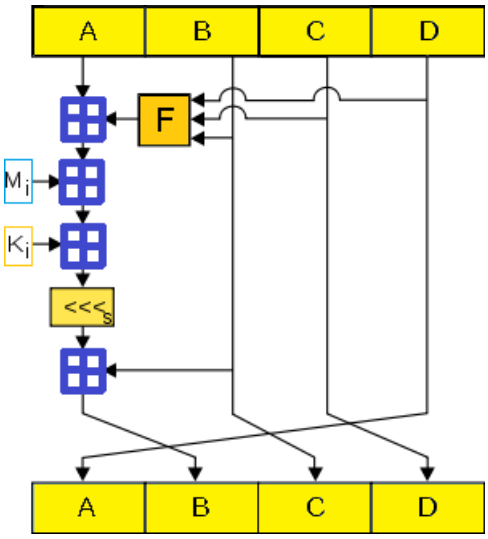


Obr. 4.4: Panely a pracovní prostředí

4.2 Hashovací funkce

4.2.1 MD5 hash

Kryptografické hashovací funkce musí mít určité vlastnosti, aby co nejvíce zvyšovaly obtížnost napadení. Ty jsou: odolnost vůči získaní předlohy, dolnost vůči získaní jiné předlohy a odolnost vůči nalezení kolize. Kolize je v stav, kdy hash nebo otisk kryptografické hashovací funkce přiřazuje stejný výstup pro různá vstupní data. Mezi nejznámější hashovací funkce patří MD5 a SHA.[12]

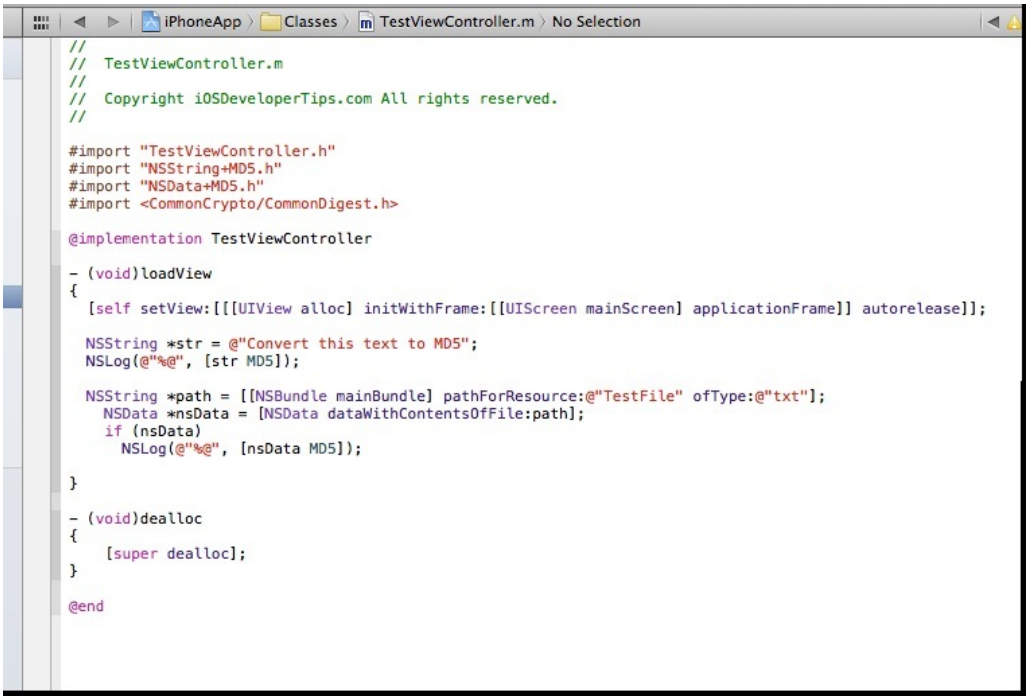


Obr. 4.5: MD5 Hash

MD5 zkráceně Message-Digest Algorithm je široce využívaná hashovací funkce. Vytváří 128-bitovou hodnotu hash klíče. Je využíván v širokém sortimentu bezpečnostních aplikací a využivá se pro kontrolu integrity dat. Jeho tvůrcem je Ronald Rivest, který v roce 1991 nahradil touto funkcí starší MD4 hash. MD5 hash je obvykle vyjádřen hexadecimálním číslem o délce 32 znaků. Nicméně bylo prokázáno, že není velmi dobře odolný, a proto ho není vhodné používat pro složitější bezpečnostní aplikace. V roce 1996 byla objevena chyba v návrhu této funkce a bylo doporučeno používat jiné, například SHA. V dalších letech bylo objeveno stále více trhlin v návrhu MD5 a v dnešní době není pro zkušené osoby velkou těžkostí toto zabezpečení překonat. S dnešními výkonnými procesory to může být otázka jen pár vteřin.[13]

Nejčastěji se využívá k ukládání hesel. Algoritmus přidá několik náhodných bitů k heslu, a tím stíží případné útoky pomocí programů, které využívají velkou zásobu slov, které postupně zkouší jako heslo. Pro další zvýšení bezpečnosti se dají kombinovat různé řetězce, jako je uživatelské jméno a heslo. Pokud dva uživatelé používají stejné heslo, jejich jiné jméno zajistí naprosto odlišný hash jejich hesel. MD5 často

využívají souborové servery nebo aplikace pro Unixové operační systémy. MD5 zpracovává různě dlouhé zprávy a převádí je na výstup v pevné délce 128 bitů. Vstupní zpráva je rozdělena do 512-bitových bloků. Hlavní algoritmus pracuje v 128-bitovém stavu, rozděleným do čtyř 32-bitových slov, které jsou označeny písmeny A, B, C a D. Algoritmus poté modifikuje každý blok o délce 512 bitů v daném pořadí. Zpracování bloku má čtyři podobné fáze, které obsahují 16 operací, které jsou založeny na nelineárních funkcích, jako je modulární sčítání nebo rotace. Algoritmus má k dispozici 4 různé nelineární funkce a v každém kole je využívaná jiná.[13]

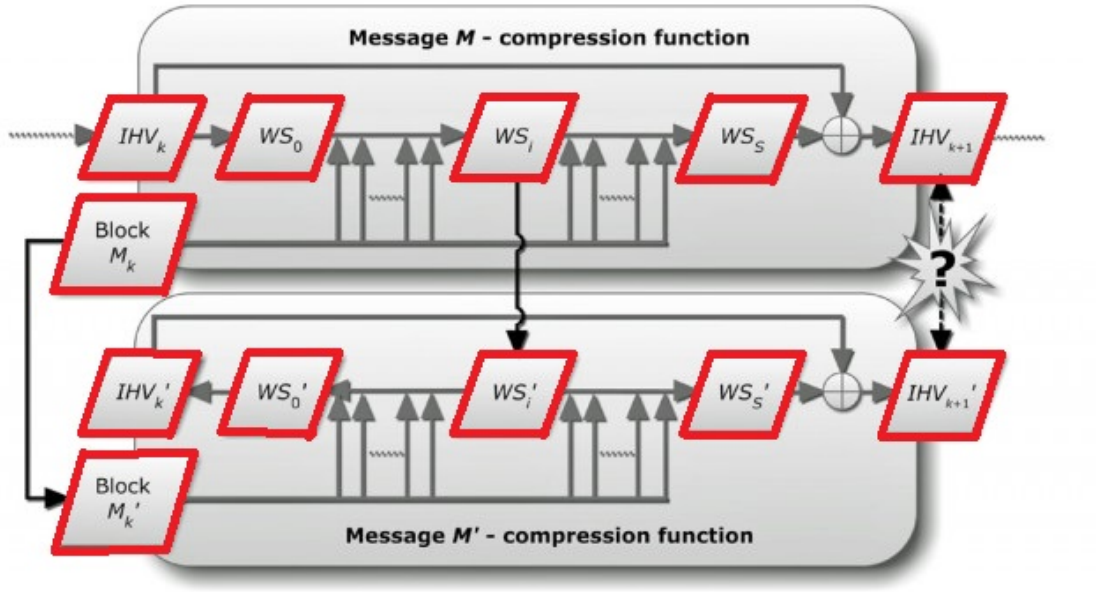


Obr. 4.6: Převedení stringu pomocí MD5

4.2.2 Hash SHA1

Tuto kryptografickou hashovací funkci navrhla Americká Národní Bezpečnostní Agentura. SHA-1 je nejrozšířenější ze současných hashovacích funkcí. V roce 2005 se ovšem objevily úspěšné útoky na algoritmus SHA-1, což snížilo jeho bezpečnost, a proto se po roce 2010 přešlo na SHA-2, který je bezpečnější.[12]

SHA-1 produkuje 160-ti bitový výstup na obdobných principech, jako algoritmus MD5 hashe. Je součástí několika bezpečnostních aplikací a protokolů, například SSL, PGP, SSH nebo IPsec. Používá se v distribuovaných systémech revizí, jako jsou Git nebo Mercurial, k zjišťování poškozených dat.[12]



Obr. 4.7: Hash SHA-1

SHA-1 rozděluje vstupní zprávu na bloky o délce 512-ti bitů, poslední blok zprávy doplňuje a zarovnává, včetně přidání údaje o délce zprávy, na které je vyhrazeno posledních 64 bitů. SHA-1 tak může zpracovávat vstupní zprávy o délce až do 2.305.840 TB. Uvedením délky hashe se výrazně ztěžuje možnost nalezení a výskytu kolizí mezi zprávami různých délek. Kolize se primárně hledají mezi zprávami zcela shodné délky.[12]

SHA-1 je podobně jako jiné hashovací funkce tzv. iterativní hashovací funkce. To znamená, že ke zpracování každého 512-ti bitového bloku se vždy použije stejný modul, tzv. vnitřní kompresní funkce. Kompresní funkce má dva vstupy, 160-ti bitový a 512-ti bitový. Po zpracování posledního bloku se výstup kompresní funkce použije jako výstupní hodnota.[12]

4.3 QR kód

Zkratka QR vychází ze slovního spojení „*Quick Response*“, což v překladu znamená kód rychlé reakce. Využívá se ke kolekci dat, ale dokáže jich zakódovat mnohem větší množství, než standardní EAN čárové kódy. Specifikace QR kódů se stala v červnu roku 2000 standardem ISO 18004, ale byla dále ještě upravována až do roku 2006. Kódy využívají velké množství technik, pomocí kterých předcházejí špatné interpretaci. Díky tomu je i poškozený kód stále čitelný. Algoritmy, které zpracovávají kódy, jsou většinou schopny číst i pootočený kód nebo kód s invertovanými či nekontrastními barvami. Je definováno 40 velikostních verzí. Každý QR kód je

tvořený čtvercovou mřížkou bodů a dalšími informačními vrstvami. Může obsahovat písmena, čísla nebo i japonské znaky. Podle typu obsahu je schopný pojmout určité množství dat.[11]

Kapacita

- Číslice - 7089 znaků
- Písmena a číslice - 4296 znaků
- 8-bitová data - 2953 znaků
- Jap. znaky - 1817 znaků

Obr. 4.8: Příklad QR kódu

4.3.1 Popis vrstev

Geometrická vrstva

Používá se pro lokalizaci pozic, ve kterých má dojít ke čtení informačních bitů. Základ této vrstvy tvoří alespoň 4 body, které ohraničují celý kód. Tato vrstva je ve většině případů reprezentována bílou a černou barvou. Rozměry bodů jednotlivých čtverečků se při dekódování primárně určí podle velikosti čtverců 7x7, které se označují jako *finders* a jsou umístěny v rozích QR kódu. Všechny verze kódů mají vyhrazen 6. řádek a sloupec na tzv. *timing* vzor, ve kterém se střídají černé a bílé body.[10]

QR kódy si prošly postupným vývojem od modelu 1 po současný model, přičemž s každým modelem přišla řada vylepšení. Například model 2 měl zvýšenou odolnost proti deformacím a došlo pak k jeho celosvětovému rozšíření. Od verze 7 a výše se číslo příslušné verze kódu zabezpečuje pomocí 18-ti bitů a zapisuje se do obdélníků o rozměrech 3x6 bodů, které se dotýkají vnějších *finders*.[10]

Informační vrstva

Tato vrstva obsahuje zakódované binární jedničky a nuly. Jedničky jsou prezentovány černými body a nuly bílými body. Hlavní datová část vyplňuje dosud nevyužité body v definovaném pořadí. Začíná v rohu s maximální souřadnicí a vždy postoupí o jednu doleva, pak se vrátí a postoupí o jednu vertikálně. Postupuje nahoru a jakmile vyplní první dva krajní sloupce, začne sestupovat v následujících dvou sloupcích. Aby nevznikaly vzory matoucí geometrickou vrstvu, je hlavní datový tok maskován 8-mi jednoduchými maskovacími vzory. Jejich negovaná hodnota je pak uložena na příslušné souřadnice.[10]

Kód	Popis	Vzorec	Perioda
000	Šachovnice	(x+y) mod 2	2×2
001	Vodorovné pruhy	y mod 2	1×2
010	Svislé pruhy ob 2	x mod 3	3×1
011	Diagonální pruhy ob 2	(x+y) mod 3	3×3
100	Šachovnice obdélníků 3×2	(⌊y/2⌋+⌊x/3⌋) mod 2	6×4
101	První velká dlaždice	(x*y) mod 3 + (x*y) mod 2	6×6
110	Druhá velká dlaždice	((x*y) mod 3 + (x*y)) mod 2	6×6
111	Třetí velká dlaždice	((x*y) mod 3 + (x+y)) mod 2	6×6

Obr. 4.9: Tabulka maskovacích vzorů

Datový tok obsahuje bitovou zprávu a další zabezpečovací bity. Od vyšších verzí se datová zpráva rozděluje do bloků a každý blok má pak vlastní zotavovací funkci z případných chyb. Každá verze QR kódu má definované 4 úrovně zabezpečení, kde každá úroveň definuje, na jaké bloky se zpráva rozpadá a kolik zabezpečovacích bitů je pro každý blok použito. Informace o zabezpečení je uložená v osmém řádku a sloupci, v patnácti bitech pomocí BCH kódu.[11]

Další vrstvy

Je normou definované, jak sdružovat několik QR kódů dohromady. Každý kód pak ví, o kolikátou z kolika částí se jedná. Začínají se používat standardy interpretace osahu zpráv, například příkazy k úhradě do banky.

4.4 Knihovna ZXing

Jedná se o open-source knihovnu, která umí pracovat s barkódy a QR kódy. Umožňuje zpracování a dekódování QR kódů nebo jejich vytváření, například ze zadaného textu. Původně byla implementována v jazyku Java s porty pro ostatní jazyky. Knihovna se zaměřuje především na využití kamer v mobilních zařízeních pro skenování a dekódování barkódů, aniž by bylo třeba komunikovat s nějakým vzdáleným serverem. V současné době knihovna podporuje velké množství formátů.[8]

Podporované formáty

- UPC-A and UPC-E
- EAN-8 and EAN-13
- Code 39 a 93
- Codabar
- Data Matrix
- Aztec
- QR code
- RSS-14 (všechny varianty)

Knihovna je rozdělena na několik hlavní komponentů, přičemž každý z nich má svoji podporu.

- core - jádro knihovny pro dekódování obrázků
- javase - klient pro J2SE
- android - klient pro Android
- androidtest - testovací aplikace pro android
- android-intergration - podpora pro integraci aplikace Barcode Scanner

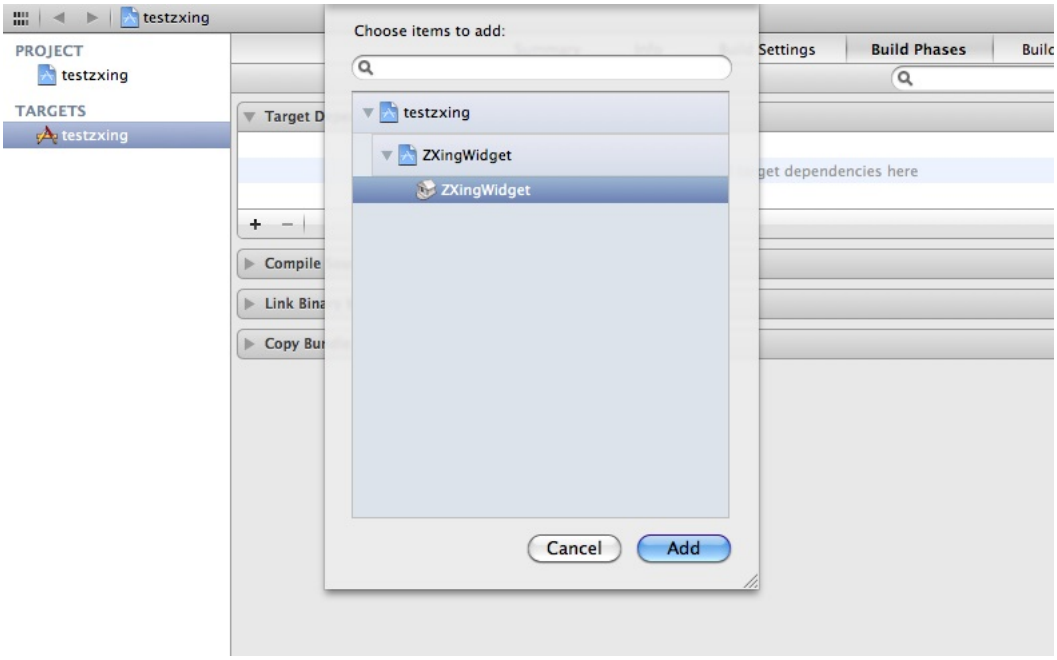
Obsahuje také další přídavné moduly, které usnadňují implementaci pro jednotlivé podporované platformy.

- cpp - podpora pro C++
- iphone - podpora pro iPhone klienta a Objective C/C++
- zxing.appspot.com - zdrojový kód pro webové generátory
- csharp - podpora pro C#
- jruby - Ruby wrapper
- actionscript - podpora pro Actionscript

4.4.1 Implementace knihovny do Xcode

Pro správnou funkčnost knihovny v projektu je třeba ji správně implementovat. Nejdůležitější je stáhnout nejnovější verzi knihovny, starší verze nelze v nejnovějších Xcode implementovat. Knihovnu je možné stáhnout například z následujícího odkazu: https://code.google.com/p/zxing/.

Po úspěšném stáhnutí a rozbalení je třeba najít *ZXingWidget.xcodeproj* v adresáři *zxing/iphone/ZXingWidget/* a ten přetáhnout do navigátoru ve svém projektu v Xcode.



Obr. 4.10: Výběr odpovídajícího targetu

Jakmile jsou všechny soubory načteny, musí se přidat *ZXingWidget.xcodeproj* do fází builderu, aby se také zkompiloval při sestavování hlavního projektu. V navigátoru vybereme náš projekt a v něm označíme odpovídající target. Následně zvolíme položku *Build Phases* a rozbalíme si *Target Dependencies*. Klikneme na tlačítko pro přidání, v novém okně označíme *ZXingWidget* a přidáme ho.[8]

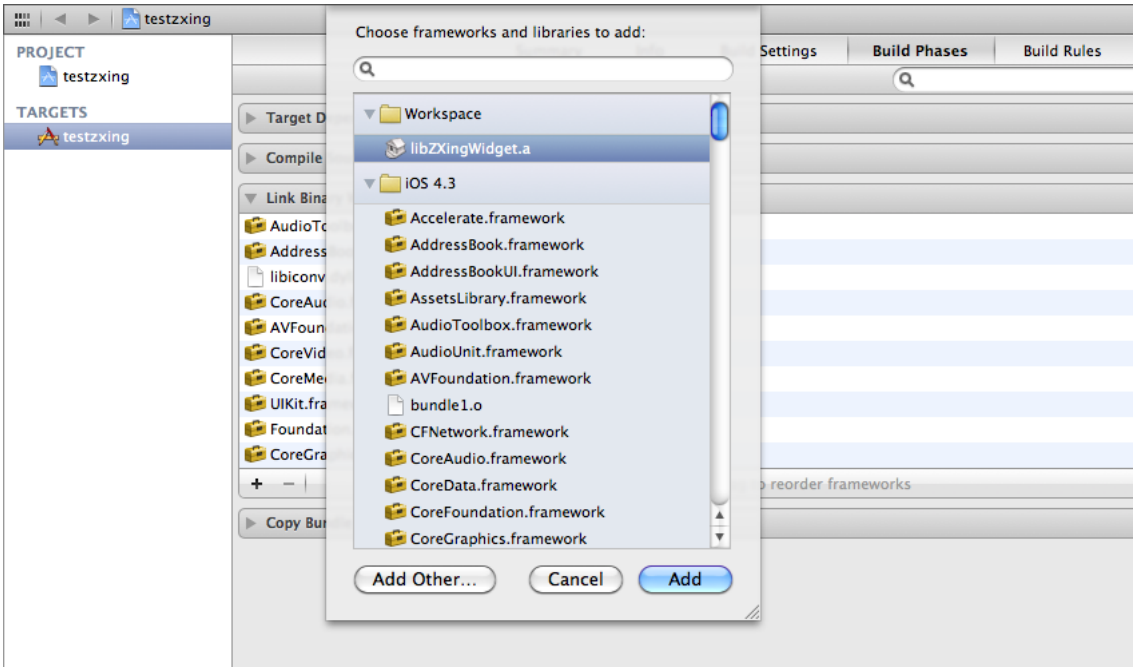
Nyní je potřeba zajistit pro knihovnu všechny požadované součásti, aby vše správně fungovalo. Stejně jako v minulém kroku vybereme *Build Phases*, ale rozbalíme si *Link Binary With Libraries*. Stiskneme tlačítko + (přidat) a vybereme *libZXingWidget.a*. Dále pak přidáme všechny další frameworky pro správnou funkčnost knihovny.[8]

Potřebné frameworky:

- AddressBook

- AddressBookUI
- AudioToolbox
- AVFoundation
- CoreMedia
- CoreVideo
- libiconv.dylib

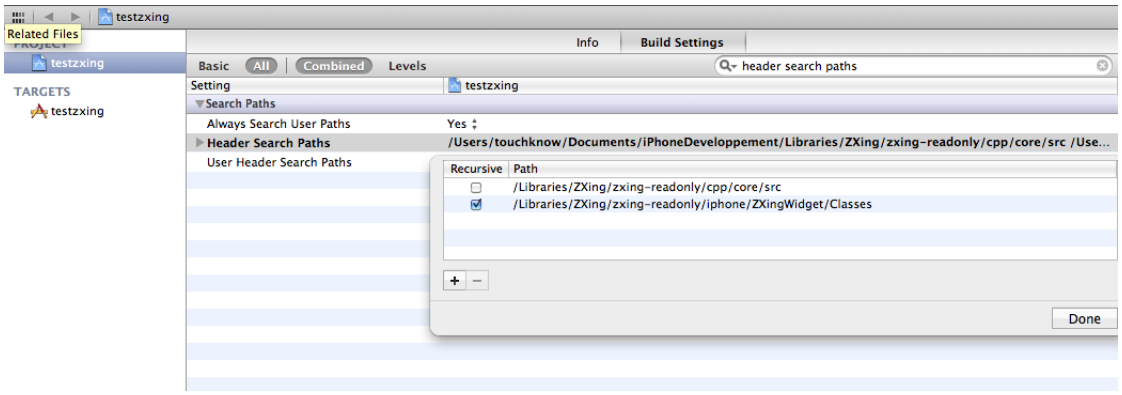
Nakonec je třeba nakonfigurovat správné vyhledání hlaviček *ZXingWidget*, aby s nimi Xcode mohl správně pracovat. V navigátoru vybereme náš hlavní projekt, nikoliv *ZXingWidget.xcodeproj*, a přejdeme na záložku *Build Settings*, ve které najdeme *Header Search Paths*. Na levou část zvoleného řádku dvakrát klikneme a přidáme plnou cestu ke složce *zxing/iphone/ZXingWidget/Classes* a také k *zxing/cpp/core/src*. Nyní už stačí jen importovat *ZXingWidgetController.h* a *QRcodeReader.h* do našeho projektu a můžeme je používat.[8]



Obr. 4.11: Přidání potřebných frameworků

4.5 Balíček ZBar SDK

ZBar je open source knihovna pro čtení,zpracovávání a generování barkódů z různých zdojů. Její vrstvené provedení umožňuje snímání čárových kódů a jejich dekódování



Obr. 4.12: Přidání cesty pro soubory knihovny

pro libovolný typ aplikace. Její součásti lze snadno itegrovat do nejrůznějších jazyků, jako například Python, Perl, C++ nebo Objective C. Její používání je velmi rychlé a efektivní. Generované kódy mají malou velikost a nejsou limitovány velikostí obrázku.[7]

Podporované formáty

- UPC-A a UPC-E
- EAN-13 a EAN-8
- Code 39 a 128
- QR Code
- Interleaved 2 of 5

4.5.1 Implementace knihovny do Xcode

Implementace je podstatně jednodušší než v případě ZXing. Stáhneme si nejnovější verzi knihovny, například z `http://zbar.sourceforge.net/iphone`. Poté, co rozbalíme dmg soubor, přetáhneme složku ZBarSDK do našeho Xcode projektu. V dialogovém okně, které se objeví, zvolíme Copy. Do našeho projektu se tak nakopírují všechny potřebné soubory.[7]

Nyní je potřeba zajistit pro knihovnu všechny součásti pro správnou funkčnost. V targetech našeho projektu vybereme naše vložené ZBarSDK a v *Build Phases* si rozbalíme *Link Binary With Libraries*. Stiskneme tlačítko + (přidat) a vybereme frameworky, které chceme přidat.

Potřebné frameworky:

- QuartzCore
- AVFoundation

- CoreMedia
- CoreVideo
- libiconv.dylib

Poté už stačí jen do hlavičky svého projektu vložit: *#import "ZBarSDK.h"*

4.6 Knihovna GMP

GMP je freewarová knihovna, která umožňuje práci s velkými celými i racionálními čísly a početní operace s nimi. Nemá žádná omezení na velikost přesnosti, jediné omezení je velikost dostupné paměti, na kterých je knihovna provozovaná. GMP knihovna obsahuje velkou sadu funkcí pro jednotlivá rozhraní.[6]

Knihovna je přednostně navržena pro kryptografické aplikace a výzkum internetového zabezpečení. Dále pak pro různé výzkumy výpočetní algebry nebo samostatné složité výpočtové systémy.[6]

Knihovna je navržena pro maximální možnou rychlost ať už velkých nebo malých početních operací. Této rychlosti je dosaženo použitím *fullwords*, jako základního aritmetického typu, a pomocí rychlých algoritmů s vysoce optimalizovaným kódem pro vnitřní práci procesorů.[6]

První verze GMP byla publikována v roce 1991 pod společností GNU LGPL. Díky této licenci je knihova volně šířitelná a modifikovatelná. Pouze pro placené aplikace má definovaná určitá pravidla, která je třeba dodržovat. Další roky byla dále vyvíjena a optimalizována pro různé používané platformy.[6]

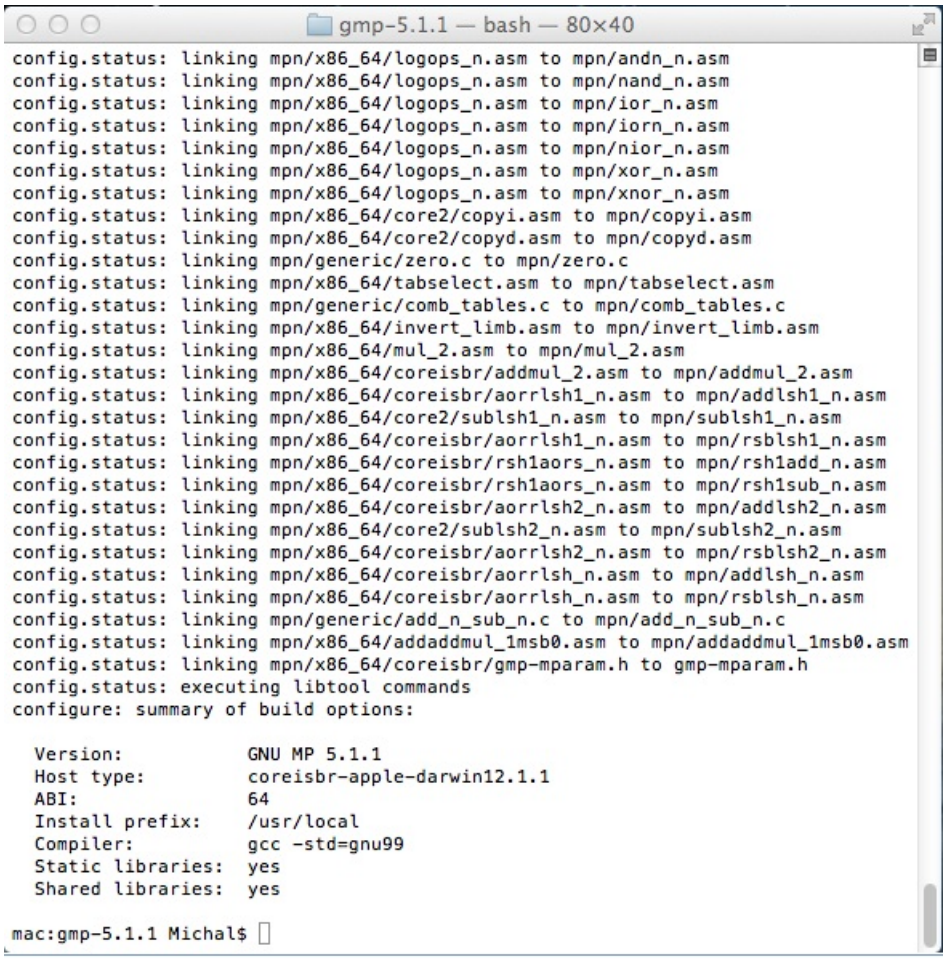
4.6.1 Funkční kategorie

- Vysoká úroveň lineární algebry, která obsahuje přes 150 logických operací.
- Vysoká úroveň racionální aritmetiky. Tato kategorie obsahuje 35 různých početních operací.
- Vysoká úroveň aritmetiky s plovoucí desetinnou čárkou. Tyto fukce lze použít, pokud typ Double nedosahuje požadované přesnosti. Tato kategorie bsahuje asi 70 funkcí.
- Vysoká úroveň aritmetiky s plovoucí desetinnou čárkou pro C++ rozhraní.
- Rozhraní pro malá čísla. Funkce pro přijímání vstupních argumentů.
- Funkce pro zaokrouhlování na vysoké úrovni s plovoucí desetinnou čárkou.

4.6.2 Implementace knihovny do OS X a Xcode

Z oficiálních stránek http://gmplib.org/ je třeba stáhnout soubor *gmp-2.1.tar.gz*. Jakmile je soubor stažený, tak ho rozbalíme a spustíme terminál.

Nyní je potřeba spustit konfigurační soubor. Rozbalený soubor přesuneme do svého domovského adresáře a poté se do něj v terminálu přepneme pomocí: *cd <název soboru>*. Následně musíme zadat příkaz *./configure* a spustí se konfigurační proces, jehož výsledek by měl vypadat přibližně jako na obrázku 4.13.



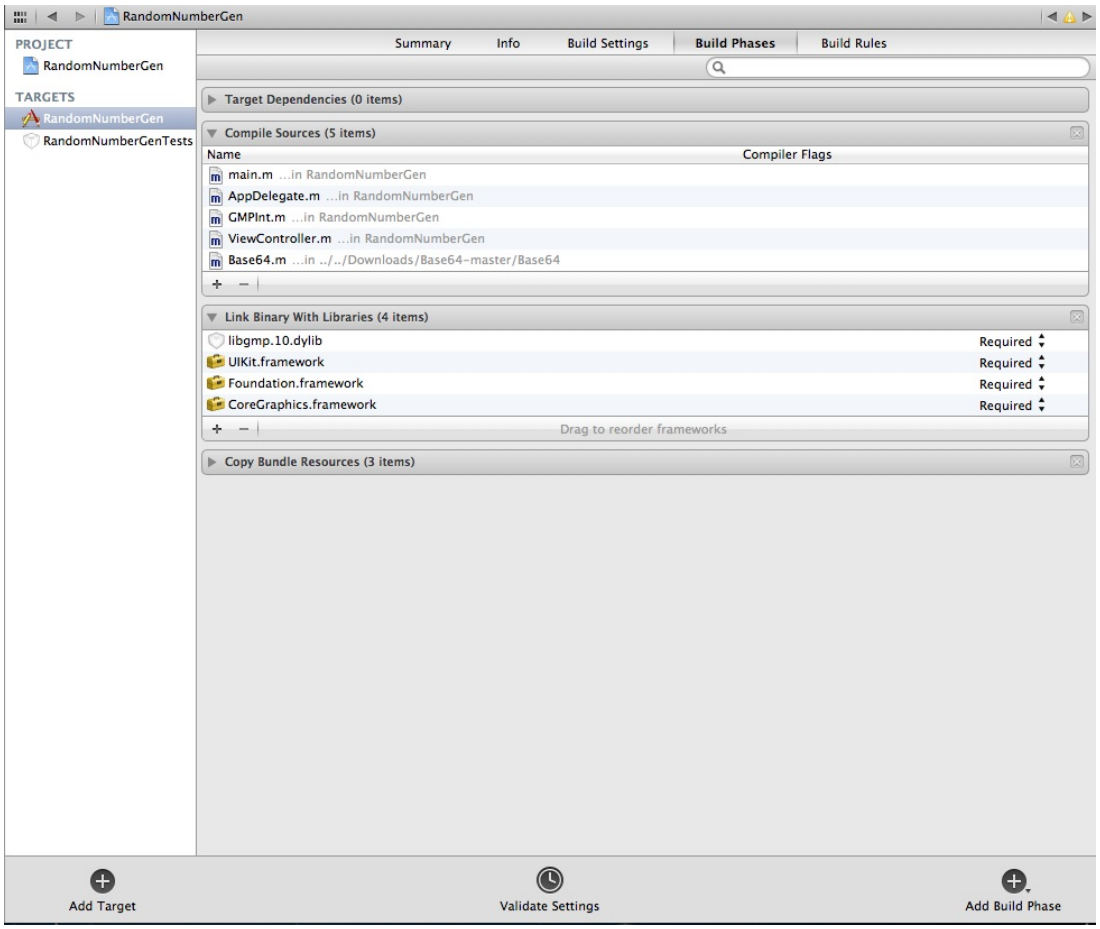
Obr. 4.13: Dokončení konfigurace.

Jakmile je proces dokončen, zadáme do terminálu příkaz *make*, který spustí build souborů. Po ukončení procesu make je doporučeno zadat příkaz *make check*, který zkontroluje jestli jsou připraveny všechny soubory k instalaci.

Pokud je všechno v pořádku, zadáme do terminálu příkaz *sudo make install*. Nestačí zadat příkaz pouze *make instal*, protože neinstalujeme jako administrátor.

Pokud instalace proběhala v pořádku, můžeme přidat knihovnu do Xcode. V navigátoru Xcode označíme náš projekt a vybereme příslušný target. Poté se přepneme

do položky *Build Phases* a zde v záložce *Link Binary With Libraries* zmáčkneme tlačítko *Add*. V novém okně je třeba najít příslušnou knihovnu *libgmp.10.dylib*. Pokud není vidět přímo mezi dostupnými knihovnami, je třeba zadat její cestu ručně. Implicitně bývá nainstalovaná v */user/Tool/Gmp/Lib/*. Po přidání knihovny už jen stačí do svého projektu přidat do hlavičky *#import "gmp.h"* a je možné začít funkce knihovny používat.[6]

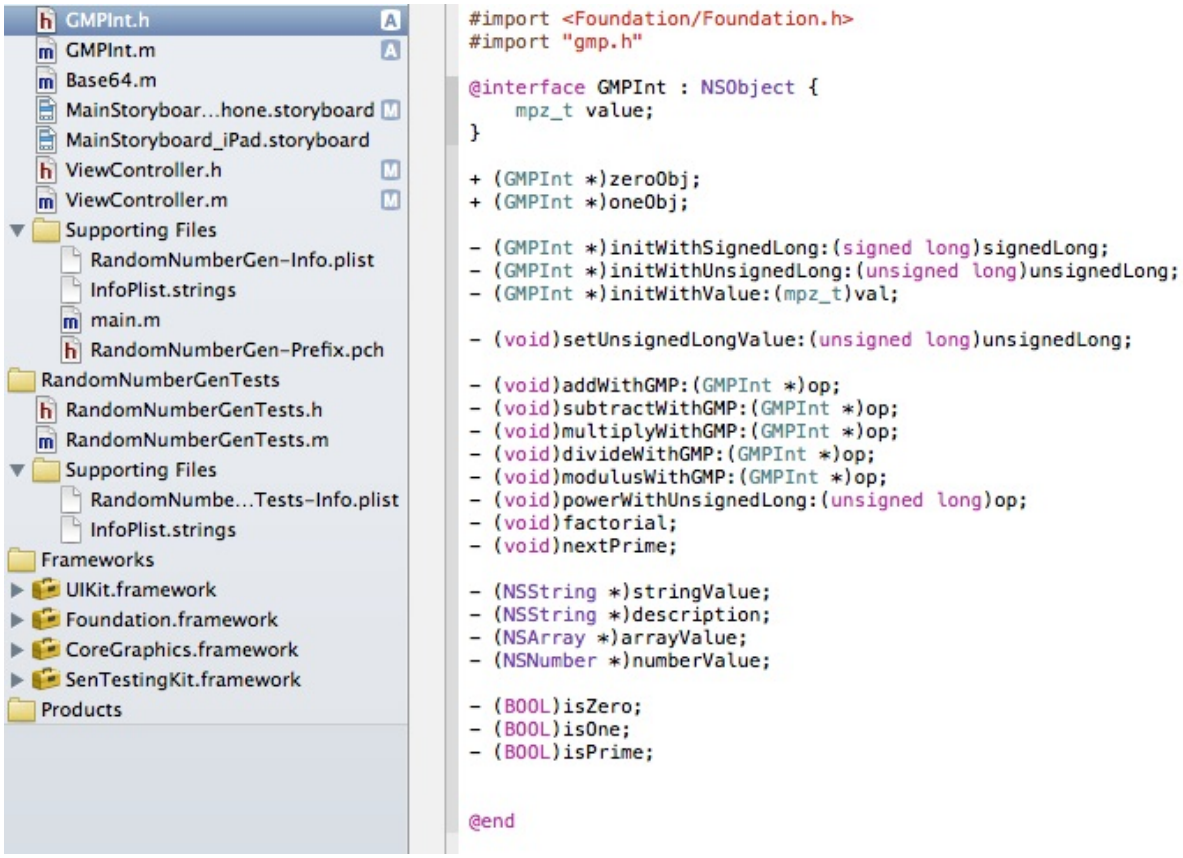


Obr. 4.14: Přidání knihovny do Build Phases

4.6.3 GMPInt

Protože samotná knihovna GMP obsahuje obrovské množství funkcí, doporučuji pro snadnější použití zvolit nějaký wrapper. Wrapper obsahuje sadu funkcí, které usnadňují použití knihovny. Jako nejlepší se jeví GMPInt, který obsahuje několik užitečných funkcí (např. převod velkých čísel na string), které budeme moci využít. Z adresy https://github.com/bmorton/GMPInt tedy stáhneme zip soubor a rozbalíme.

Rozbalený soubor obsahuje tři soubory, *GMPInt.h*, *GMPInt.m* a README, ve kterých jsou jednoduché příklady použití wrapperu. Soubory přetáhneme do navigátoru v našem projektu v Xcode. Poté stačí do hlavičky našeho projektu přidat *#include "GMPInt.h"* a můžeme začít používat funkce wrapperu. Funkce, které nyní můžeme používat jsou zobrazeny na obrázku 4.15.

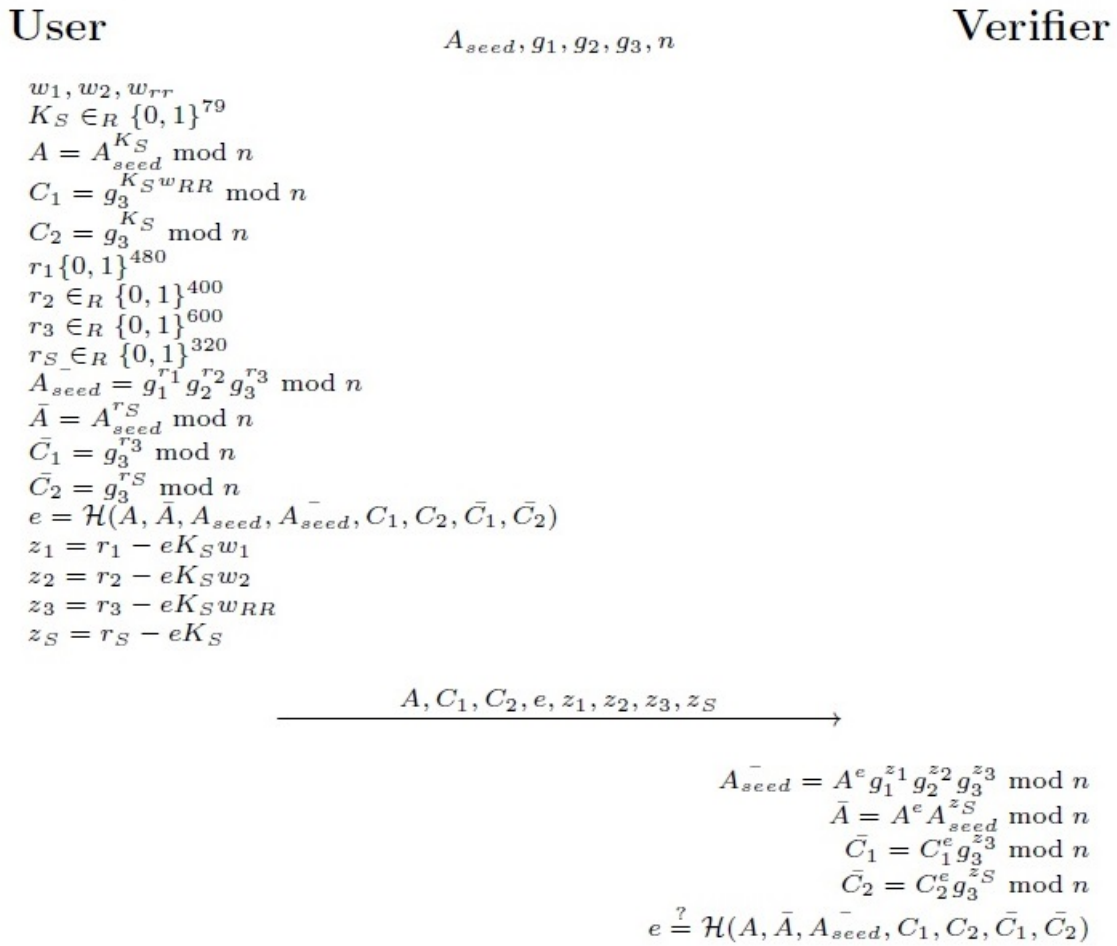


Obr. 4.15: Funkce wrapperu GMPInt

5 PRAKTICKÁ ČÁST

5.1 Implementovaný protokol

Úkolem bylo implementovat zadaný bezpečnostní protokol, který je zobrazen na obrázku 5.1. Tento protokol pracuje s velkými až 1024-mi bitovými čísly a pro výpočty nejčastěji využívá modulárního mocnění. Některé hodnoty, které využívá, jsou staticky dané a pomocí nich generuje další hodnoty pro další výpočty. Z vybraných hodnot poté sestaví SHA1 hash (e) s výstupem o velikosti 160 bitů. Pomocí této hodnoty spočítá zbylé hodnoty, které jsou třeba pro konečné vygenerování QR kódu $(A, C_1, C_2, e, z_1, z_2, z_3, z_s)$.



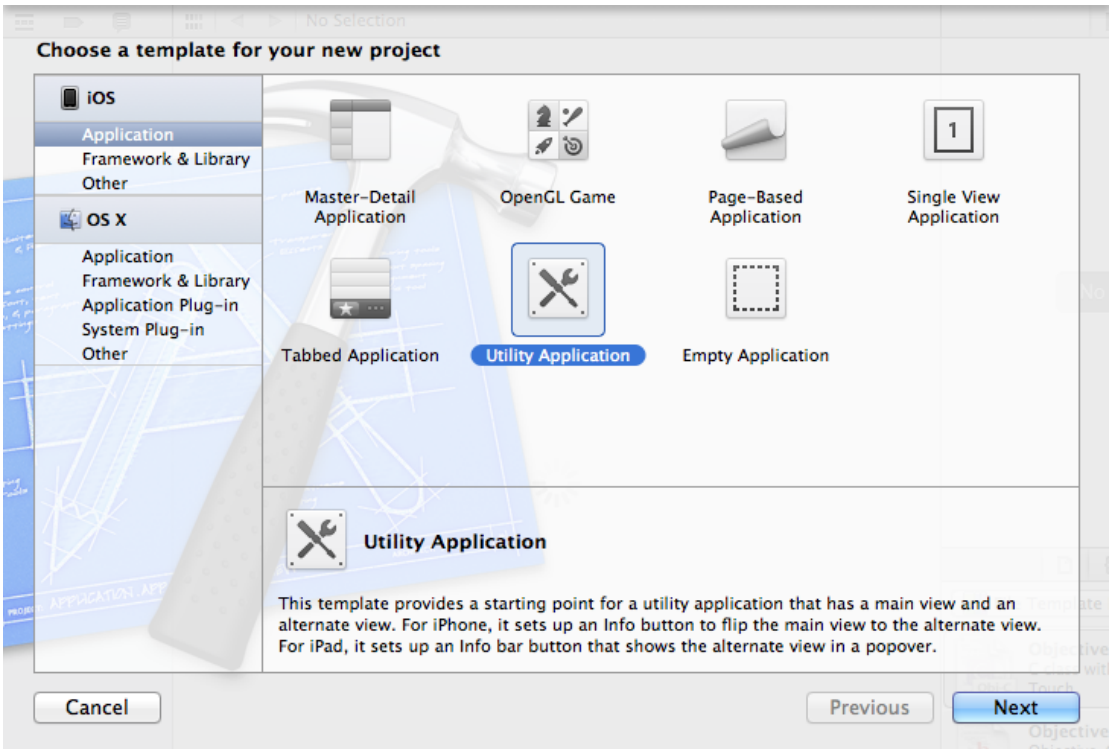
Obr. 5.1: Implementovaný protokol

5.2 Tvorba programu

Úkolem je vytvoření aplikace, která bude realizovat zadaný bezpečnostní protokol, jehož výstupem bude QR kód, který bude přenášet všech osm požadovaných hodnot. Program bude mít velmi jednoduché rozhraní. Bude obsahovat jedno tlačítko, které po stisknutí vygeneruje QR kód z hodnot vypočítaných v protokolu. Hodnoty počítané programem se budou pro každý stisk tlačítka měnit a s nimi samozřejmě i QR kód.

Před vytvořením projektu v Xcode, je třeba mít v systému nainstalované potřebné knihovny pro implementaci protokolu. Pro ukládání a operace s velkými čísly potřebujeme knihovnu GMP. Pro usnadnění používání samotné knihovny doporučuji stáhnout wrapper GMPInt, který obsahuje řadu užitečných funkcí, například převod výsledného velkého čísla na string, který pak můžeme zobrazit nebo použít pro další operace, jako je hashování. Ke generování QR kódu využijeme knihovnu Zbar SDK a pro tvorbu hashe SHA1 použijeme *base64.h*. Kde knihovny stáhnout a jak je správně implementovat, je popsáno v teoretické části.

V Xcode otevřeme nový projekt a pojmenujeme ho. V levém sloupci zvolíme správnou cílovou platformu a vybereme vytvoření aplikace. Konkrétní aplikaci použijeme *Utility Application*, která obsahuje prostředí s vhodnými nástroji pro tvorbu naší aplikace 5.2.



Obr. 5.2: Výběr typů aplikace

Jakmile naběhne prostředí pro editaci jednotlivých souborů, přidáme všechny potřebné knihovny, které budeme používat. Po úspěném nakopírovaní souborů do projektu přes navigátor musíme ještě přidat všechny potřebné frameworky, které budeme potřebovat. Aby vše správně fungovalo, musí se knihovny zkompilovat zároveň s projektem. Takže v *Build Phases* v položce *Link With Binary Libraries* vložíme všechny potřebné knihovny ke kompilaci.

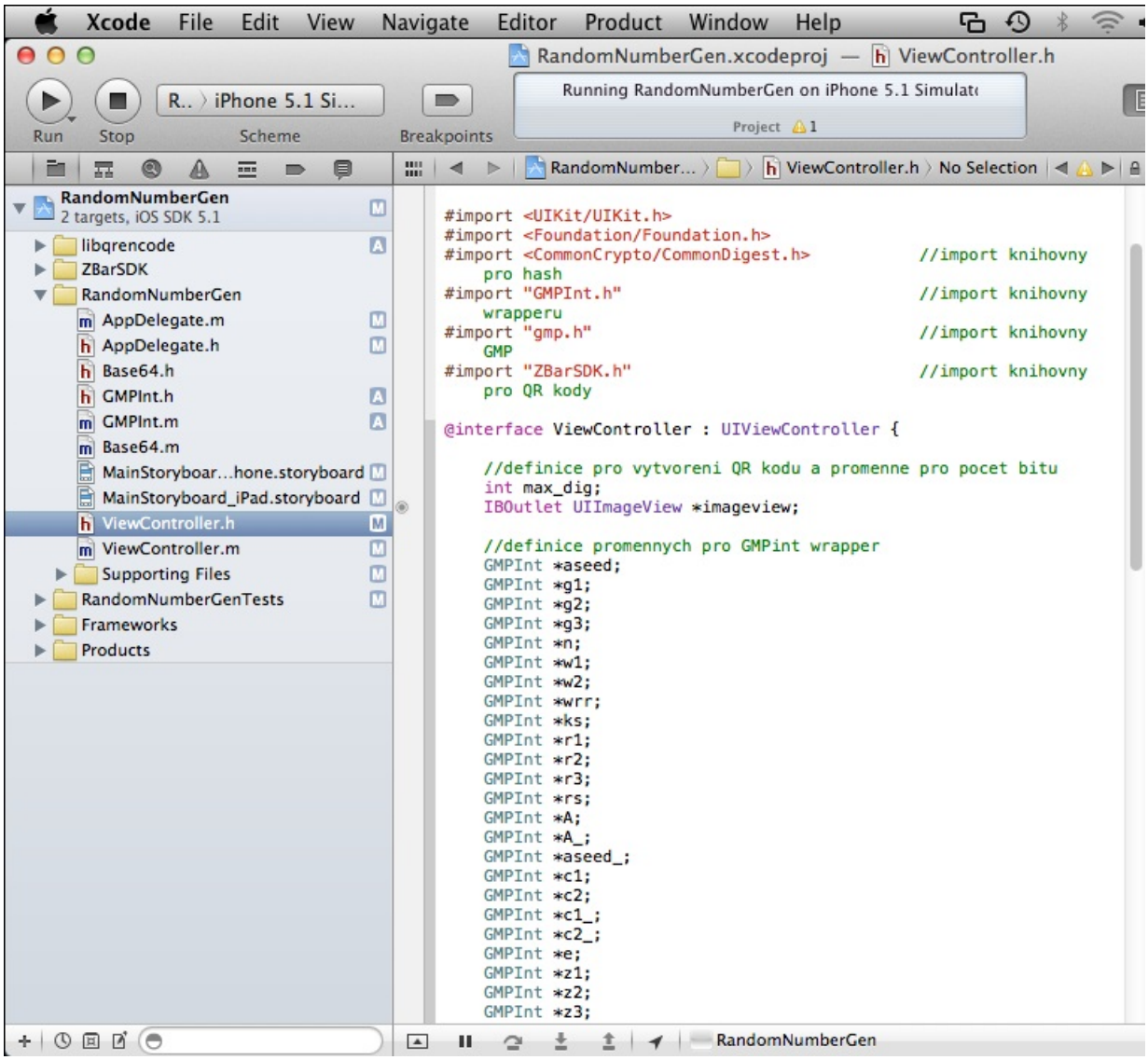
Aby bylo možné knihovny a jejich funkce používat, musíme je přidat do hlavičky našich tříd viewcontroller.h a viewcontroller.m pomocí *#import "název knihovny.h"*.

Třída `viewcontroller.h` slouží k definování jednotlivých proměnných, akcí a outletů, které se budou v projektu používat. Outlety *(IBOutlet)* vytváří odkazy pro spojení mezi jednotlivými prvky v grafickém rozhraní projektu. Akce *(IBAction)* definují reakce na nějakou událost, například zmáčknutí tlačítka.

Třída `viewcontroller.m` slouží k přímé implementaci jednotlivých funkcí a výpočetních operací. Inicializují se zde jednotlivé proměnné, které byly definovány v třídě *viewcontroller.h* a provádí se zde programování vytvořených akcí *IBAction*.

Pro naší aplikaci definujeme v třídě `viewcontroller.h` všechny potřebné proměnné. Budeme potřebovat dvě proměnné typu **Int**. Jedna bude definovat počet bitů pro náhodně generovaná čísla, druhá bude sloužit jako návratová hodnota pro funkce. Pak vytvoříme jeden **IBOutlet** pro **ImageView**, ve kterém se bude zobrazovat výsledný QR kód. Dále budeme potřebovat velké množství proměnných typu **mpz_t** z knihovny GMP, do kterých se budou ukládat velká čísla ať už z výpočetních operací nebo staticky daných pro protokol. Téměř pro každou **mpz_t** se musí vytvořit i proměnná typu **GMPInt**, aby se hodnoty ze struktur mohly převádět na stringy, ať už pro hash nebo pro QR kód. Pro statické naalokování velkých čísel je třeba vytvořit několik typů **char**, do kterých se číslo uloží jako řetězec, a později pomocí jedné z funkcí GMP převede na číslo do struktury **mpz_t**.

V třídě `viewcontroller.m` začneme postupně s implementací. Jako první vytvoříme odkaz na naši nadefinovanou IBAction, ve které inicializujeme všechny proměnné a budeme zapisovat zbytek kódu. Do jednotlivých typů const char pak nadefinujeme příslušné zadané konstanty, které jsou uvedené níže. Pak musíme inicializovat všechny nadefinované proměnné ze struktury mpz_t pomocí funkce **mpz_init(mpz_t value)**. Poté už můžeme všechny chary převést na hodnoty uložené do struktury mpz_t pomocí funkce **mpz_set_str(mpz_t number, char cnumber, int base)**. Zápis funkce vypadá například:
`err = mpz_set_str(mpz_aseed, chaseed, 10)`. *Err* je pouze návratová hodnota



Obr. 5.3: Definování konstant ve třídě viewcontroller.h

a hodnota v *chaseed* se naalokuje do proměnné *mpz_aseed* ze struktury mpz_t.

Zadané konstanty

$Aseed$ = 333289058122624068941073299883236219862110426775919510218457967665071829398250818690567236662100199295522785052819782665223787002929414172213533984483671382536247182286280447182888315318196498059459474203648683261833308667542000254122087693849424254101083203257869228591969228312232027550369793835388315204 45

$g_1 = 24946843320126007864150201941090929902416361128527839991103894461666$
$658770380262570602655121180755307603650110779369015015909858301870272668$
$781180059252432767296681935273045357370527018947570943691802994413001099 48$
$577764586579320219809381800477957178165051874099297371941782853778509091 9$
961859050140165675849

$g_2 = 360837347661176417593362629721473037868512339533268353077912397602 33$
$469024214086339571248435290126881259207726710826846280942999936750140172 1$
$448540223062015789331488824111312822532783725807821620252596091301092838 6$
$627022666888951274155291094350118650710497970064594969700086931881079893 2$
714099966551092607246

$g_3 = 484191671068152324456888385398720434031544869775048127863785800989 78$
$046671110574966366056355609740730254534301322614516260204555365935829489 7$
$572282116970033960466638005436624262848551507389397522252764653224248595 5$
$480614721252380431869101078048757789115854427294999198177188993973281687 3$
353762477494002958710

$n = 904504749130068921178621949147287745825965789055326413936023141331 78$
$674567850521157818522450328358993328738863682268593854940761376193126742 6$
$444995887635845534801807080308459842794607139764271077070562048707336072 0$
$633724650560337942354036164726517735286205543163821223214704123390739055 5$
459320034157134941607

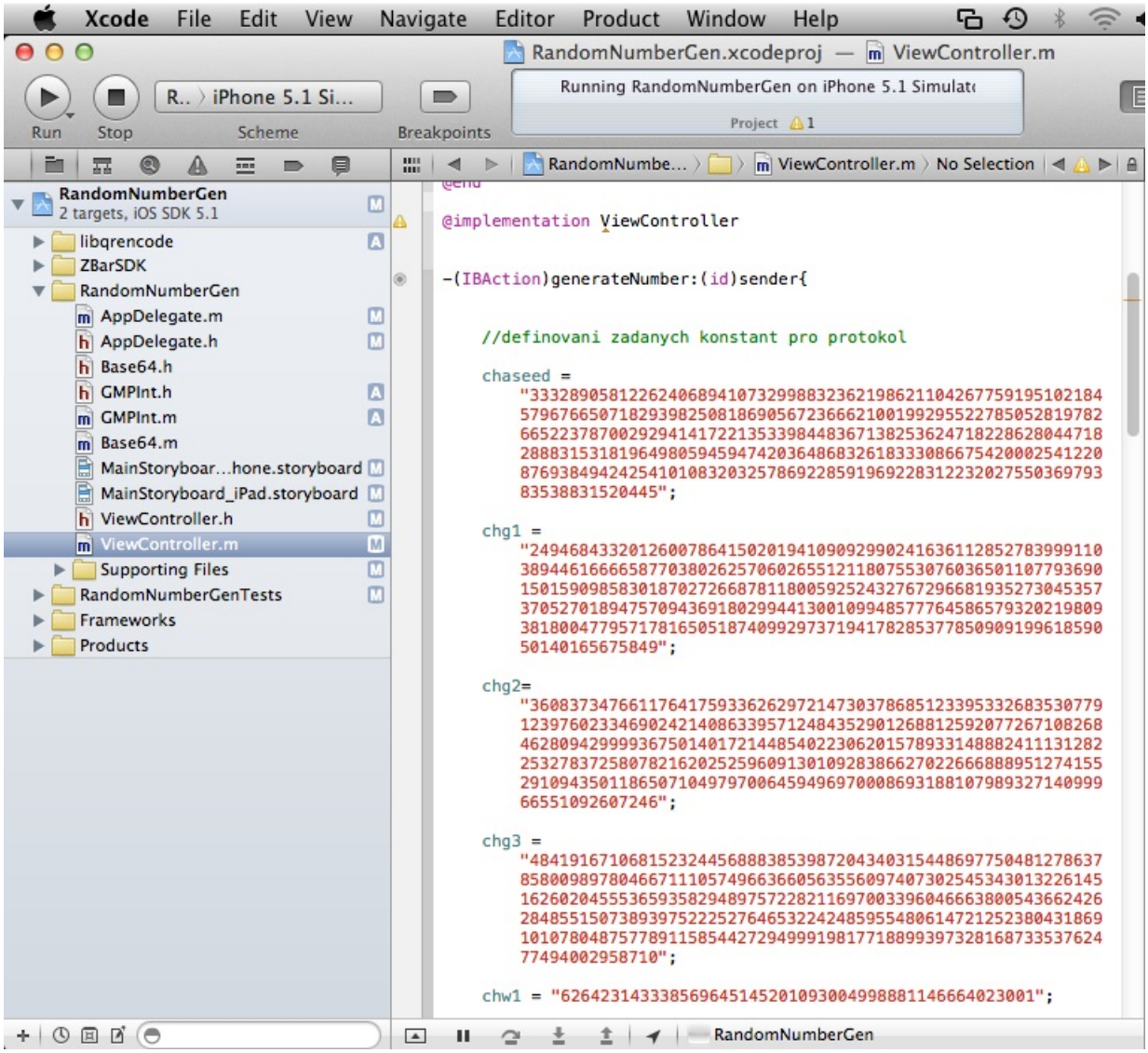
$w_1 = 626423143338569645145201093004998881146664023001$

$w_2 = 993181089445785543872483$

$w_{rr} = 162726494200232420924155224919928278999016482153812207802402882 4$
330849798456550935775

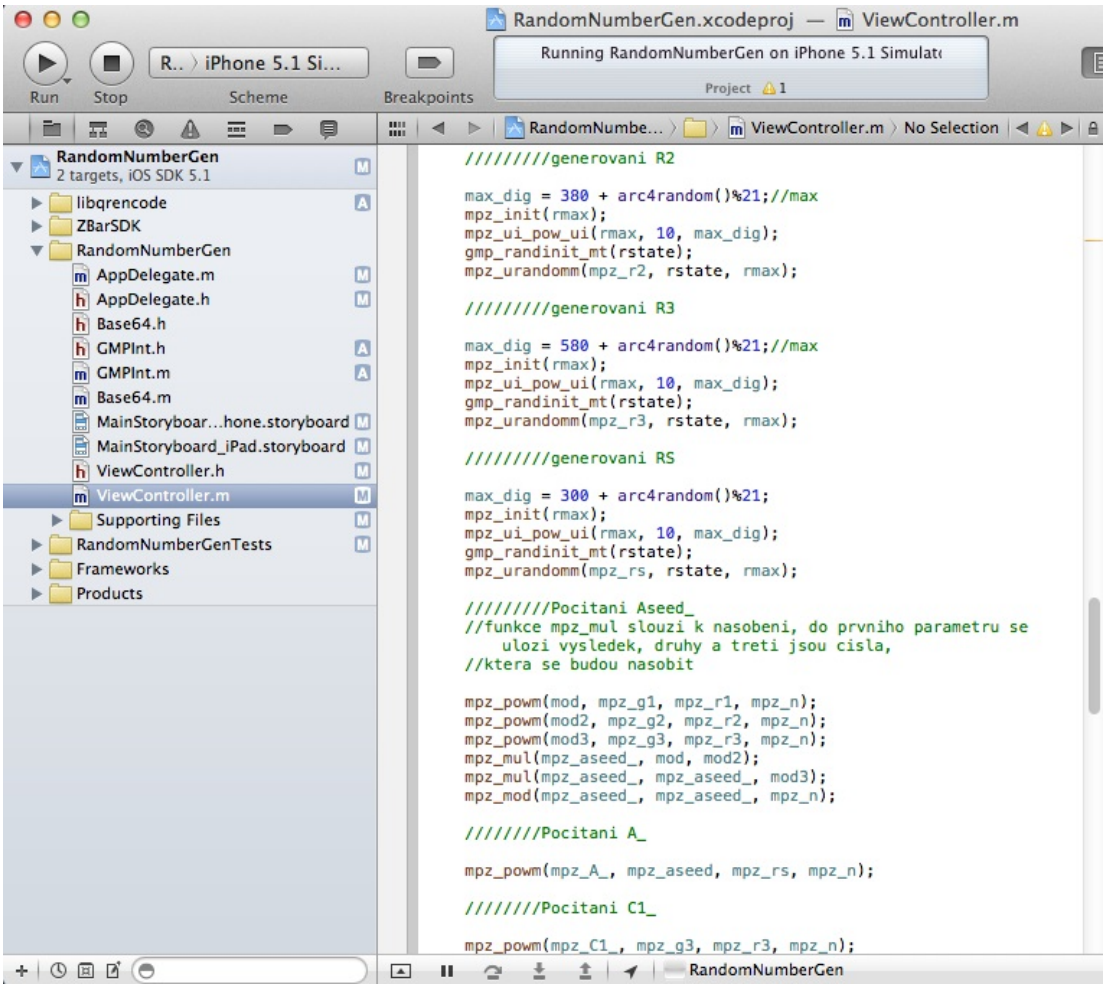
Nyní již můžeme přejít ke konkrétním výpočtům jednotlivých hodnot. Pro vygenerování čísla Ks využijeme funkci **mpz_urandomm(randNum, gmpRandState, rndBnd)**. Do proměnné *int max_dig* uložíme hodnotu, která bude odpovídat délce v bitech generovaného čísla. Funkce **gmp_randstate_t** se používá v konstrukci funce mpz_urandomm k vytvoření náhodného čísla pomocí knihovny GMP.

Pro výpočet hodnot, jako je A, C_1, $A\_$, $C_1\_$, $C_2\_$,se využívá funkce



Obr. 5.4: Plnění konstant a vytvoření IBAction ve třídě viewcontroller.m

mpz_powm(mpz_t rop, mpz_t base, mpz_t exp, mpz_t mod). Je to funkce, která provádí modulární mocnění. První parametr je proměnná, do které se výsledek uloží, druhý parametr je základ mocniny, třetí parametr je exponent mocniny a poslední parametr je modulo. Jak je vidět z definice funkce, všechny parametry musejí být proměnné typu mpz_t.



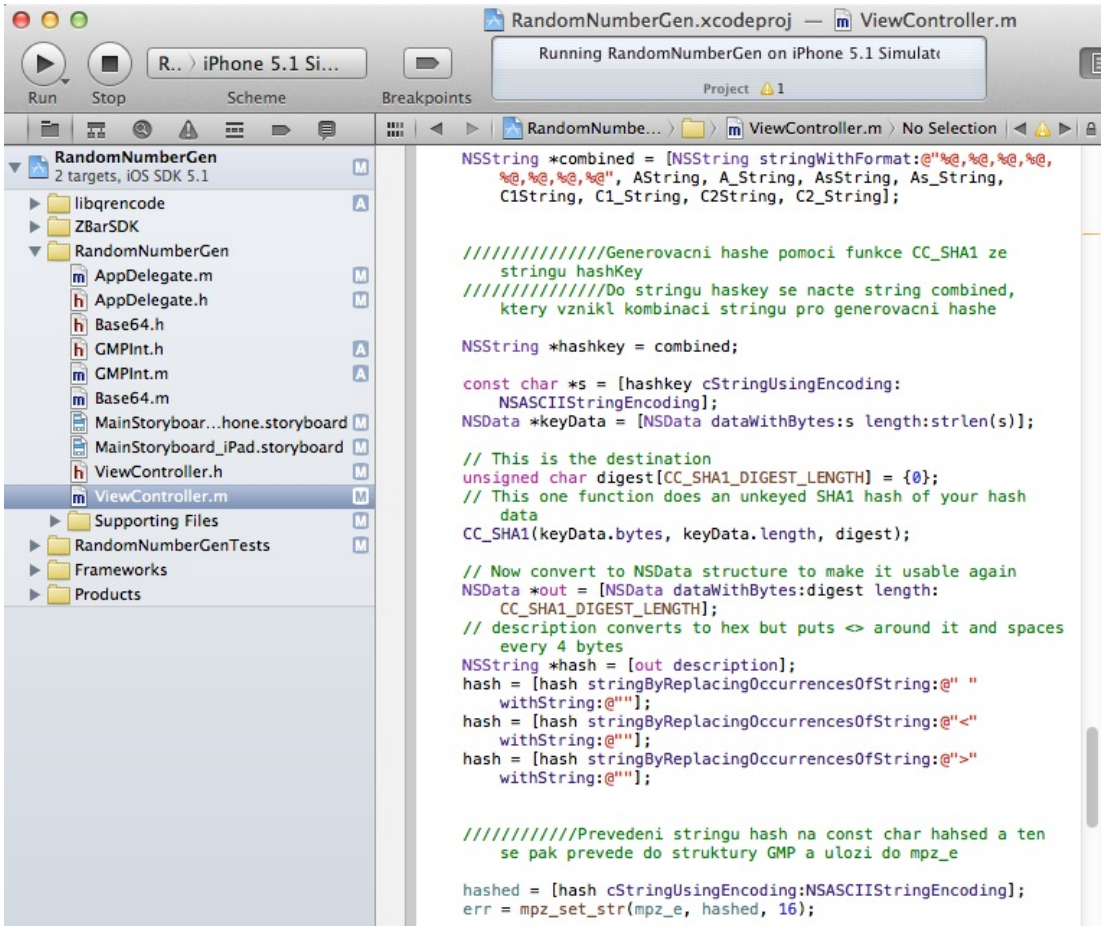
Obr. 5.5: Výpočty hodnot ve třídě viewcontroller.m

Generování hodnot r_1, r_2, r_3 a rs je skoro stejné jako generování hodnot Ks. Jediný rozdíl je v použití funkce **arc4rand()**, která ovlivňuje počet posledních bitů.

Výpočet hodnoty Aseed_ je trochu složitější. Je provedena třikrát funkce **mpz_powm()**. První, druhý a třetí výsledek se mezi sebou vynásobí pomocí funkce **mpz_mul(mpz_t rop, mpz_t p1, mpz_t p2)**. Funkce vynásobí druhý a třetí parametr a výsledek se uloží do prvního parametru funkce. Nakonec je použita funkce **mpz_mod()**, která udělá modulo celého výsledku.

Pro výpočet hashe je třeba všechny požadované hodnoty převést na stringy. K tomu využijeme wrapper GMPInt a jeho funkce **(GMPInt *)initWithValue:(mpz_t)val** a **(NSString *)stringValue**. První načte hodnotu ze struktury mpz_t a přiřadí ji své hodnotě GMPInt. Druhá funkce potom tuto hodnotu převede na stringový řetězec. Nakonec všechny potřebné hodnoty převedené na stringy

sloučíme do jednoho velkého řetězce, který předáme na vstup hashovací funkce. K hashování použijeme funkci **CC_SHA1**, která patří mezi nejrozšířenější.



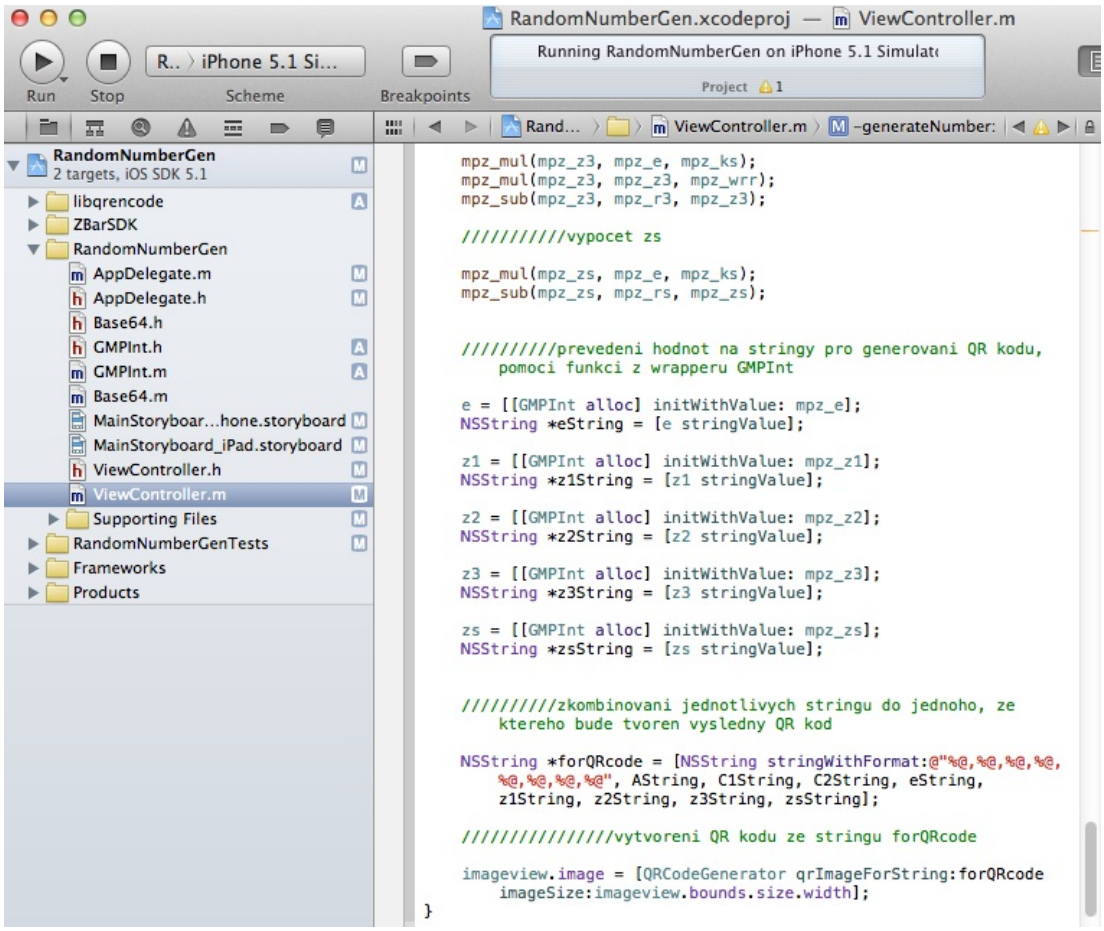
Obr. 5.6: Vytvoření hashe ve třídě viewcontroller.m

Nakonec výslednou hodnotu hashe musíme převést ze stringu na char, abychom ho mohli naalokovat do struktury mpz_t pomocí již známé funkce **mpz_set_str()**. Jediný rozdíl bude v zápisu parametru `base`, ten zvolíme 16. Tím dosáhneme převedu hexa hodnoty přímo na dekadickou viz. obr. 5.6.

Pro výpočty hodnot z_1, z_2, z_3 a zs se mimo již známou funkci **mpz_mul()** používá funkce **mpz_sub (MP_INT *difference, MP_INT *minuend, MP_INT *subtrahend)**.

Jakmile máme vypočítané všechny hodnoty protokolu, je třeba převést na stringy ty, které se budou používat pro generování QR kódu. Stejným postupem jako pro hash převedeme požadované hodnoty na stringy pomocí GMPInt wrapperu a nakonec z nich vytvoříme jeden velký string. Tento string poté předáme funkci, která do

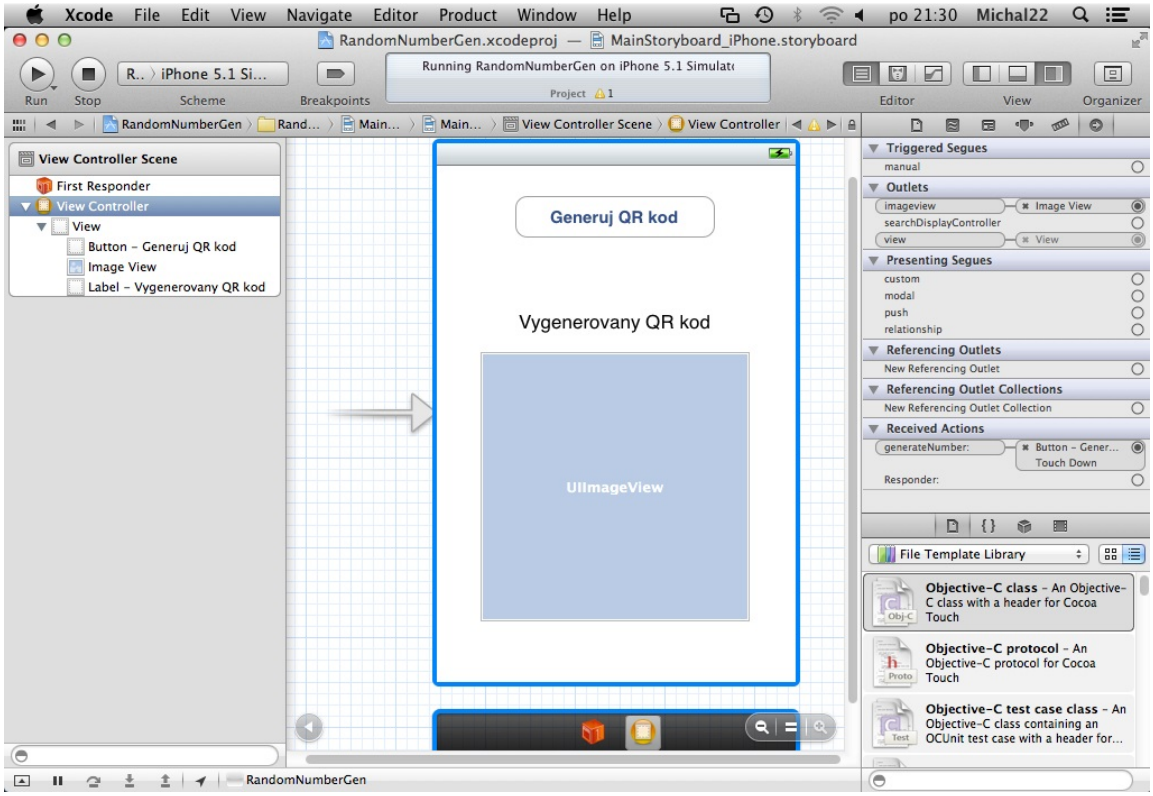
námi vytvořeného imageView vygeneruje výsledný QR kód.



Obr. 5.7: Vytvoření QR kódu ve třídě viewcontroller.m

Tímto jsme hotovi s programovou částí projektu. Nyní je třeba navrhnout grafické rozhraní a propojit vytvořené IBOutlety a IBAction s příslušnými funkcemi. V navigátoru se přepneme do položky **MainStoryboard_iPhone**. Tím se dostane do návrhu grafické podoby naší aplikace. Je tu předem připravené pozadí, které odpovídá přesně tomu, jak bude vypadat na přístroji. Z výběrového panelu na pravé straně dole vybereme vhodné objekty pro nás projekt. Na plochu přetáhneme jeden **Button**, jeden **Label** a jeden **UIImageView**. V pravém panelu ve vrchní části si můžeme přepínat mezi několika menu. Nás bude zajímat hlavně to úplně vpravo, které slouží k propojování akcí a objektů. V záložce **Outlets** tedy propojíme *imageview* s naším Image View na pracovní ploše táhnutím myší, a pak ještě v záložce **Received Actions** propojíme naši IBAction s tlačítkem na ploše. Tím je zajištěno, že po stisknutí tlačítka se provede tato akce. Pak nás bude zajímat ikona třetí z prava, kde se nastavují příslušné vlastnosti zvoleného objektu na pracovní ploše.

Tlačítku tedy můžeme nastavit barvu, název, umístění textu a mnoho dalších věcí. Podobná nastavení jsou k dispozici i pro label, kterému nastavíme příslušný text. Samotné prostřední nám nabízí pomocné kóty pro umístění jednotlivých objektů, aby byly například všechny stejně centrované nebo jinak zarovnané.

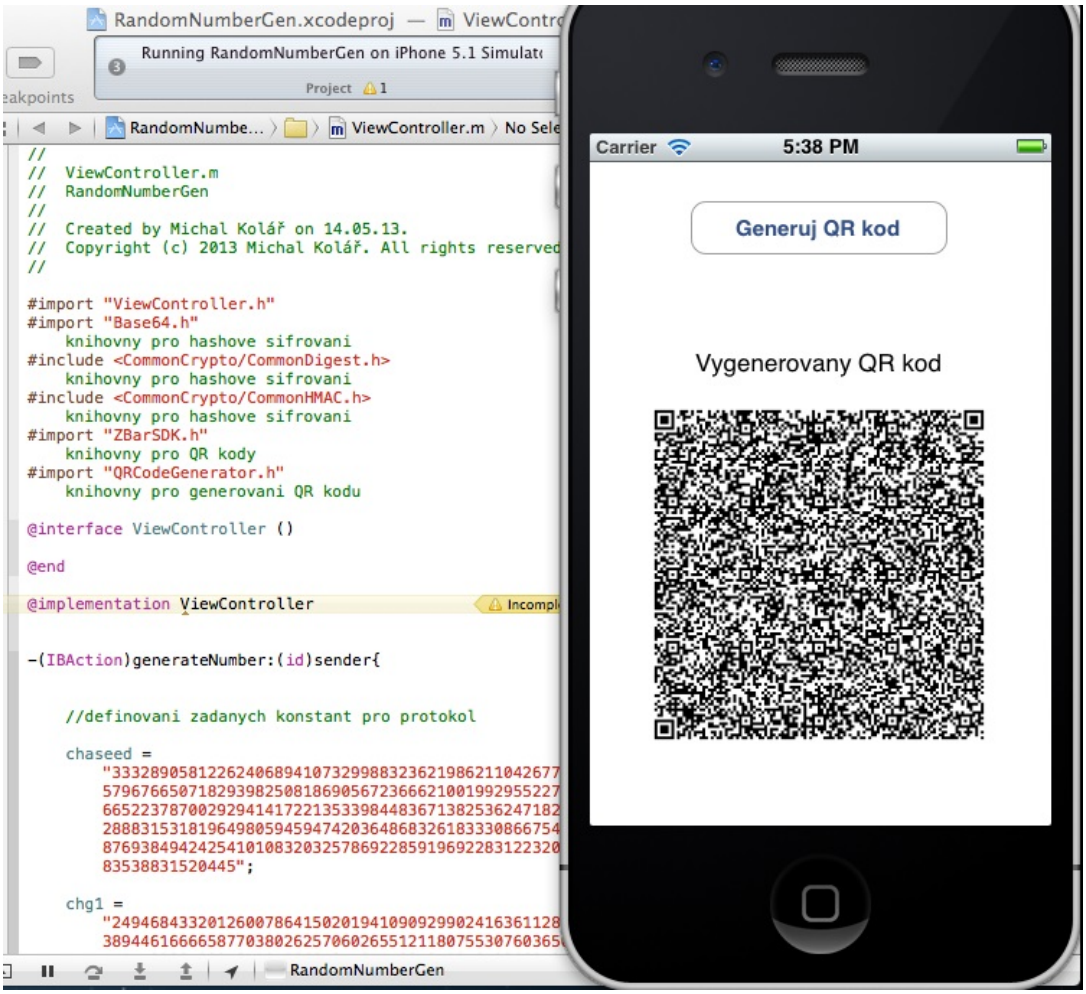


Obr. 5.8: Návrh grafického rozhraní aplikace

Pokud jsou všechna nastavení v pořádku a kód neobsahuje žádné chyby, je aplikace připravena na zkompilování a spuštění v iOS simulátoru. Spuštěný simulátor vypadá přesně jako skutečný iPhone a má i stejné ovládání (obr. 5.9). Po stisknutí tlačítka se v imageview objeví vygenerovaný QR kód, který je možné pomocí mnoha aplikací naskenovat.

5.3 Ověření výstupních hodnot

K ověření výstupních hodnot použijeme aplikaci **Wolfram Mathematica**. Výsledný QR kód naskenujeme například pomocí aplikace Barcodes, která je zdarma dostupná na iTunes ke stáhnutí. V naskenovaném QR kódu jsou jednotlivé hodnoty



Obr. 5.9: Zobrazení aplikace v simulátoru

odděleny čárkami. Ty, které potřebujeme si překopírujeme do prostředí Mathematica, stejně jako ty, které jsou pevně zadané a budou třeba k ověření.

Pravá část rovnic vychází ze zadaných a vypočítaných hodnot, které přenáší QR kód. Hodnoty levé strany rovnic si zobrazíme v pomocném labelu v programu, poté jejich hodnotu ověříme výpočtem v Mathematice. Po ověření odpovídaly všechny proměnné vypočtené pomocí hodnot z pravé strany rovnice těm, které jsem si zobrazil v pomocném labelu přímo v aplikaci. Protokol tedy pracuje přesně jak má.

Výpočet ověřovacích hodnot

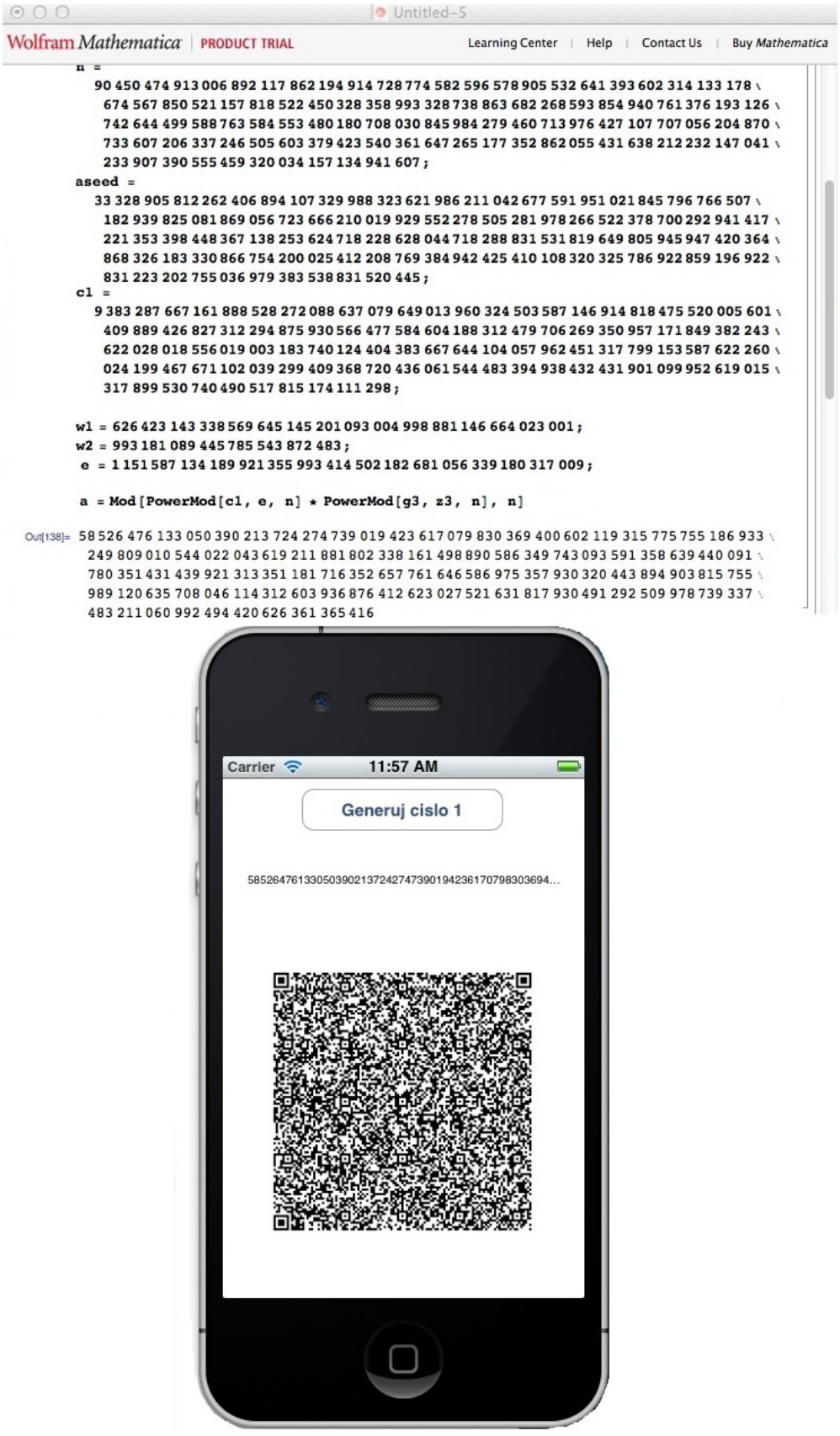
$Aseed\_ = A^e\, g_1{}^{z_1}\, g_2{}^{z_2}\, g_3{}^{z_3} \bmod n$

$A\_ = A^e\, Aseed\_{}^{zs} \bmod n$

$C_1\_ = C_1{}^e\, g_3{}^{z_3} \bmod n$

$C_2\_ = C_2{}^e g_3{}^{zs} \bmod n$

$e =? H(A, A\_, Aseed\_, C_1, C_2, C_1\_C_2\_)$



Obr. 5.10: Kontrolní výpočet hodnoty v Mathematica

6 ZÁVĚR

Systém iOS patří k nejbezpečnějším mobilním systémům. Shrnutím všech bezpečnostních technologií vzniká dobrý bezpečnostní celek. Apple dokázal zkombinovat velké množství bezpečnostních prvků a neomezil tím příliž uživatele nebo rychlost zařízení. Systém iOS od začátku patřil k nejrychlejšímu a i v současné době mi připadá ze všech konkurenčních systémů nejsvižnější.

Přehled všech probraných bezpečnostních technologií:

Secure Boot Chain	Zabezpečený bootovací proces, který ověřuje základní bezpečnostní složky.
System Software Personalization	Zabezpečení instalace nejnovějších aktualizací a softwaru.
App Code Signing	Zabezpečení aplikací pomocí podpisů a certifikátů
Runtime Process Security	Zabezpečení aplikací a jejich souborů za běhu přístroje.
File Data Protection	Kryptografické zabezpečení dat na všech úrovních.
Passcodes	Zabezpečení zařízení pomocí pinu a ochrana před útoky hrubou silou.
Classes	Ochrana souborového systému pomocí tříd s různým typem zabezpečení.
Keychain Data Protection	Zabezpečení přihlašovacích údajů a správa všech bezpečnostních klíčů.
Keybags	Rozdělení bezpečnostních klíčů, zabezpečení zálohy přes iTunes.

Tab. 6.1: Shrnutí bezpečnostních technologií

Zabezpečení sítě vychází především ze standardních síťových protokolů. Veškerá komunikace je plně šifrována nejmodernějšími metodami.

Tato práce slouží především k seznámení se s nejdůležitějšími bezpečnostními technologiemi systému iOS, k seznámení se a zorientování ve vývojovém prostředí Xcode, jazyku Objective C a frameworku Cocoa. Obsahuje také popis používaných knihovem pro kryptografické operace a jejich implementaci do systému i do vývojového prostředí Xcode.

Byl vytvořen program, který využívá funkcí knihovny GMP a balíčku ZbarSKD, který umožňuje tvorbu téměř neomezeně velkých QR kódů. Knihovnu GMP, i přes dva roky starou aktualizaci, se povedlo na systém nainstalovat s drobnými úpravami. Implementace balíčku ZbarSDK byla bezproblémová. Zadaný bezpečnostní protokol se podařilo správně implementovat a hodnoty přenesené QR kódem jsou správné, což

jsem ověřil pomocí výpočtů ověřovacích hodnot v programu Wolfram Mathematica. Serverová část, která měla ověřovat hodnoty přenesené QR kódem, nebyla zpracována v plném rozsahu, kvůli vyšší náročnosti klientské aplikace. Vedoucí práce s tím souhlasí. Protože výsledný QR kód přenáší několik obrovských čísel, je také velmi složitý. Je nutné ke skenování použít vhodný program, protože některé mají s tak velkými QR kódy problém. Doporučil bych program Barcodes, který využívá funkcí knihovny ZXing. Ta je podle mého úsudku nejlepší ke skenování a zpracovávání QR kódů. Naopak balíček ZBar SDK je nejvhodnější pro jejich generování.

LITERATURA

[1] CONWAY, J., HILLEGASS, A. *iOS Programing: The big nerd ranch guide*. 3. vydání. Indianapolis: Big Nerd Ranch Guides, 2012. 590 s. ISBN 0321821521

[2] Xcode 4 User Guide. [online]. 2012 [cit. 2012-09-19]. Dostupné z URL: <http://developer.apple.com/library/mac/#documentation/ToolsLanguages/Conceptual/Xcode4UserGuide/000-About_Xcode/about.html >

[3] IOS Security. [online]. 2012, poslední aktualizace 19. 05. 2012 [cit. 2012. 10. 09]. Dostupné z URL: <http://images.apple.com/ipad/business/docs/iOS_Security_May12.pdf>

[4] Security Overview. [online]. 2012, poslední aktualizace 19. 09. 2012 [cit. 2012. 11. 12]. Dostupné z URL: <http://developer.apple.com/library/mac/documentation/Security/Conceptual/Security_Overview/Security_Overview.pdf>

[5] MARK, Dave.,LAMARCHE, Jeff. *iPhone SDK*. 1. vydání. Brno: Computer Press, a.s., 2010. 480 s. ISBN 978-80-251-2820-6

[6] The GNU MP Bignum Library. [online]. 1991, poslední aktualizace 20. 05. 2013 [cit. 2013. 23. 05]. Dostupné z URL: <http://gmplib.org/>

[7] ZBar bar code reader. [online]. 2007, poslední aktualizace 15. 06. 2011 [cit. 2013. 23. 05]. Dostupné z URL: <http://zbar.sourceforge.net/>

[8] ZXing Multi-format 1D/2D barcode image processing library. [online]. 2007, poslední aktualizace 18. 05. 2013 [cit. 2013. 23. 05]. Dostupné z URL: <https://code.google.com/p/zxing/>

[9] Stack Overflow. [online]. 2005, poslední aktualizace 22. 05. 2013 [cit. 2013. 23. 05]. Dostupné z URL: <http://stackoverflow.com/>

[10] QR kódy a QR code technologie. [online]. 2009, poslední aktualizace 12. 03. 2013 [cit. 2013. 23. 05]. Dostupné z URL: <http://www.qr-kody.cz/>

[11] QRcode.com. [online]. 2008, poslední aktualizace 22. 04. 2013 [cit. 2013. 23. 05]. Dostupné z URL: <http://www.qrcode.com/en/>

[12] KLÍMA, Vlastimil. Hašovací funkce, principy, příklady a kolize [online]. 2005, poslední aktualizace 19. 03. 2005 [cit. 2013. 23. 05]. Dostupné z URL: <http://cryptography.hyperlink.cz/2005/cryptofest_2005.htm>

[13] NEPŠINSKÝ, Roman. MD5 algorithm overview. [online]. 2008, poslední aktualizace 17. 10. 2002 [cit. 2013. 23. 05]. Dostupné z URL: <http://www.nepsin.com/mem/projects/md5.htm>

[14] The Xcode Guide. [online]. 2013, poslední aktualizace 23. 05. 2013 [cit. 2013. 23. 05]. Dostupné z URL: <http://xcodeguide.com/>

[15] Xcode Quick Start Guide. [online]. 2010, poslední aktualizace 05. 07. 2011 [cit. 2013. 23. 05]. Dostupné z URL: <http://developer.apple.com/library/mac/#documentation/IDEs/Conceptual/xcode_quick_start/000-About_Xcode/introduction.html>