Universiteti i Evropës Juglindore

Fakulteti i Shkencave dhe Teknologjive Bashkëkohore

Programi i magjistraturës:
Aplikimi i teknologjisë informative dhe komunikimit në procesin arsimor

TEZA E MAGJISTRATURËS

Dizajnimi i aplikacioneve të sigurta në platformën android dhe analiza e veglave për siguri

Kandidati:
Altrim SHABANI

Mentori:
Prof. dr. Artan LUMA

Tetor 2019

Deklarata e autorësisë

Unë, Altrim SHABANI, deklaroj se kjo tezë e magjistraturës e titulluar, "Dizajnimi i aplikacioneve të sigurta në platformën android dhe analiza e veglave për siguri" dhe puna e paraqitur në të janë të mija. Unë konfirmoj që:

- Kjo punë është bërë tërësisht gjatë periudhës së kandidaturës sime për një tezë magjistrature që kamë bërë një hulumtim shkencor në këtë Universitet.
- Kur ndonjëherë më parë nuk është paraqitur ndonjë pjesë e kësaj teze të magjistraturës në ndonjë punim shkencor, tezë të magjistraturës, temë diplome ose ndonjë kualifikim tjetër në këtë universitet ose në ndonjë institucion tjetër, kjo është shumë e vërtetë.
- Kur kam konsultuar literaturën shkencore të botuar të të tjerëve, gjithmon e kam referuar sipas standarteve për referenca në mënyrë të qartë.
- Kur kam cituar nga punimet shkencore të të tjerëve, gjithmonë e kam dhënë burrimin. Me përjashtim të citimeve të tilla, kjo tezë e magjistraturës është tërësisht puna ime.
- Ku teza e magjistraturës bazohet në punën e bërë nga unë bashkë me të tjerët, unë e kam bërë të qartë saktësisht atë që është bërë nga të tjerët dhe atë që kam kontribuar unë.

Nënshkruar: _____

Data: _____

Komisioni i vlerësimit

Kjo tezë e magjistraturës është kontrolluar nga një komision i vlerësimit i krijuar nga Këshilli Mësimor Shkencor i Fakultetit të Shkencave dhe Teknologjive Bashkëkohore, si vijon:

Prof. dr. Azir ALIU, kryetar

Prof. dr. Artan LUMA, antarë dhe mentor

Prof. dr. Halil SNOPÇE, antarë

Abstrakti

Përdorimi i telefonave të mençur në gjithë botën është në rritje shumë të madhe, duke u bërë ata shumë të rëndësishëm dhe duke zë një vend të veçantë për çdo njërin nga ne. Me përodrimin e tyre, duke përdorë aplikacione të ndryshme përveç që përfitojmë lehtësime të mëdha për kryerjen e procesëve të ndryshme, po ashtu mundësia që të keqpërdoren të dhënat konfidenciale apo personale është shumë e madhe. Për këtë, siguria është një nga faktorët kyç në dizajnimin e një aplikacioni të përshtatshëm për telefonat e mençur. Ekzistojnë sisteme operative të ndryshme të dizajnuara për telefona të mençur ose për pajisje mobile të ndjeshme ndaj prekjeve, i till është Android i cili është një nga më të përdorurit. Platforma Android është e hapur dhe kërkon një arkitekturë të fortë sigurie.

Synimi apo objektivi i kësaj teme është përpilimi dhe përshkrimi i një cikli të sigurtë të jetës së zhvillimit të softuerit për platformën Android. Tema përmban gjithashtu disa seksione të fokusuara në faktorin e sigurisë nga këndvështrime të ndryshme e në veçanti nga aspekti i algoritmeve kriptografike, me theks në proceset e modelimit të kërcënimeve, si dhe analiza statike dhe dinamike e kodit. Jemi përqendruar më shume në rëndësinë e përdorimit të algoritmeve kriptografike për të fshehur dhe për të siguruar të dhënat e përdoruesve që do të ruhen apo transportohen. Fillimisht do mbulojmë bazat e algoritmeve kriptografike dhe si ato aplikohen për ne në kontekstin e zhvillimit të aplikacioneve dhe pastaj do të shikojmë mekanizmat e ndryshëm të ruajtjes së të dhënave në platformën Android. Është shumë e rëndësishme për t'u mbajtur mend që kur nuk duhet të përpiqeni të shkruani rutinat tuaja të algoritmeve kriptografike nëse nuk jeni të njoftuar me terma e algoritmeve kriptografike. Nëse përpiqeni të bëni këtë, do përfundoni me aplikacione të cenueshme për pajisjet mobile.

Përmes algoritmeve kriptografike, mund të i bëni të dhënat tuaja sensitive të palexueshme për përdoruesit e paautorizuar. Duke përdorë algoritme të ndryshme kriptografike, me qëllim të mbrojtjes se të dhënave është zhvilluar një aplikacion në platformën Android, i cili do të përshkruhet në këtë temë se si është zhvilluar. Veglat e disponueshme për këto procese më pas janë krahasuar duke përdorë matje të ndryshme.

Falënderim

Unë do të doja që të falënderoj familjen time për përkrahjen e pandërprerë gjatë këti hulumtimi shkencor dhe mentorin me antarët e komisionit për ndihmën e tyre gjatë konsultimeve, sugjerimeve dhe kohën e kaluar duke më këshilluar për këtë punë shkencore që e kanë ndarë nga jeta e tyre për mua.

Përmbajtja

Lista e figurave

Lista e tablave

Fjalor i termeve

API	Application Programming Interface
OWASP	Open Web Application Security Project
OHA	Open Handset Alliance
ART	Android Runtime
DEX	Dalvik Exectuables
NDK	Native Development Kit
OS	Operating System
SMS	Short Message Service
UI	User Interface
UX	User Experience
AS	Android Studio
IDE	Integrated Development Environment
APK	Android Package Kit
HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
ASLR	Address Space Layout Randomization
XML	Extensible Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
SSL	Secure Sockets Layer
AES	Advanced Encryption Standard
RSA	Rivest Shamir Adleman
IPC	Interprocess Communication

iOS	iPhone Operating System
MASVS	Mobile Application Security Verification Standard
MSTG	Mobile Security Testing Guide
SMDG	Secure Mobile Development Giudelines
JCA	Java Cryptography Architecture
URL	Uniform Resource Locator
EC	Elliptic Curve
CBC	Cipher Block Chaining
CTR	Counter
GCM	Galois/Counter Mode
IV	Initialization Vector
HMAC	Hash Message Authentication Code
SHA	Secure Hash Algorithm
CSP	Cryptographic Service Provider
CSPRNG	Cryptographically Secure Pseudo Random Number Generator
DSA	Digital Signature Algorithm
MAC	Message Authentication Code
PBE	Password Based Encryption
HD	Diffie Hellman
ECDH	Elliptic Curve Diffie Hellman
CA	Certification Authority
CRL	Certificate Revocation List
JSSE	Java Secure Socket Extension
RI	Reference Implementation
RIPEMD	RACE Integrity Primitives Evaluation Message Digest
CMAC	Cipher based Message Authentication Code
LDAP	Lightweight Directory Access Protocol
TLS	Transport Layer Security
BC	Bouncy Castle
SDLC	Software Development Life Cycle
UAT	User Acceptance Testing

QA	Quality Assurance
STRIDE	Spoofing Tampering Repudiation Information DOS Elevantion
DREAD	Damage Reproducibility
SA	Static Analysis
UID	Unique Identifier
JAR	Java Archive
BPMN	Business Process Model Notation
TMT	Threat Modeling Tool
CWE	Common Weaknesses Enumeration
PMD	Programming Mistake Detector
MobSF	Mobile Security Framework

Kapitulli 1

Hyrje

Statistikat përshkruajnë se numri i përdoruesve të telefonave të mençur në gjithë botën është në rritje duke u bazuar që në vitin 2014 ishin 1.57 bilion përdorues dhe tani ne vitin 2017 janë 2.32 bilion [4]. Duke përdorë aplikacionet në telefon, ne shpesh me vetëdije dhe pa vetëdije ruajmë ose dërgojmë të dhëna, të cilat mund të jenë personale ose konfidenciale. Këto të dhëna mund të vjedhën dhe të keqpërdorën, dhe që duhet të ketë një faktor sigurie për të parandaluar veprime të tilla.

Platforma Android në vetvete ofron disa shtresa të mbrojtjes për të shmangur këto ngjarje. Megjithatë, kur bëhet fjalë për vetë aplikacionet, siguria është një gjë që duhet të ofrohet dhe të implementohet nga krijuesit e aplikacioneve. Për ta bërë këtë, siguria duhet të merret ne konsideratë që nga fillimi, gjë që zakonisht sigurohet duke ndjekur një cikël të sigurtë të zhvillimit të softuerit.

Qëllimi i kësaj teme është dizajnimi i një cikli të jetës të tillë, që si target primar do ketë platformën Android, dhe për të gjetur vegla që do të ndihmonin për të garantuar sigurinë e një aplikacioni, që u krijua duke përdorur ciklin e jetës së propozuar për të zhvilluar një softuer të sigurt.

Fillimisht, një pasqyrë të shkurtër të arkitekturës Android, Android API dhe

mundësitë e zhvillimit janë paraqitur. Më pas, bazat më themelore të sigurisë në Android janë të listuara e në veçanti algoritmet kriptografike së bashku me detajet lidhur me zhvillimin e aplikacionit në platformën Android duke përdorë algoritme kriptografike, si dhe *Open Web Application Security Project* (*OWASP*) është prezantuar me projekte të ndryshme të orientuara drejtë sigurisë mobile (celulare).

Në vazhdimësi shqyrtohen ciklet e jetës ekzistuese të zhvillimit të softuerit dhe prezantohen proceset e modelimit të kërcënimeve, analiza statike dhe dinamike e kodit, si dhe nënshkrimi dhe publikimi i një aplikacioni. Kapitulli më radhë harton një cikël të jetës së sigurtë të zhvillimit të softuerit me target në platformën Android. Kapitulli i fundit, krahason dhe vlerëson kandidatët më të mirë për përdorim për secilën nga aktivitet kryesore në ciklin e jetës së sigurt të zhvillimit të softuerit, përkatësisht në modelimin e kërcënimeve, analizën e kodit statik dhe dinamik.

1.1 Lënda e hulumtimit

Përdorimi i telefonave të mençur në gjithë botën është në rritje shumë të madhe, duke u bërë ata shumë të rëndësishëm dhe duke zë një vend të veçantë për çdo njërin nga ne. Me përodrimin e tyre, duke përdorë aplikacione të ndryshme përveç që përfitojmë lehtësime të mëdha për kryerjen e procesëve të ndryshme, po ashtu mundësia që të keqpërdoren të dhënat konfidenciale apo personale është shumë e madhe. Për këtë, siguria është një nga faktorët kyç në dizajnimin e një aplikacioni të përshtatshëm për telefonat e mençur.

Ekzistojnë sisteme operative të ndryshme të dizajnuara për telefona të mençur ose për pajisje mobile të ndjeshme ndaj prekjeve, i till është Android i cili është një nga më të përdorurit. Platforma Android është e hapur dhe kërkon një arkitekturë të fortë sigurie.

Synimi apo objektivi i kësaj teme është përpilimi dhe përshkrimi i një cikli të sigurtë të jetës së zhvillimit të softuerit për platformën Android. Tema do të përmban gjithashtu disa seksione të fokusuara në faktorin e sigurisë nga këndvështrime të ndryshme e në veçanti në algoritmet kriptografike, me theks në proceset e modelimit të kërcënimeve, si dhe analiza statike dhe dinamike e kodit. Do të përqendrohemi më shumë në rëndësinë e përdorimit të algoritmeve kriptografike për të fshehur dhe për të siguruar të dhënat e përdoruesve që do të ruhen apo transportohen.

Fillimisht do mbulojmë bazat e algoritmeve kriptografike dhe si ato aplikohen në kontekstin e zhvillimit të aplikacioneve dhe pastaj do të shikojmë mekanizmat e ndryshëm të ruajtjes së të dhënave në platformën Android. Është shumë e rëndësishme për t'u mbajtur mend që kurr nuk duhet të përpiqeni të shkruani rutinat tuaja kriptografike nëse nuk jeni të njoftuar me terma kriptografik. Nëse përpiqeni të bëni këtë, do përfundoni me aplikacione të cenueshme për pajisjet mobile.

Përmes algoritmeve kriptografike, mund të i bëjmë të dhënat tuaja sensitive të palexueshme për përdoruesit e paautorizuar. Duke përdorë algoritme të ndryshme kriptografike, me qëllim të mbrojtjes se të dhënave do të dizajojmë një aplikacion në platformën Android, i cili do të përshkruhet në këtë temë se si është dizajnuar.

Veglat e disponueshme për këto procese më pas do ti krahasojmë duke përdorë matje të ndryshme.

1.2 Qëllimet e hulumtimit

Kjo temë ka dy qëllime kryesore, e para është për të dizajnuar një jetëgjatësi të sigurt të zhvillimit të softuerit të targetuar në platformën Android dhe e dyta për të gjetur vegla që do të mbështesin këtë jetëgjatësi. Para dizajnimit të jetëgjatësisë, ne kalojmë nëpër disa tema të fokusuara në platformën Android dhe sigurinë e saj

nga pikëpamje të ndryshme. *Open Web Application Security Project* ($OWASP$) dhe projektet e saj me orientim mobile janë futur për të dhënë një pasqyrë të kërcënimeve të përbashkëta në platformën Android dhe opsionet se si të garantojnë më tej sigurinë e një aplikacioni të zhvilluar.
Qëllimi i kësaj teme është dizajnimi i një cikli të jetës të tillë, që si target primar do ketë platformën Android dhe për të gjetur vegla që do të ndihmonin për të garantuar sigurinë e një aplikacioni që do të krijohet duke përdorur ciklin e jetës së propozuar për të zhvilluar një softuer të sigurt.

Fillimisht, do të japim një pasqyrë të shkurtër të arkitekturës Android, Android API dhe mundësitë e dizajnimit të një aplikacionit. Më pas, do ti ilustrijmë bazat më themelore të sigurisë në Android në veçanti algoritmet kriptografike së bashku me detajet lidhur me dizajnimin e aplikacionit në platformën Android duke përdorë algoritme kriptografike, si dhe *Open Web Application Security Project* ($OWASP$).

Po ashtu do të shqyrtohen ciklet e jetës ekzistuese të dizajnimit të softuerit dhe prezantohen proceset e modelimit të kërcënimeve, analiza statike dhe dinamike e kodit, si dhe nënshkrimi digjital dhe publikimi i një aplikacioni.

Modelimi i kërcënimeve është një qasje për të analizuar sigurinë e një aplikacioni. Ndihmon në identifikim, përcaktimin dhe adresimin e rreziqeve të sigurisë që lidhen me një aplikacion. Përfshirja e modelimit të kërcënimeve në $SDLC$ mund të ndihmojë për të siguruar që aplikacionet janë duke u zhvilluar me sigurinë e ndërtuar që nga fillimi.

Kritik për identifikimin e kërcënimeve është përdorimi i një metodologjie kategorizimi të kërcënimeve të tilla si $STRIDE$ dhe Korniza e Sigurisë së Aplikacionit, ndërsa përcaktimi i rrezikut të sigurisë mund të vlerësohet duke përdorur një model rreziku siç është $DREAD$.
Në trend tani janë më shumë telefonat e mençur, ku edhe përdorimi dhe avancimimi i tyre është në rritje çdo ditë e më shumë. Rëndësi të veçantë kanë marrë

edhe aplikacionet të cilat dizajnohen dhe që janë të përshtatshme për telefonat e mençur, dhe që janë të përdorshme nga ne si shfrytëzues. Por, faktorë kyç në dizajnimin e një aplikacioni të tillë është siguria.

1.3 Hipotezat

1) Dizajnimi i një jetëgjatësi të sigurt të zhvillimit të softuerit është me rëndësi primare në funksionimin e platformën Android.

2) Aplikacionet e siguris ndikojnë pozitivisht në jetëgjatësi.

3) Siguria e aplikacioneve të ndryshme që shfrytëzohen për telefonat e mençur në shumicën e rasteve nuk janë implementuar në mënyrën e duhur.

4) Të dhënat tona personale që i shfrytëzojmë në telefonat e mençur nuk janë në sasi të mjaftueshme të sigurtë.

5) Shfrytëzimi i veglave që na ofrojnë mundësin e mbrojtjes së të dhënave në telefonat e mençur gjatë dizajnimit të aplikacioneve nuk janë në nivel të duhur.

1.4 Metodologjia e hulumtimit

Duke pasë parasysh që qëllimi i kësaj teme është se si të dizajnojm një jetëgjatësi të sigurt të dizajnimit të softuerit të targetuar në platformën Android dhe si të gjejmë vegla që do të mbështesin këtë jetëgjatësi, atëherë ne kemi përdorë një kombinim të hulumtimit dytësor dhe analizës cilësore të të dhënave.

Aktivitetet kryesore të përfshira në proceset që përdoren në një jetëgjatësi të dizajnimit të softuerit janë modelimi i kërcënimeve, analiza statike dhe dinamike e kodit, ku janë përcaktuar matjet për të krahasuar veglat në dispozicion, do të krahasohen ato më pas dhe do ti vlerësojmë alternativat më të mira. Duke përdorur

këtë metodologji arritëm të gjejmë vegla për secilën nga këto aktivitete.

Do të hulumtojmë me kujdes për përcaktim të matjeve, krahasim dhe në fund vlerësimi do të bëhet po ashtu me kujdes të madhë.

Analizimi i metodave të kërkimit do të ndihmon në minimizim e gabimeve gjatë mbledhjes, interpretimit ose burimit të të dhënave.

Kritik për identifikimin e kërcënimeve është përdorimi i një metodologjie kategorizimi të kërcënimeve të tilla si $STRIDE$ dhe Korniza e Sigurisë së Aplikacionit, ndërsa përcaktimi i rrezikut të sigurisë mund të vlerësohet duke përdorur një model rreziku siç është $DREAD$.

$STRIDE$ është një metodologji e kategorizimit të kërcënimeve, duke identifikuar kërcënimet nga perspektiva e sulmuesve, që rrjedh nga një akronim i gjashtë kategorive të mëposhtme:

- *Spoofing* (me tallje).
- *Tampering* (ngatërrime).
- *Repudiation* (mohim).
- *Information Disclosure* (shpalosja e informacionit).
- *Denial of Service* (mohimi i shërbimit).
- *Elevation of Privilege* (ngritja e privilegjit).

Korniza e Sigurisë së Aplikacionit është kërcënim për kategorizimin e metodologjisë, duke identifikuar kërcënimet nga perspektiva e mbrojtësve. Ajo përcakton tetë lloje kërcënimi:

- Autentifikimi.

- Autorizimi.
- Menaxhimi i konfigurimit.
- Mbrojtja e të dhënave gjatë ruajtjes dhe transaksioneve.
- Validimi i të dhënave dhe validimi i parametrave.
- Menaxhimi i gabimeve dhe menaxhimi i përjashtimeve.
- Menaxhimi i përdoruesve dhe sesioneve.
- Auditimi dhe prerjet.

$DREAD$ është një model i rangut të kërcënimit me rrezik të krijuar nga $Microsoft$, i quajtur nga faktorët e rrezikut që përcakton. Faktorizimi i rrezikut lejon caktimin e vlerave tek faktorët e ndryshëm të ndikimit të një kërcënimi. Për të përcaktuar renditjen e një kërcënimi, analisti i kërcënimit duhet t'i përgjigjet një pyetjeje për secilin faktor rreziku:

- *Damage Potential* (potenciali i dëmtimit).
- *Reproducibility* (riprodhueshmëria).
- *Exploitability* (përdorshmëria).
- *Affected Users* (përdoruesit e prekur).
- *Discoverability* (zbulueshmëria).

Rezultati i përgjithshëm i $DREAD$ llogaritet duke caktuar një vlerë prej 1 deri 10 në secilën pyetje dhe duke pjestu shumën e tyre me 5. Sa më i lartë rezultati, aq më të larta janë rreziqet e kërcënimit.

Procesi i modelimit të kërcënimeve mund të dekompozohet në tre hapa:

- Dekompozo aplikacionin.

- Përcaktoni kërcënimet.
- Përcaktoni kundërmasat dhe zbutjen.

Gjatë dizajnimit të aplikacionit duhet pas kujdes për një jetëgjatësi të sigurtë të tij. Megjithatë, kjo nuk mund të arrihet pa përdorimin e veglave të ndryshme, të cilat e mbështesin këtë jetëgjatësi.

Një vegël programimi ose një vegël për zhvillimin e softuerit është një program kompjuterik i përdorur nga zhvilluesit për të krijuar, rregulluar (*debuguar*), mbajtur ose ndryshe të mbështetur procesin e zhvillimit. Për të vlerësuar alternativën ose alternativat më të mira, duhet të përcaktuar matje të veglave dhe duhet krahasuar ato vegla më pastaj.

Lista e proceseve të përdorura në një jetëgjatësi të zhvillimit të softuerit në platformën Android përfshin tri aktivitete kryesore, përkatësisht modelimin e kërcënimeve, analizën statike dhe dinamike të kodit.

Matjet për modelimin e kërcënimeve do të krijohen duke identifikuar faktorët kryesorë në procesin e modelimit të kërcënimeve, të cilat janë të renditura sipas rëndësisë së tyre dhe që do të përdoren për krahasimin dhe vlerësimin e veglave për modelimin e kërcënimeve.

Ndryshe nga veglat e modelimit të kërcënimeve, numri i velgave të analizës statike të kodit është i madh. Për shkak të këtij fakti, krahasimi do të bëhet në dy xhiro, së pari ndarjen e veglave që nuk kanë gjasa të jenë të dobishme për qëllimet e SA në platformën Android, e dyta duke krahasuar rezultatet e testimit në një projekt me kod të hapur. Projekti quhet $InsecureBankV2$ i krijuar nga *Sinesh Shetty*, i cili është një aplikacion i dobët Android që është bërë për entuziastët e sigurisë dhe zhvilluesit për të mësuar pasiguritë Android duke testuar këtë aplikacion. Aktualisht përmban 24 dobësi.

Krahasimi i veglave të analizës dinamike të kodit do të bëhet përsëri në projektin $InsecureBankV2$.

1.5 Rëndësia e punimit

Rëndësia e kësaj teme do të jetë dizajnimi dhe përshkrimi i një cikli të sigurtë të zhvillimit të aplikacioneve për platformën Android.

Fokus në faktorët e sigurisë nga këndvështrime të ndryshme e në veçanti në algoritmet kriptografike, në proceset e modelimit të kërcënimeve, si dhe analiza statike dhe dinamike të kodit.

Përqendrim shumë i rëndësishëm në shfrytëzimin e algoritmeve kriptografike për të fshehur dhe për të siguruar të dhënat e përdoruesve që do të ruhen apo transportohen.

Shfrytëzimi i algoritmeve kriptografike dhe si ato ti aplikojmë në kontekstin e zhvillimit të aplikacioneve dhe pastaj do të shohim mekanizmat e ndryshëm të ruajtjes së të dhënave në platformën Android.

Përmes algoritmeve kriptografike, mund ti bëjmë të dhënat sensitive të pakuptimt për përdoruesit e paautorizuar.

Veglat të cilat do të shfrytëzohen në këto procese, do të krahasohen duke bërë matje të ndryshme.

1.6 Përfundimi

Do të listojmë proceset e përbashkëta të përfshira në një jetëgjatësi të zhvillimit të softuerit dhe përshkruajmë procesin e nënshkrimit digjital dhe publikimit të

aplikacionit. Bazuar në rishikimin e cikleve ekzistuese të jetëgjatësisë, të cilat na japin një kuptim më të mirë se çfarë duhet bërë dhe kur duhet të bëhet, në dizajnimin e jetëgjatësis të re të zhvillimit të softuerit të sigurtë, që është me target në platformën Android dhe modelin e shkathët të zhvillimit të softuerit.

Çdo fazë e jetëgjatësisë të dizajnuar justifikon ose përmban këto detyra dhe i cakton këto detyra personave që punojnë në një projekt. Për çdo fazë gjithashtu krijojmë një diagram që përfaqëson fazën dhe aktivitetet e tij, duke përdorur shënimin e *Business Process Model Notation*.

Lista e proceseve të përdorura në një jetëgjatësi të zhvillimit të softuerit përfshin tri aktivitete kryesore, përkatësisht modelimin e kërcënimeve, analizën statike dhe dinamike të kodit. Për secilën nga këto aktivitete, përcaktojmë matjet për të krahasuar veglat në dispozicion, krahasojmë ato dhe vlerësojmë alternativat më të mira. Duke përdorur këtë qasje arrim të gjejmë vegla për secilën nga këto aktivitete.

Puna shtesë në këtë fushë mund të jetë më specifike dhe të përcaktojë këtë proces në platformën Android. Kjo mund të përfshijë krijimin e një modeli specifik të platformës Android për *Microsoft Threat Modeling Tool 2016*. Pastaj, mund të krijohet një analizë më e thellë e përcaktimit të kërkesave dhe kontrolleve të sigurisë, së bashku me krijimin e një udhëzuesi të testimit për të verifikuar këto kërkesa, në mënyrë të ngjashme me ato të ofruara nga *OWASP*.

Më abicioz mund të jetë krijimi i një vegle statike për analizën e kodeve specifike të Androidit të orientuar në sigurinë, pasi opsionet në këtë fushë janë shumë të kufizuara.

Kapitulli 2

Shqyrtimi i literaturës

Pas burimeve njerëzore, informacioni është aseti më i rëndësishëm i një organizate. Të gjitha përpjekjet për të mbrojtur sistemet dhe rrjetët, përpiqen të arrijnë disponueshmërinë e të dhënave, integritetin dhe konfidencialitetin si rezultate. Gjithashtu, siç e dim, asnjë kontroll i sigurisë së infrastrukturës nuk është 100% efektiv. Në një model të shtresuar të sigurisë, shpesh është e nevojshme të implementohet një kontroll përfundimtar i parandalimit i mbështjellë rreth informacionit të ndjeshëm dhe që është i kriptuar.

Kriptimi nuk është një zgjidhje për sigurinë. Kjo nuk do të zgjidhë të gjitha çështjet e sigurisë tuaj me bazë të të dhënave. Përkundrazi, është thjeshtë një kontroll në mes shumë të tjerëve.

Kriptografia është një shkencë që implementon matematikën dhe logjikën komplekse për të dizajnuar metoda të forta të algoritmeve për kriptimi. Arritja e kriptimit të fortë, fshehja e kuptimit të të dhënave, gjithashtu kërkon hapa intuitivë që lejojnë aplikim kreativ të metodave të njohura ose të reja. Pra, kriptografia është gjithashtu një art.

Qëllimi i kësaj teme është se si të dizajnojm një jetëgjatësi të sigurt të zhvillimit të softuerit të targetuar në platformën Android dhe si të gjejmë vegla që do

të mbështesin këtë jetëgjatësi.

Në mes të proceseve që përdoren në një jetëgjatësi të zhvillimit të softuerit, përfshihen modelimi i kërcënimeve, analiza statike dhe dinamike e kodit si aktivitete kryesore. Duke përcaktuar matjet për të krahasuar veglat në dispozicion, duke krahasuar ato, ne po ashtu vlerësojmë alternativat më të mira.

Seksioni i modelimit të kërcënimeve përmban një përshkrim bazë të asaj që është dhe si zakonisht bëhet. Çfarë mbetet të bëhet në këtë fushë më tej është që të jetë më specifike dhe të përcaktojë këtë proces në platformën Android, duke krijuar një model specifik të platformës Android për *Microsoft Threat Modeling Tool 2016*.

Pastaj, mund të krijohet një udhëzues i testimit për të verifikuar kërkesat e përcaktuara të sigurisë, sikurse ato të ofruara nga $OWASP$. Duke pas parasysh që opsionet lidhur me veglat statike për analizën e kodeve specifike të Androidit të orientuar në sigurinë janë shumë të kufizuara, krijimi i një vegle statike në këtë fushë do të ishte shumë ambicioz.

2.1 Android

Android është një *Open Source* (burim i hapur), i një sistem operativ i bazuar në *Linux* i krijuar për një koleksion të gjerë të pajisjeve, që përfshin *middleware* dhe aplikacionet kyçe [5][1]. Fillimisht, ajo u zhvillua nga *Android Inc.*, e cila u ble nga *Google* në vitin 2005. Versioni i parë i Android ishte lëshuar në vitin 2007, së bashku me themelimin e *Open Handset Alliance* (OHA), e cila ishte një partneritet i kompanive të përkushtuara për avancimin e standardeve të hapura për pajisjet mobile [6].

Arkitektura e platformës Android mund të ndahet në këto shtresa:

- **Linux Kernel**, përdorimi i *Linux Kernel-it* lejon platformën Android të përfitojë nga disa veçori të tilla si siguria, menaxhimin e memories, menaxhimin e procesit, menaxhimin e rrjetit dhe një model transmisioni.

- **Shtresa e abstraksionit të harduerit**, siguron ndërfaqe standarde që ekspozojnë aftësitë e pajisjeve harduerike në kuadër të nivelit më të lartë. Përbehet nga biblioteka të shumta që sigurojnë një ndërfaqe për një komponent të veçantë harduerike, për shembull, një aparat fotografik.

- **Android RunTime**, që nga versioni 5.0 (niveli API 21), çdo aplikacion shkon në procesin e vet dhe me instancën e vet të *Android RunTime* (ART). ART është shkruar për të drejtuar makina të shumta virtuale në pajisjet me memorie të ulëta duke ekzekutuar skedarët *Dalvik Exectuables* (DEX). Siguron veçori të mëdha të tilla si përpilimi parakohe dhe në kohë, mbledhjen e mbeturinave të optimizuar ose mbështetje më të mirë të korrigjimit të gabimeve me funksionalitet të dedikuar. Para versionit 5.0, *Dalvik* është përdorur për këto qëllime.

- **Native C/C++ bibliotekat**, shumë nga komponentët kryesore janë ndërtuar nga kodi bazë që kërkon biblioteka bazë të shkruara në C dhe $C++$. Zhvilluesit duke përdorë *Android NDK* mund t'i qasen disa prej këtyre bibliotekave dhe të shkruajnë kodin e tyre në C dhe $C++$.

- **Korniza Java API**, i gjithë tipari i vendosur i *OS Android* është i gatshëm përmes API-ve që janë shkruar në gjuhën *Java*. Krijon blloqet e nevojshme të ndërtimit për të krijuar Android aplikacione.

- **Aplikacionet e sistemit**, grupi i aplikacioneve bazë për funksionalitetet bazik të tilla si mesazheve SMS, kalendarëve dhe kontakteve. Ato punojnë edhe si aplikacione për përdoruesit, ashtu edhe për aftësitë kyçe për zhvilluesit për të hyrë në aplikacionin e tyre.

Android ofron kornizë të aplikacionit të pasur që lejon zhvilluesit të ndërtojnë dhe krijojnë aplikacione të reja apo inovative [7]. Procesi i zhvillimit për platformën

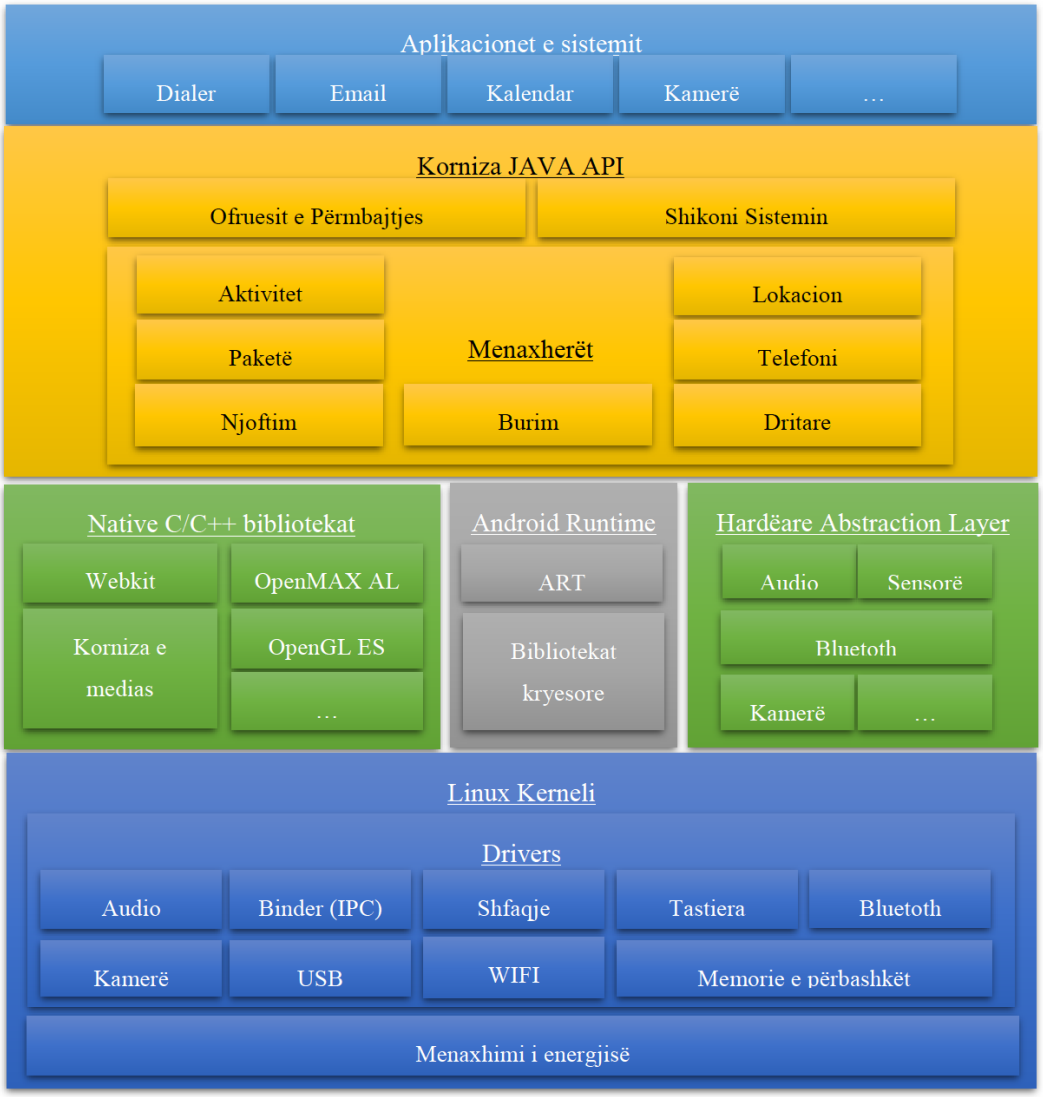

FIGURA 2.1: Arkitektura e Platformës Android [1]

Android ndodh rreth familjeve njoftuese, të cilat përdorin emrat kodues të renditura në mënyrë alfabetike pas trajtimeve.

Aktualisht, ekzistojnë 26 versione të *Android API*, me versionin më të ri 8.0.0 (niveli *API* 26) të quajtur *Oreo*. Versioni i parë alfa i Android është lëshuar me 23 Shtator 2008, dhe përditësimet e para të ndjekura në dy vitet e ardhshme. Që nga viti 2011 një version i vetëm është lëshuar çdo vit [8]. Për momentin, rreth 32% e përdoruesve kanë versionin 6.0 të quajtur *Marshmallow* (*API* versioni 23), duke pasuar versionet 5.0 dhe 5.1 (përkatësisht *API* versionet 21 dhe 22) me shpërndarje rreth 6.7% dhe 21% të quajtur *Llollipop* [3]. Duke përdorë versionet

më të reja te një *Android API* jo vetëm që përmirëson procesin e zhvillimit duke bërë veçoritë më të reja të qasshme, por gjithashtu rrit edhe sigurinë e një pajisjeje. Për shembull, në versionin 6.0 (API versioni 23), autorizimet apo lejet e kërkuara *RunTime* u shtuan.

Versioni	Emri kodues	API	Shpërndarja
2.3.3-2.3.7	Gingerbread	10	0.6%
4.0.3-4.0.4	Ice Cream Sandëich	15	0.6%
4.1.x	Jelly Bean	16	2.3%
4.2.x	Jelly Bean	17	3.3%
4.3	Jelly Bean	18	1.0%
4.4	KitKat	19	14.5%
5.0	Llollipop	21	6.7%
5.1	Llollipop	22	21.0%
6.0	Marshmallow	23	32.0%
7.0	Nougat	24	15.8%
7.1	Nougat	25	2.0%
8.0	Oreo	26	0.2%

Tabela 2.1: Shpërndarja e Android API versioneve në Tetor 2017 [3]

Zhvillimi i aplikacioneve mobile apo celulare zakonisht ndodh duke përdorur një nga metodat që do ti përmendim në vijim. Së pari, është zhvillimi *native* (amtar), duke përdorur një gjuhë specifike të platformës për të ndërtuar një aplikacion për një platformë të vetme. E dyta është qasja ndër platformare, e cila synon njëkohësisht platforma të shumta. Të dyja këto qasje kanë anavantazhet dhe disavantazhet dhe duhet të përdorin procesin e duhur të zhvillimit [9].

Zhvillimi *native* apo amë për platformën Android ofron një optimizim më të saktë, duke rezultuar në një ndërfaqe të përdoruesit (UI) më të mirë, përvoja e përdoruesit (UX) dhe integrimi i platformës. Përdorimi i kësaj qasjeje gjithashtu u mundëson zhvilluesve të përdorin përditësimet më të reja të platformës Android, të cilat përfshijnë shpesh përmirësimet të lidhura me sigurinë, sa më shpejt që ato të jenë në dispozicion [9].

Zhvillimi i aplikacioneve *native* bëhet zakonisht duke përdorur *Android Studio* (AS), që aktualisht është i vetmi mjedis i zhvillimit të integruar në Android (IDE) zyrtar. *Android Studio* është bazuar në *IntelliJ* ($IDEA$), dhe në krye të editorit

të kodit bazë dhe veglave të zhvillimit, ai ofron disa veçori shtesë që rrisin produktivitetin gjatë zhvillimit të Android aplikacioneve të tilla si sistemi i ndërtimit i bazuar në *Grandle*, drejtimin e momentit dhe veglat e testimit, të tilla si *Android Lint.* [10].

Përdorimi i zhvillimit ndër platformë lejon shkrimin e një kodi të vetëm që më vonë të kompilohet për të përshtatur sistemin operativ të platformave të synuara. Aplikacionet e krijuara duke përdorur zhvillimin ndër platformë shpesh nuk integrohen njësoj në platformat dhe kërkojnë më shumë kohë për të implementuar karakteristikat specifike të platformës [9].

Aktualisht, ka platforma të shumta që përdorin këtë qasje, të tilla si *PhoneGap*, i njohur gjithashtu si *Apache Cordova*, *Titanium Studio* ose, kohët e fundit e përdorur gjerësisht, korniza *Xamarin* [9].

Në kohën e lansimit të *API*-të të parë komercial në Android, *Java* ishte e vetmja gjuhë programuese mbështetëse [11]. Megjithatë, numri i gjuhëve, në të cilat mund të zhvillohen aplikacionet Android, është rritur shumë me ardhjen e kornizave ndër platformë [9].

Java është aktualisht gjuha kryesore për programimi të platformës Android [11]. Përdorimi i *Android Studio* i lejon zhvilluesit të përdorin të gjitha veçoritë gjuhësore *Java 7* dhe një nëngrup të veçorive gjuhësore *Java 8* të cilat ndryshojnë sipas versionit të platformës. Megjithatë, më shumë nga karakteristikat gjuhësore *Java 8* do të shtohen në lansimet e ardhshme të *AS IDE* [12].

Të dyja gjuhët, C dhe $C++$ mund të përdoren për të krijuar aplikacione duke kompiluar atë në një bibliotekë *native* apo amtare që *Grandle* mund të paketojë me një *Android Package Kit* (APK) të gjeneruar [13]. Përdorimi i C dhe $C++$ kërkon *Native Development Kit* (NDK) dhe është me e dobishme për rastet në të cilat zhvilluesit duhet të marrin përformancën shtesë, gjendje latente apo zhvillim

i fshehtë ose për të ripërdorur një tjetër bibliotekë C ose $C++$ [14].

Zhvillimi i aplikacioneve Android duke përdorur zhvillimin ndër platformë lejon zhvilluesit të përdorin gjuhë shtesë të specifikuara nga korniza e përdorur. Për shembull, *Xamarin* përdor $C\#$ të kombinuar me kornizën *.NET*, dhe *Titanium Studio* ose *PhoneGap* përdorin teknologji të tilla si $HTML5$, $CSS3$ ose *JavaScript* [9].

2.2 Siguria në android

Sigurimi i një platforme të hapur kërkon arkitekturë të fortë të sigurisë dhe programe të forta të sigurisë. Platforma Android është projektuar me siguri shumë shtresore që është mjaft fleksibile për të mbështetur një platformë të hapur[15].

Android ka ndërtuar karakteristika sigurie që zvogëlojnë frekuencën dhe ndikimin e çështjeve të sigurisë së aplikacionit.

Këto karakteristika janë:

- *Sandbox* Android Aplikacioni i cili ofron izolim për të dhënat e aplikacionit dhe ekzekutimit të kodit.
- Korniza e aplikacionit me implementim të fuqishëm të funksionaliteteve të përbashkëta sigurie të tilla si algoritmet kriptografike ose komunikim ndër procesor.
- Teknologjitë për të implementuar rreziqet që lidhen me gabimet e zakonshme të menaxhimit të memories, për shembull $ASLR$ ose NX.
- Një sistem i koduar apo kriptuar i skedarit që mund të aktivizohet për të mbrojtur të dhënat mbi pajisjet e humbura.
- Autorizimet apo lejet e dhëna nga përdoruesit për të kufizuar qasjen në karakteristikat e sistemit dhe të dhënat e përdoruesit.

- Autorizimet apo lejet për përcaktimin e aplikacioneve për të kontrolluar të dhënat e aplikacionit në një bazë për aplikacion [16].

Shqetësimi më i zakonshëm i sigurisë është nëse të dhënat e ruajtura në pajisje janë të qasshme për aplikacione të tjera.

Ekzistojnë tri mënyra për të ruajtur të dhënat në pajisje:

- **Ruajtje apo depo e brendshme**, si parazgjedhje, të gjitha skedarët e një aplikacioni në ruajtjen e brendshme janë të qasshme vetëm për aplikacionin.
- Nëse kërkohet të ndani apo shpërndani këto skedarë me një aplikacion tjetër, duhet të përdoret ofruesi i përmbajtjes.
- Për të siguruar shtresë shtesë të sigurisë, mund të përdoret kriptimi i skedarëve lokal duke përdorur një çelës që nuk është në dispozicion për aplikacionin.
- **Ruajtje apo depo e jashtme**, dosjet e ruajtura në depo të jashtme, siç janë SD kartat, janë të lexueshme dhe të shkruara në nivel global.
- **Ofruesit e përmbajtjes**, një mekanizëm ruajtës i strukturuar që mund të jetë i kufizuar në aplikacion ose i eksportuar për të lejuar qasje nga aplikacione të tjera.
- Ofruesit e përmbajtjes kanë autorizimet apo lejet e tyre të përcaktuara në një skedar konfigurimi XML të veçantë, që ofron lejet e leximit ose të shkrimit në shtigjet e specifikuara [16].

Lejet janë një mënyrë për të hyrë në burimet dhe të dhënat që nuk janë të disponueshme nga *Sandbox* bazë. Para versionit 6.0 (API versioni 23), të gjitha lejet e kërkuara në Android $Manifest$ u dhanë menjëherë në momentin e lëshimit të aplikacionit. Që nga versioni 23, lejet u ndanë në dy grupe: të zakonshme dhe të rrezikshme. Lejet e rrezikshme duhet të kërkohen në mënyrë eksplicite pasi aplikacioni niset nëpërmjet një dialogu të sistemit dhe përdoruesi mund ta mohojë atë. Ato të zakonshme jepen pa asnjë kërkesë.

Rregulli i përgjithshëm thotë se aplikacioni nuk duhet të kërkojë leje që nuk janë të nevojshme. Kufizimi i qasjes në leje të ndjeshme zvogëlon rrezikun e keqpërdorimit të këtyre lejeve dhe e bën aplikacionin më pak të ndjeshëm ndaj sulmuesve [16].

Rrjetëzimi në Android nuk është shumë i ndryshëm nga mjediset e tjera të *Linux*-it. Rekomandohet përdorimi i protokolleve të përshtatshme për transportimin e të dhënave të ndjeshme dhe $HTTPS$ mbi $HTTP$ kudo që të jetë e mundur. Komunikimi i autentikuar, i kriptuar në nivel *socket* mund të implementohet duke përdorur klasën *SSL Socket*. Duke pasur parasysh frekuencën me të cilën pajisjet Android lidhen në rrjetet pa tela të pa sigurta duke përdorur $WiFi$, inkurajohet përdorimi i rrjeteve të sigurta [15].

Për më tepër, Android ofron Konfigurimin e Sigurisë së Rrjetit që lejon aplikacionet të rregullojnë parametrat e sigurisë së rrjetit në një skedar konfigurimi deklarues pa modifikuar kodin e aplikacionit. Përdorimi i kësaj karakteristike mundëson përshtatjen e autoriteteve çertifikuese të cilave u besohet, në mënyrë të sigurtë të korigjoj lidhjet e sigurta pa shtuar rrezikun për bazën e instaluar, të mbrojë aplikacionin nga përdorimi aksidental i trafikut të qartë, ose të kufizojë një lidhje të aplikacionit të sigurtë me çertifikata të veçanta [17].

Android ofron një koleksion të gjerë të algoritmeve kriptografike për mbrojtjen e të dhënave duke përdorur algoritme me çelës simetrik dhe asimetrik. Duke përdorur një algoritëm kriptografik ekzsitues të siguruar nga klasa $Cipher$, të tilla si algoritmi AES dhe RSA, inkurajohet. Nëse keni nevojë të ruani një çelës për përdorim të përsëritur, përdorni një mekanizëm, si $KeyStore$, që siguron një mekanizëm për ruajtjen afatgjate dhe rikthimin e çelësave kriptografikë. [16]

Komunikimi i ndërsjellë (IPC) në Android është trajtuar duke pëdorur $Intents$, $Binders$, $Messengers$ me një Shërbim dhe *Broadcast Receivers*. Mekanizmi IPC i Android lejon të verifikojë identitetin e aplikacionit që lidhet me çdo IPC dhe

të vendosë politikat e sigurisë për secilin mekanizëm IPC. Mekanizmi i IPC që nuk është i destinuar për përdorim nga aplikacione të tjera duhet të ketë bashkësi të atribuar e të eksportuar në të rreme [16].

Që nga versioni i Android 2.2 (API versioni 8) platforma ofron aftësi të menaxhimit të pajisjeve të sistemit përmes sistemit API të administrimit të pajisjes. Duke përdorur këtë funksion, aplikacionet bëhen të vetëdijshme për sigurinë dhe kështu mund të menaxhojnë qasjen në përmbajtjen e saj. Për shembull, një aplikacion mund të kërkojë një fjalëkalim të bllokimit të pamjes me fuqi të mjaftueshme për të shfaqur një përmbajtje të ndjeshme tek përdoruesi [18].

2.2.1 Projekti i Sigurisë së Hapur të Ueb Aplikacionit

The Open Web Application Security Project ($OWASP$) (Projekti i Sigurisë së Hapur të Ueb Aplikacionit) është një organizatë bamirëse botërore jo fitimprurëse e fokusuar në përmirësimin e sigurisë së softuerit. Qëllimi i tyre është të bëjnë të dukshme sigurinë e softuerit në mënyrë që individët dhe organizatat të jenë në gjendje të marrin vendime të informuara lidhur me sigurinë [19].

Në shtator të vitit 2010, $OWASP$ filloi Projektin e Sigurisë Mobile që ofroi burime të centralizuara për të ndërtuar dhe mbajtur aplikacione të sigurta mobile. Fokusi kryesor është në shtresën e aplikacionit, ku zhvilluesit mund të bëjnë një ndryshim. Përveç kësaj, projekti fokusohet në integrimin në mes të aplikacionëve mobile, shërbimeve të autentifikimit në distancë, dhe karakteristikave specifike të platformës së re. Projekti është i fokusuar në platformën Android dhe *iOS* [20].

Që nga viti 2010, $OWASP$ lanson çdo vit një listë të njohur zakonisht si *Top 10*. Kjo listë përmban *Top 10* çështjet më të shpeshta të sigurisë në aplikacionet në një vit të caktuar. Secila nga kjo çështje përmban përshkrimin e vet të agjentëve të kërcënimit, vektorëve të sulmit, dobësitë e sigurisë, ndikimet teknike dhe ndikimet e biznesit. Gjithashtu, çdo artikull në listë përmban një udhëzues

të thjeshtë se si të identifikojë nëse aplikacioni përmban defektin e sigurisë, si të parandalojë atë dhe një shembull të një skenari të sulmit [21].

Versioni i fundit është nga viti 2016 dhe përmban elementët vijues të renditur sipas faktorit të rrezikut të llogaritur si një kombinim i gjasave dhe ndikimit:

- Përdorimi i gabuar i platformës.
- Ruajtja e pasigurt e të dhënave.
- Komunikimi i pasigurt.
- Autentifikimi i pasigurt.
- Implementimi i algoritmeve kriptografike në nivel të pamjaftueshme.
- Cilësia e kodit të klientit.
- Ngatërrimi i kodit.
- Inxhinieri e kundërt.
- Funksionalitet i jashtëm [21].

Siç mund të shihet, lista përmban shumë çështje që vijnë nga bazat e sigurisë të Android-it.

Mobile Application Security Verification Standard ($MASVS$) vendos kërkesat bazë të sigurisë për aplikacionet mobile. Këto kërkesa ndahen në tri shtresa të verifikimit, secila duke mbuluar një lloj tjetër të kërcënimit:

- **MASVS-L1**, duke mbuluar dobësitë e përbashkëta.
- **MASVS-L2**, i cili merret me sulmet më të sofistikuara si pllombimi SSL.
- **MASVS-R**, që përmbajnë një sërë kërkesash për elasticitet inxhinierik të anasjelltë.

Megjithatë, ajo kurr nuk duhet të përdoret si zëvendësim për kontrollet e sigurisë [22].

Kërkesat ndahen në kategori, ku secila përmban disa kërkesa specifike dhe një lidhje me seksionin e Udhëzuesit të Testimit të Sigurisë Mobile për këtë çështje. Çdo kërkesë përmbanë një përshkrim dhe vendosje në një shtresë verifikimi [22].

Lista aktuale e kategorive të përcaktuara është si më poshtë:

- Arkitektura, dizajni dhe kërkesat e modelimit të kërcënimeve.
- Kërkesat për ruajtjet e të dhënave dhe privatësinë.
- Kërkesat e kriptografisë.
- Kërkesat e autentifikimit dhe menaxhimit të sesioneve.
- Kërkesat e komunikimit në rrjet.
- Kërkesat e ndërveprimit të platformës.
- Cilësia e kodeve dhe ndërtimi i kërkesave për vendosjen.
- Elasticitet kundrejt kërkesave inxhinierike të kundërta [22].

OWASP Mobile Security Testing Guide ($MSTG$) është një manual gjithëpërfshirës për të testuar sigurinë e aplikacioneve mobile. Ajo përshkruan proceset teknike për verifikimin e kontrolleve në $MASVS$. Libri ende nuk është lansuar zyrtarisht, kështu që përmban disa hapësira boshe që do të mbushen në të ardhmen [23].

Përveç proceseve për të verifikuar kontrollet, ai përmban një përshkrim të platformës Android dhe metodave bazë të testimit të sigurisë [23].

Secure Mobile Development Giudelines ($SMDG$) japin udhëzime se si të ndërtohen aplikacione të sigurta mobile, duke pasur parasysh dallimet në sigurinë në platformat mobile dhe desktop (kompjuterike) [24].

Aktualisht, *SMDG* mbulon temat e mëposhtme:

- Autentifikimi dhe menaxhimi i fjalëkalimeve.
- Mashtrimi i kodit.
- Siguria e komunikimit.
- Ruajtja dhe mbrojtja e të dhënave.
- Kontrollet *Paywall*.
- Kontrollet e serverit.
- Menaxhimi i sesionit.
- Përdorimi i bibliotekave ose kodit të palëve të treta [24].

2.3 Kriptografia dhe ofruesit kriptografik

Android ofron një koleksion të gjerë të algoritmeve kriptografike për mbrojtjen e të dhënave. Në përgjithësi, ju duhet të dini se cilin ofrues të sigurisë *Java Cryptography Architecture* (*JCA*) të përdorni në softuerin tuaj. Mundohuni të përdorni nivelin më të lartë të kornizës implementuese para ekzistuese që mund të mbështesë rastin tuaj të përdorimit. Nëse është e aplikueshme, përdorni ofruesit e siguruar nga *Google* në rendin e specifikuar nga *Google*. Nëse keni nevojë të merrni një skedar të sigurtë nga një vend i njohur, një *URL HTTPS* i thjeshtë mund të jetë adekuat dhe nuk kërkon njohuri për algoritmet kriptografike. Nëse keni nevojë për një tunel të sigurtë, merrni parasysh përdorimin e *Https URL Connection* ose *SSLSocket* në vend që të shkruani protokollin tuaj.

Duke përdorur një algoritëm kriptografik ekzsitues të siguruar nga klasa *Cipher*, të tilla si algoritmet *AES* dhe *RSA*, inkurajohet. Përveç kësaj, ju duhet të ndiqni këto praktika më të mira:

- Përdorni algoritmin AES 256 *bit* për qëllime komerciale (Nëse nuk disponohet përdorni AES 128 *bit*).
- Përdorni madhësinë e çelësave publike prej 224 ose 256 *bit* për algoritmet kriptografike duke përdorur lakoret eliptike (EC).
- Dini kur përdorni CBC, CTR, ose modalitetet e bllokimit GCM.
- Shmangni kundër IV ripërdorimin në mënyrë CTR.
- Kur përdorni algoritmet për kriptim, implementoni integritetin duke përdorur mënyrën CBC ose CTR me një nga funksionet e mëposhtme:
  - $HMAC\ SHA1$
  - $HMAC\ SHA$ 256 *bit*
  - $HMAC\ SHA$ 512 *bit*
  - Mënyra GCM

Përdorni një gjenerator të sigurt të numrave të rastësishëm, $SecureRandom$, për të inicializuar çdo çelës kriptografik të gjeneruar nga $KeyGenerator$. Përdorimi i një çelësi që nuk është gjeneruar nga një gjenerator i numrave të rastit të sigurt, dobëson ndjeshëm fuqinë e algoritmit kriptografik dhe mund të lejojë sulme $offline$.

Nëse keni nevojë të ruani një çelës për përdorim të përsëritur, përdorni një mekanizëm, si $KeyStore$, që siguron një mekanizëm për ruajtjen afatgjate dhe rikthimin e çelësave kriptografikë. [16]

Kjo pjesë prezanton arkitekturën e ofruesit kriptografikë të Android dhe diskuton ofruesit e integruar dhe algoritmet kriptografike që ata mbështesin. Për shkak se Android ndërton në Arkitekturën Kriptografike $Java$ (JCA), ne e prezantojmë shkurtimisht hartimin e tij, duke filluar me kornizën e ofruesit të shërbimit kriptografik (CSP). Pastaj diskutojmë klasat dhe ndërfaqet kryesore të JCA, dhe

primitivët kriptografikë që zbatojnë. Pastaj, ne paraqesim ofruesit e *JCA* të Android dhe bibliotekat kriptografike si dhe algoritmet kriptografike që mbështesin secilin ofrues. Së fundi, ne tregojmë se si të përdorim algoritme kriptografike shtesë të kriptografisë duke instaluar një ofrues të personalizuar *JCA*. [25]

2.3.1 Arkitektura e ofruesit JCA

JCA ofron një kornizë të dhënat kriptografike të zgjerueshme dhe një sërë *API* që mbulojnë primitivët kryesorë kriptografikë në përodrim sot. Kjo arkitekturë synon të jetë e pavarur nga implementimi dhe mund të zgjerohet. Aplikacionet që përdorin *API*-të standarte të *JCA*-së vetëm duhet të specifikojnë algoritmin kriptografik që ata duan të përdorin dhe (në shumicën e rasteve) nuk varen nga një implementim i caktuar i ofruesve. Mbështetja për algoritme të reja kriptografike mund të shtohet thjesht duke regjistruar një ofrues tjetër që implementon algoritmet kriptografike të kërkuara. Përveç kësaj, shërbimet kriptografike të siguruara nga ofruesit e ndryshëm janë në përgjithësi të ndërveprueshme (me kufizime të caktuara kur çelësat janë të mbrojtur nga hardueri ose materialet kyçe, nuk janë të disponueshme drejtpërdrejtë) dhe aplikacionet janë të lira për të përzjerë dhe të përputhen shërbimet nga ofruesit e ndryshëm sipas nevojës. Le të shohim arkitekturën e *JCA* në më shumë detaje.

JCA ndan funksionalitetin kriptografik në një numër shërbimesh abstrakte kriptografike të quajtura bazë dhe përcakton *API* për çdo shërbim në formën e një klase bazike. Për shembull, nënshkrimet dixhitale përfaqësohen nga klasa bazike. Nënshkrim, dhe kriptimi është modeluar me klasën *Cipher*.

Në kontekstin e *JCA*, një ofrues shërbimi kriptografik është një paketë (ose bashkësi paketash) që ofron një implementim konkret të disa shërbimeve kriptografike. Secili ofrues reklamon shërbimet dhe algoritmet që inplementohen, duke lejuar kornizën e *JCA* të mbajë një regjistër të algoritmeve kriptografike të mbështetura dhe ofruesve të tyre të implementimit. Ky regjistër mban një porosi preferenciale

për ofruesit, kështu që nëse një algoritëm kriptografik i caktuar sigurohet nga më shumë se një ofrues, ai me urdhër më të lartë preferencial i kthehet aplikacionit kërkues. Një përjashtim nga ky rregull është bërë për klasat e bazëve që mbështesin përzgjedhjen e ofruesve të vonuar (*Cipher*, *KeyAgreement*, dhe Nënshkrimi). Me zgjedhjen e ofruesve të vonuar, ofruesi nuk zgjedhet kur një instancë e klasës së bazës është krijuar, por kur klasa e bazës është inicializuar për një operacion të veçantë kriptografik. Inicjimi kërkon një *Key* instancë, të cilin sistemi e përdor për të gjetur një ofrues që mund të pranojë objektin e specifikuar *Key*. Zgjedhja e vonuar e ofruesve është e dobishme kur përdoren çelësat që ruhen në harduer sepse sistemi nuk mund të gjejë ofruesin e mbështetur në harduer bazuar vetëm në emrin e algoritmit kriptografik. Sidoqoftë, *Key* instancat konkrete të kaluara në metodat e inicializimit zakonisht kanë informacion të mjaftueshëm për të përcaktuar ofruesin themelor.

Shënim: *Versionet e tanishme të Android nuk mbështesin përzgjedhjen e ofruesve të vonuar, por disa punë të ngjashme janë duke u kryer në degën kryesore dhe përzgjedhja e ofruesve të vonuar ka mundësi të mbështetet në një version të ardhshëm.*

Le të shohim një shembull duke përdorë konfigurimin e ofruesit të ilustruar në figurën më poshtë. Nëse një aplikacion kërkon një implementim të algoritmit krip-

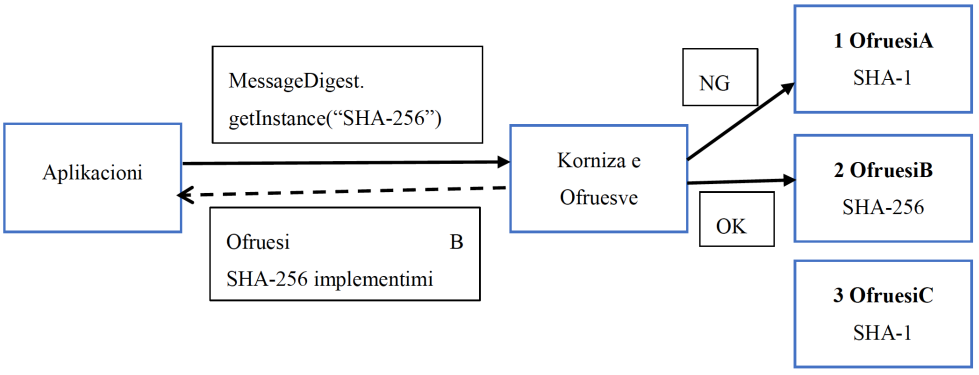

FIGURA 2.2: Zgjedhja e implementimit të algoritmeve të JCA kur ofruesi nuk është i specifikuar

tografik të klasifikuar SHA 256 pa specifikuar një ofrues, korniza e ofruesve e kthen implementimin e gjetur në $OfruesiB$, jo ai në $OfruesiC$, i cili gjithashtu

mbështet SHA 256.

Kërkimi i një implementimi $SHA-256$ pa specifikuar një ofrues:

$$MessageDigest\ md = MessageDigest.getInstance(\text{"}SHA-256\text{"});$$

Nga ana tjetër, nëse aplikacioni kërkon në mënyrë specifike $ProviderC$, implementimi i tij do të kthehet edhe pse $ProviderB$ ka një renditje më të lartë preferenciale.

Kërkimi i një implementimi $SHA-256$ nga një ofrues specifik:

$$MessageDigest\ md = MessageDigest.getInstance(\text{"}SHA{-}256\text{"}, \text{"}ProviderC\text{"});$$

Në përgjithësi, aplikacionet nuk duhet të kërkojnë në mënyrë eksplicite një ofrues, përveç nëse ata përfshijnë ofruesin e kërkuar si pjesë të aplikacionit ose mund të trajtojnë rezervimin nëse ofruesi i preferuar nuk është i disponueshëm.

Korniza e JCA garanton pavarsinë e implementimit duke kërkuar që të gjitha implementimet e një shërbimi ose algoritmi të caktuar kriptografik të jenë në përputhje me një ndërfaqe të përbashkët. Për çdo klasë bazaësh që paraqet një shërbim të veçantë kriptografik, korniza përcakton një klasë abstrakte përkatëse të *Ndërfaqes së Shërbimit të Ofruesve* (SPI). Ofruesit që japin një shërbim të veçantë kriptografik implementojnë dhe reklamojnë klasën përkatëse SPI. Për shembull, një ofrues që implementon një algoritëm të caktuar për kriptim do të kishte një implementim të klasës $CipherSpi$ që korrespondon me klasën bazë $Cipher$. Kur një aplikacion e quan këtë metodën $Cipher.getInstance()$, korniza e JCA gjen furnizuesin e duhur duke përdorur procesin e përshkruar në *"Ofruesit e Shërbimeve të Kriptografisë"* dhe jep një instancë $Cipher$ që drejton të gjitha thirrjet e metodës së tij në nënklasën $CipherSpi$ të implementuara në ofruesin e përzgjedhur.

Përveç klasave të implementimit të SPI, çdo ofrues ka një nënklasë të klasës abstrakte *java.security.Provider* që përcakton emrin dhe versionin e ofruesit dhe, më e rëndësishmja, një listë të algoritmeve të mbështetura dhe përputhjen e klasave të implementimit të SPI. Korniza e ofruesve të JCA përdorë këtë Provider klasë për të ndërtuar regjistrin e ofruesve, dhe e kërkon atë kur kërkon për implementimet e algoritmit për t'u kthyer tek klientët e tij. [25]

2.3.2 Klasa bazë e JCA

Një klasë bazë e ofron ndërfaqen për një lloj specifik të shërbimit kriptografik. JCA baza oforjnë një nga shërbimet e mëposhtme:

- Operacionet kriptografike, kriptim/dekriptim, shenjë/verifikim, hash dhe kështu më radhë.
- Gjenerimi ose konvertimi i veglave kriptografik, çelësat dhe parametrat e algoritmeve.
- Menaxhimi dhe ruajtja e objekteve kriptografike, të tilla si çelësat dhe çertifikatat digjitale. [25]

Përveç kësaj për të siguruar një ndërfaqe të unifikuar për operacionet kriptografike, klasat bazë që ndërrojnë kodin e klientit nga implementimi themelor, e cila është arsyeja pse ato nuk mund të instancohen drejtëpërdrejtë, në vend të kësaj, ata ofrojnë një metodë statike të gjenerimit të quajtur *getInstance*() që ju lejon të kërkoni një implementim në mënyrë indirekte.

Metoda e nënshkrimit të gjeneruar së klasës bazë të JCA:

```
static EngineClassName getInstance(String algorithm) ...(1)
throws NoSuchAlgorithmException
static EngineClassName getInstance(String algorithm, String provider) ...(2)
throws NoSuchAlgorithmException, NoSuchProviderException
static EngineClassName getInstance(String algorithm, Provider provider) ...(3)
throws NoSuchAlgorithmException
```

Zakonisht, ju do të përdorni nënshkrimin në (1) dhe të specifikoni vetëm emrin e algoritmit. Nënshkrimet në (2) dhe (3) ju lejojnë të kërkoni një implementim nga një ofrues specifik. Të gjitha variantet hedhin një $NoSuchAlgorithmException$ nëse një implementim për algoritmin e kërkuar nuk është i disponueshëm dhe (2) hedh $NoSuchProviderException$ nëse një ofrues me emrin e specifikuar nuk është regjistruar. [25]

Parametri i alogritmit të vargut që të gjitha metodat e gjenerimit marrin harta për një algoritëm ose transformim të veçantë kriptografik ose përcakton një strategji implementimi për objektet e nivelit të lartë që menaxhojnë koleksionet e çertifikatave ose çelësave. Zakonisht hartografia është e drejtëpërdrejtë. Për shembull, $SHA-256$ harton një implementim të algoritmit *hashing* $SHA-256$ dhe AES kërkon një implementim të algoritmit të kodimit AES. Megjithatë, disa emra të algoritmeve kanë strukturë dhe specifikojnë më shumë se një parametër të implementimit të kërkuar. Për shembull, $SHA-256$ me RSA specifikon një implementim të nënshkrimit që përdor $SHA-256$ për hashimin e mesazhit të nënshkruar dhe RSA për të kryer operacionin e nënshkrimit. Algoritmet gjithashtu mund të kenë emërtimet, dhe më shumë se një emër algoritmi mund të hartohet në të njëjtin implementim.

Emrat e algoritmeve janë pa ndjeshmëri ndaj rastit. Emrat e algoritmeve standarde të mbështetura nga secila klasë bazë e JCA janë përcaktuar në *Dokumentimin e Emrit të Algoritmit Standard JCA*. Ju mund të përdorni kodin në listimin e mëposhtëm për të rënditur të gjithë ofruesit, emrat e algoritmeve të shërbimeve kriptografike të siguruara nga secili ofrues, dhe klasat e implementimit të të cilave ato planifikojnë. [25]

Listimi i të gjithë ofruesve të JCA dhe algoritmeve që ata mbështesin:

```
Provider[] providers = Security.getProviders();
for (Provider p : providers) {
        System.out.printf("%s/%s/%f\n", p.getName(),
```

```
        p.getInfo(), p.getVersion());
        Set<Service> services = p.getServices();
        for (Service s : services) {
                System.out.printf("\t%s/%s/%s\n", s.getType(),
                s.getAlgorithm(), s.getClassName());
        }
}
```

Klasa *SecureRandom* paraqet një *Gjenerator të Numrave të Rastësishëm Kriptografik*, e cila përdoret së brendshmi nga shumica e operacionëve kriptografike për të gjeneruar çelësa dhe veglat të tjera kriptografike. Implementimi tipik i softuerit është zakonisht një *Cryptographically Secure Pseudo Random Number Generator* ($CSPRNG$), i cili gjeneron një sekuencë numrash që përafrojnë vetit e numrave të rastësishëm të vërtetë bazuar në një vlerë fillestare të quajtur një farë. Meqë cilësia e numrave të rastësishëm të gjeneruar nga një $CSPRNG$ në mas të madhe varet nga rrënjët e saj, ajo zgjidhet me kujdes, zakonisht bazuar në prodhimin e një RNG të vërtetë.

Në Android, implementimet e $CSPRNG$ mblidhen duke lexuar bajte të rrenjës nga skedari standard i pajisjes $LINUX/dev/urandom$, i cili është një ndërfaqe ndaj kernelit $CSPRNG$. Ndërsa vetë $CSPRNG$ kerneli mund të jetë në një gjendje mjaft të parashikueshme menjëherë pas fillimit, Android periodikisht ruan gjendjen (që është 4096 *byte* si nga Android 4.4) e $CSPRNG$ kernelit në skedarin $/data/system/entropy.dat$. Përmbajtja e atij skedari është shkruar përsëri në $/dev/urandom$ në *boot* në mënyrë që të mbajnë gjendjen e mëparshme të $CSPRNG$. Kjo kryhet nga shërbimi i sistemit $EntropyMixer$. Ndryshe nga shumica e klasave bazë, *SecureRandom* ka konstruktorë publikë që mund t'i përdorni për të krijuar një instancë. Në listimin e mëposhtëm, është përdor metoda $nextBytes()$ për të pasur varg të bajtëve të gjeneruar rastësisht. [25]

Përdorimi i *SecureRandom* për të gjeneruar bajt të rastit:

```
SecureRandom sr = new SecureRandom();
byte[] output = new byte[16];
sr.nextBytes(output);
```

Klasa *MessageDigest* përfaqëson funksionalitetin e një mesazhi *digest* kriptografik, gjithashtu i referuar si një *hash* funksion. Një mesazh *digest* kriptografik merr një sekuencë arbitrare të gjatë të bajtëve dhe gjeneron një sekuencë bajtësh me madhësi fikse që quhet një *digest* ose *hash*. Një funksion i mirë *hash* garanton se edhe një ndryshim i vogël në inputin e tij rezulton në rezultate krejtësisht të ndryshme dhe se është shumë e vështirë të gjesh dy inpute që janë të ndryshme por gjenerojn të njejtën vlerë *hash* (rezistencë përplasjeje), ose të gjenerojë një hyrje që ka një *hash* të dhënë (rezistencë para imazhit). Një tjetër veti e rëndësishme e funksionëve *hash* është rezistenca e dytë para imazhit.

Përdorimi i *MessageDigest* për të hashuar të dhënat:

```
MessageDigest md = MessageDigest.getInstance("SHA-256");
byte[] data = getMessage();
byte[] digest = md.digest(data);
```

Një instancë *MessageDigest* është krijuar duke kaluar emrin e *hash* algoritmit në metodën e gjenerimit të *getInstance*(). Inputi mund të sigurohet në *chunks* duke përdorur një nga metodat *update*(), dhe pastaj duke e quajtur një nga metodat *digest*() për të marrë vlerën e llogaritur *hash*. Nga ana tjetër, nëse madhësia e të dhënave të hyrjes është fikse dhe relativisht e shkurtër, ajo mund të hashohet në një hap duke përdorur metodën *digest*(*byte*[] *input*) ashtu siç tregohet në listimin e më sipërm. [25]

Klasa *Signature* siguron një ndërfaqe të përbashkët për algoritmet e nënshkrimit digjital bazuar në kriptimin asimetrik. Një algoritëm i nënshkrimit digjital merr një mesazh arbitrar dhe një çelës privat dhe gjeneron një varg të fiksuar me bajt të quajtur një nënshkrim. Nënshkrimet digjitale zakonisht aplikojnë një algoritëm *digest* në mesazhin hyrës, kodojnë vlerën *hash* të llogaritur, dhe pastaj

përdorin një çelës privat operacional për të gjeneruar nënshkrimin. Nënshkrimi pastaj mund të verifikohet duke përdorur çelësin përkatës publik duke implementuar operacionin e kundërt, duke llogaritur vlerën *hash* të mesazhit të nënshkruar dhe duke e krahasuar atë me atë të koduar në nënshkrim. Verifikimi i suksesshëm garanton integritetin e mesazhit të nënshkruar dhe, me kusht që çelësi privat i nënshkrimit të mbetet në të vërtetë privat, origjinaliteti i tij.

Instancat e nënshkrimit krijohen me metodën standarde të gjenerimit $getInstance()$. Emri i algoritmit që përdoret zakonisht është në formën $< digest >$ me $< encryption >$, ku $< digest >$ është një emër algoritmi *hash* siç përdoret nga $MessageDigest$ (si $SHA-256$), dhe $< encryption >$ është një algoritëm kriptues asimetrik (të tilla si RSA ose DSA). Listimi i më poshtëm tregon se si të përdorësh klasën $Signature$ për të gjeneruar dhe verifikuar një nënshkrim kriptografik. [25]

Gjenerimi dhe verifikimi i një nënshkrimi me klasën $Signature$:

```
PrivateKey privKey = getPrivateKey();
PublicKey pubKey = getPublicKey();
byte[] data = "sign me".getBytes("ASCII");
Signature sig = Signature.getInstance("SHA256WithRSA");
sig.initSign(privKey);
sig.update(data);
byte[] signature = sig.sign();
sig.initVerify(pubKey);
sig.update(data);
boolean valid = sig.verify(signature);
```

Klasa *Cipher* siguron një ndërfaqe të përbashkët për operacionet kriptuese dhe dekriptuese. Kriptimi është procesi i përdorimit të një algoritmi (të quajtur një shifër) dhe një çelësi për të tranformuar të dhënat (të quajtura teksti i thjeshtë ose mesazhi i thjeshtë) në një formë të rastësishme (tekst i shifruar). Operacioni invers, i quajtur dekriptim, transformon tekstin e shifruar përsëri në tekstin origjinal. Dy

llojet kryesore të kriptimit që përdoren gjerësisht sot janë kriptimi simetrik dhe asimetrik. Çelësi simetrik ose sekret, kriptimi përdor të njejtin çelës për të kriptuar dhe dekriptuar të dhënat. Kriptimi asimetrik përdor një palë çelësash: një çelës publik dhe një çelës privat. Të dhënat e kriptuara me një nga çelësat mund të dekriptohet vetëm me çelësin tjetër të palës. Klasa Cipher mbështet të dy, kriptimin simetrik dhe asimetrik. Në varësi të mënyrës se si ata përpunojnë të dhëna, shifrat mund të jenë blloqe ose rrjedhës. Bllok shifrat punojnë në grupe të fiksuara me madhësi të të dhënave të quajtura blloqe. Nëse hyrja nuk mund të ndahet në një numër të plotë të blloqeve, blloku i fundit është i mbushur me shtimin e numrit të nevojshëm të bajtëve që përputhen me madhësinë e bllokut. Si operacioni ashtu edhe bajtët e shtuar quhen mbushje. Mbushja është hequr në procesin e dekriptimit dhe nuk është e përfshirë në tekstin e thjeshtë të dekriptuar. Nëse një algoritëm mbushjeje është përcaktuar, klasa *Cipher* mund të shtojë automatikisht dhe të heqë mbushjet. Nga ana tjetër, shifrat e transmetimit përpunojnë të dhënat një bajtëshe (ose madje bit) në një kohë dhe nuk kërkojnë mbushje. [25] Klasa Mac siguron një ndërfaqe të përbashkët për algoritmet e *Message Authentication Code*. Një MAC përdoret për të kontrolluar integritetin e mesazheve të transmetuara mbi një kanal jo të besueshëm. Algoritmet MAC përdorin një çelës sekret për të llogaritur një vlerë, MAC (i quajtur edhe një etiketë), i cili mund të përdoret për të vërtetuar mesazhin dhe për të kontrolluar integritetin e tij. I njejti çelës përdoret për të kryer verifikimin, prandaj duhet të ndahet midis palëve të komunikimit.

Përdorimi i klasës MAC për të gjeneruar një kod për autentifikim të mesazhit:

```
KeyGenerator keygen = KeyGenerator.getInstance("HmacSha256");
SecretKey key = keygen.generateKey();
Mac mac = Mac.getInstance("HmacSha256");
mac.init(key);
byte[] message = "MAC me".getBytes("UTF-8");
byte[] tag = mac.doFinal(message);
```

Një instancë MAC është marrë me metodën $getInstance()$ duke kërkuar një implementim të algoritmit $HMAC6$ MAC që përdorë $SHA-256$ si funksion hash. Pastaj fillohet me një instancë $SecretKey$, që mund të gjenerohet me një $KeyGenerator$, që rrjedhin nga një fjalëkalim ose instancohet drejtëpërdrejt nga bajtat e çelësit të papërpunuar. Për implementimet MAC bazuar në funksionet hash, lloji i çelësit nuk ka rëndësi, por implementimet që përdorin një shifër simetrike mund të kërkojnë një lloj çelësi të ngjashëm që duhet kaluar. Ne pastaj mund të kalojmë mesazhin në copa duke përdorur një nga metodat $update()$ dhe thirrjen $doFinal()$ për të marrë vlerën përfundimtare MAC, ose të kryeni operacionin në një hap duke kaluar bajtat e mesazhit drejtëpërdrejt në $doFinal()$. [25] Ndërfaqja Key përfaqëson çelësat e errët në kornizën e JCA, dhe që përcakton vetëm tre metoda:

- *String getAlgorithm()* – Kthen emrin e algoritmit të kriptimit (simetrik apo asimetrik) që ky çelës mund të përdoret. Shembujt janë AES ose RSA.
- *[byte] getEncode()* – Kthen një formë standarde të koduar të çelësit që mund të përdoret kur transmeton çelsëin në sisteme tjera. Kjo mund të përdoret për çelësa privatë. Për implementimet e mbështetura nga harduerët që nuk lejojnë eksportimin e materialit kryesor, kjo metdoë zakonisht kthen *null*.
- *String getFormat()* – Kthen formatin e çelësit të koduar. Kjo zakonisht është RAW për çelësat që nuk janë koduar në ndonjë format të veçantë. Formate të tjera të përcaktuara në JCA janë $X.509$ dhe $PKCS\#8$.

Ju mund të merrni një Key instancë në mënyrat e mëposhtme:

- Gjeneroni çelësat duke përdorur një $KeyGenerator$ ose një $KeyPairGenerator$.
- Konvertoni nga një përfaqësim i koduar duke përdorur një $KeyFactory$.
- Marrja e një çelësi të ruajtur nga një $KeyStore$. [25]

Ndërfaqja $SecretKey$ përfaqëson çelësat e përdorur në algoritme simetrike. Kjo është një ndërfaqe shënuesish dhe nuk shton asnjë metodë për ato të ndërfaqes

Key të prindit. Ajo ka vetëm një implementim që mund të instancohet direkt, konkretisht *SecretKeySpec*. Nën ndërfaqja *PBEKey* përfaqëon çelësat që rrjedhin duke përdorur *Password Based Encryption – PBE*. *PBE* përcakton algoritme që nxjerrin çelësa të fortë kriptografikë nga fjalëkalimet dhe frazat kaluese, të cilat zakonisht kanë entropi të ulët dhe kështu nuk mund të përdoren direkt si çelësa. *PBE* bazohet në dy ide kryesore: përdorimi i një kripe për të mbrojtur nga sulmet e fjalëve të asistuara, dhe duke përdorur një numër të madh iterimesh për të bërë derivim kyç në mënyrë kompjuterike të shtrejtë. Kalkulimi i kripës dhe iterimi përdoren si parametra për algoritmet *PBE* dhe kështu duhet të mbahen në mënyrë që të gjenerojnë të njëjtin çelës nga një fjalëkalim i caktuar. Kështu implementimet *PBEKey* janë të nevojshme për të implementuar *getSalt*() dhe *getIterationCount*() së bashku me *getPassword*(). [25] Çelësat publik dhe privatë për algoritmet e kodimit asimetrik janë modeluar me ndërfaqet *PublicKey* dhe *PrivateKey*. *JCA* përcakton klasa të specializuara për algoritme asimetrike konkrete që mbajnë parametrat e çelësave përkatës, të tilla si *RSAPublicKey* dhe *RSAPrivateCrtKey*. Ndërfaqja *KeyPair* është thjeshtë një enë për një çelës publik dhe një çelës privat. [25] Ndërfaqja *JCA Key* përfaqëson çelësat e errët. Nga ana tjetër, *KeySpec* modifikon një specifikacion çelësi, i cili është një përfaqësim transparent i çelësit që ju lejon të keni qasje në parametrat e çelësit individual. Në praktikë, shumica e ndërfaqëve *Key* dhe *KeySpec* për algoritmet konkrete mbivendosen në mënyrë të konsiderueshme, sepse parametrat e çelësit duhet të jenë të arritshëm në mënyrë që të implementojnë algoritmet e kriptimit. Për shembull, të dy *RSAPrivateKey* dhe *RSAPrivateKeySpec* përcaktojnë metodat *getModulus*() dhe *getPrivateExponent*(). Dallimi është i rëndësishëm vetëm kur një algoritëm implementohet në harduer, në të cilin rast *KeySpec* do të përmbajë vetëm një referencë për çelësin e menaxhuar me harduer dhe jo parametrat e çelësit aktual. Çelësi përkatës do të mbajë një dorezë për çelësin e menaxhuar me harduer dhe mund të përdoret për të kryer operacione kriptografike, por nuk do të mbajë asnjë material kryesor. Për shembull, një *RSAPrivateKey* që ruhet në harduer do të kthen *null* kur të thirret metoda e saj *getPrivateExponent*(). Implementimet *KeySpec* mund të mbajnë një përfaqësim të çelësit të koduar, në të cilin rast

ato janë algoritme të pavarura. Për shembull, $PKCS8EncodedKeySpec$ mund të mbajë ose një çelës RSA ose një çelës DSA në formatin e $PKCS\#8$ të koduar DER. Nga ana tjetër, një algoritëm $KeySpec$ specifik mban të gjitha parametrat e çelësit si fusha. Për shembull, $RSAPrivateKeySpec$ përmban modulin dhe eksponentin privat për një çelës RSA, i cili mund të merret duke përdorur metodat $getModulus()$ dhe $getPrivateExponent()$, respektivisht. Pavarësisht nga lloji i tyre, $KeySpecs$ konvertohen në objekte kryesore duke përdorur një $KeyFactory$. [25] Një $KeyFactory$ përmbledh një rutinë konvertimi që nevojitet për të kthyer një përfaqësim të çelësit publik ose privat transparent në një objekt çelësi të errët që mund të përdoren për të kryer një operacion kriptografik ose anasjelltas. Një $KeyFactory$ që konverton një çelës të koduar zakonisht analizon të dhënat e koduara kryesore dhe ruan çdo parametër çelësi në fushën përkatëse të klasës Key konkrete. Për shembull, për të analizuar një kod publik të RSA të koduar $X.509$, ju mund të përdorni kodin e mëposhtëm.

Përdorimi i një $KeyFactory$ për të kthyer një çelës të koduar $X.509$ në një objekt $RSAPublicKey$:

```
KeyFactory kf = KeyFactory.getInstance("RSA");
byte[] encodedKey = readRsaPublicKey();
X509EncodedKeySpec keySpec = new X509EncodedKeySpec(encodedKey);
RSAPublicKey pubKey = (RSAPublicKey) kf.generatePublic(keySpec);
```

Një $KeyFactory$ gjithashtu mund të konvertojë një algoritëm $KeySpec$ specifik, të till si $RSAPrivateKeySpec$, në një shembull të përputhshëm, por në atë rast thjesht kopjon parametrat e çelësit nga njëra klasë në tjetrën. Thirrja e metodës $KeyFactory.getKeySpec()$ konverton një objekt Key në një $KeySpec$, por ky përdorim nuk është shumë i zakonshëm sepse një përfaqësim i koduar i çelësit mund të merret thjeshtë duke thirrur $getEncoded()$ direkt në objektin çelës, dhe algoritmi $KeySpec$ specifik në përgjithësi nuk ofron më shumë informacione sesa një klasë konkrete Key. [25] $SecretKeyFactory$ është shumë e ngjashme me $KeyFactory$ përveç se vepron vetëm në çelësat sekret (simetrik). Ju mund të

përdorni atë për të kthyer një specifikim simetrik në një objekt Çelës dhe anasjelltas. Megjithatë, në praktikë, nëse keni qasje në materialin kryesor të një çelësi simetrik, është shumë më e lehtë ta përdorni atë për të ilustruar drejtëpërdrejt një *SecretKeySpec* që është gjithashtu një Çelës, kështu që nuk përdoret shumë shpesh në këtë mënyrë. Një rast shumë më i zakonshëm i përdorimit po krijon një çelës simetrik nga një fjalëkalim i dhënë nga përdoruesi duke përdorur *PBE*. [25]

Gjeneremi i një çelësi sekret nga një fjalëkalim duke përdour *SecretKeyFactory*:

```
byte[] salt = generateSalt();
int iterationCount = 1000;
int keyLength = 256;
KeySpec keySpec = new PBEKeySpec(Password.toCharArray(), salt,
iterationCount, keyLength);
SecretKeyFactory skf = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA1");
SecretKey key = skf.generateSecret(keySpec);
```

Klasa *KeyPairGenerator* gjeneron dy çifte të çelsave publik dhe privat. *KeyPairGenerator* është instancuar duke kaluar një emër algoritmi asimetrik në metodën *getInstance*().

Inicimi i *KeyPairGenerator* me parametra të algoritmit specifik:

```
KeyPairGenerator kpg = KeyPairGenerator.getInstance("ECDH");
ECGenParameterSpec ecParamSpec = new ECGenParameterSpec("secp256r1");
kpg.initialize(ecParamSpec);
KeyPair keyPair = kpg.generateKeyPair();
```

Ka dy mënyra për të inciuar një *KeyPairGenerator*: duke specifikuar madhësinë e dëshiruar të çelësit dhe duke specifikuar parametrat specifik të algoritmit. Në të dy rastet, ju mund të kaloni opsionalisht një instancë *SecureRandom* që do të përdoret për gjenerimin e çelësit. Nëse specifikohet vetëm një madhësi e çelësit, gjenerimi i çelësit do të përdorë parametrat e parazgjedhur (nëse ka). Për të

specifikuar parametra shtesë, duhet të instanconi dhe konfiguroni një instancë *AlgorithmParameterSpec* e përshtatshme për algoritmin simetrik që po përdorni dhe ta kaloni atë në metodën *Initialize*(), siç tregohet në listimin e mësipërm. [25] *KeyGenerator* është shumë i ngjashëm me klasën e *KeyPairGenerator*, përveç se gjeneron çelësa simetrikë. Ndërsa ju mund të gjeneroni më shumë çelësa simetrikë duke kërkuar një sekuencë të bajtëve të rastësishëm nga *SecureRandom*, implementimet e *KeyGenerator*-it kryejnë kontrolle shtesë për çelësat e dobët dhe vendosin bajtët e paritetit të çelësit aty ku është e përshtatshme dhe mund të përfitojnë nga hardueri i kriptografisë në dispozicion, kështu që është mirë të përdorni *KeyGenerator* në vend që të krijoni manualisht çelësat. Listimi i mëposhtëm tregon se si të gjenerosh një çelës *AES* duke përdorur *KeyGenerator*.

Gjenerimi i një çelësi *AES* me *KeyGenerator*:

```
KeyGenerator keygen = KeyGenerator.getInstance("AES");
kg.init(256);
SecretKey key = keygen.generateKey();
```

Për të gjeneruar një çelës duke përdorur *KeyGenerator*, duhet të krijohet një instancë, të përcaktohet madhësia e dëshiruar e çelësit me *init*(), dhe pastaj të thirret *generateKey*() për të gjeneruar çelësin. [25] Klasa *KeyAgreement* përfaqëson një protokoll çelësi të marrëveshjes që lejon dy ose më shumë palë të gjenerojnë një çelës të përbashkët pa pasur nevojë të shkëmbejnë informacion sekret. Ndërsa ka protokolle të ndryshme të marrëveshjes çelës, ato që përdoren më gjerësisht sot bazohen në shkëmbimin çelës të *Diffie-Hellman (HD)* – ose një origjinialitet i bazuar në kriptografinë diskrete të logaritmit (të njohur si *HD*), ose variantin më të ri të bazuar në kriptografinë eliptike të çelësit (*ECDH* 12). Të dy variantet e protokollit modelohen në *JCA* duke përdorur klasën *KeyAgreement* dhe mund të kryhen në të njejtën mënyrë, me të vetmin ndryshim që janë çelësat. Për të dy variantet, secila palë e komunikimit duhet të ketë një çelës pale, me të dy palët e çelësit të gjeneruara me parametra të njejtë të çelësit. Pastaj palët

kanë nevojë vetëm për të shkëmbyer çelësat publikë dhe për të ekzekutuar algoritmin e marrëveshjes së çelësit për të arritur një sekret të përbashkët. Listimi i mëposhtëm ilustron përdroimin e klasës *KeyAgreement* për të gjeneruar një sekret të përbashkët duke përdorur *ECDH*.

Përdorimi i *KeyAgreement* për të gjeneruar një sekret të përbashkët:

```
PrivateKey myPrivKey = getPrivateKey();
PublicKey remotePubKey = getRemotePubKey();
KeyAgreement keyAgreement = KeyAgreement.getInstance("ECDH");
keyAgreement.init(myPrivKey);
keyAgreement.doPhase(remotePubKey, true);
byte[] secret = keyAgreement.generateSecret();
```

Ky listim tregon rrjedhën e thirrjeve vetëm për një nga palët (A), por pala tjetër (B) duhet të ekzekutojë të njejtën sekuencë duke përdorur çelësin e vet privat për të inicializuar marrëveshjen dhe duke kaluar çelësin publik të A për *doPharse*(). [25] *JCA* përdor termin keystore për t'iu referuar një baze të dhënash të çelësave dhe çertifikatave. Një *KeyStore* menaxhon objekte të shumta kriptografike, të referuara si shënime që janë të lidhura secila me një varg ndryshe. Klasa *KeyStore* ofron një ndërfaqe të përcaktuar mirë në një *KeyStore* që përcakton tri lloje të hyrjeve:

- *PrivateKeyEntry* – Një çelës privat me një çertfifikatë të lidhur zinxhir. Për një implementim të softuerit, materiali kryesor privat zakonisht është i koduar dhe i mbrojtur nga një frazë e kalimit të dhënë nga përdoruesi.
- *SecretKeyEntry* – Një çelës sekret (simetrik). Jo të gjitha implementimet e *KeyStore* mbështesin ruajtjen çelësave të fshehtë.
- *TrustedCertificateEntry* – Një çertifikatë e çelësit publik e një pale tjetër. *TrustedCertificateEntry*-të shpesh përmbajnë çertifikatat *CA* që mund të përdoren për të krijuar marrëdhënie besimi. Një *KeyStore* që përmban vetëm *TrustedCertificateEntry*-të quhet një truststore. [25]

$CertificateFactory$ vepron si një çertifikatë dhe CRL analizues dhe mund të ndërtojë një çertifikatë zinxhirore nga një listë e çertifikatave. Mund të lexojë një transmetim që përmban çertifikata të koduara ose CRL dhe nxjerr një koleksion (ose një instancë të vetme) të objektëve $java.security.cert.Certificate$ dhe $java.security.cert.CRL$. Zakonisht, vetëm një implementim $X.509$ që analzion çertifikatat $X.509$ dhe CRL-të është në dispozicion. Listimi i mëposhtëm tregon se si të analizohet një skedar çertifikate duke përdorur $CertificateFactory$.

Analiza e një skedar çertifikimi $X.509$ me $CertificateFactory$:

```
CertificateFactory cf = CertificateFactory.getInstance("X.509");
InputStream in = new FileInputStream("certificate.cer");
X509Certificate cert = (X509Certificate) cf.generateCertificate(in);
```

Për të krijuar një $CertificateFactory$, ne e kalojmë $X.509$ si lloj metode për $getInstance()$, dhe pastaj thirrim $generateCertificate()$, duke kaluar një $InputStream$ nga e cila lexohet. Për shkak se kjo është një çertifikatë $X.509$, objekti i marrë mund të hidhet në mënyrë të sigurtë në $java.security.cert.X509Certificate$. Nëse skedari i lexuar përfshin çertifikata të shumta që formojnë një zinxhir çertifikate, një objekt $CertPath$ mund të merret duke thirr metodën $generateCertPath()$. [25] Klasa $CertPathValidator$ përmbledh një algoritëm validimi të zinxhirit të çertifikimit siç përcaktohet nga Infrastruktura $Public - Key$ ($X.509$) ose standardi $PKIX$. Listimi i mëposhtëm tregon se si të përdoret $CertificateFactory$ dhe $CertPathValidator$ për të ndërtuar dhe vërtetuar një zinxhir çertifikatë. [25]

Ndërtimi dhe validimi i një zinxhir çertifikate me $CertPathValidator$:

```
CertPathValidator certPathValidator = CertPathValidator.getInstance("PKIX");
CertificateFactory cf = CertificateFactory.getInstance("X.509");
X509Certificate[] chain = getCertChain();
CertPath certPath = cf.generateCertPath(Arrays.asList(chain));
Set<TrustAnchor> trustAnchors = getTrustAnchors();
```

```
PKIXParameters result = new PKIXParameters(trustAnchors);
PKIXCertPathValidatorResult result = (PKIXCertPathValidatorResult)
certPathValidator.validate(certPath, pkixParams);
```

2.3.3 Ofruesit JCA të Android

Ofruesit e kriptografisë së *Android* bazohen në *JCA* dhe ndjekin arkitekturën e saj me disa përjashtime relativisht të vogla. Ndërsa komponentët e nivelit të ulët të *Android* përdorin direkt biblotekat e fshehta të kriptografisë (si psh. *OpenSSL*), *JCA* është *API* kryesor kriptografike dhe përdoret nga komponentët e sistemit dhe aplikacionet e palëve të treta. *Android* ka tre ofruesit kryesor të *JCA* që përfshijnë implementimin e klasave të përshkruara në seksionin e mëparshëm dhe dy ofruesve të *Java Secure Socket Extension (JSSE)* që implementojnë funksionalitetin e *SSL*. [25] Implementimi i bibliotekës runtime të *Android* rrjedh nga projekti *Apache Harmony* i veçuar, i cili përfshin gjithashtu një ofrues të kufizuar *JCA* thjeshtë i quajtur *Crypto* që siguron implementime për shërbime bazike kriptografike si gjenerimi i numrave të rastit, *hashing*, dhe nënshkrimet dixhitale. *Crypto* është ende i përfshirë në *Android* për pajtueshmërinë prapavajtëse por ka prioritetin më të ulët të të gjithë ofruesve *JCA*, kështu që implementimet e klasës së *Crypto* nuk kthehen nëse kërkohet shprehimisht. Tabela e mëposhtme tregon klasat e *Crypto* algoritmeve që mbështesin librarit e *Crypto*. [25]

Emri i klasës së motorit	**Algoritmet e mbështetura**
KeyFactory	DSA
MessageDigest	SHA-1
SecureRandom	SHA1PRNG
Signature	SHA1WithDSA

Tabela 2.2: Algoritmet e mbështetura nga Crypto Ofruesi si nga Android 4.4.4

Para versionit të *Android* 4.0, i vetmi ofrues i plotë *JCA* në *Android* ishte ofruesi *Bouncy Castle*. Ofruesi *Bouncy Castle* është pjesë e *Bouncy Castle Crypto API*-ve, një sërë implementimesh *Java* me burim të hapur të algoritmeve dhe protokolleve kriptografike. *Android* përfshin një version të modifikuar të ofruesit

Bouncy Castle, i cili rrjedhë nga versioni i zakonshëm duke aplikuar një grup të *Android* pjesëve të veçanta. Ato pjesë mbahen në pemën burimore të *Android* dhë përditësohen për çdo lansim të ri të ofruesit *Bouncy Castle*. Dallimet kryesore nga versioni kryesor janë përmbledhur më poshtë.

- Algoritmet, mënyrat dhe parametrat e algoritmeve që nuk mbështeten nga implementimi i referencës së $Java$ (RI) janë hequr ($RIPEMD$, $SHA-224$, $GOST3411$, $Twofish$, $CMAC$, $El\ Gamal$, $RSA-PSS$, $ECMQV$ dhe kështu më radhë).
- Algorimet e pasigurta të tilla si $MD2$ dhe $RC2$ janë hequr.
- Implementimet e bazuara në $Java$ të $MD5$ dhe familjes SHA të algoritmeve *digest* janë zëvendësuar me një implementim autokton.
- Disa algoritme PBE janë hequr (për shembull, $PBEWithHmacSHA256$).
- Mbështetja për të hyrë në çertifikatat e ruajtura në $LDAP$ është hequr.
- Mbështetja për listat e zeza të çertifikatës është shtuar.
- Jaën bërë optimizime të ndryshme të përformancës.
- Emri i paketës është ndryshuar në *com.android.org.bouncycastle* për të shmangur konfliktet me aplikacionet pako në *Bouncy Castle* (që nga *Android* 3.0).

Klasat dhe algoritmet e mbështetura nga *Bouncy Castle* Ofruesi i *Android* si në versionin 4.4.4 janë të listuara mëposhtë në tabelë. [25]

Emri i klasës	**Algoritmet e mbështetura**
CertPathBuilder PKIX	CertPathBuilder PKIX
CertPathValidator PKIX	CertPathValidator PKIX
CertStore Collection	CertStore Collection
CertificateFactory X.509	CertificateFactory X.509
Cipher	AES
	AESWRAP
	ARC4
	BLOWFISH
	DES
	DESEDE
	DESEDEWRAP
	PBEWITHMD5AND128BITAES-CBC-OPENSSL
	PBEWITHMD5AND192BITAES-CBC-OPENSSL
	PBEWITHMD5AND256BITAES-CBC-OPENSSL
	PBEWITHMD5ANDDES
	PBEWITHMD5ANDRC2
	PBEWITHSHA1ANDDES
	PBEWITHSHA1ANDRC2
	PBEWITHSHA256AND128BITAES-CBC-BC
	PBEWITHSHA256AND192BITAES-CBC-BC
	PBEWITHSHA256AND256BITAES-CBC-BC
	PBEWITHSHAAND128BITAES-CBC-BC
	PBEWITHSHAAND128BITRC2-CBC
	PBEWITHSHAAND128BITRC4
	PBEWITHSHAAND192BITAES-CBC-BC
	PBEWITHSHAAND2-KEYTRIPLEDES-CBC
	PBEWITHSHAAND256BITAES-CBC-BC
	PBEWITHSHAAND3-KEYTRIPLEDES-CBC
	PBEWITHSHAAND40BITRC2-CBC
	PBEWITHSHAAND40BITRC4
	PBEWITHSHAANDTWOFISH-CBC
	RSA
KeyAgreement	DH
	ECDH
KeyFactory	DH
	DSA
	EC
	RSA
KeyGenerator	AES
	ARC4
	BLOWFISH
	DES
	DESEDE
	HMACMD5
	HMACSHA1
	HMACSHA256

Emri i klasës	**Algoritmet e mbështetura**
KeyGenerator	AES
	HMACSHA384
	HMACSHA512
KeyPairGenerator	DH
	DSA
	EC
	RSA
KeyStore	BKS (default)
	BouncyCastle
	PKCS12
Mac	HMACMD5
	HMACSHA1
KeyPairGenerator	DH
	DSA
	EC
	RSA
KeyStore	BKS (default)
	BouncyCastle
	PKCS12
Mac	HMACMD5
	HMACSHA1
	HMACSHA256
	HMACSHA384
	HMACSHA512
	PBEWITHHMACSHA
	PBEWITHHMACSHA1
MessageDigest	MD5
	SHA-1
	SHA-256
	SHA-384
	SHA-512
SecretKeyFactory	DES
	DESEDE
	PBEWITHHMACSHA1
	PBEWITHMD5AND128BITAES-CBC-OPENSSL
	PBEWITHMD5AND192BITAES-CBC-OPENSSL
	PBEWITHMD5AND256BITAES-CBC-OPENSSL
	PBEWITHMD5ANDDES
	PBEWITHMD5ANDRC2
	PBEWITHSHA1ANDDES
	PBEWITHSHA1ANDRC2
	PBEWITHSHA256AND128BITAES-CBC-BC
	PBEWITHSHA256AND192BITAES-CBC-BC
	PBEWITHSHA256AND256BITAES-CBC-BC
	PBEWITHSHAAND128BITAES-CBC-BC
	PBEWITHSHAAND128BITRC2-CBC

Emri i klasës	**Algoritmet e mbështetura**
SecretKeyFactory	DES
	PBEWITHSHAAND128BITRC4
	PBEWITHSHAAND192BITAES-CBC-BC
	PBEWITHSHAAND2-KEYTRIPLEDES-CBC
	PBEWITHSHAAND256BITAES-CBC-BC
	PBEWITHSHAAND3-KEYTRIPLEDES-CBC
	PBEWITHSHAAND40BITRC2-CBC
	PBEWITHSHAAND40BITRC4
	PBEWITHSHAANDTWOFISH-CBC
	PBKDF2WithHmacSHA1
	PBKDF2WithHmacSHA1And8BIT
Signature	ECDSA
	MD5WITHRSA
	NONEWITHDSA
	NONEWithECDSA
	SHA1WITHRSA
	SHA1WithDSA

Tabela 2.3: Algoritmet e mbështetura nga Ofruesi Bouncy Castle i Android si Android 4.4.4

Për të përmirësuar më tej përformancën kriptografike, numri i klasave dhe algoritmeve të mbështetur në ofruesin autokton *AndroidOpenSSL* ka qenë vazhdimisht në rritje me secilin lansim që nga versioni 4.0. Fillimisht, *AndroidOpenSSL* u përdor vetëm për të implementuar bazat *SSL*, por si nga *Android* 4.4, ajo mbulon shumicën e funksionalitetit të ofruar nga *Bouncy Castle*. Për shkak se është ofruesi më i preferuar (me prioritetin më të lartë, 1), klasat që nuk kërkojnë në mënyrë ekspicite *Bouncy Castle* marrin një implementim nga ofruesi *AndroidOpenSSL*. Siç nënkupton emri, funksionaliteti i tij kriptografik sigurohet nga biblioteka e *OpenSSL*. Implementimi i ofruesve përdor *JNI* për të lidhur kodin e origjinës së *OpenSSL*-it me klasat *Java SPI* që kërkohen për të implementuar një ofrues *JCA*. Pjesa më e madhe e implementimit është në klasën *NativeCrypto Java*, e cila thirret nga shumica e klasave *SPI*. *AndroidOpenSSL* është pjesë e bibliotekës *Libcore* të *Android*, e cila implementon pjesën kryesore të bibliotekës runtime të *Android*. Duke filluar me *Android* 4.4, *AndroidOpenSSL* është çvendosur nga *Libcore* në mënyrë që të mund të përpilohet si një bibliotekë e pavarur dhe të përfshihet në aplikacione që kërkojnë një implementim të qëndrueshëm kriptografik

që nuk varet nga versioni i platformës. Ofruesi i pavarur quhet *Conscrypt* dhe gjendet në paketën *org.conscrypt*, i riemëruar në *com.android.org.conscrypt* kur është ndërtuar si pjesë e platformës *Android*. Klasat dhe algoritmet të mbështetura nga ofruesi i *AndroidOpenSSL* si në versionin 4.4.4 janë të listuara në tabelën e mëposhtme. [25]

Emri i klasës së motorit	**Algoritmet e mbështetura**
CertificateFactory X509	CertificateFactory X509
Cipher	AES/CBC/NoPadding
	AES/CBC/PKCS5Padding
	AES/CFB/NoPadding
	AES/CTR/NoPadding
	AES/ECB/NoPadding
	AES/ECB/PKCS5Padding
	AES/OFB/NoPadding
	ARC4
	DESEDE/CBC/NoPadding
	DESEDE/CBC/PKCS5Padding
	DESEDE/CFB/NoPadding
	DESEDE/ECB/NoPadding
	DESEDE/ECB/PKCS5Padding
	DESEDE/OFB/NoPadding
	RSA/ECB/NoPadding
	RSA/ECB/PKCS1Padding
KeyAgreement	ECDH
KeyFactory	DSA
	EC
	RSA
KeyPairGenerator	DSA
	EC
	RSA
Mac	HmacMD5
	HmacSHA1
	HmacSHA256
MessageDigest	MD5
	SHA-1
	SHA-256
	SHA-384
	SHA-512
SecureRandom	SHA1PRNG
Signature	ECDSA
	MD5WithRSA
	NONEWithRSA
	SHA1WithRSA

Emri i klasës së motorit	**Algoritmet e mbështetura**
Signature	ECDSA
	SHA1WithDSA
	SHA256WithRSA
	SHA256WithECDSA
	SHA384WithRSA
	SHA384WithECDSA
	SHA512WithRSA
	SHA512WithECDSA

Tabela 2.4: Algoritmet e përkrahura nga Ofruesi AndroidOpenSSL si nga Android 4.4.4

OpenSSL është një mjet kriptografik me burim të hapur që implementon protokollet *SSL* dhe *TLS* dhe përdoret gjerësisht si një bibliotekë kriptografike me qëllim të përgjithshëm. Ai është përfshirë në *Android* si një bibliotekë sistemi dhe përdoret për të implementuar ofruesin e *AndroidOpenSSL JCA* që u prezantua në *"Ofruesi AndroidOpenSSL"* më herët, si dhe nga disa komponentë të tjerë të sistemit. Lansime të ndryshme në *Android* përdorin versione të ndryshme të *OpenSSL*, me një implementim të caktuar të pjesëve. Prandaj, *Android* nuk ofron një *OpenSSL API* të qëndrueshëm publik, kështu që aplikacionet që duhet të përdorin *OpenSSL* duhet të përfshijnë bibliotekën dhe nuk lidhen me versionin e sistemit. *API* e vetme kriptografike është ajo e *JCA*, e cila ofron një ndërfaqe të qëndrueshme të çkyçur nga implementimi themelor. [25]

2.3.4 Duke përdorë një Ofrues me Porosi

Përderisa ofruesit e integruar të *Android* mbulojnë algoritmet kriptografike më të përdorur gjerësisht, ato nuk mbështesin disa algoritme ekzistues dhe madje disa standarde më të reja. Aplikacionet *Android* mund të regjistrojnë ofruesit me porosi për përdorimin e tyre, por nuk mund të ndikojnë në ofruesit e sistemit të gjerë. Një nga ofruesit më të përdorur gjerësisht dhe të plotë me *JCA* është *Bouncy Castle*, gjithashtu baza e një prej ofruesve të integruar të *Android*. Megjithatë, siç është diskutuar në *"Ofruesit Bouncy Castle të Android"*, versioni i ngarkuar me *Android* ka hequr disa algoritma. Nëse keni nevojë të përdorni

ndonjë nga këto algoritme, mund të përpiqeni thjesht të lidhni bibliotekën e plotë të *Bouncy Castle* me aplikacionin tuaj – por kjo mund të shkaktojë konflikte në ngarkim të klasës, veçanarisht në versionet e *Android* më herët se 3.0, të cilat nuk ndryshojnë emrin e paketës së sistemit *Bouncy Castle*. Për të shmangur këtë, ju mund të ndryshoni paketën e rrënjëve të bibliotekës me një metodë të till si $jarjar$, ose të përdorni *Spongy Castle*. [25] *Spongy Castle* është një version i ripaketuar i *Bouncy Castle*. Ajo përmban të gjitha emrat e paketave nga $org.bouncycastle.*$ në $org.spongycaste.*$, në mënyrë që të shmangen konfliktet e ngarkuesve të klasës, dhe ndryshon emrin e ofruesit nga BC në SC. Asnjë emër i klasës nuk ndryshohet, kështu që API është i njejtë me *Bouncy Castle*. Për të përdorur *Spongy Castle*, ju thjesht duhet të regjistroni atë me kornizën JCA duke përdorur $Security.addProvider()$ ose $Security.insertProviderAt()$. Ju pastaj mund të kërkoni algoritme që nuk implementohen nga ofruesit e integruar të *Android* thjeshtë duke kaluar emrin e algoritmit tek metoda përkatëse $getInstance()$. Për të kërkuar në mënyrë eksplicite një implementim nga *Spongy Castle*, kaloni vargun SC si emrin e ofruesit. Nëse e paketoni bibliotekën e *Spongy Castle* me aplikacionin tuaj, mund të përdorni direkt edhe API-në e lehtë kriptografië të *Bouncy Castle* pa kaluar nëpër klasat e motorëve të JCA. Përveç kësaj, disa operacione kriptografike, të tilla si nënshkrimi i një çertifikate $X.509$ ose krijimi i një mesazhi $S/MIME$, nuk kanë API JCA përputhëse dhe mund të kryhen vetëm dukë përdorur API-të e *Bouncy Castle* të nivelit më të ulët. Listimi i mëposhtëm tregon se si të regjistrohet ofruesi *Spongy Castle* dhe të kërkojë një implementim të nënshkrimit $RSA - PSS$, i cili nuk mbështetet nga asnjë prej ofruesve të ndërtuar në JCA të *Android*. [25] Regjistrimi dhe përdorimi i ofruesit *Spongy Castle*:

```
static {
        Security.insertProviderAt(
        new org.spongycastle.jce.provider.BouncyCastleProvider(), 1);
}
Signature sig = Signature.getInstance("SHA1WithRSA/PSS", "SC");
```

2.4 Proceset

Cikli i jetës së zhvillimit të softuerit ($SDLC$), i njohur edhe si proces i zhvillimit të softuerit, është një kornizë që përcakton detyrat, aktivitetet dhe proceset e kryera në çdo hap të secilës fazë të një $SDLC$. Ajo ndiqet nga një ekip i zhvillimit dhe përbëhet nga një plan i detajuar që përshkruan si të zhvillohet, mirëmbahet dhe zëvendësohet softueri specifik. Përdorimi i përgjithshëm i një $SDLC$ përmirëson cilësin përfundimtare të softuerit dhe procesit të përgjithshëm të zhvillimit [26]. Cikli i jetës së zhvillimit të softuerit të sigurtë ($SSDLC$) është një $SDLC$ me aktivitete shtesë të lidhura me sigurinë, të tilla si përcaktimi i kërkesave të sigurisë, krijimi i një model kërcënimi, kryerja e analizës së kodeve statike ose dinamike. Përdorimi i një $SSDLC$ ndihmon në ndërtimin e softuerit më të sigurtë dhe adresimin e kërkesave të pajtueshmërisë së sigurisë duke reduktuar koston e zhvillimit [27].

2.4.1 Përmbledhje e një SDLC ekzistuese

Aktualisht, nuk ekziston asnjë udhëzues ose $SDLC$ i krijuar nga *Google Inc.*, që do të ishte në shënjestër në platformën *Android*. Prandaj, për të hartuar një $SSDLC$ së pari duhet të shikoni në zgjidhjet ekzistuese dhe pastaj t'i përshtatni ose t'i kombinoni ato. *Microsoft SDLC* është një proces i zhvillimit të softuerit që i ndihmon zhvilluesit të ndërtojnë softuere më të sigurta dhe të adresojnë kërkesat e pajtueshmërisë së sigurisë duke reduktuar koston e zhvillimit [27]. *Microsoft SDLC* përmban 7 faza në një sekuencë, shumica e të cilave përmbajnë 3 detyra për të kryer:

- **Trajnimi** – Një parakusht për përdorimin e $SDLC$. Konceptet themelore për ndërtimin e softuerit më të mirë që duhet të mbulohen përfshijnë dizajnin e sigurtë, modelimin e kërcënimeve, kodimin e sigurtë, testimin e sigurisë dhe praktikat më të mira në lidhje me privatësinë [28].

- **Kërkesat** – Koha më e mirë për të shqyrtuar çështjet themelore të sigurisë dhe privatësisë, për të analizuar mënyrën e harmonizimit të cilësisë dhe kërkesave rregullatore me kostot dhe nevojat e biznesit.
  - **Vendos kërkesat e sigurisë** – Përcaktimi dhe integrimi i kërkesave të sigurisë dhe privatësisë në fillim minimizon ndërprerjet ndaj planeve dhe orareve. Kjo detyrë përfshinë përcaktimin e ekspertëve të sigurisë, përcaktimin e kriterëve të sigurisë dhe privatësisë dhe vendosjen e sistemit të cenueshmërisë/gjurmimit të sendeve të punës.
  - **Kirjo rrjetë cilësore porte/virus** – Nivelet e pranueshme minimale të sigurisë dhe cilësisë së privatësisë ndihmojnë në identifikimin dhe rregullimin e defekteve të sigurisë gjatë zhvillimit dhe aplikimit të standardeve në tërë projektin. Vendosja e një rrjeti kuptimplotë përfshin përcaktimin e qartë të pragut të ashpërsisë së dobësive të sigurisë. Për shembull, asnjë dobësi e shënuar si\"*kritike*" nuk do të njihet në kohën e lansimit.
  - **Krijo vlerësimet e rrezikut për sigurinë dhe privatësinë** – Shqyrtimi i dizajnit të softuerit bazuar në kërkesat ndihmon për të identifikuar se cilat pjesë do të kërkojnë modelimin e kërcënimeve dhe shqyrtimet e dizajnit të sigurisë [29].
- **Dizajni** – Kritik për krijimin e praktikave më të mira rreth dizajnit dhe specifikimit funksional dhe kryerjes së analizës së rrezikut.
  - **Vendos kërkesat e dizajnimit** – Trajtimi i shqetësimeve të sigurisë dhe privatësisë së hershme ndihmon në minimizimin e rrezikut të ndërprerjeve të orarit dhe reduktimin e shpenzimeve. Kjo përfshin një specifikim të saktë dhe të plotë të dizajnit, duke përfshirë kërkesat minimale të dizajnit kriptografik dhe një rishikim të specifikimeve.
  - **Kryej analiza/reduktime të sipërfaqes së sulmeve** – Duke analizuar në tërësi sipërfaqen e përgjithshme të sulmit, paaftësimin ose kufizimin e qasjes në shërbimet e sistemit, duke implementuar parimin

e privilegjit më të vogël, dhe duke përdorur mbrojtje të shtresuar ul mundësitë për sulmuesit të shfrytëzojnë një dobësi.

  - **Përdor modelimin e kërcënimeve** – Aplikimi i një qasjeje të strukturuar ndaj skenarëve të kërcënimeve gjatë fazës se dizajnimit ndihmon një ekip në mënyrë më efikase dhe më pak të shtrenjtë të identifikojë dobësitë e sigurisë, të përcaktojë rreziqet nga këto kërcënime dhe të krijojë zbutje adekuate [30].

- **Implementimi** – Fokusi i kësaj faze është krijimi i praktikave më të mira për zbulimin dhe largimin e çështjeve të sigurisë dhe implementimin e funksionit të kërkuar.

  - **Përdor vegla të aprovuara** – Mbajtja e një liste të përditësuar të veglave të aprovuara dhe kontrolleve të sigurisë ndihmon automatizimin dhe implementimin e praktikave më të mira të sigurisë dhe përfshirjen e funksionalitetit dhe mbrojtjeve të reja të analizës së sigurisë.

  - **Kundërshto funksionet e pasigurta** – Analizimi i funksionëve të projektit dhe *API*-të që ndalojnë ato që janë të përcaktuar të jenë të pasigurta zvogëlojnë gabimet (*bug*) potenciale të sigurisë me kosto të vogël inxhinierike. Për shembull, duke kontrolluar një kod për funskionet në listën e ndaluar dhe duke i zëvendësuar ato me alternativa më të sigurta.

  - **Kryej analiza statike të kodit** – Analiza e kodit burimor para përpilimit siguron një metodë të shkallëzuar të rishikimit të kodit të sigurisë dhe ndihmon në sigurimin e ndjekjes së politikave të kodimit të sigurtë [31].

- **Verifikimi** – Faza që siguron që kodi i plotëson parimet e sigurisë dhe të privatësisë të përcaktuara në fazat e mëparshme.

  - **Kryej analiza dinamike** – Verifikimi i kohës së programimit duke përdorur veglat që monitorojnë sjelljen e aplikacionit për korrupsionin e memorjes, çështjet e privilegjit dhe problemet të tjera kritike të sigurisë.

  - **Kryej testimin push (fuzz)** – Nxitjen e dështimit të programit duke paraqitur qëllimisht informacion të formuluar keq ose të rastësishëm.
  - **Administro shqyrtimin e sipërfaqes së sulmit** – Rishikimi i sipërfaqes së sulmit pas përfundimit të kodit ndihmon për të siguruar që çdo ndryshim i dizajnit ose implementimit të ndryshojë në një aplikacion ose sistem është marr parasysh dhe se çdo vektori i ri sulmesh i krijuar si rezultat i ndryshimeve është shqyrtuar dhe implementuar duke përsfhirë modele kërcënimi [32].

- **Lansimi** – Përgatitja e projektit për lansim publik, planifikime për të kryer në mënyrë efektive detyrat e servisimit pas lansimit, të tilla si trajtimi i reagimeve të përdoruesve ose fiksimin e gabimeve (*bug*) dhe adresimi i dobësive të sigurisë ose të privatësisë që mund të ndodhin më vonë.
  - **Kirjo një plan për përgjigje të incidentit** – Identifikimi i kontakteve të përshtatshme emergjente të sigurisë dhe vendosja e planeve të shërbimit të sigurisë për kodin e trashëguar nga grupet e tjera ose kodin e lincensuar të palës së tretë.
  - **Kryej shqyrtimin përfundimtar të sigurisë** – Shqyrtimi i të gjitha aktiviteteve të sigurisë që janë kryer. Ajo përfshin shqyrtimin e modeleve të kërcënimeve, rezultateve të veglave dhe përformancës pas portave dhe shiritave të gabimeve (*bug*) të cilësisë së përcaktuar gjatë fazës së kërkesës.
  - **Çertifiko, lanso dhe arkivo** – Arkivimi i të gjitha specifikimeve, kodit burimor, binareve, simboleve private, modeleve të kërcënimit, dokumentacionit, planëve të reagimit emergjent, licensave dhe kushteve të servisimit për çdo softuer të palëve të treta para lansimit, ndihmon në garantimin e sigurisë dhe kërkesave të privatësisë [33].

- **Përgjigje** – Një fazë pas lansimit që qendrat në ekipin e zhvillimit janë në gjendje dhe në dispozicion për t'iu përgjigjur në mënyrë të përshtatshme çdo raporti të kërcënimeve dhe dobësive në zhvillim [34].

$MSDLC$ për zhvillim të shkathët është riorganizuar në tri kategori. Secila kategori përcakton se sa shpesh duhet të kryhen detyrat nga $MSDLC$. Kategoritë me detyrat janë:

- **Praktikat e çdo sprinti** – Detyrat e përcaktuara nga kjo kategori duhet të kryhet çdo lansim *sprint*.
  - Përdor modelimin e kërcënimeve.
  - Përdor veglat e aprovuara.
  - Kundërshto funksionet e pasigurta.
  - Kryej analiza statike të kodit.
  - Kryej shqyrtimin përfundimtar të sigurisë.
  - Çertifikon lansimin dhe arkivimin [35].
- **Praktikat e kovës** – Praktikat më të rëndësishme të sigurisë që duhet të plotësohen në baza të rregullta, por mund të përhapen nëpër *sprint*-a të shumta gjatë jetës.
  - Krijo rrjetë cilësore porte/virus.
  - Kryej analiza dinamike.
  - Kryej testimin *push* ($fuzz$).
  - Kryej shqyrtimin e sipërfaqes së sulmeve [36].
- **Praktikat një herë** – Praktikat e sigurisë themelore që duhet të krijohen në fillim të çdo projekti të ri të shkathët.
  - Vendos kërkesat e sigurisë.
  - Kryej vlerësimet e rrezikut për sigurinë dhe privatësinë.
  - Vendos kërkesat e dizajnimit.
  - Kryej analiza/reduktim të sipërfaqes së sulmeve.
  - Krijo një plan për përgjigje të incidentit [37].

XAMARIN SDLC mobile përmban pesë faza dhe nuk përshkruhet si *Microsoft SDLC*. Çdo fazë përmban më shumë konsiderata dhe përshkrime se çfarë duhet bërë, në vend se detyrat specifike. Gjithashtu, *XAMARIN SDLC mobile* përcakton ndonjë detyrë të lidhur me sigurinë.

- **Fillimi** – Kjo fazë ka të bëjë me përcaktimin dhe përsosjen e idësë se një aplikacioni. Në këtë fazë duhet të parashtrohen disa pyetje themelore si:
  - A ka aplikacione të ngjashme?
  - Pse është i ndryshëm ky dhe më i mirë?
  - Cila infrastrukturë ekzistuese do të integrohet apo zgjerohet?
  - Çfarë vlere i sjell ky aplikacion përdoruesit?
  - Si do ta përdorin atë?
  - Si funksionon ky aplikacion në një rrethanë mobile?
  - A mund të përdorin teknologjitë *mobile* si vetëdijesimi i lokacionit për të rritur vlerën?

Për të ndihmuar në dizajnimin e funksionalitetit të një aplikacioni, mund të jetë e dobishme të ndërtohet një diagram $Use - Case$.

- **Dizajni** – Faza e dizajnimit përbëhet nga përcaktimi i aplikacionëve *User Experience (UX)* dhe duke e kthyer atë në një *User Interface (UI)*, zakonisht me ndihmën e një dizajneri grafik. UX mund të bëhet përmes $wireframes$ (një model tridimensional skeletor) dhe zakonisht rezulton në një skicë të zezë dhe të bardhë të një aplikacioni. UI është më i rafinuar, përmban ngjyra, grafikë dhe forma. Të dy UX dhe UI duhet të ndjekin udhëzimet e dizajnimit të platformës dhe duhet të marrin në konsideratë kufizime fizike të pajisjes.
- **Zhvillimi** – Faza më burimore-intensive, që aktualisht është duke ndërtuar aplikacionin. Shpesh fillon shumë herët, duke dhënë një prototip të punës që validon funksionalitetin, supozimet dhe jep një kuptim të qëllimit të punës.

- **Stabilizimi** – Procesi i përpunimit të gabimeve (*bugs*) në aplikacion, nga pikëpamja e funksionalitetit, përdorshmërisë dhe përformancës. Duhet të fillojë sa më shpejtë që të jetë e mundur brenda procesit të zhvillimit. Ajo përcakton katër faza e aplikacionit:
  - *Prototipi* – Aplikacioni ende është në fazën provuese të konceptit. Përmban vetëm funksionalitetin bazë ose pjesë specifike. Mangësitë kryesore janë të pranishme.
  - *Alfa* – Funksioni bazë është i plotë. Funskionaliteti i jashtëm mund të mungojë ende. Përmban gabime të shumta.
  - *Beta* – Shumica e funksionalitetit është e plotë dhe ka pasur të paktën testimin e dritës dhe rregullimin e gabimeve. Çështjet e rëndësishme ende mund të jenë të pranishme.
  - *Lanso kandidatin* – Të gjitha funksionet janë të plota dhe të testuara.
- **Shpërndarja** – Sapo të bëhet stabilizimi, aplikacioni mund të lansohet [38].

Ngjashëm me *Xamarin SDLC*, përcakton pesë faza, secila që përmban përshkrimin e aktiviteteve që duhet të bëhen. Megjithatë, *WhiteHat SDLC Sigurisë Mobile* përmban një integrim të sigurisë.

- **Kërkesat** – Me zhvillimin e kërkesave të softuerit, duhet të përcaktohen edhe kërkesat përkatëse të sigurisë. Kjo fazë gjithashtu thekson nevojën për një trajnim bazik të sigurisë, ku shkallëzueshmëria dhe përsëritshmëria duhet të jenë aspekte kritike.
- **Dizajni** – Pasi të dihen kërkesat, duhet të krijohen një arkitekturë që i korrespondon këtyre kërkesave. Kontrollet e nevojshme të sigurisë duhet të identifikohen dhe shpërndahen si pjesë e aplikacionit. Kjo fazë gjithashtu duhet të përfshijë modelimin e kërcënimeve për të identifikuar dhe adresuar rreziqet që lidhen me aplikacion.
- **Implementimi** – Pasi të gjitha kërkesat janë përcaktuar dhe një dizajn arkitektural është në vend, implementimi i aplikacionit fillon. Idealisht,

zhvilluesit duhet të marrin reagime të sigurisë gjatë programimit, që do të thotë se vlerësimet e sigurisë së kodit burimor statik dhe analiza statike e kodit duhet të rrjedhin sa më shpesh që të jetë e mundur. Marrja e reagimeve sa më shpesh që të jetë e mundur ndihmon në identifikimin e çështjeve në kohë afërsisht reale dhe siguron që aplikacioni të mos ndërtohet mbi kodin e gabuar.

- **Testimi i pranimit të përdoruesit (UAT) / Sigurimi i cilësisë (QA)** – Testimi i kodit për të siguruar se vepron ashtu siç pritet. Kërkesat funksionale dhe të sigurisë duhet të jenë pjesë e testimit. Kjo fazë duhet të përfshijë analiza dinamike.

- **Prodhimi** – Në fazën e vendosjes, testimi i vazhdueshëm është thelbësor për ruajtjen e sigurisë. Çdo përditësim i kodit gjithashtu duhet t'i nënshtrohet analizës së kodit statik, sigurimit të cilësisë dhe testimit të prodhimit [39].

Në *Multi Konferencën Ndërkombëtare të Inxhinierëve dhe Shkenctarëve Kompjuterikë 2014*, në Mars të vitit 2014 në *Hong Kong* u botua një vepër që synonte ciklin jetësor të zhvillimit të aplikacionëve mobile, qka është thelbësore një $SSDLC$. $SSDLC$ përbëhet nga shtatë faza, ku secila përshkruan dhe propozon detyra që duhet të bëhen.

- **Identifikimi** – Faza e parë që duhet të rezultojë në një përmirësim të aplikacionit ekzistues, duke menduar idenë, duke analizuar aplikacione ose funkisone të ngjashme. Rezultatet duhet të dokumentohen.

- **Dizajni** – Ideja është zhvilluar në një dizajn fillestar të aplikacionit. Detyrat në rend kronologjik duhet të përfshijnë:

  - Zgjidhni platformën ose platformat e synuara.
  - Zgjidhni një lloj lansimi pa pagesë, provues, premium.
  - Funskionaliteti i aplikacionit ndahet në module dhe prototipa.
  - Përcaktoni kërkesat funksionale.

  - Krijoni arkitekturën softuerike të aplikacionit.
  - Përcaktoni prototipe dhe modulet e lidhura.
  - Krijoni *storyboards* (sekuencë vizatimesh) për ndërfaqen e përdoruesit.

- **Zhvillimi** – Në këtë fazë, aplikacioni është implementuar. Procesi i zhvillimit mund të jetë në një fazë të kodimit për kërkesat funksionale ose kodimin për kërkesat e ndërfaqes së përdoruesit. Një zhvillim paralel mund të bëhet për modulet e prototipit të njëjtë që janë të pavarur nga njëri tjetri. Më pas, këto module mund të integrohen. Së fundi, dokumentacioni i zhvillimit është krijuar dhe përcjellë në fazën e prototipimit.

- **Prototipi** – Analizimi i kërkesave funksionale të secilit prototip duke testuar dhe dërguar prototipet tek klienti për reagime. Pas pranimit të reagimeve, ndryshimet e kërkuara implementohen përmes fazës së zhvillimit. Prototipi i dytë integrohet me të parën dhe pastaj i dërgohet klientit për reagime shtesë. Ky proces përsëritet derisa prototipi përfundimtar të jetë gati.

- **Testimi** – Një nga fazat më të rëndësishme të çdo *SDL*. Testimi duhet të kryhet duke përdorur si emulatorin/simulatorin ashtu edhe një pajisje të vërtetë. Pajisjet e testimit duhet të jenë me dendësi të ndryshme të ekranit, versionet e *Android API* dhe modelet. Rastet e testimit duhet të dokumentohen dhe të përcillen tek klienti për reagime.

- **Shpërndarja** – Është faza përfundimtare e procesit të zhvillimit. Para lansimit për shkarkim apo përdorim nga përdoruesit, aplikacioni duhet të kontrollohet nësë ai plotëson rregulloret dhe kushtet. Gjithashtu, të gjitha skedaret e dokumentimit të udhëtimit dhe komentet duhet të hiqen.

- **Mirëmbajtja** – Një proces i vazhdueshëm që përbëhet nga mbledhja e reagimeve të përdoruesit, korigjimi i gabimeve, riparimi i çështjeve të sigurisë, përmirësimi i përformancës, funksionaliteti i shtuar dhe ripunimi i ndërfaqes së përdoruesit. Faza e mirëmbajtjes përfshin gjithashtu edhe marketingun e aplikacionit [40].

2.4.2 Modelimi i kërcënimeve

Modelimi i kërcënimeve është një qasje për të analizuar sigurinë e një aplikacioni. Ndihmon në identifikim, përcaktimin dhe adresimin e rreziqeve të sigurisë që lidhen me një aplikacion. Përfshirja e modelimit të kërcënimeve në $SDLC$ mund të ndihmojë për të siguruar që aplikacionet janë duke u zhvilluar me sigurinë e ndërtuar që nga fillimi. Modelimi i kërcënimeve i ofron recensuesit një kuptim më të madh të sisitemit dhe i lejon atij të shohë se ku janë pikat hyrëse të aplikacionit dhe cilat janë kërcënimet e lidhura. Kryerja e modelimit të kërcënimeve lejon përqendrimin e vëmendjes tek komponentët me rrezikun më të lartë të lidhur. Kritik për identifikimin e kërcënimeve është përdorimi i një metodologjie kategorizimi të kërcënimeve të tilla si $STRIDE$ dhe Korniza e Sigurisë së Aplikacionit, ndërsa përcaktimi i rrezikut të sigurisë mund të vlerësohet duke përdorur një model rreziku siç është $DREAD$[41]. $STRIDE$ është një metodologji e kategorizimit të kërcënimeve, duke identifikuar kërcënimet nga perspektiva e sulmuesve, që rrjedh nga një akronim i gjashtë kategorive të mëposhtme:

- **Spoofing** – Kur një proces ose entitet është diçka tjetër nga identiteti i pretenduar i saj. Për shembull, duke përdorur infromacionin e autentifikimit të blerjes ilegale të një përdoruesi tjetër për të kryer një detyrë në emrin e tij.

- **Tampering** – Një veprim i ndryshimit të *bit*-ëve. Ngatërrimi me një proces përfshin ndërrimin e *bit*-eve në procesin e funksionimit. Në mënyrë të ngjashme, ngatërrimi me një rrjedhë të dhënash përfshin ndërrimin e *bit*-eve në tel ose midis dy proceseve të përdorimit. Për shembull, ndryshimet e paautorizuara të bëra në të dhënat e vazhdueshme ose ndryshimi i të dhënave kur rrjedh midis dy kompjuterëve në një rrjet të hapur.

- **Repudiation** – Një kundërshtar që mohonte se diqka ndodhi në pa mundësin për të provuar ndryshe. Një shmebull mund të kryejë një oepracion të paligjshëm që nuk mund të gjurmohet nga sistemi.

- **Information disclosure** – Leximi i paautorizuar i informacionit konfidencial. Për shembull, leximi i të dhënave në tranzit ndërmjet dy kompjuterëve.

- **Denial of service** – Kur procesi ose një depo e të dhënave, nuk është në gjendje t'i shërbejë kërkesave hyrëse. Për shembull, duke kryer një sasi ekstreme të kërkesave për një kohë të shkurtër në një server derisa të mos jetë më në gjendje t'i shërbejë.

- **Elevation of privilege** – Një subjekt përdoruesi fiton aftësi ose privilegje të rritura duke përfituar nga një gabim implementimi. Një shembull mund të jetë një sulmues që ka deprtuar me sukses në sistem dhe është bërë pjesë e besueshme e sistemit[42].

Korniza e Sigurisë së Aplikacionit është kërcënim për kategorizimin e metodologjisë, duke identifikuar kërcënimet nga perspektiva e mbrojtësve. Ajo përcakton tetë lloje kërcënimi:

- **Autentifikimi** – Akti i konfirmimit të identitetit të një përdoruesi. Mund të parandalohet, për shmebull, duke përdorur kredencialet e koduara dhe argumentet e legalizimit, politikat e forta të fjalëkalimit ose duke përdorur një autentifikim të besuar të serverit në vend të SQL autentifikimit.

- **Autorizimi** – Caktimi i nivelit korrekt të privilegjeve ose burimeve për një përdorues. Aplikimi i parimit të privilegjit më të vogël ose përdorimi i kontrolleve të qasjes në bazë të roleve mund të ndihmojë për të kundërshtuar këtë çështje.

- **Menaxhimi i konfigurimit** – Kontabiliteti i duhur i skedarëve të konfigurimit. Ndërfaqet e administratorit dhe skedarët e konfigurimit duhet të kufizohen vetëm tek administratorët dhe të gjitha aktivitetet e administratorit duhet të auditohen dhe regjistrohen.

- **Mbrojtja e të dhënave në ruajte dhe tranzit** – Mbrojtja e të dhënave ose të dhënave të vazhdueshme në tranzit midis dy kompjuterëve, për shembull, duke përdorur algoritme të kriptimit të forcës dhe madhësisë së mjaftueshme.

- **Validimi i të dhënave dhe validimi i parametrave** – Konfirmimi i integritetit të të dhënave duke aplikuar kontrolle në llojin e të dhënave, formatin, gjatësin ose gamën. Asnjë vendim sigurie nuk duhet të bazohet në parametrat që mund të manipulohen.

- **Menaxhimi i gabimeve dhe menaxhimi i përjashtimeve** – Trajtimi i strukturuar i të gjitha përjashtimeve. Niveli i duhur i privilegjit duhet të rivendoset pas ndeshjes me një gabim dhe mesazhet e gabimit nuk duhet të përmbajnë ndonjë informacion të ndjeshëm.

- **Menaxhimi i përdoruesve dhe sesioneve** – Sigurimi se sesioni i një përdoruesi të vërtetuar nuk do të vjedhet ose përdoret nga një kundërshtar. Për shembull, sesionet duhet të përfundojnë pasi një përdorues të dalë jashtë dhe *cookies* nuk duhet të përmbajnë ndonjë informacion të ndjeshëm të ruajtur në tekst të qartë.

- **Auditimi dhe prerjet** – Niveli i duhur i auditimit dhe evidentimit të informacionit. Kontrollet e hyrjes dhe kontrollet e integritetit duhet të implementohen në skedarët e regjistrit dhe nuk duhet të regjistrojn informacione të ndjeshme, të tilla si fjalëkalime [41].

$DREAD$ është një model i rangut të kërcënimit me rrezik të krijuar nga $Microsoft$, i quajtur nga faktorët e rrezikut që përcakton [43]. Faktorizimi i rrezikut lejon caktimin e vlerave tek faktorët e ndryshëm të ndikimit të një kërcënimi. Për të përcaktuar renditjen e një kërcënimi, analisti i kërcënimit duhet t'i përgjigjet një pyetjeje për secilin faktor rreziku:

- **Damage potential (potenciali i dëmtimit)** – Sa i madh do të ishte dëmi nëse sulmi do të kishte sukses?

- **Reproducibility (gjenerimi)** – Sa e lehtë është të gjeneroni një sulm?

- **Exploitability (përdorshmëria)** – Sa kohë, përpjekje dhe ekspertizë nevojiten për të shfrytëzuar kërcënimin?

- **Affected users (përdoruesit e prekur)** – Çfarë përqindje e përdoruesve do të preken nga sulmi?
- **Discoverability (zbulueshmëria)** – Sa e lehtë është të zbulosh këtë kërcënim?

Rezultati i përgjithshëm i *DREAD* llogaritet duke caktuar një vlerë prej 1 deri 10 në secilën pyetje dhe duke pjestu shumën e tyre me 5. Sa më i lartë rezultati, aq më të larta janë rreziqet e kërcënimit [41]. Procesi i modelimit të kërcënimeve mund të dekompozohet në tre hapa:

- **Dekompozo aplikacionin** – Së pari, kërkohet një kuptim i aplikacionit dhe si ndërvepron me entitetet e jashtme. Kjo mund të ndahet në disa aktivitete:
  - Krijo diagramin e përdorimit (*use-case diagram*) për të kuptuar se si përdoret aplikacioni.
  - Identifiko pikat e hyrjes në aplikacion.
  - Identifiko pasuritë që sulmuesi do të jetë i interesuar.
  - Identifiko nivelet e besimit që përfaqësojnë të drejtat e qasjes që aplikacioni do t'u japë subjektëve të jashtme.
  - Informacioni i fituar më parë duhet të dokumentohet dhe përdoret për të prodhuar një diagram të rrjedhës së të dhënave (*data flow diagram*).
- **Përcaktoni dhe gradoni kërcënimet** – Përdorni kërcënimin duke kategorizuar metodologjinë dhe duke renditur kërcënime duke përdorur një model të rënditjes së rrezikshmërisë.
- **Përcaktoni kundërmasat dhe zbutjen** – Mungesa e mbrojtjes kundër një kërcënimi mund të tregojë një cenueshmëri, rreziku i ekspozimit të të cilit mund të zbutet me një kundërmasë, e cila mund të identifikohet duke përdorur një listë të hartës kërcënuese kundërmasash. Strategjia e zbutjes së rrezikut pastaj përcakton se si do të menaxhohen kërcënimet [41].

2.4.3 Analiza statike e kodit

Analiza statike e kodit (SA) zakonisht i referohet drejtimit të mjetëve të analizës së kodit statik që përpiqen të gjejnë dhe të nxjerrin në pah çështjet e mundshme në kodin burimor. Zakonisht kryhet në fazën e implementimit të SDL dhe njihet edhe si testim i bardhë. Kryerja e analizës bëhet duke përdorur teknikat e mëposhtme:

- **Analiza e rrjedhës së të dhënave** – Mbledh informacionin mbi kohën e duhur në lidhje me të dhënat në softuer ndërsa është në një fazë statike.
- **Grafiku i rrjedhës së kontrollit** – Një përafqësim abstrakt i një softueri, ku nyjet konsiderohen si blloqe. Nyjet janë të lidhura nga skajet, duke përfaqësuar hedhje midis blloqeve.
- **Analiza e infeksonit** – Përpjekja për të identifikuar variabla që janë infektuara me të hyra të kontrollueshme të përdoruesit dhe gjurmimi i tyre në funksione të mundshme të cenueshme dhe të njohura si një lavaman. Nëse variabla e infektuar merr kalimin në një lavaman pa u pastruar fillimisht ajo shënohet si një dobësi.
- **Analiza leksikore** – Sintaksa e kodit burimor konvertohet në tregues të informacionit, që duhet të jetë më e lehtë për t'u manipuluar.

Në mënyrë ideale, analiza statike e kodit do të gjente të gjitha gabimet me një shkallë të lartë besimi se ajo që është gjetur është me të vërtetë një e metë. Megjithatë, realiteti është shumë i ndryshëm, dhe SA shpesh raporton një numër të madh të polëve pozitive të gabuara. Mund të mos gjeni probleme të lidhura me skedarët e konfigurimit, pasi ato nuk janë të pranishme me kodin dhe shpesh nuk punojnë në një kod që nuk mund të përpilohet. Nga këndvështrimi i sigurisë, SA aktualisht mund të identifikojë vetëm një përqindje të vogël të defekteve të sigurisë dhe shpesh nuk mund të vërtetojë se këto të meta janë një cenueshmëri aktuale. Nga ana tjetër, SA është shumë e shkallëzuar, zakonisht shumë e konfigurueshme, dhe lehtë për t'u përdorur. Gjithashtu, çështjet më të zakonshme identifikohen me një nivel të lartë besimi dhe mund të vendosen pothuajse në çast [44].

2.4.4 Analiza dinamike e kodit

Analiza dinamike e kodit është testimi dhe vlerësimi i një program duke ekzekutuar të dhënat në kohë reale [45]. Ekzistojnë dy lloje kryesore të analizës dinamike [46]:

- **Skanimi i cenueshmërisë** – Një proces i automatizuar i identifikimit të dobësive të sigurisë në mënyrë që të përcaktohet nëse dhe ku një sistem mund të shfrytëzohet ose kërcënohet. Ajo përdorë softuer që kërkon çështje që bazohen në një bazë të dhënash me të meta të njohura, teston objektivat për shfaqjet e këtyre defekteve dhe më në fund gjeneron një raport të gjetjeve të tij [47].
- **Testimi i penetrimit** – Është një përpjekje për të vlerësuar sigurinë e një IT infrastrukture duke shfrytëzuar në mënyrë të sigurtë dobësitë dhe duke vlerësuar efikasitetin e mekanizmit mbrojtës. Qëllimi themelor është matja e fizibilitetit të sistemeve dhe vlerësimi i çdo pasojave të incidentëve të mundshme [48].

Android kërkon që të gjitha APK-të të nënshkruhen në mënyrë digjitale me një çertifikatë para se të mund të publikohen dhe të instalohen [49]. Nënshkrimi i aplikacionëve lejon identifikimin e autorit të aplikacionit dhe përditësimin e aplikacionit pa ndërfaqe dhe leje të ndërlikuara. Çdo aplikacion që funksionon në platformën *Android* duhet të nënshkruhet dhe ato që nuk janë do të refuzohen nga *Google Play* ose nga instaluesi i paketës në pajisjen *Android*. Nënshkrimi është hapi i parë për vendosjen e një aplikacioni në *Sandboxin* e Aplikacionit. Ajo përcakton se cila ID e përdoruesit është e lidhur me cilin aplikacion dhe siguron që një aplikacion nuk mund të hyjë në ndonjë tjetër përveç me mekanizmin IPC [2]. Kur një aplikacion është instaluar, Menaxheri i Paketëve verifikon që APK është nënshkruar siç duhet me çertifikatën e përfshirë në APK. Nëse çelësi publik i çertifikatës përputhet me çelësin kryesor të përdorur në çdo APK tjetër në pajisje, APK-ja e re ka mundësi në $manifest$ që do të ndajë një identifikues unik (UID) me APK-të e tjera të nënshkrurara në mënyrë të ngjashme. *Android* mbështet dy skema të nënshkrimit të aplikacionëve, një në bazë të nënshkrimit (skema v1)

të *Java Archive (JAR)* dhe *APK* Skemës së Nënshkrimit v2 (skema v2), e cila u prezantua në versionin *Android 7.0 (API versioni 24)*. Për pajtueshmërinë më të mirë, aplikacionet duhet të nënshkruhen me të dy skemat [2]. Nënshkrimi i skemës v1 nuk mbron disa pjesë të *APK*-së, të tilla si metadata (bashkësi e të dhënave) të paketuara. Verifikuesi i *APK*-së duhet të përpunojë shumë struktura të të dhënave të pa sigurta dhe pastaj të hedhë poshtë ato që nuk mbulohen nga nënshkrimi, ajo që ofron një sipërfaqe të madhe sulmesh. Për më tepër, verifikuesi *APK* duhet të shkyçë të gjitha hyrjet e ngjeshura, duke konsumuar më shumë kohë dhe memorie [2]. Për të adresuar këto çështje, u prezantua skema v2, ku përmbajtjet e *APK*-së janë prerë, nënshkruar dhe pastaj Blloku i Nënshkrimit të *APK*-së është futur në *APK*. Gjatë validimit, skedari *APK* trajtohet si një pikë dhe kontrolli i nënshkrimit kryhet në të gjithë skedarin, çka mundëson zbulimin e më shumë klasave të modifikimeve të paautorizuara. Ky format është gjithashtu kompatibil i prapambetur për aq kohë sa *APK* është gjithashtu e nënshkruar nga skema $v1$ [2]. Një çertifikatë me çelës publik, që përmbanë çelësin publik e një pale të çelësit publik apo privat, si dhe disa *metadata* tjera që identifikojnë pronarin e çelësit, bashkangjitet përmes veglës së nënshkrimit të *APK*-së. Çertifikata me çelës publik i shërben qëllimit të shoqërimit unik të *APK*-së tek zhvilluesi dhe tek çelësi privat përkatës. Një skedar që përmban një ose më shumë çelësa privatë quhet një *keystore*, që kërkohet në procesin e vetëm të nënshkrimit. I njejti *keystore* duhet të përdoret gjatë gjithë jetëgjatësisë së aplikacionit [49].

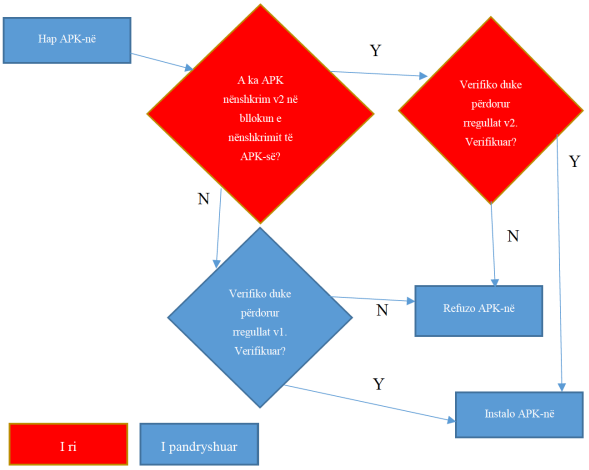

FIGURA 2.3: Procesi i verifikimit të nënshkrimeve APK. Ngjyra e kuqe shënon hapat e rinj të shtuar në skemën v2 [2]

AS-ja automatikisht nënshkruan një APK me një çertifikatë rregullimi të gjeneruar nga veglat e *Android SDK* gjatë drejtimit ose rregullimin e projektit nga IDE. Kjo çertifikatë nuk duhet të përdoret për publikimin pasi që është e pa sigurtë nga dizajni. Procesi i nënshkrimit të lansimit është një aktivitet i thjeshtë, pak aktivitet, i cili mund të konfigurohet më tej për t'u bërë me parametra të ndryshëm gjatë procesit të ndërtimit të aplikacionit [49]. Publikimi i aplikacionit është procesi i përgjithshëm që e bën aplikacionin e zhvilluar *Android* të disponueshëm për përdoruesit. Para një lansimi, duhet të kryhen disa detyra, të tilla si heqja e thirrjeve të regjistrit ose vendosja e atributit *debuggable* në $false$. Pas testimit të aplikacionit, përfshirë testimin e një serveri të mundshëm të largët, mund të nënshkruhet dhe të lansohet [50]. Procesi i lansimit të një aplikacioni zakonisht bëhet nëpërmjet një tregu, të tillë si *Google Play*, e cila është një platformë e fuqishme publikuese, duke ofruar vegla për të analizuar shitjet, për të identifikuar tendencat e tregut, për të kontrolluar shpërndarjen, për të përdorur një faturim në aplikacion ose licensim të aplikacionëve. Lansimi i aplikacionit në *Google Play* kërkon një sërë materialesh promovuese, konfigurimin e opsioneve dhe ngarkimin e aseteve [50]. Mundësitë e tjera për të lansuar aplikacionin përfshijnë një mundësi për të shkarkuar nga një faqe interneti ose për ta dërguar atë përmes postës elektronike. Megjithatë, kjo qasje kërkon që përdoruesit të kenë instalimin nga burimet e panjohura të aktivizuara në pajisjet e tyre [50].

2.5 Dizajnimi i një SSDLC

Cikli jetësor i dizajnuar gjithashtu duhet të caktojë aktivitete për personat e lidhur me projektin. Prandaj, pas përbërjes së njerëzve do të merren parasysh dhe përdoren:

- **Klienti** – Krijuesi i idesë për aplikacion, ose veçori.
- **Udhëheqësi i ekipit zhvillimor** – Shefi i ekipit të zhvillimit. Duhet të jetë i njohur me çdo aspect të projektit dhe duhet të jetë përgjegjës për tërë procesin e zhvillimit.

- **Ekipi i zhvillimit** – Zhvilluesit. Asnjë zhvillim apo vendim sigurie nuk duhet të bëhet nga ekipi zhvillimor pa udhëheqësin e ekipit zhvillimor ose ekipin e sigurisë.
- **Ekipi i sigurisë** – Ekspertët e sigurisë. Duhet të jetë i njohur me aspektin e sigurisë të platformës *Android*, të jetë në gjendje të përcaktojë kërkesat e sigurisë dhe të jetë në gjendje të verifikojë këto kërkesa.

SSDLC-ja e dizajnuar është krijuar për një model të shkathët të zhvillimit të softuerit, kështu që çdo detyrë përmban gjithashtu sa shpesh duhet të kryhet. Për këto qëllime do të përdoret kategorizimi:

- **Praktika një herëshe** – Detyrat e kryera vetëm një herë, zakonisht në fillim të projektit. Projekti mund të përmbajë shumë epike. Këto detyra mund të anashkalohen në *SSDLC* për epikën e ardhshme pasi të jenë bërë.
- **Praktikat e çdo epike** – Detyrat që duhet të kryhen për çdo epike. Një epike mund të shtrihet nëpër *sprints* të shumta. Këto detyra mund të anashkalohen në *SSDLC* për *sprint*-et e ardhshme sapo të jenë bërë.
- **Praktika çdo sprinti** – Detyrat që duhet të kryhen në çdo *sprint*.

Seksionet e mëposhtme do të përcaktojnë fazat në *SSDLC* dhe do t'i caktojnë aktivitetet e personave të veçantë. Detyrat krijohen sipas roleve të përcaktuara dhe përshkrimeve të tyre. Gjithashtu, për secilën fazë, një skemë që përfaqëson fazën krijohet duke përdorur shënimin e *Business Process Model Notation – BPMN*. Tre nga ciklet e jetës së rishikuar përbmajnë si fazë të parë fazën fillestare, e cila duhet të përmirësojë dhe thjeshtojë idenë. Megjithatë, *WhiteHat SDLC* Sigurisë Mobile gjithashtu thekson nevojën për një trajnim themelor të sigurisë, ku skalabiliteti dhe përsëritshmëria duhet të jenë aspekti kritik. Gjithashtu, *Microsoft SDLC* përcakton fazën e parë si Trajnimi, i cili duhet të mbulojë konceptet themelore dhe *SDL* të përdorura. Prandaj, faza e parë e përcaktuar, e cila është gjithashtu një kërkesë e nevojshme për përdorimin e *SSDLC*, është Trajnimi. Duke parë listën e *Top 10* të *OWASP* nga viti 2016, mund të shihet se shumë çështje vijnë

nga bazat e *Android*. Prandaj, trajnimi duhet të mbulojë të paktën bazat e sigurisë të *Android* dhe *OWASP Top 10*. Ekipi i zhvillimit duhet të jetë gjithashtu i njohur me Udhëzimet për Zhvillimin e Sigurisë të *OWASP Mobile*. Gjithashtu, nëse përdoren Standardet e Verifikimit të Sigurisë së Aplikacionit për *OWASP Mobile* dhe Udhëzuesi për Testimin e Sigurisë *Mobile* të $OWASP$, ato duhet të përfshihen në trajnimin për ekipin e sigurisë, për të siguruar që ato të përdoren në mënyrë korrekte dhe efektive. Të tre $SDLC$-të mobile përcaktojnë si fazë të parë, fazën fillestare, që duhet të përmirësojë dhe thjeshtëzojë funksionin ose aplikacionin duke ideizuar atë dhe duke analizuar aplikacione ose funksione të ngjashme. Gjithashtu, të dyja, *Microsoft SDLC* dhe *WhiteHat SDLC* Sigurisë Mobile përmendin se kërkesat për aplikacionin dhe sigurinë duhet të përcaktohen, atë që përshkruhet më mirë dhe në detaje në *Microsoft SDLC*. Megjithatë, vetëm *XAMARIN SDLC Mobile* përmend aspektin grafik të aplikacionit, i cili është gjithashtu kritik për fazën e zhvillimit, prandaj duhet të përfshihet në këtë fazë pasi ideja themelore e aplikacionit duhet të jetë tashmë e njohur. Të kombinuara së bashku, faza fillestare përmban detyrat e mëposhtme të kryera në një rend të caktuar:

- Udhëheqësi i ekipit duhet të krijojë një përshkrim bazë të funksionit që kërkohet nga klienti. I gjithë ekipi i zhvillimit, duke përfshirë udhëheqësin e ekipit, duhet të mendojnë ide, të mendojnë se si mund ai të përmirësohet dhe thjeshtëzohet. Shikoni zgjidhjet ekzistuese dhe krahasoni idenë me ta, a ofron zgjidhja aktuale ndonjë vlerë shtesë? Përmirësimet e propozuara duhet të diskutohen me klientin. Ideja përfundimtare duhet të dokumentohet në një mënyrë jo teknike, duke qenë e lehtë për tu lexuar dhe kuptuar edhe për një person të rregullt. Praktika e çdo epike.

- Pas përfundimit të përshkrimit bazë, duhet t'u dergohet dizajnerëve që të fillojnë të punojnë së skica të para për aplikacion. Krijimi i *Use Case* diagramit për këtë qëllim thjeshton procesin. Praktika e çdo epike.

- Merrni parasysh çështjet themelore të sigurisë dhe privatësisë. Analizoni se si të përshtatni kërkesat e cilësisë dhe rregullatorit me kostot dhe nevojat e biznesit. Përcaktoni dhe integroni kërkesat e sigurisë dhe privatësisë. Përdorimi i Standardit të Verifikimit të Sigurisë së Aplikacionit – *OWASP* mund të ndihmojë me këtë detyrë. Caktoni ekspertë të sigurisë në projekt, përcaktoni kriteret e sigurisë dhe privatësisë, vendosni një sistem ndjekjeje dhe identifikoni se cilat vegla duhet të përdoren. Kjo duhet të kryhet nga udhëheqësi i ekipit zhvillimor dhe ekipi i sigurisë. Praktika një herëshe.

- Krijoni një gabim kuptimplotë dhe shirita cilësore për të gjithë projektin. Pasi të krijohen, ato duhet të modifikohen. Të kryera nga ekipi i zhvillimit dhe udhëheqësi i ekipit zhvillimor. Praktika një herëshe.

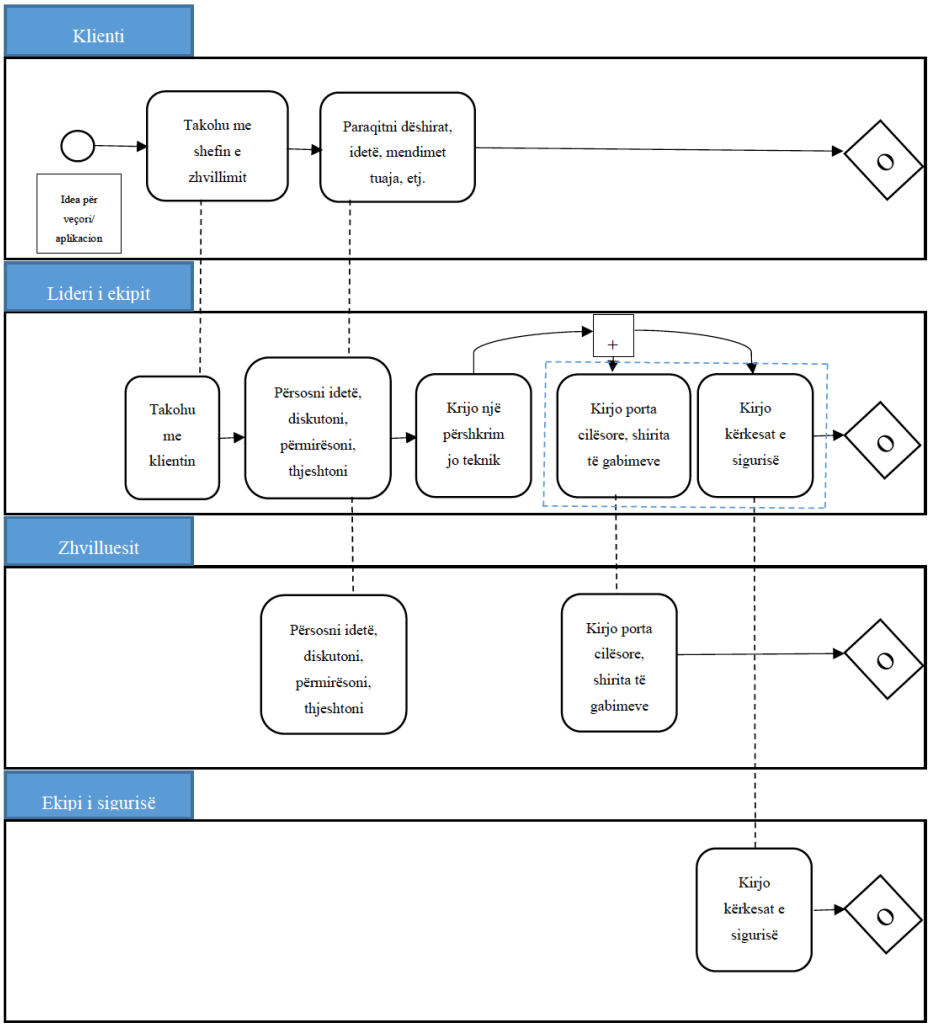

FIGURA 2.4: Faza fillestare

Të dyja *Microsoft SDLC* dhe *WhiteHat SDLC* Sigurisë Mobile përcaktojnë në fazën e ardhshme, që duhet të krijohet një arkitekturë që korrespondon me kërkesat, kontrollet e duhura të sigurisë duhet të shpërndahen si pjesë e aplikacionit dhe duhet të kryhet një model kërcënimi. *IMEC SDLC* në këtë fazë është më e orientuar në thyerjen e aplikacionit në module, çka mund të përfshihet si nën-detyrë në përcaktimin e një arkitekture të duhur, ndërsa *Xamarin SDLC Mobile* nuk paraqet ndonjë gjë që mund të përdoret në këtë moment. Prandaj, detyrat e mëposhtme duhet të kryhen:

- Duhet të kryhet analiza teknike e funksionit dhe një zgjidhje implementimi duhet të miratohet nga ekipi i zhvillimit dhe udhëheqësi i ekipit zhvillimor. Praktika e çdo epike.
- Pasi që zgjidhja e implementimit është e njohur tashmë, kontrollet e duhura të sigurisë duhet të përcaktohen nga udhëheqësi i ekipit zhvillimor dhe një ekip i sigurisë. Për shembull, duhet të përcaktohen kërkesat kriptografike ose siguria e rrjedhës së të dhënave. Praktika e çdo epike.
- Nëse kërkohet një prototip për të testuar zgjidhjen, ajo duhet të krijohet nga ekipi i zvhillimit. Praktika e çdo epike.
- Modeli i kërcënimeve duhet të kryhet nga ekipi i zhvillimit, udhëheqësi i ekipit zhvillimor dhe ekipi i sigurisë. Nëse gjenden probleme që nuk plotësojnë portat e cilësisë, të përcaktuara në fazën fillestare, pjesa problematike duhet të riformohet duke kryer përsëri të gjitha hapat e kësaj faze. Nëse nuk u gjeten probleme, modeli i kërcënimit dhe çdo dokument tjetër duhet të arkivohet nga udhëheqësi i ekipit zhvillimor. Praktika e çdo epike.
- Një specifikim teknik i thelluar bazuar në rezultatet e deritanishme duhet të krijohet nga udhëheqësi i ekipit zhvillimor. Praktika e çdo epike.

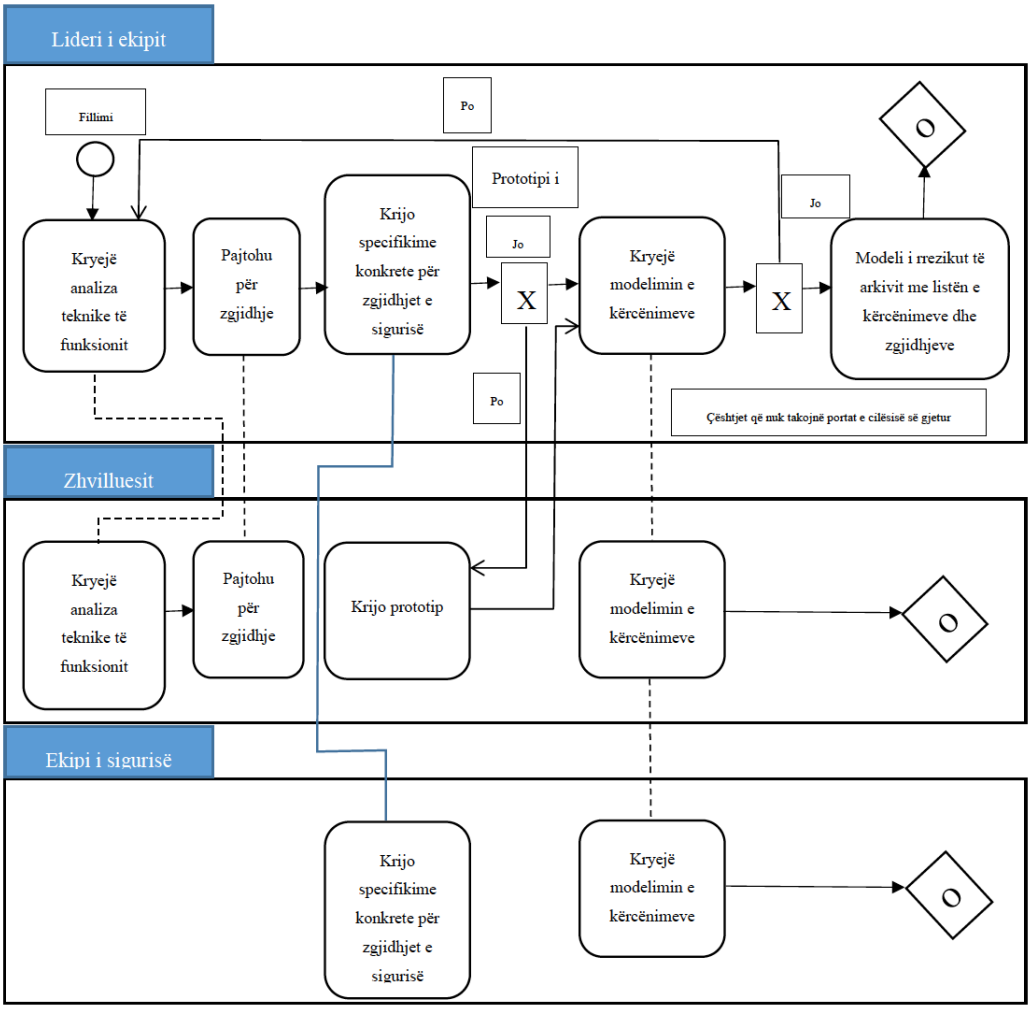

FIGURA 2.5: Faza e përpunimit

Të gjitha kërkesat janë të përcaktuara dhe një dizajn arkitektural është në vend, prandaj ndërtimi mund të fillojë, siç përcaktohet nga *WhiteHat SDLC Sigurisë Mobile*, e cila gjithashtu thekson nevojën për një analizë statike të kodit në këtë fazë. Përfshirja e një analize statike të kodit përmendet gjithashtu në *Microsoft SDLC*. Përveç kësaj, *Microsoft SDLC* gjithashtu përcakton se duhet të përdorur vetëm vegla të aprovuara gjatë kësaj faze. Kjo listë e veglave të aprovuara duhet të përcaktohet në fillim të fazës së ndërtimit. Të dyja *Xamarin SDLC Mobile* dhe *IMEC SDLC* janë më të orientuar se si mund të bëhen gjërat, dhe jo se cilat detyra duhet të bëhen. Për më tepër, pas përfundimit të implementimit të një verifikimi të duhur të kërkesave duhet të bëhet siç thuhet nga të dyja *Microsoft SDLC* dhe *WhiteHat SDLC* Sigurisë. *Xamarin SDLC* dhe *IMEC SDLC* e përcaktojnë

këtë fazë si prototipim dhe stabilizim. Megjithatë, procesi i stabilizimit duhet të inkorporohet tashmë gjatë zhvillimit për të siguruar që çdo çështje të zgjidhet sa më parë. Gjithashtu, reagimet nga klienti nuk duhet të ndikojnë në zhvillimin e tanishëm dhe duhet të përpunohen për ndryshimet e ardhshme, pasi që çdo ndryshim mund të ndryshojë arkitekturën, çka do të kërkonte kryerjen e të gjitha detyrave të fazës së përpunimit përsëri për të siguruar përdorimin e kontrolleve të duhura të sigurisë. Faza e ndërtimit përcakton detyrat e mëposhtme:

- Një listë e veglave të aprovuara që do të përdoren gjatë zhvillimit duhet të përcaktohet nga udhëheqësi i ekipit zhvillimor dhe ekipi i zhvillimit. Gjithashtu, duhet të mbulojë çdo bibliotekë apo vegla të palëve të treta, dhe të sigurojë që këto objekte të kalojnë nivelin e kërkuar të kërkesave të sigurisë. Për shembull, lista mund të aprovojë një përdorim të *Lint Android* dhe të mos aprovojë përdorimin e një PMD. Praktika një herëshe.

- Ekipi i zhvillimit duhet të fillojë me implementimin e zgjidhjes së rënë dakord. Praktika çdo epike.

- Në fund të çdo *sprint*-i, një analizë statike e kodit duhet të kryhet nga ekipi i zhvillimit. Praktika çdo *sprint*-i.

- Në fund të çdo *sprint*i, në qoftë se ekziston një ekzaminim manualisht i testuar, duhe të kryhet edhe një testim manual me qëllim që të sigurohet që kodi të sillet siç pritet. Praktika çdo *sprint*i.

- Pas përfundimit të implementimit një analizë dinamike duhet të kryhet nga ekipi i zhvillimit dhe i sigurisë. Nëse ndonjë çështje që nuk takon portat e cilësisë gjendet, zgjidhja për pjesët problematike duhet të riformulohet dhe verifikohet përsëri. Caktimi në ekipin e sigurisë është për shkak të nivelit të më të lartë të aftësisë që kërkohet për të kryer analizën dinamike, e cila mund të mungojë nga një zhvillues i përbashkët. Praktika çdo epike.

- Nëse nuk u gjetën probleme, kryerja e shqyrtimit të sipërfaqëve të sulmeve duhet të kryhet nga ekipi i zhvillimit dhe i sigurisë. Nëse ndonjë çështje që nuk plotëson portat e cilësisë gjendet, zgjidhja për pjesët problematike duhet

të riformulohet dhe verifikohet përsëri. Kryerja e rishikimit të sipërfaqes së sulmeve duhet të sigurojë që çdo ndryshim në një aplikacion të merret parasysh dhe se nuk ka kërcënime të reja që dalin nga këto ndryshime. Praktika e çdo epike.

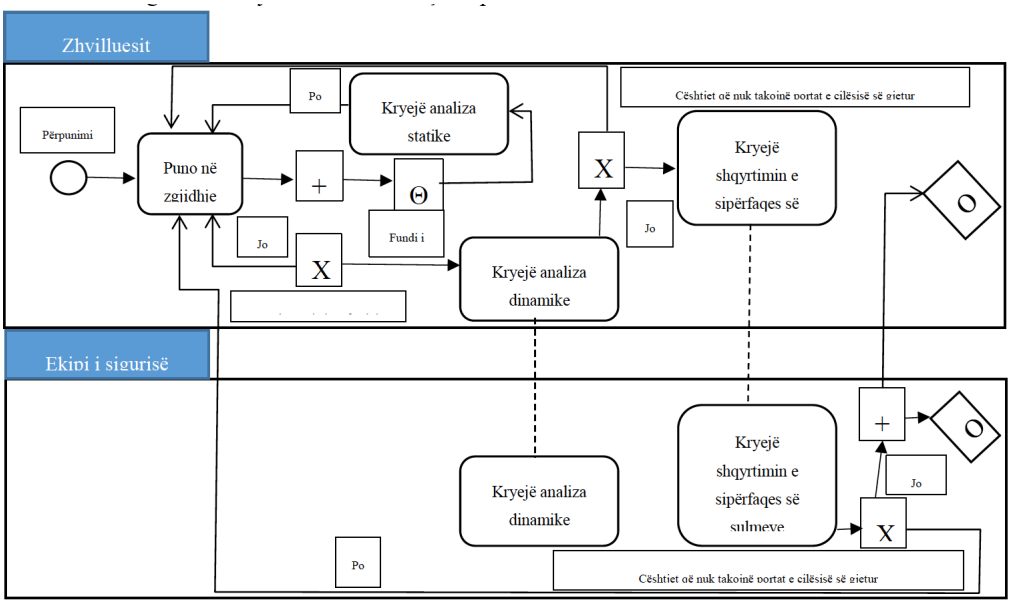

FIGURA 2.6: Faza e ndërtimit

Të tre *SDLC mobile* shkojnë në fazën e lansimit. Megjithatë, *Microsoft SDLC* përcakton para lansimit dy veprime shtesë. Së pari, një rishikim i fundit i sigurisë, i cili duhet të shqyrtojë të gjitha rezultatet nga të gjitha aktivitetet kundër portëve të definuara të cilësisë dhe shiritave të gabimeve. Dhe e dyta, një plan përgjigjeje për incidentin. Aktivitetet e mëposhtme duhet të kryhen:

- Udhëheqsi i ekipit zhvillues dhe një ekip i sigurisë duhet të kryejnë shqyrtimin përfundimtar të sigurisë, duke mbuluar të gjitha aktivitetet e kryera gjatë ciklit të jetës, i cili siguron që të gjitha proceset dhe aktivitetet janë bërë në përputhje me parimet e zhvillimit të sigurt dhe se të gjitha portat e cilësisë dhe shiritat e gabimeve u përmbushën. Çdo problem i gjetur duhet të dokumentohet dhe riparohet në *sprint*-a të ardhshëm dhe në përputhje me *SSDLC* të përdorur. Të gjitha dokumentimet, modelet e kërcënimeve, rezultatet e analizës dinamike dhe statike të kodeve, rezultatet e shqyrtimit

të sipërfaqes së sulmeve, rezultatet e shqyrtimit përfundimtar të sigurisë duhet të arkivohen për përdorim të ardhshëm nga udhëheqësi i ekipit zhvillimor. Praktika e çdo epike.

- Plani i reagimit të incidentit duhet të krijohet nga udhëheqësi i ekipit zhvillimor dhe klienti. Çështjet që dalin duhet të trajtohen sipas këtij plani. Praktika e çdo epike.
- Aplikacioni mund të nënshkruhet dhe të publikohet. Praktika e çdo epike.

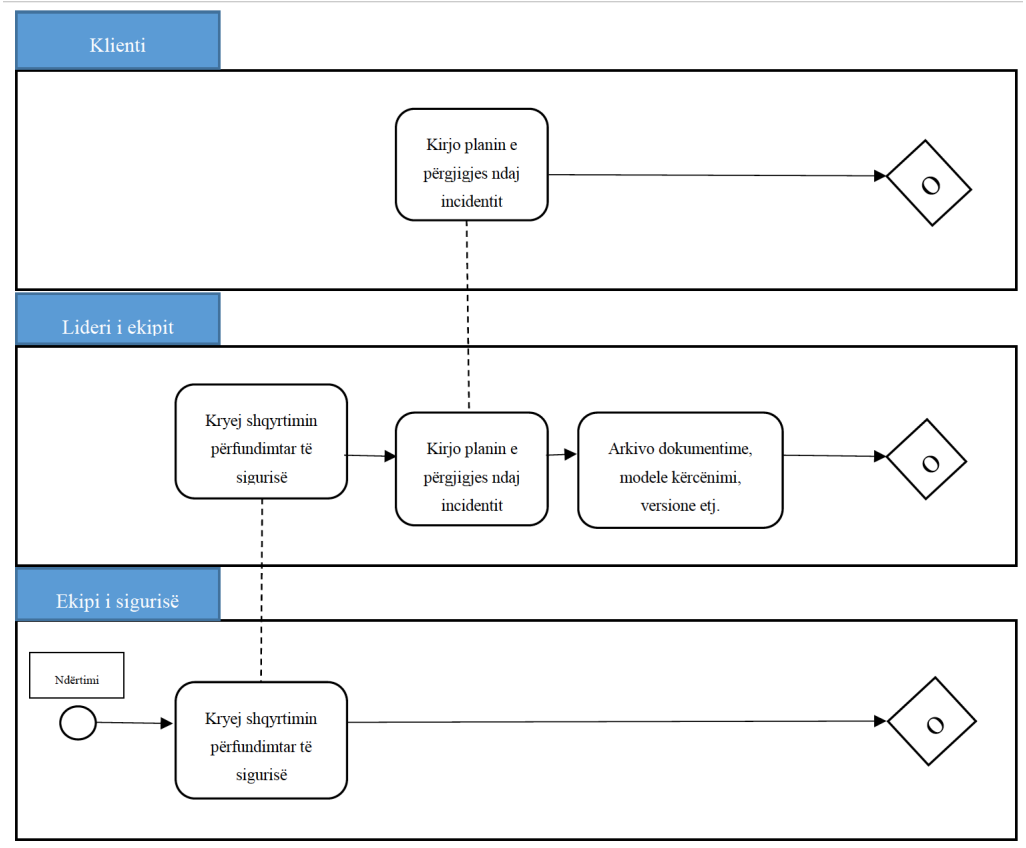

FIGURA 2.7: Faza e ndryshimit

Përveç *Xamarin SDLC Mobile*, të gjitha të definuara si faza e fundit një lloj mirëmbajtjeje e cila duhet të reagojë ndaj çdo problemi që shfaqet. Prandaj kjo fazë përmban një aktivitet të vetëm na vazhdim:

- Funksion ose një aplikacion u implementua me sukses duke plotësuar të gjitha kërkesat, ndërsa përmbushjen e të gjitha portëve të definuara të cilësisë dhe

shiritave të gabimit. Çdo çështje duhet të trajtohet sipas planit të reagimit të incidentit dhe çdo ndryshim i ardhshëm në aplikacion ose kod duhet të bëhet në përputhje me *SSDLC* të përdorur.

2.6 Aplikacioni "KriptoS"

Në vijim do të paraqesim një përshkrim të shkurtër në lidhje me dallimin e përgjithshëm të koncepteve simetrike dhe asimetrike.
Paketat kryesore të *API* kriptografik të Androidit janë:

- *javax.crypto* – Kjo paketë siguron klasat dhe ndërfaqet për aplikacionet kriptografike që implementojn algoritme për kriptim dhe dekriptim, apo çelës simetrik [51].
- *javax.crypto.interfaces* – Kjo paketë siguron ndërfaqet e nevojshme për të implementuar algoritmin e çelës simetrik [52].
- *javax.crypto.spec* – Kjo paketë siguron klasa dhe ndërfaqe të nevojshme për të specifikuar çelësat dhe parametrat për kriptim [53].

Këto *API* janë të shtuara që nga niveli 1 dhe nuk ka nevojë të specifikohet versioni minimal *SDK*.
Klasa ListCryptoAlgorithms

```
public class ListCryptoAlgorithms extends Activity
{
    static final String TAG = "ListCryptoAlgorithms";
    @Override
        public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.algs);
        ListSupportedAlgorithms();
        }
        public void ListSupportedAlgorithms()
    {
        String result = "";
        // merrni të gjithë ofruesit
```

```
            Provider[] providers = Security.getProviders();
        for (int p = 0; p < providers.length; p++)
        {
                    //        merrni të gjitha llojet e shërbimeve për një ofrues specifik
                    Set<Object> ks = providers[p].keySet();
                        Set<String> servicetypes = new TreeSet<String>();
                        for (Iterator<Object> it = ks.iterator(); it.hasNext();)
                {
                        String k = it.next().toString();
                                k = k.split(" ")[0];
                                if (k.startsWith("Alg.Alias."))
                            k = k.substring(10);
                                        servicetypes.add(k.substring(0, k.indexOf('.')));
                }
                        //        merrni të gjitha algoritmet për një lloj të shërbimit specifik
                    int s = 1;
                        for (Iterator<String> its = servicetypes.iterator(); its.hasNext();)
                    {
                     String stype = its.next();
                                Set<String> algorithms = new TreeSet<String>();
                                for (Iterator<Object> it = ks.iterator(); it.hasNext();)
                        {
                         String k = it.next().toString();
                                        k = k.split(" ")[0];
                                        if (k.startsWith(stype + "."))
                                 algorithms.add(k.substring(stype.length() + 1));
                                        else
                                 if (k.startsWith("Alg.Alias." + stype +"."))
                                     algorithms.add(k.substring(stype.length() + 11));
                                  }
                                    int a = 1;
                                    for (Iterator<String> ita = algorithms.iterator(); ita.hasNext();)
                        {
                         result += ("[P#" + (p + 1) + ":" + providers[p].getName() + "]"
                         + "[S#" + s + ":" + stype + "]" + "[A#" + a + ":" + ita.next() + "]\n");
                            a++;
                        }
                        s++;
                    }
                }
            }
}
```

Figura e mëposhtme paraqet një pamje që liston algoritmet e kriptografis në dispozicion në pajisjen tonë ku "$P\#$" tregon numrin e sekuencës për ofruesit, "$S\#$" për llojet e shërbimit dhe "$A\#$" për algoritme.

FIGURA 2.8: Algoritmet Kriptografike të Suportuara

Dallimi kryesor teknik në mes të algoritmeve kriptografike me çelës simetrike dhe asimetrike është se në algoritmin asimetrik, çelësi që përdoret për të kriptua një mesazh nuk është i njejtë me çelësin që përdoret për ta dekriptuar. Secili përdoruers ka një çift çelësash kriptografik, pra një çelës publik për kriptimi dhe një çelës privat për dekriptimi. Kjo është bërë e mundur matematikisht nga faktorizimi i numrave të plotë. Një algoritëm shumë i përdorur, i quajtur *RSA*, është demonstruar më vonë.
Në të kundërtën, algoritmet me çelës simetrike përdorin një çelës të vetëm për të dy kriptim dhe dekriptim. Një algoritëm quhet *AES*, i cili do të mbulohet më vonë në këtë seksion. Këto algoritme zakonisht janë më efikase, por nuk mund të jenë aq të sigurta sa homologët e tyre asimetrik. Ka disa qasje hibride për të përfituar nga përparësitë e të dy llojeve të algoritmeve.
Algoritmi simetrik shumë i popullarizuar dhe shumë efikas quhet *AES* (*Advanced Encryption Standard*. Bazuar në algoritmin *Rijnadel* të zhvilluar nga dy kriptografët belg, *AES* është miratuar nga qeveria e *SHBA* dhe tani përdoret në gjithë botën [54]. *Klasa SymmetricAlgorithmAES*

```
public class SymmetricAlgorithmAES extends Activity
{
    static final String TAG = "SymmetricAlgorithmAES";
    @Override
```

```
   public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.symm);
        // Teksti origjinal

       String theTestText = "Kushtrim Sadriu student i kolegjit UBT.";
        TextView tvorig = (TextView)findViewById(R.id.tvorig);
        tvorig.setText("\n[Teksi origjinal]:\n" + theTestText + "\n");

       // Vendos çelësin privat specifik për enkriptim dhe dekriptim AES 128bit SecretKeySpec
        sks = null;
        try
        {

           SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");
            sr.setSeed("any data used as random seed".getBytes());
            KeyGenerator kg = KeyGenerator.getInstance("AES");
            kg.init(128, sr);

           sks = new SecretKeySpec((kg.generateKey()).getEncoded(), "AES");
        }
        catch (Exception e)
        {
            Log.e(TAG, "AES secret key spec error");
        }
        // Kodo të dhënat origjinale me AES

       byte[] encodedBytes = null;
        try
        {
            Cipher c = Cipher.getInstance("AES");
            c.init(Cipher.ENCRYPT_MODE, sks);
            encodedBytes = c.doFinal(theTestText.getBytes());

       }
        catch (Exception e)
        {
            Log.e(TAG, "AES encryption error");
        }

       TextView tvencoded = (TextView)findViewById(R.id.tvencoded);
        tvencoded.setText("[Enkodimi]:\n" +
Base64.encodeToString(encodedBytes, Base64.DEFAULT) + "\n");

       // Dekodo të dhënat e koduara me AES
        byte[] decodedBytes = null;
        try
```

```
        {

           Cipher c = Cipher.getInstance("AES");
            c.init(Cipher.DECRYPT_MODE, sks);
            decodedBytes = c.doFinal(encodedBytes);
        }
        catch (Exception e)
        {
            Log.e(TAG, "AES decryption error");
        }
    }
}
```

Figura e mëposhtme paraqet një pamje pas rrjedhës së algoritmit 128 *bit AES* së pari me kriptim dhe pastaj me dekriptim.

Figura 2.9: Algoritmi AES

RSA e dizajnuan *Ron Rivest, Adi Shamir* dhe *Leonard Adleman.* Ata zhvilluan algoritmin duke përdorur teknikën e madhe të faktorizimit të numrit të plotë në vitin 1977. Që nga ajo kohë është bërë kaq e popullarizuar sa që ne pothuajse varemi nga teknologjitë e ngjashme të përdorura në jetën e përditshme, siç janë bankat, mesazhet etj. Ky lloj algoritmi përdor një palë çelësash që përdoren për

kriptim dhe dekriptim respektivisht [55].
Klasa AsymmetricAlgorithmRSA:

```
public class AsymmetricAlgorithmRSA extends Activity
{
        static final String TAG = "AsymmetricAlgorithmRSA";
        final Context context = this;
        private TextView result;
        @Override
        public void onCreate(Bundle savedInstanceState)
        {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.asym);
                result = (TextView) findViewById(R.id.tvorig); // Teksti origjinal
                final String theTestText="Kushtrim Sadriu student i kolegjit UBT.";
                TextView tvorig = (TextView)findViewById(R.id.tvorig);
                tvorig.setText("\n[Teksti origjinal]:\n" + theTestText + "\n");
                //        Gjeneroni një pale çelësash për enkriptim dhe dekriptim RSA 1024bit Key publicKey = null;
                Key privateKey = null;
                try
                {
                        KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
                        kpg.initialize(1024);
                        KeyPair kp = kpg.genKeyPair();
                        publicKey = kp.getPublic(); privateKey = kp.getPrivate();
                }
                catch (Exception e)
                {
                        Log.e(TAG, "RSA key pair error");
                }
                //        Kodo të dhënat origjinale me çelës privat RSA
                byte[] encodedBytes = null;
                try
                {
                        Cipher c = Cipher.getInstance("RSA");
                        c.init(Cipher.ENCRYPT_MODE, privateKey);
                        encodedBytes = c.doFinal(theTestText.getBytes());
                }
                catch (Exception e)
                {
                        Log.e(TAG, "RSA encryption error");
                }
                TextView tvencoded = (TextView)findViewById(R.id.tvencoded);
                tvencoded.setText("[Enkodimi]:\n" + Base64.encodeToString(encodedBytes, Base64.DEFAULT) + "\n");
                //        Dekodo të dhënat e koduara me çelës publik RSA byte[] decodedBytes = null;
                try
                {
                        Cipher c = Cipher.getInstance("RSA");
```

```
                    c.init(Cipher.DECRYPT_MODE, publicKey);
                    decodedBytes = c.doFinal(encodedBytes);
                }
                catch (Exception e)
                {
                    Log.e(TAG, "RSA decryption error");
                }
        }
}
```

Figura e mëposhtme paraqet një pamje pas rrjedhës së algoritmit *RSA* së pari me kriptim dhe pastaj me dekriptim.

FIGURA 2.10: Algoritmi RSA

Kjo pjesë kishte të bëjë kryesisht për të ecur nëpër metodën e gjetjes së mbështetjes së dhënë për pajisjet *Android*, si dhe disa përdorime themelore të qasjeve të popullarizuara në algoritmet simetrike *AES* dhe asimetrike *RSA*. Disa qasje përdorin algoritme hibride dhe shumë shpesh funksionet e mbushjes janë inkorporuar gjithashtu për të mbushur të dhënat në përmasat specifike të bllokut.

Teknologjia e kriptografisë mbulon shumë fusha të njohurive dhe kërkon studime të thella.

Kapitulli 3

Definimi i problemit

Në trend tani më shumë të madh janë telefonat e mençur, ku edhe përdorimi dhe avancimimi i tyre është në rritje çdo ditë e më shumë. Rëndësi të veçantë kanë marrë edhe aplikacionet të cilat dizajnohen dhe që janë të përshtatshme për telefonat e mençur, dhe që janë të përdorshme nga ne si shfrytëzues. Por, faktorë kyç në dizajnimin e një aplikacioni të tillë është siguria. Gjatë dizajnimit të aplikacionit duhet pas kujdes për një jetëgjatësi të sigurtë të tij. Megjithatë, kjo nuk mund të arrihet pa përdorimin e veglave të ndryshme, të cilat e mbështesin këtë jetëgjatësi.

Një vegël programimi ose një vegël për zhvillimin e softuerit është një program kompjuterik i përdorur nga zhvilluesit për të krijuar, rregulluar (*debuguar*), mbajtur ose ndryshe të mbështetur procesin e zhvillimit [56]. Për të vlerësuar alternativën ose alternativat më të mira, duhet të përcaktuar matje të veglave dhe duhet krahasuar ato vegla më pastaj.

Lista e proceseve të përdorura në një jetëgjatësi të zhvillimit të softuerit në platformën *Android* përfshin tri aktivitete kryesore, përkatësisht modelimin e kërcënimeve, analizën statike dhe dinamike të kodit.

Matjet për modelimin e kërcënimeve janë krijuar duke identifikuar faktorët kryesorë në procesin e modelimit të kërcënimeve, të cilat janë të renditura sipas rëndësisë së tyre dhe që do të përdoren për krahasimin dhe vlerësimin e veglave për modelimin e kërcënimeve.

Ndryshe nga veglat e modelimit të kërcënimeve, numri i velgave të analizës statike të kodit është i madh. Për shkak të këtij fakti, krahasimi do të bëhet në dy xhiro, së pari ndarjen e veglave që nuk kanë gjasa të jenë të dobishme për qëllimet e SA në platformën *Android*, e dyta duke krahasuar rezultatet e testimit në një projekt me kod të hapur. Projekti quhet $InsecureBankV2$ i krijuar nga *Sinesh Shetty*, i cili është një aplikacion i dobët *Android* që është bërë për entuziastët e sigurisë dhe zhvilluesit për të mësuar pasiguritë *Android* duke testuar këtë aplikacion. Aktualisht përmban 24 dobësi [57].

Krahasimi i veglave të analizës dinamike të kodit do të bëhet përsëri në projektin $InsecureBankV2$.

Kapitulli 4

Metodologjia

Kjo temë është bërë duke u bazuar në literaturën shkencore dhe profesionale nga fusha e teknologjisë informative, përkatësishtë nga fusha e sigurisë. Duke pasë parasysh që qëllimi i kësaj teme ishte se si të dizajnojm një jetëgjatësi të sigurt të zhvillimit të softuerit të targetuar në platformën *Android* dhe si të gjejmë vegla që do të mbështesin këtë jetëgjatësi, atëherë ne kemi përdorë një kombinim të hulumtimit dytësor dhe analizës cilësore të të dhënave.

Aktivitetet kryesore të përfshira në proceset që përdoren në një jetëgjatësi të zhvillimit të softuerit janë modelimi i kërcënimeve, analiza statike dhe dinamike e kodit, ku janë përcaktuar matjet për të krahasuar veglat në dispozicion, janë krahasuar ato më pas dhe kemi vlerësuar alternativat më të mira. Duke përdorur këtë metodologji arritëm të gjejmë vegla për secilën nga këto aktivitete.

Kemi hulumtuar me kujdes për përcaktim të matjeve, krahasim dhe ne fund vlerësimi është bërë po ashtu me kujdes të shtuar.

Analizimi i metodave të kërkimit na ka ndihmuar në minimizim të gabimeve gjatë mbledhjes, interpretimit ose burimit të të dhënave.

Kapitulli 5

Rezultatet

5.1 Veglat

Një vegël programimi ose një vegël për zhvillimin e softuerit është një program kompjuterik i përdorur nga zhvilluesit për të krijuar, *debuguar* (rregulluar), mbajtur ose ndryshe të mbështetur procesin e zhvillimit [56]. Brenda fushës së kësaj teme, kërkesa kryesore për veglat është identifikimi i sa më shumë çështjeve të lidhura me sigurinë të jetë e mundur. Çdo seksion përcakton matje të përdorura për kategorinë e veçantë të veglave, krahason veglat dhe vlerëson alternativën ose alternativat më të mira.

5.1.1 Modelimi i kërcënimeve

Matjet për këtë kategori janë krijuar duke identifikuar faktorët kryesorë në procesin e modelimit të kërcënimeve. Matjet e mëposhtme, të renditura sipas rëndësisë së tyre, do të përdoren për krahasimin dhe vlerësimin e veglave për modelimin e kërcënimeve:

- **Lehtësia e modelimit** – Një përfaqësim i ngushtë i aplikacionit të synuar duke krijuar një diagram të rrjedhës së të dhënave është një nga pjesët kryesore të modelimit të kërcënimeve. Kështu që ky proces duhet të jetë i thjeshtë dhe intuitiv. Megjithatë, ajo duhet të jetë ende në gjendje të ofrojë thellësi të mjaftueshme për të shprehur entitetet dhe rrjedhat e të dhënave të përfshira në diagram.

- **Alternativat shtesë** – Mundësia për të vendosur një atribut specifik në një njësi specifike ose një rrjedhë të dhënash duhet të ndihmojë në identifikimin e kërcënimeve me një nivel më të lartë besimi.

- **Ndërfaqja e përdoruesit** – Një kërkesë që rezulton nga lehtësia e modelimit. Ndërfaqja e përdoruesit duhet të jetë e pastër dhe e lehtë për të lundruar për të thjeshtuar procesin e përgjithshëm të modelimit dhe rishikimit.

- **Procesi i analizës** – Vegla duhet të jetë në gjendje të vlerësojë dhe të identifikojë kërcënimet e pranishme në modelin vetvetiu.

- **Zgjerueshmëria** – Mundësia për të përcaktuar ose zgjeruar më tej modelet bazë të ofruara nga vegla.

- **Raportet e gjeneruara** – Gjenerimi i një raporti të strukturuar nga modeli i kërcënimit të krijuar thjeshton përdorimin e ardhshëm.

- **Dokumentimi** – Një dokumentacion i duhur siguron që vegla të përdoret si duhet.

Do të krahasohen veglat e modelimit të kërcënimeve:

- **Microsoft Threat Modeling Tool 2016** (TMT 2016) – Një vegël e modelimit të kërcënimeve e krijuar nga $Microsoft$, që përdor modelin $STRIDE$ në ndërveprim për identifikimin e kërcënimeve [58].

- **IriusRisk** – Konsolë e integruar e vetme për të menaxhuar rrezikun e sigurisë së aplikacionit gjatë gjithë procesit të zhvillimit të softuerit. Përdor

modelimin e kërcënimeve bazuar në pyetësorët. E vetmja vegël komerciale në këtë listë [59].

- **SeaSponge** – Një vegël e modelimit të kërcënimeve të bazuara në ueb [60].
- **Platforma e Vlerësimit të Rrezikut Coras** (*Coras*) – Platforma për analizën e rrezikut të sistemit të sigurisë së *IT* kritike duke përdorur *UML*, bazuar në metodologjinë e vlerësimit të rrezikut bazuar në modelin *CORAS* [61].
- **Trike** – Një metodologji dhe vegël për modelimin e kërcënimeve me burim të hapur [62].

Matje	TMT 2016	IriusRisk	SeaSponge	Coras	Trike
Lehtësia e modelimit	Po	Po	Jo	Jo	Jo
Alternativat shtesë	Po	Po	Po	Jo	Jo
Ndërfaqja e përdoruesit	Po	Jo	Po	Jo	Jo
Procesi i analizës	Po	Po	Jo	Jo	Jo
Zgjerueshmëria	Po	Jo	Jo	Jo	Jo
Raportet e gjeneruara	Po	Po	Po	Jo	Jo
Dokumentimi	Po	Po	Jo	Jo	Jo

Tabela 5.1: Rezultatet e veglave të modelimit të kërcënimeve të testuara

Nga tabela është e qartë, se *TMT* 2016 ofron alternativat më të mira për modelimin e kërcënimeve. I dyti në tabelë është *IriusRisk*, që siguron një qasje të pazakontë, por të lehtë, për modelimin e kërcënimeve përmes pyetësorëve. Megjithatë, nuk u gjet alternativa për të krijuar një paraqitje vizuale të modelit, çka zvogëlon mundësinë e zgjerimit të së ardhmës, pasi leximi i faqeve të shumta nuk ofron thjeshtësi duke shikuar diagramin e rrjedhes së të dhënave. Gjithashtu, *UI* është sporadikisht kaotik dhe përmban shumë funksione duplikate. Për më tepër, kjo vegël nuk siguron një funksion për të përcaktuar entitetet ose kërcënimet doganore. *SeaSponge* ofron modelimin më themelor të kërcënimeve, duke ofruar elemente bazë, por nuk ofron ndonjë kërcënim të parazgjedhur ose mundësi për të analizuar projektin. Platforma e Vlerësimit të Rrezikut *Coras* dhe *Trike* nuk përputhen me asnjë matje.

Si rezultat i testimit, *Microsoft Threat Modeling Tool 2016* është vegël e këshilluar

për t'u përdorur për procesin e modelimit të kërcënimeve.

TMT 2016 është e lehtë për t'u përdorur si vegël për të krijuar diagram të rrjedhës së të dhënave për produktet ose shërbimet, analizuar diagramet e rrjedhës së të dhënave për të gjeneruar automatikisht një sërë kërcënimesh të mundshme, sugjeruar zbutje potenciale për të dizajnuar dobësitë, prodhuar raporte për kërcënimet e identifikuara dhe të zbutura dhe për të krijuar modele të përsonalizuara për modelimin e kërcënimeve. Përdor metodën $STRIDE$ për modelimin e ndërveprimit, që analizon kërcënimet në kontekstin e ndërveprimit ndërmjet dy elementëve në model.

Modeli është krijuar duke përdorur një *Design View*, që lejon të vizatojë skema dhe siguron objekte dhe prona për të përfaqësuar në mënyrë adekuate dizajnin e një komponenti. Çdo diagram duhet të përmbajë të paktën:

- Një proces.
- Rrjedhjet e të dhënave të drejtuara ndërmjet interaktorëve, proceseve dhe ruajtjes së të dhënave.
- Ruajtja e të dhënave të rëndësishme.
- Kufijtë e besimit. [58]

Krijimi i një modeli të ri kërkon marrjen e një shablloni për të. Çdo shabllon përcakton:

- Shabllonet e lejuara (objektet) me atributet, vetitë dhe përmbajtet e tyre.
- Llojet e kërcënimeve dhe kushtet kur duhet të përfshihen ose përjashtohen nga modeli.
- Vetitë kërcënuese. [58]

Shabllonet e modelit të parazgjedhur ndahen në 6 kategori, secila përmban atribute të ndryshme me porosi që ndikojnë në listën finale të kërcënimeve të gjeneruara.

- Procesi gjenerik.
- Interaktori i jashtëm gjenerik.
- Ruajtja e të dhënave gjenerike.
- Rrjedhja e të dhënave gjenerike.
- Kufiri i besimit gjenerik.
- Kufiri i skajit të besimit gjenerik. [58]

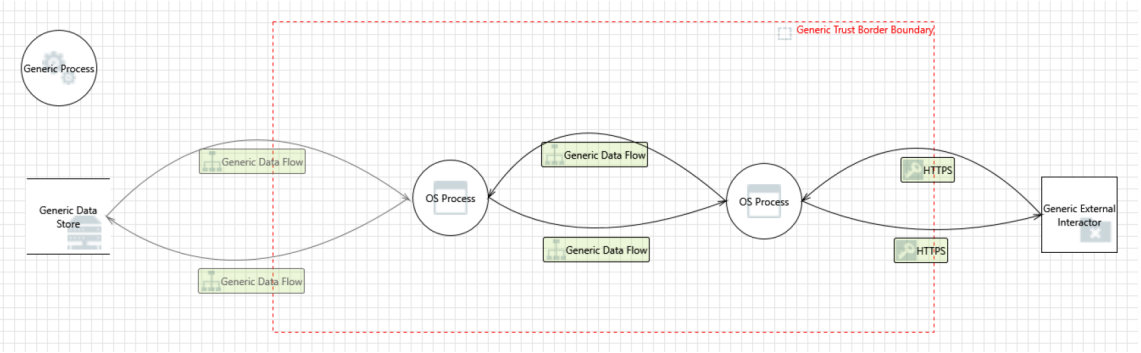

FIGURA 5.1: Shembull i një modeli kërcënimi të krijuar nga TMT 2016

Aktualisht, *TMT 2016* përmban vetëm një skicë që është shënjestruar në aplikacionet e zakonshme të desktopit. Shablloni është gjenerik dhe është pak a shumë i zbatueshëm për platformën *Android*. Mangësi të kësaj qasjeje janë se modeli do të përmbajë çështje që nuk janë të zbatueshme për platformën *Android* dhe se do të humbasë çështjet e sigurisë specifike të *Android*.
Sidoqoftë, është gjithashtu e mundur të krijohet një shabllon me porosi, me target të veçantë në platformën *Android*, që do të përmbushë detyrën më saktësisht.
Analiza e modelit të kërcënimit bëhet duke kaluar në pamjen e Analizës, e cila fsheh veglat e editimit dhe tregon listën e kërcënimeve të gjeneruara për modelin. Çdo kërcënim ka të caktuar një *id*, datën e modifikimit të fundit, gjendjen, titullin, kategorinë, përshkrimin, ndërveprimin dhe prioritetin. Secila prej këtyre fushave, me përjashtim të *id*, është e modifikueshme. [58]

Threat List

	Diagram	Last Modified	State	Title	Category	Description	Interaction	Priority
18	Diagram 1	Generated	Not Started	The Generic D...	Tampering	Data flowing across...	Generic Data Flow	High
19	Diagram 1	Generated	Not Started	Data Store Den...	Repudiation	Generic Data Store cl...	Generic Data Flow	High
20	Diagram 1	Generated	Not Started	Data Flow Sniff...	Information Disclosu...	Data flowing across...	Generic Data Flow	High
21	Diagram 1	Generated	Not Started	Potential Exces...	Denial Of Service	Does OS Process or G...	Generic Data Flow	High
22	Diagram 1	Generated	Not Started	Data Flow Gen...	Denial Of Service	An external agent int...	Generic Data Flow	High
23	Diagram 1	Generated	Not Started	Data Store Inac...	Denial Of Service	An external agent pre...	Generic Data Flow	High
24	Diagram 1	Generated	Not Started	Elevation Usin...	Elevation Of Privilege	OS Process may be a...	Generic Data Flow	High
25	Diagram 1	Generated	Not Started	Elevation Usin...	Elevation Of Privilege	OS Process may be a...	Generic Data Flow	High
26	Diagram 1	Generated	Not Started	Spoofing the G...	Spoofing	Generic External Inter...	HTTPS	High
27	Diagram 1	Generated	Not Started	Potential Data...	Repudiation	OS Process claims th...	HTTPS	High
28	Diagram 1	Generated	Not Started	Potential Proc...	Denial Of Service	OS Process crashes,...	HTTPS	High

FIGURA 5.2: Shembull i listës së kërcënimeve të gjeneruara nga TMT 2016

Si pjesë e kësaj teme, një modelim kërcënimi duke përdorur *TMT 2016* është kryer në një nga aplikimet e korporatës *Y Soft*. Procesi i vetëm i modelimit të kërcënimeve ishte intuitiv dhe në një kohë të shkurtër u krijua dhe u analizua një diagram i plotë i të dhënave që përfaqësonte aplikacionin.

5.1.2 Analiza statike e kodit

Ndryshe nga veglat e modelimit të kërcënimeve, numri i velgave të analizës statike të kodit është i madh. Për shkak të këtij fakti, krahasimi do të bëhet në dy xhiro, së pari ndarjen e veglave që nuk kanë gjasa të jenë të dobishme për qëllimet e *SA* në platformën *Android*, e dyta duke krahasuar rezultatet e testimit në një projekt me kod të hapur.
Xhiro e parë e krahasimit përdor matjet e mëposhtme:

- **AS mbështetjen** – *AS* është *Android IDE* zyrtare, e cila është e favorshme duke pasur kështu një vegël që mund të integrohet direkt në *IDE* përmes një shtojce.
- **Android specifike** (*Aspec*) – Vegla duhet të jetë e targetuar në platformën *Android* për të ndihmuar në identifikimin e çështjeve me nivel më të lartë besimi.
- **Përditësimi** – Treguesi nëse vegla është e përditësuar. Një projekt që nuk është përditësuar për të paktën një vit do të konsiderohet i vjetëruar për këto qëllime.

- **Kërkesat** – Në mënyrë ideale, numri i kërkesave për të drejtuar veglën duhet të jetë minimal, për të lehtësuar maksimalisht procesin e përgjithshëm të instalimit dhe përdorimit. Vlerësuar nga vlerat e ulëta (shtojca *AS* pa konfigurim të kërkuar), mesatare (shotjca *AS* me konfigurimin e kërkuar) dhe të larta (jo shtojca *AS*, konfigurimi i kërkuar).

Vegla	AS mbështetje	Aspec	Kërkesat	Përditësimi
Android Lint	Po	Po	Ulët	Po
FindBugs	Po	Jo	Mesatar	Po
FlowDroid	Jo	Po	Lartë	Jo
PMD	Po	Jo	Ulët	Po
SPARTA	Jo	Po	Lartë	Jo
MobSF	Jo	Po	Lartë	Po
Madrolyzer	Jo	Po	Lartë	Jo
Androwarn	Jo	Po	Lartë	Jo
Checkstyle	Po	Jo	Mesatar	Po
Devknox Lite	Po	Po	Ulët	Po

Tabela 5.2: Rezultatet e xhiros së parë të testeve në analizuesit statik të kodimit

Siç mund të shihet nga tabela, vetëm dy vegla arritën të kenë rezultatin më të mirë të mundshëm, përkatësisht *Android Lint* dhe *Devknox Lite*. Më pas, PMD i mungon vetëm orientimi në platformën *Android*. $FindBugs$ dhe $Checkstyle$ janë në të njejtin nivel, të integruar në *AS*, jo në *Android*, me kërkesa të mesme. Vegla e fundit e përdorshme është *Mobile Security Framework* ($MobSF$). Projektet e mbetura janë të vjetëruara, më përditësimin më të fundit ose të kryer më shumë se një vit më parë, kështu që ato nuk do të përfshihen në xhiron e dytë të testimit. Nënseksioni përmbledhje e veglave përmban një përshkrim bazë të veglave të zgjedhura që rezultojnë nga xhiro e parë e testeve. I pari është *Android Lint*, që është një vegël e integruar e parazgjedhur në *Android Studio*, që përmban 281 rregulla në 6 kategori. Është gjithashtu aktualisht e vetmja vegël ekzistuese e analizës statike specifike për *Android* që punon me kodin burimor dhe jo me APK. [63]
Tjetra, $FindBugs$ është një nga veglat më të njohura dhe të përdorura gjerësisht për analizën e kodeve statike, që kërkojnë gabime në programet $Java$ dhe janë të integruara drejtpërdrejt në *Android Studio* nga një shtojcë. Kjo bazohet në konceptin e modeleve të gabimeve dhe punon duke analizuar Java bytecode, duke

kërkuar kështu qasje në skedarët e klasës së përpiluar. Sipas faqes së internetit të *FindBugs*, analiza mund të jetë ndonjëherë e pasaktë dhe mund të ketë normë pozitive false deri në 50%. *FindBugs* aktualisht përdor 425 rregulla të ndara në 9 kategori. [64]

Duke shfrytëzuar shtojcën *Find Security Bugs* në shtojcën *FindBugs*, është e mundur të shtoni 113 kontrolle shtesë të sigurisë. *Find Security Bugs* mbulon gjithashtu *OWASP Top 10* dhe *Common Weaknesses Enumeration* (*CWE*). [65]

PMD është një analizues i kodit burimor që gjen gabime të zakonshme programore si ndryshore të papërdorura, krijim të panevojshëm të objektëve e kështu me radhë. Mbështet gjuhët e shumta, duke përfshirë *Java*. Është në dispozicion në *Android Studio* përmes një shtojce. [66]

Për momentin, *PMD* përdor 274 rregulla të ndara në 25 kategori, me vetëm 5 rregulla që janë specifike për *Android* dhe vetëm 2 rregulla të orientuara drejt sigurisë. [67]

Devknox është një shtojcë e *Android Studio* që ndihmon në zbulimin dhe zgjidhjen e çështjeve të sigurisë në aplikacione gjatë shkrimit të kodit. *Devknox Lite*, versioni pa pagesë, përmbanë 30 kontrolle sigurie, ndërsa versioni i plotë me pagesë përmban 60 kontrolle shtesë të sigurisë dhe alternativën për të kryer testimin e sigurisë statike të aplikacionit. [68]

Mobile Security Framework (*MobSF*) është inteligjent, aplikacion mobil me kod të hapur i automatizuar me mprehtësi duke testuar të dy kornizën e aftë ose përformuese, analizën statike dhe dinamike. [69]

Checkstyle ëhstë një vegël zhvillimi për të ndihmuar programuesit të shkruajnë *Java* kodin që i përmbahet një standardi kodues. Automatizon procesin e kontrollimit të *Java* kodit, çka e bën atë ideal për projektet që duan të zbatojnë një standard kodues. *Checkstyle* është shumë e konfigurueshme dhe mund të mbështesë pothuajse çdo standard kodimi. Grupi i rregullave përcaktohet vetëm nga skedari i konfigurimit të përzgjedhur. Për shkak të kësaj natyre, ajo nuk do të përfshihet në xhiron e dytë të testeve. [70]

Krahasimi i dytë i veglave të zgjedhura është në një projekt *InsecureBankV2* të krijuar nga *Dinesh Shetty*. Është një aplikacion i dobët *Android* që është bërë

për entuziastët e sigurisë dhe zhvilluesit për të mësuar pasiguritë *Android* duke testuar këtë aplikacion. Aktualisht përmban 24 dobësi. [57]
Matjet e mëposhtme do të përodren për të krahasuar veglat:

- **Numri i rregullave të orientuara drejtë sigurisë** – Tregon se sa me rëndësi është vegla e orientuar në siguri.

- **Koha e analizës** – Koha totale e ekzekutimit të analizës në sekonda.

- **Çështjet me prioritet të ulët të gjetura** – Numri i çështjeve me prioritet të ulët që gjenden nga totali i numërimit. Këto çështje mund të konsiderohen si paralajmërime.

- **Çështjet me prioritet të lartë të gjetura** - Numri i çështjeve me prioritet të lartë që gjenden nga totali i numërimit. Këto çështje janë kritike dhe duhet të hiqen nga projekti, para një lansimi.

- **Çështjet e sigurisë të gjetura** – Numri i çështjeve të sigurisë të gjetura në projekt.

Çështjet e gjetura sipas prioritetit. *Android Lint* gjeti një sasi ekstreme të çështjeve

Vegla	Rregullat e sigurisë	Koha e ekzekutimit	Ulët	Mesëm	Lartë	Siguria
Android Lint	37	12	11505	434	16	5
FindBugs	11	3	17	2	0	0
FindBugs+ Shtojca Android	11	4	17	2	0	0
FindBugs+ Sthojca FindSec	124	6	17	2	14	14
PMD	2	3	2	54	3	0
MobSF	E panjohur	107	2	1	12	15
Devknox Lite	30	0	0	10	10	10

Tabela 5.3: Rezultatet e xhiros së dytë të testeve në analizuesit statik të kodit

me prioritet të ulët, megjithatë, shumica e tyre ishin vetëm një gabim shkrimi dhe kështu i parëndësishëm. *FindBugs* në vetvete nuk arriti të gjejë ndonjë çështje të sigurisë. Shtimi i shtojcës së *Android* nuk ka ndryshuar fare rezultatet. Megjithatë, duke shtuar *Find Security Plugin*, ai arriti të gjejë 14 çështje të lidhura me sigurinë. *PMD* me 2 rregullat e saj të sigurisë nuk gjetën ndonjë çështje të sigurisë. *Devknox Lite* arriti të gjejë 10 çështje të sigurisë. Shumica e

të gjitha veglave u gjetën nga *MobSF*, i cili arriti të gjejë 15 çështje të lidhura me sigurinë. *InsecureBankV2* përmban 24 çështje të orientuara nga siguria, të cilat u përcaktuan në çështjet e sigurisë të gjetura nga veglat. Përveç *MobSF*, i cili identifikon 12 çështje të sigurisë, secila vegël gjeti më së shumti 6 çështje nga lista. Kombinuar së bashku janë identifikuar 14 nga 24 çështje.
Bazuar në rezultatet e testeve, zhvilluesit duhet gjithmonë të përdorin *MobSF*, *Android Lint*, *FindBugs* me shtojcën *Find Security Bugs* dhe *Devknox Lite*. Shtimi i *PMD* me shumë gjasa nuk do të ndikojë në rezultatet e sigurisë, për shkak të sasisë së saj të vogël të rregullave të orientuara të sigurisë. Megjithatë, për shkak të kohës së ekzekutimit të vogël, ajo është ende një vegël e qëndrueshme për qëllime të ndryshme nga siguria. Përdorimi i një *Checkstyle* nuk do të ndikojë drejtëpërdrejt në sigurinë, por të kesh një kod që ndjek rigorozisht një sërë rregullash përmirëson cilësinë e kodit, gjë që në kthim thjeshton modifikimet dhe rregullimet e gabimeve.

5.1.3 Analiza dinamike e kodit

Krahasimi i veglave të analizës dinamike të kodit do të bëhet përsëri në projektin *InsecureBankV2* dhe do të përdoren matjet në vijim:

- **Përditësimi** – Vegla nuk duhet të jetë e vjetëruar dhe idealisht duhet të pranojë përditësime në baza të rregullta. Një projekt që nuk është përditësuar për të paktën një vit do të konsiderohet i vjetëruar për këto qëllime.
- **Çështjet e gjetura** – Numri i çështjeve të gjetura në 30 minuta të kryerjes së analizës dinamike duke përdorur veglën.
- **Lehtësia e përdorimit** – Ndjenja e përgjithshme se sa e lehtë është të përdoret vegla. Vlerësohet me vlera nga 1 (më e vështira) deri në 10 (më e lehta).
- **Dokumentimi** – Një dokumentim i duhur siguron që vegla të përdoret si duhet dhe ndihmon në fillim me mësimet se si të përdoret vegla.

- **Kërkesat** – Koha e përafërt e kërkuar për të vendosur veglën dhe për të filluar përdorimin e tij në minuta, duke përfshirë instalimin e të gjitha kërkesave. Nuk përfshin kohën e kërkuar për të shkarkuar veglat.

E para e veglave të krahasuara do të jetë *MobSF*, e cila tashmë është futur në seksionin e përmbledhjes së veglave të analizës statike të kodit. *MobSF* përdor një makinë virtuale të personalizuar që simulon një pajisje të rrënjosur *Android* me versionin *Android 4.4.2* (*API versioni 19*). Kjo përdorë gjithashtu Kornizën *Xposed*, e cila përdor module për të ndryshuar sjelljen e sistemit dhe aplikacionëve pa prekur asnjë *APK*. [71] Asnjë specifikë tjetër rreth procesit të analizës dinamike nuk është gjetë.

Vegla e dytë e testuar është *Drozer*, një auditim gjithëpërfshirës i sigurisë dhe një kornizë sulmesh për *Android*, që ndihmon në sigurimin e besimit se aplikacionet *Android* dhe pajisjet që po zhvillohen, ose të shpërndara përtej, nuk paraqesin një nivel të papranueshëm rreziku. Kjo lejon bashkëveprimin me *Dalvik VM*, pikat fundore *IPC* të aplikacioneve tjera dhe *OS* themelor. Kjo ndihmon në shfrytëzimin nga distanca të pajisjeve *Android*, duke ndërtuar skema me qëllim të keq që shfrytëzojnë dobësitë e njohura. Ngarkesa e përdorur në këto shfrytëzime është një agjent mashtrues që është në thelb një mjet administrator nga distanca. [72]

Drozer përdoret shpesh për qëllime të analizës dinamike nga veglat ose kornizat e tjera, të tilla si *Android 4B* [73] ose *Appie* [74].

Tjetra, *Inspeckage* një modul për Kornizën *Xposed*, që ofron analizë dinamike të aplikacioneve *Android* duke aplikuar grepa në funksionet e *Android API*. Aktualisht, grepat mbulojnë fushat e Preferencave të Përbashkëta, Serializimit, Kriptografisë, Hashes, *SQLite*, *HTTP*, Sistemi i Skedarit, Të ndryshme (për shembull, *Clipboard*), *WebView* dhe *IPC*. Gjithashtu lejon përcaktimin e grepeve me porosi. Të gjitha informatat e mbledhura tregohen përmes një Ndërfaqeje të Përdoruesit të thjeshtë të uebit. [75]

Devknox Pro, një version i paguar i *Devknox*, i cili ishte u prezantua tashmë. Versioni i plotë gjithashtu ofron mundësinë e kryerjes së një analize dinamike përmes një ueb faqe të *Devknox*. [68]

E fudnit, *AppUse aVMdeveloped* nga *AppSec Labs*. Kjo është një platformë për testimin e sigurisë së aplikacioneve mobile në mjedisin *Android*. Versioni pa pagesë përmban vetëm një funksionalitet të kufizuar. [76]

Si hap i parë i përbashkët për të gjitha veglat, serveri *backend python* i quajtur

Matje	MobSF	Drozer	Inspeckage	Devknox	AppUse
Përditësimi	Po	Po	Po	Po	Po
Çështjet e gjetura	12	6	9	-	-
Lehtësia e përdorimit	8	3	8	5	3
Dokumentimi	Po	Po	Po	Po	Po
Kërkesat	30	10	25	1	20

Tabela 5.4: Rezultatet e matjeve të analizës dinamike të kodit

AndroLab, i cili është i pranishëm në $InsecureBankV2$, duhej të ishte strukturë (*setup*). Tjetra një konfigurim i komunikimit në mes aplikacionit dhe *backend* është siguruar duke vendosur një IP adresë të duhur dhe numrin e portit në aplikacion. Të gjitha veglat pastaj ndoqën një parim të njejtë gjatë analizës – kryejnë operacione dhe analizojnë të dhënat e ruajtura ose të përpunuara nga aplikacioni.
Me $MobSF$, 12 çështje u identifikuan në kohën e caktuar, për shkak të raportit të strukturuar, gjë që e bën analizën më të lehtë. $Inspeckage$ kryhet në mënyrë të ngashme. Të dyja këto vegla ishin të lehta dhe intuitive për të punuar me to.
Me $Drozzer$ u gjetën vetëm 6 çështje, meqë kërkon të përdoren komanda specifike dhe një kuptim më i lartë se ku duhet të shikojë analisti dhe çfarë duhet bërë.
$Devknox$ nuk mund të testohej në këtë projekt pasi lidhja midis *AndroidLab* dhe analizuesit dinamik nuk mund të krijohej. UI i emulatorit ishte jashtëzakonisht i vogël dhe gjithashtu i vonuar.
Me $AppUse$ pa pagesë, analiza nuk mund të kryhet, pasi përmban vetëm një funksionalitet të kufizuar i cili nuk ofroi ndonjë mundësi të dobishme. UI i përgjithshëm është kaotik, i vështirë për t'u naviguar dhe përdorur.
Bazuar në rezultatet, aktualisht, rekomandimi për një analizë dinamike të kodit është të përdorë Kornizën e Sigurisë Mobile. Është e lehtë për t'u përdorur, intuitive, gjeneron raporte strukturore, dhe përmban vegla shumë më tepër në krye të analizës dinamike të kodit.
Alternativa e dytë më e mirë është $Inspeckage$, që aktualisht ofron funksionalitete

shumë të ngjashme në fushën e analizës dinamike. *Drozzer* mund të bëjë krahasime me dy veglat e mëparshme, megjithatë, është shumë më e vështirë për t'u përdorur dhe kërkon një shkallë më të lartë të aftësive.

Kapitulli 6

Diskutime dhe përfundime

Kjo temë kishte dy qëllime kryesore, e para ishte për të dizajnuar një jetëgjatësi të sigurt të zhvillimit të softuerit të targetuar në platformën *Android* dhe e dyta për të gjetur vegla që do të mbështesin këtë jetëgjatësi. Para dizajnimit të jetëgjatësisë, ne kalojmë nëpër disa tema të fokusuara në platformën *Android* dhe sigurinë e saj nga pikëpamje të ndryshme. *OWASP* (*Open Web Application Security Project*) dhe projektet e saj me orientim mobile janë futur për të dhënë një pasqyrë të kërcënimeve të përbashkëta në platformën *Android* dhe opsionet se si të garantojnë më tej sigurinë e një aplikacioni të zhvilluar.

Tjetra, ne listojmë proceset e përbashkëta të përfshira në një jetëgjatësi të zhvillimit të softuerit dhe ne përshkruajmë procesin e nënshkrimit dhe publikimit të aplikacionit. Bazuar në rishikimin e cikleve ekzistuese të gjetëgjatësisë, të cilat na japin një kuptim më të mirë se çfarë duhet bërë dhe kur duhet të bëhet, ne dizajnojmë një jetëgjatësi të re të zhvillimit të softuerit të sigurtë, që është me target në platformën *Android* dhe modelin e shkathët të zhvillimit të softuerit. Çdo fazë e jetëgjatësisë të dizajnuar justifikon ose përmban këto detyra dhe i cakton këto detyra personave që punojnë në një projekt. Për çdo fazë ne gjithashtu krijojmë një diagram që përfaqëson fazën dhe aktivitetet e tij, duke përdorur shënimin e *Business Process Model Notation*.

Lista e proceseve të përdorura në një jetëgjatësi të zhvillimit të softuerit përfshin tri aktivitete kryesore, përkatësisht modelimin e kërcënimeve, analizën statike dhe

dinamike të kodit. Për secilën nga këto aktivitete, ne përcaktojmë matjet për të krahasuar veglat në dispozicion, krahasojmë ato dhe vlerësojmë alternativat më të mira. Duke përdorur këtë qasje arritëm të gjejmë vegla për secilën nga këto aktivitete.

Ka disa mënyra se si të zgjerojmë më tej këtë temë. Seksioni i modelimit të kërcënimeve përmban një përshkrim bazë të asaj që është dhe si zakonisht bëhet. Puna shtesë në këtë fushë mund të jetë më specifike dhe të përcaktojë këtë proces në platformën *Android*. Kjo mund të përfshijë krijimin e një modeli specifik të platformës *Android* për *Microsoft Threat Modeling Tool 2016*. Pastaj, mund të krijohet një analizë më e thellë e përcaktimit të kërkesave dhe kontrolleve të sigurisë, së bashku me krijimin e një udhëzuesi të testimit për të verifikuar këto kërkesa, në mënyrë të ngjashme me ato të ofruara nga $OWASP$. Më shumë abicioz mund të jetë krijimi i një vegle statike për analizën e kodeve specifike të Androidit të orientuar në sigurinë, pasi opsionet në këtë fushë janë shumë të kufizuara.

Kapitulli 7

Referencat

[1] A. Developers, "Platform Architecture", 2017. [Online].

Available: https://developer.android.com/guide/platform/index.html

[2] A. Source, "Application Signing,", 2017. [Online].

Available: https://source.android.com/security/apksigning/

[3] A. Developers, "Dashboards", 2017. [Online].

Available: https://developer.android.com/about/dashboards/index.html

[4] Online, "Statista", 2017. [Online].

Available: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/

[5] A. Developers, "What is Android", 2017. [Online].

Available: http://sites.google.com/site/amitsahain/Andoid–tech.pdf

[6] Wikipedia, "Android (operating system)", 2017. [Online].

Available: https://en.wikipedia.org/wiki/Android_(operating_system)

[7] A. Developers, "Introduction to Android", 2017. [Online].

Available: https://developer.android.com/guide/index.html

[8] A. Source, "Codenames, Tags, and Build Numbers", 2017. [Online].

Available: https://source.android.com/source/build-numbers

[9] P. KHATRI, "Native vs Cross-Platform App Development: Pros" 2017. [Online].

Available: https://dzone.com/articles/native-vs-cross-platform-app-development-pros-and

[10] A. Developers, "Meet Android Studio", 2017. [Online].

Available: https://developer.android.com/studio/intro/index.html

[11] A. Source, "Requirements", 2017. [Online].

Available: https://source.android.com/source/requirements.html

[12] A. Developers, "Use Java 8 language features", 2017. [Online].

Available: https://developer.android.com/studio/write/java8-support.html

[13] A. Developers, "Add C and C++ Code to Your", 2017. [Online].

Available: https://developer.android.com/studio/projects/add-native-code.html

[14] A. Developers, "Getting Started with the NDK", 2017. [Online].

Available: https://developer.android.com/ndk/guides/index.html

[15] A. Source, "Security", 2017. [Online].

Available: https://source.android.com/security/

[16] A. Developers, "Security Tips", 2017. [Online].

https://developer.android.com/training/articles/security-tips.html

[21] OWASP, "Top 10 Mobile Risks - OWASP Mobile Security Project", 2017. [Online].

Available: https://www.owasp.org/index.php/OWASP_Mobile_Security_Project

[22] OWASP, "OWASP Mobile Application Security Verification Standard", 2017. [Online].

Available: https://github.com/OWASP/owasp-masvs

[23] OWASP, "OWASP Mobile Security Testing Guide", 2017. [Online].

Available: https://github.com/OWASP/owasp-mstg

[24] OWASP, "OWASP Mobile Security Project. Secure M-Development", 2017. [Online].

Available: https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Secure_ M-Development

[25] N. Elenkov, Android Security Internals, San Francisco: William Pollock, 2015.

[26] Techopedia, "Software Development Life Cycle (SDLC)", 2017. [Online].

Available: https://www.techopedia.com/definition/22193/software-development-life-cycle-sdlc

[27] Microsoft, "What is the Security Development Lifecycle?", 2017. [Online].

Available: https://www.microsoft.com/en-us/sdl/

[28] Microsoft, "SDL PROCESS: TRAINING", 2017. [Online].

Available: https://www.microsoft.com/en-us/SDL/process/training.aspx?cf76f7d4-5430-4a81-99ed-343310bc8374=True

[29] Microsoft, "SDL PROCESS: REQUIREMENTS", 2017. [Online].

Available: https://www.microsoft.com/en-us/SDL/process/requirements.aspx?cf76f7d4-5430-4a81-99ed-343310bc8374=True&Accordionitem1=True&Accordionitem2=True

[17] A. Developers, "Network Security Configuration", 2017. [Online].

Available: https://developer.android.com/training/articles/security-config.html

[18] A. Developers, "Enhancing Security with Device Management Policies", 2017. [Online].

Available: https://developer.android.com/work/device-management-policy.html

[19] OWASP, "Welcome to OWASP", 2017. [Online].

Available: https://www.owasp.org/index.php/Main_Page

[20] OWASP, "OWASP Mobile Security Project", 2017. [Online].

Available: https://www.owasp.org/index.php/OWASP_Mobile_Security_Project

[30] Microsoft, "SDL PROCESS: DESIGN", 2017. [Online].

Available: https://www.microsoft.com/en-us/SDL/process/design.aspx?41ed9ed1-398e-403a-fe06-ed3c3f48404d=True&Accordionitem1=True

[31] Microsoft, "SDL PROCESS: IMPLEMENTATION", 2017. [Online].

Available: https://www.microsoft.com/en-us/SDL/process/implementation.aspx?

Accordionitem1=True&cf76f7d4-5430-4a81-99ed-343310bc8374=True&Accordionitem2=True

[32] Microsoft, "SDL PROCESS: VERIFICATION", 2017. [Online].

Available: https://www.microsoft.com/en-us/SDL/process/verification.aspx?cf76f7d4-5430-4a81-99ed-343310bc8374=True&Accordionitem1=True&Accordionitem2=True

[33] Microsoft, "SDL PROCESS: RELEASE", 2017. [Online].

Available: https://www.microsoft.com/en-us/SDL/process/release.aspx?cf76f7d4-5430-4a81-99ed-343310bc8374=True&Accordionitem1=True&Accordionitem2=True

[34] Microsoft, "SDL PROCESS: RESPONSE", 2017. [Online].

Available: https://www.microsoft.com/en-us/SDL/process/response.aspx?Accordionitem1=True

[35] Microsoft, "SDL FOR AGILE. EVERY-SPRINT PRACTICES", 2017. [Online].

Available: https://www.microsoft.com/en-us/SDL/discover/sdlagile.aspx

[36] Microsoft, "SDL FOR AGILE. BUCKET PRACTICES", 2017. [Online].

Available: https://www.microsoft.com/en-us/SDL/discover/sdlagile-bucket.aspx

[37] Microsoft, "SDL FORAGILE. ONE-TIME PRACTICES", 2017. [Online].

Available: https://www.microsoft.com/en-us/SDL/discover/sdlagile-onetime.aspx

[38] Xamarin, "Introduction to the Mobile Software Development Lifecycle", 2017. [Online].

Available: https://developer.xamarin.com/guides/cross-platform/getting_

started/introduction_to_mobile_sdlc/

[39] W. Security, "Integrating Application Security into the Mobile Software Development Lifecycle", 2017. [Online].

Available: https://www.whitehatsec.com/wp-content/uploads/2016/01/Mobile_

SDLC_White_Paper.pdf

[40] T. K. A. I. VITHANI, "Modeling the Mobile Application Development Lifecycle", 2017. [Online].

Available: http://www.iaeng.org/publication/IMECS2014/IMECS2014_pp596-600.pdf

[41] OWASP, "Application Threat Modeling", 2017. [Online].

Available: https://www.owasp.org/index.php/Application_Threat_Modeling

[42] Microsoft, "The STRIDE Threat Model", 2017. [Online].

Available: https://msdn.microsoft.com/en-us/library/ee823878(v=cs.20).aspx

[43] Microsoft, "Chapter 3 - Threat Modeling", 2017. [Online].

Available: https://msdn.microsoft.com/en-us/library/ff648644.aspx

[44] OWASP, "Static Code Analysis", 2017. [Online].

Available: https://www.owasp.org/index.php/Static_Code_Analysis

[45] TechTarget, "DEFINITION: dynamic analysis", 2017. [Online].

Available: http://searchsoftwarequality.techtarget.com/definition/dynamic-analysis

[46] OWASP, "Dynamic Analysis", 2017. [Online].

Available: https://www.owasp.org/index.php/Dynamic_Analysis

[47] Webopedia, "vulnerability scanning", 2017. [Online].

Available: https://www.webopedia.com/TERM/V/vulnerability_scanning.html

[48] C. security, "Penetration Testing Overview", 2017. [Online].

Available: https://www.coresecurity.com/content/penetration-testing

[49] A. Developers, "Sign Your App", 2017. [Online].

Available: https://developer.android.com/studio/publish/app-signing.html

[50] A. Developers, "Publish Your App", 2017. [Online].

Available: https://developer.android.com/studio/publish/index.html

[51] ORACLE, "ORACLE", 2017. [Online].

Available: https://docs.oracle.com/javase/7/docs/api/javax/crypto/package-summary.html

[52] ORACLE, "ORACLE", 2017. [Online].

Available: https://docs.oracle.com/javase/7/docs/api/javax/crypto/interfaces/package-summary.html

[53] ORACLE, "ORACLE", 2017. [Online].

Available: https://docs.oracle.com/javase/7/docs/api/javax/crypto/spec/package-summary.html

[54] WIKIPEDIA, "WIKIPEDIA", 2017. [Online].

Available: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard.

[55] WIKIPEDIA, "WIKIPEDIA", 2017. [Online].

Available: https://en.wikipedia.org/wiki/RSA_(cryptosystem)

[56] Wikipedia, "Programming tool", 2017. [Online].

Available: https://en.wikipedia.org/wiki/Programming_tool

[57] D. SHETTY, "Android-InsecureBankv2", 2017. [Online].

Available: https://github.com/dineshshetty/Android-InsecureBankv2

[58] Microsoft, "Microsoft Threat Modeling Tool 2016 User Guide", 2017. [Online].

Available: https://www.microsoft.com/en-us/download/details.aspx?id=49168

[59] Continuumsecurity, "IRIUSRISK - THREAT MODELING", 2017. [Online].

Available: https://www.continuumsecurity.net/threat-modeling-tool/

[60] Mozilla, "SeaSponge", 2017. [Online].

Available: https://github.com/mozilla/seasponge

[61] CORAS, "The CORAS Tool", 2017. [Online].

Available: http://coras.sourceforge.net/coras_tool.html

[62] Trike, "Trike", 2017. [Online].

Available: http://octotrike.org/

[63] A. Developers, "Improve Your Code with Lint", 2017. [Online].

Available: https://developer.android.com/studio/write/lint.html

[64] FindBugs, "FindBugs Fact Sheet", 2017. [Online].

Available: http://findbugs.sourceforge.net/factSheet.html

[65] P. ARTEAU, "Find Security Bugs", 2017. [Online].

Available: http://find-sec-bugs.github.io/

[66] PMD, "PMD DON'T SHOOT THE MESSENGER", 2017. [Online].

Available: https://pmd.github.io/

[67] PMD, "PMD DON'T SHOOT THE MESSENGER. Current Rulesets", 2017. [Online].

Available: http://pmd.sourceforge.net/pmd-4.3.0/rules/index.html

[68] Devknox, "Documentation", 2017. [Online].

Available: https://devknox.io/documentation/

[69] A. S. D. D. M. ABRAHAM, "Mobile Security Framework", 2017. [Online].

Available: https://github.com/MobSF/Mobile-Security-Framework-MobSF

[70] Checkstyle, "Overview", 2017. [Online].

Available: http://checkstyle.sourceforge.net/

[71] X. M. Repository, "Xposed Installer", 2017. [Online].

Available: http://repo.xposed.info/module/de.robv.android.xposed.installer

[72] M. Labs, "Drozer", 2017. [Online].

Available: https://labs.mwrinfosecurity.com/tools/drozer/

[73] A. SHAHIN, "Androl4b V.2", 2017. [Online].

Available: https://github.com/sh4hin/Androl4b

[74] A. AGRAWAL, "Appie – Android Pentesting Portable Integrated Environment", 2017. [Online].

Available: https://manifestsecurity.com/appie/

[75] acpm, "Inspeckage - Android Package Inspector", 2017. [Online].

Available: https://github.com/ac-pm/Inspeckage

[76] A. Labs, "AppUse", 2017. [Online].

Available: https://appsec-labs.com/appuse/