BILINGUAL PUBLISHING CO.
Pioneer of Global Academics Since 1984

Journal of Computer Science Research
https://ojs.bilpublishing.com/index.php/jcsr

ARTICLE

A Case Study of Mobile Health Applications: The OWASP Risk of Insufficient Cryptography

Suzanna Schmeelk[1*] Lixin Tao[2]

1. St. John's University, United States
2. Pace University, United States

ARTICLE INFO

Article history
Received: 25 December 2021
Accepted: 9 February 2022
Published Online: 24 February 2022

Keywords:
OWASP mobile threats
Cryptography
Mobile application
mHealth
Healthcare
Android

ABSTRACT

Mobile devices are being deployed rapidly for both private and professional reasons. One area of that has been growing is in releasing healthcare applications into the mobile marketplaces for health management. These applications help individuals track their own biorhythms and contain sensitive information. This case study examines the source code of mobile applications released to GitHub for the Risk of Insufficient Cryptography in the Top Ten Mobile Open Web Application Security Project risks. We first develop and justify a mobile OWASP Cryptographic knowledge-graph for detecting security weaknesses specific to mobile applications which can be extended to other domains involving cryptography. We then analyze the source code of 203 open source healthcare mobile applications and report on their usage of cryptography in the applications. Our findings show that none of the open source healthcare applications correctly applied cryptography in all elements of their applications. As humans adopt healthcare applications for managing their health routines, it is essential that they consider the privacy and security risks they are accepting when sharing their data. Furthermore, many open source applications and developers have certain environmental parameters which do not mandate adherence to regulations. In addition to creating new free tools for security risk identifications during software development such as standalone or compiler-embedded, the article suggests awareness and training modules for developers prior to marketplace software release.

1. Introduction

Smart mobile devices such as phones and tablets are being integrated rapidly into human life. The device usages range in mobility in that some are carried around daily for communications and others rest standalone to coordinate and provide human interaction for smart devices. Mobile applications are therefore employed for a wide-range of activities. The software assurance and resulting security risks of these mobile applications continue to

*Corresponding Author:
Suzanna Schmeelk,
St. John's University, United States;
Email: schmeels@stjohns.edu

DOI: https://doi.org/10.30564/jcsr.v4i1.4271

increase every year far out pacing legal regulations and ethical data training for the storage, use, and transfer of such private and sensitive information.

This paper explores building a knowledge-graph specific to mobile-device applications for the mobile risk of insufficient cryptography which can result in the loss of both data confidentiality and integrity. We report on a software assurance case study of healthcare specific mobile applications hosted on GitHub with respect to the OWASP Mobile Risk of Insecure Cryptography. Specifically we examine Android Java application source code as Android is reported to have over 2.8 billion active users with a global market share of 75 percent [1]. In fact, Curry [1] reports that over one billion Android smartphones were shipped in 2020. The loss of healthcare data confidentiality and integrity is further exacerbated with the fact that mobile applications can be connected to device identifiers and subsequently tracked. These aspects add higher degrees of risk to humans storing data and communicating information with mobile device applications.

2. Literature Review

Literature related to the OWASP Top 10 mobile risk of insufficient cryptography spans at least three pillars: software assurance, weakness analysis with ontology development, and other mobile device cryptography studies.

2.1 Cryptography Software Assurance

Software assurance specifically cryptographic best practices for software development one domain of literature for developing higher degrees of software assurance. A. M. Braga and R. Dahab [2] propose a methodology for development of secure cryptographic software agnostic to any programming language. The methodology is designed to provide a structured way to approach cryptography into secure development methods. The research is useful to inform the software development process and lifecycle.

Haney, Garfinkel, and Theofanos [3] identified challenges organizations face when developing cryptographic products. They are conducting a web-based survey of 121 individuals representing organizations involved in the development of products that include cryptography. The research found that participants used cryptography for a wide range of purposes, with most relying on generally accepted, standards-based implementations as guides. Their surveys reported on participants developing their own cryptography implementations by drawing on non-standard based resources during their development and testing processes. These results show that perhaps due to the lack of adequate resources and standardized training, cryptographic development and software assurance remains challenging to implement correctly.

Damanik and Sunaringtyas [4] reviewed the Open Web Application Security Testing Guide to determine and defend vulnerabilities identified in a web application, Sistem Informasi Akademik dan Pengasuhan (SIAP). Their research was specific to one particular web application.

2.2 Cryptography Ontologies and Weaknesses

The development of known cryptographic weaknesses and ontologies is another literature domain. Bojanova, Black, Yesha [5] reported on Cryptography Classes in Bugs Framework (BF): Encryption Bugs (ENC), Verification Bugs (VRF), and Key Management Bugs (KMN) but building a novel BF ontology with cryptography concerns at the National Institute of Standards and Technology (NIST). The ontology is currently updated and is linked to related risks identified in the Common Weakness Enumeration (CWE) [6], for example the 'CWE-780 Use of RSA Algorithm without OAEP.' The NIST BF encryption ontology remains under development and is agnostic to mobile devices.

Similar to categorizing weaknesses as in the CWE, the Common Vulnerability Enumeration (CVE) is a MITRE program to identify, define, and catalog publicly disclosed cybersecurity vulnerabilities. Lazar, Chen, Wang, and Zeldovich [7] examined reports to the 269 cryptographic vulnerabilities reported in the MITRE CVE from January 2011 to May 2014. Their results show that 17% of the bugs were in cryptographic libraries, and 83% of the reports were individual application misuses of cryptographic libraries. Overall, properly implementing cryptographic libraries and APIs remains a challenge across many domains.

2.3 Mobile Application Cryptography Studies

Mobile application research studies for the improvement of cryptography have been researched in the past few years. As cryptographic best practices change nearly annually, study reanalysis is necessary to keep pace with the changing cryptographic landscape. Egele, Brumley, Fratantonio, and Kruegel [8] introduced a static analysis tool CryptoLint to automatically check programs on the Google Play marketplace. They found that 88% applications of employing cryptographic APIs did not implement cryptography correctly.

Shuai, Guowei, Tao, Tianchang and Chenjie [9] introduced Cryptography Misuse Analyzer (CMA). Gao, Kong, Li, Bissyandé, and Klein [10] introduced CogniCryptSAST. Singleton, R. Zhao, M. Song and H. Siy, [11] introduced

FIREBugs. These static analysis tools were built to identify weaknesses in cryptography development based on best practices of that timeframe.

Gajrani, Tripathi, Laxmi, Gaur, Conti, and Rajarajan [12] introduce sPECTRA as an automated framework for analyzing wide range of cryptographic vulnerabilities in Android finding that 90% of the applications analyzed had cryptographic weaknesses.

As cryptography industry requirements change rapidly with changes to language APIs and the identification of both novel attacks and found weaknesses, actual weakness identification through static analysis tool pattern matching also must be updated to reflect the changing industry landscape causing the need for program reanalysis based on current best practices, regulations, and industry needs.

3. OWASP Top Mobile Risk Ontologies

The Open Web Application Security Project (OWASP) is a nonprofit foundation that works to improve the security of software with global participation and collaboration. The organization creates a forum for industry, academic, and government leads to discuss current best computing practices. One of the projects maintained by OWASP is a list of the reported Top 10 Mobile Risks to mobile applications. The list notes security concerns for mobile applications' data, internal/external device communications, among other risks. The actual OWASP risks have remained since the last publication in 2016. The last risk iteration was a variation from the risks reported in 2014. Although the risks remain the same, the supporting OWASP best practice guidance appears to be dynamically updated periodically. We develop a knowledge graph based on the OWASP guidance. A useful attribute of knowledge graphs is that they can expand with time so that we can see what has changed in security concerns over time. Building such a domain graph aids both software assurance tools and techniques. Deprecated security concerns can easily be traced in the graph along with design changes benefiting all phases of the secure software development lifecycle (sSDLC).

3.1 2014 Threat 6: Broken Cryptography

The OWASP 2014 Mobile Threat 6 is Broken Cryptography. Broken cryptography can potentially lead to data compromise in both confidentiality and integrity. To control data confidentiality, cryptography is primarily implemented with key-centric encryption/decryption methodologies. To mitigate from data integrity risks, cryptography can be used to generate cryptographic message digests to numerically validate data. These techniques coexist with repudiation techniques, for example with digital signatures.

Other security concerns in the CIA-model revolve around data and service availability. Availability is typically controlled with other primary mitigation controls; however, if there exists a lack of direct mitigating controls, further cryptographic weaknesses further expose services breaking defense-in-depth.

Figure 1 shows our knowledge graph for the OWASP threat of Broken Cryptography, labeled *M6_Broken_Cryptography*. Since the knowledge graph for the OWASP threat of Broken Cryptography is extensive, we review each sub-tree from Figure 1 in different figures, specifically Figures 2-9. Figure 1 is shown to give a full overview of the breadth and inter-connections for the OWASP threat.

From the perspective of an application, there are four main relationships for insufficient cryptography. First, insufficient cryptography can potentially *resultFrom* device specific issues such as compromised hardware. In addition, insufficient cryptography can *resultFrom* cryptography application programming interface (API) weaknesses or misuses. Third, insufficient cryptography can *resultFrom* improper key generation and management. Forth, broken cryptography can *resultFrom* entirely not using cryptography when it is needed.

Cryptography algorithms with weak environment parameters, shown in Figure 2, can cause higher security risks. The mobile threat of broken cryptography due to the implementation of a weak parameters can *resultFrom* from four main issues. First, cryptographic parameters such as weak initialization vector (IV) and improper salts will increase the risk of the output cipher text to be easily decoded. Second, weak algorithms [13] (e.g. DES, 3DES, SHA1, MD5) are known to have exploits and have been deprecated by industry and the U.S. Federal government. Third, weak key generation (e.g. less than 128-bits, non-random, etc.) and management are also known susceptible to brute force attacks [14]. Fourth, other predictable environment components such as imported flawed libraries or flawed cryptographic providers are means for security concerns. These four predominate issues can cause weak cryptographic output increasing the risk of information exposure to the loss of integrity and confidentiality.

Improperly implemented cryptography algorithms, shown in Figure 3, can cause higher security risks. Two main groups of algorithms that fall into this category are improperly implemented message digests and ciphers. Figure 4 shows sample best practice code for encryption from the Carnegie Mellon University Software Engineering Institute rules and recommendations [13].

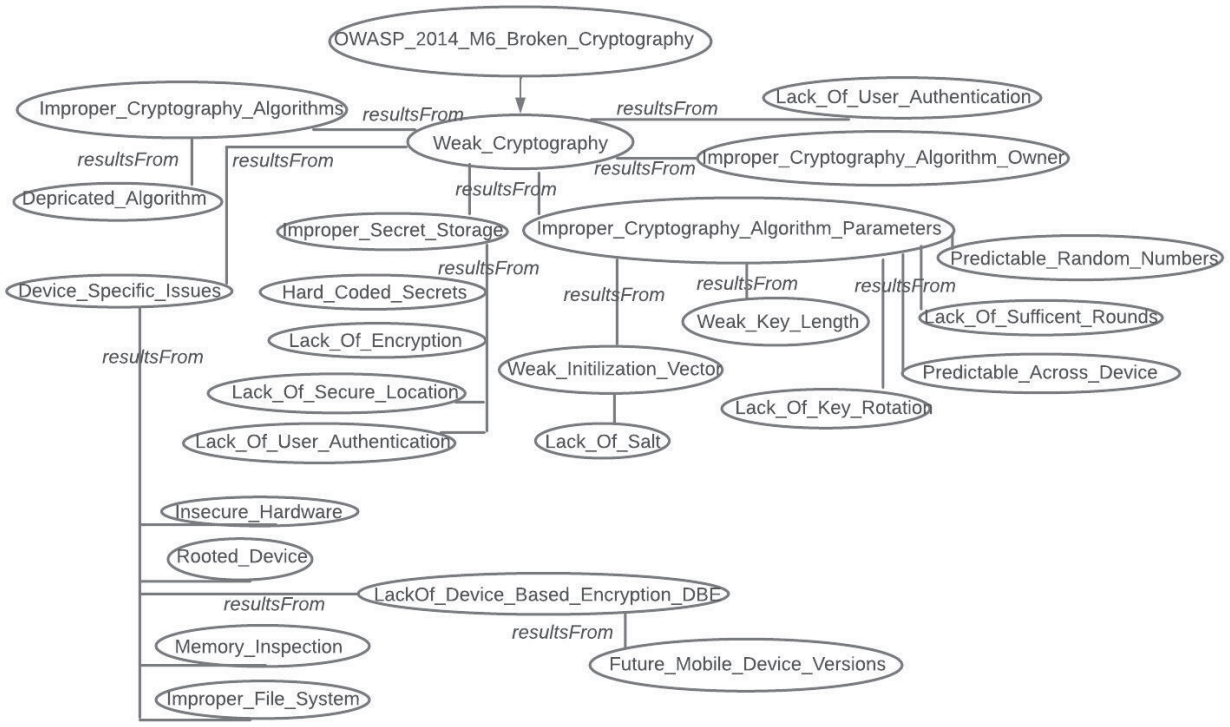

Figure 1. OWASP Mobile 2014 Threat: Broken Cryptography

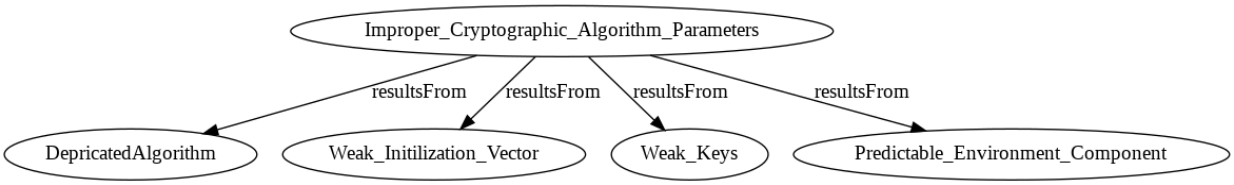

Figure 2. Improper_Cryptographic_Algorithm_Parameters

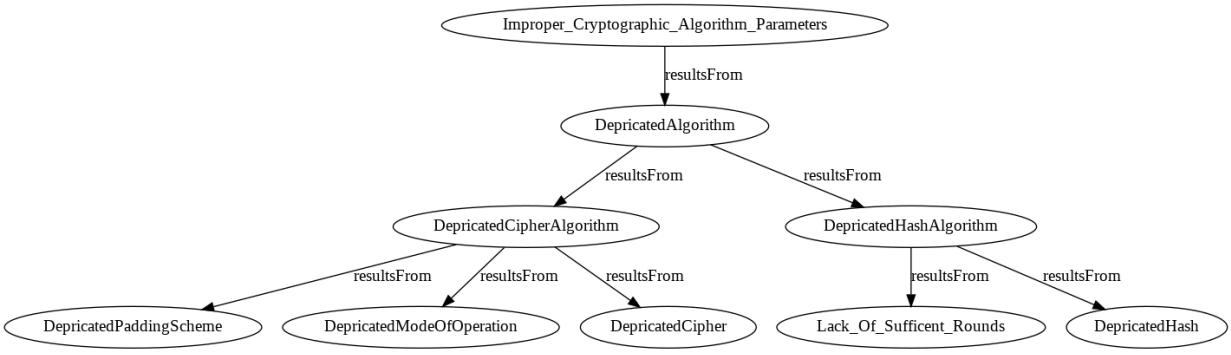

Figure 3. Deprecated Algorithm Parameters

Five common risks can be specific to devices, our knowledge-graph can be seen in Figure 5. Although we show the relationship with broken cryptography in our knowledge graph, software analysis of these underlying concerns breaking cryptography are directions for future research. First, a common device specific concern is related to hardware—either directly through compromised hardware, or indirectly through a side channel attack on the system power analysis. This issue is difficult to detect in a mobile application unless a watchdog application is involved but it faces similar issues. Insufficient hardware-specific power constraints can also cause in-

effective cryptography. Devices can linger on networks for many years [18]. Device platforms may not be able to keep up with modern cryptographic requirements for multiple reasons [19]. Second, a rooted or jail-broken device compromises application access controls. In such cases, applications should detect that they are running on a compromised system. Third, tools that harvest keys and passwords from memory are another device specific concern. Fourth, and improperly constructed file system for data storage can result in broken cryptography. Lastly, lack of device and/or file based encryption can also cause broken cryptography.

```
public static byte[] encrypt_cbc(SecretKey skey, String
plaintext) {
 /* Precondition: skey is valid … . */
 try {
  byte[] ciphertext = null; Cipher =
Cipher.getInstance("AES/CBC/PKCS5Padding");
  final int blockSize = cipher.getBlockSize();
  byte[] initVector = new byte[blockSize];
  (new SecureRandom()).nextBytes(initVector);
  IvParameterSpec ivSpec = new
IvParameterSpec(initVector);
  cipher.init(Cipher.ENCRYPT_MODE, skey, ivSpec);
 byte[] encoded =
plaintext.getBytes(java.nio.charset.StandardCharsets.UTF_8);
 ciphertext = new byte[initVector.length +
cipher.getOutputSize(encoded.length)]; ….
  // Perform encryption
  cipher.doFinal(encoded, 0, encoded.length, ciphertext,
initVector.length);
  return ciphertext; } catch …
```

Figure 4. Condensed CMU SEI AES Implementation [13]

Insufficient cryptography can also arise from not properly encrypting certain sensitive data (i.e. physical domain). This lack of sufficient cryptography can occur on an endpoint communication channel during transmission. This lack of sufficient cryptography can occur on device without device based encryption (DBE) as DBE is a feature of only Android 5 [20]. Google has also issued a warning for pre-Android5 devices which have been upgraded, "Caution: Devices upgraded to Android 5.0 and then encrypted may be returned to an unencrypted state by factory data reset." [20] DBE will be deprecated in future versions of Android, perhaps due to performance constraints [86]. Google currently has posted, "Caution: Support for full-disk encryption is going away. If you're creating a new device, you should use file-based encryption." [21] This lack of sufficient cryptography can occur on a file without file based encryption (FBE) [22]. Google has already issued OS version specific issues in relation to FBE. For example, the Android Application API currently reads, "Caution: On devices running Android 7.0-8.1, file-based encryption can't be used together with adoptable storage [22]". On devices using FBE, new storage media (such as an external card) must be used as traditional storage. Devices running Android 9 and higher can use adoptable storage and FBE [22].

Cipher keys and passwords are known to have software assurance concerns for different reasons, as shown in Figure 6. First, keys may not be stored correctly (e.g. in Android KeyStore [14]) and therefore subject to compromise. An example of such a broken scenario is when a cryptographic key or password is stored in plaintext on a shared space next to the encrypted data.

Weak keys are known to cause other insufficient cryptographic problems. Second, the key derivation function may not be best practice, based on cryptographic random numbers, or have sufficient iterations. Third, the key length changes with industry best practices based on computational power. In such cases, legacy systems relying on shorter keys increase the risk around real time brute force attempts [15]. Finally, key rotations may not follow best practices.

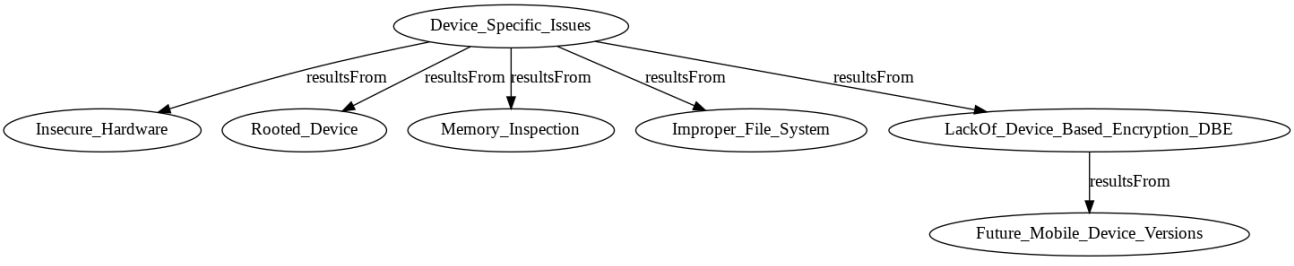

Figure 5. Device Specific Issue

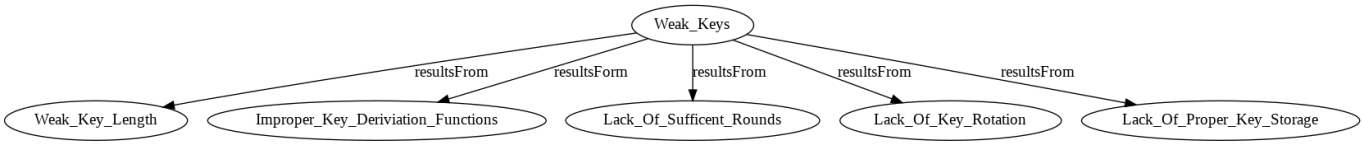

Figure 6. Knowledge graph for weak keys concerns

```
public static SecretKey generateKey() {
try {
 KeyGenerator kgen =
 KeyGenerator.getInstance("AES");
 kgen.init(256);
 return kgen.generateKey();
 } catch (NoSuchAlgorithmException e)
{ throw new
IllegalStateException(e.toString());} }
```

Figure 7. Condensed CMU SEI Key Generation [14]

CMU SEI [16] provides an example of a more secure implementation for storing passwords. Software can be analyzed by employing static analysis techniques (e.g. context sensitive analysis, string analysis, variable propagation, etc.) to detect the concerns on to password algorithms, iterations, salts, and other issues.

```
final class Password {
private SecureRandom random = …
private final int SALT_BYTE_LENGTH = 12;
private final int ITERATIONS = 100000;
private final String ALGORITHM =
"PBKDF2WithHmacSHA256";
/* Set password to new value, zeroing … */
void setPassword(char[] pass)  throws …
byte[] salt = new
byte[SALT_BYTE_LENGTH];
random.nextBytes(salt);
saveBytes(salt, "salt.bin");
byte[] hashVal = hashPassword(pass, salt); …
} ...
/* Encrypts password & salt and zeroes both */
private byte[] hashPass (char[] pass, byte[]
salt)
throws GeneralSecurityException {
KeySpec spec = new PBEKeySpec(pass, salt,
ITERATIONS); …
SecretKeyFactory f =
SecretKeyFactory.getInstance(ALGORITHM);
return f.generateSecret(spec).getEncoded();} ...
```

Figure 8. Condensed CMU Passwords Implementation [16]

3.2 2016 Threat 5: Insufficient Cryptography

The OWASP 2016 Mobile Threat Insufficient Cryptography (IC) is the fifth risk, labeled in Figure 9 as *OWASP_2016_M5_Insufficient_Cryptography*. The 2016 threat has the same implications as the 2014 [17]. The knowledge graph shows that the same general cryptographic concerns from 2014 directly translates into the risks of 2016, unlike many other risks from 2014 that were rearranged, removed, or merged together in the 2016 OWASP list. Differences between the years lie in identified weaknesses within the ciphers, digests, devices, and key management.

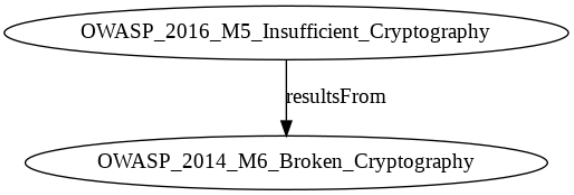

Figure 9. OWASP 2016 M5 Insufficient Cryptography

In summary, we have identified sub-areas where software assurance methodologies can be developed and improved to detect the OWASP Mobile Threat of Insufficient Cryptography. The standard Android encryption API calls include creating keys, encrypting, and decrypting, are all detectable using assurance methodologies such as program analysis.

4. Analysis Results

We examined the source code of 203 mobile applications written to store, track, and communicate healthcare related data. Healthcare data is typically sensitive information and is only regulated under certain conditions. For example, research shows that HIPAA and HITECH only apply to covered entities [23]. For data not covered under HIPAA, the FCC becomes involved when breaches affect > 500 individuals [24]. Smaller mobile applications, which may only serve a small population segment, may not fall under any regulations.

Specifically, we examined the source code of healthcare applications with publicly available source code to gain a sense of how they were implementing cryptography, if at all, in their programs.

The analyzed applications stored health data for many health-related concerns including mental health, pregnan-

cy, exercise management, hypertension, among other sensitive issues.

In total, we examined each application based on the knowledge-graphs reported in Section 3 to gain insights into these applications' source code confidentiality and integrity security for data-at-rest. We applied pattern-matching criteria to identify source code with concerns reported in our knowledge-graph.

Our main finding was that some of the 203 mobile applications made attempts at data-at-rest cryptography but were unsuccessful in perfectly implementing all data-at-rest knowledge graph elements reported in Section 3. These weaknesses in implementation cause a breakdown in confidentiality and integrity for people storing, using, and transmitting their health data with any of these applications.

4.1 Application Cryptography Utilization

Historically there have been two main packages in the Oracle Application Programming Interface (API) for Java cryptographic implementations. The message digests (hash) functions, secure random number generator for cryptography, certificates, key management implementations are contained in the java.security packages, known as the Java Cryptography Architecture (JCA). The key generation, agreement, and cipher algorithms are contained in the javax.crypto package, known as the Java Cryptography Extension (JCE). "Prior to JDK 1.4, the JCE was an unbundled product, and as such, the JCA and JCE were regularly referred to as separate, distinct components. As JCE is now bundled in the JDK, the distinction is becoming less apparent. Since the JCE uses the same architecture as the JCA, the JCE should be more properly thought of as a part of the JCA. [25]" When analyzing source code, one key indicator of properly implemented cryptography is by employing the standard Oracle API. Implementing one's own cryptographic algorithms can be successful but is highly prone to error.

Of the 203 mobile application source code analyzed, 25 imported the Oracle API cryptography libraries in their java source code, with three of these applications importing only cryptography policies, keys, or crypto related exceptions rather than importing libraries needed for cipher and/or hash algorithms. Analysis showed that a few source repositories may contain cryptography within embedded mobile application bytecode and is outside the scope of this research due to its low-likelihood that it adequately protects the confidentiality and integrity of the contained healthcare data.

4.2 Proper Confidentiality Implementations

Confidentiality mitigation implementation which results in low risk of data exposure have many elements that need to be satisfied as reported in Section 3. In this section we report on the 22 mobile applications which imported the proper cryptography libraries. Of these 22 applications, only 10 applications imported JCE extensions. We report on these 10 applications cryptography implementations in the following subsections.

4.2.1 Proper Cipher Algorithms

One important aspect to properly implementing cryptography is to employ the non-deprecated ciphers. Sheth [26] and CMU SEI [27] indicated that AES remains a compliant symmetric key algorithm for storing data-at-rest. Other properly implemented algorithms such as RSA are not entirely wrong given certain data-exchange use cases but may not be the best choice meeting same-device data-at-rest requirements. One application did interface with a blockchain therefore an RSA implementation could be needed. Of the 10 applications which imported JCE libraries, 6 applications employed the AES (5 apps) only, RSA (1 app) only, and 2 applications used both AES and RSA algorithms. One application employed DES algorithm which has been deprecated for years. The remainder either had their own encryption generation or did not import JCE libraries for encryption.

4.2.2 Proper Cipher Modes

In general, a cipher mode of operation lowers risks of generating predictable ciphertext. Sheth [26] reports on GCM and CBC mode as having lower risks from cryptanalyses attacks. The JCR currently supports other non-best practice mode operations, perhaps for legacy systems. Of the six applications that implemented non-deprecated ciphers, only four had known proper cipher modes implemented. Two applications relied on the defaults by employing either Cipher.getInstance ("AES") or "RSA" which do not default to best practices as guided by CMU SEI [27]. Four applications properly implemented the symmetric cipher mode. One of these four proper implementations also had an improper implementation of their key storage using the symmetric algorithm with transformation string "AES/ECB/PKCS5Padding" for key storage rather than storing a salted hash of the password as described by CMU SEI in Figure 7. Of the three applications with RSA implementations, one application employed the default "RSA" mode, one employed "ECB" mode, and one employed "NONE" mode subject to conflicting RSA

mode guidance.

4.2.3 Proper Cipher Padding.

Padding schemes can be employed to pad cleartext into acceptable cipher algorithm block sizes. Sheth [26] reports the best practice of employing PKCS5Padding or OAEP-With* padding schemas, although the JCR supports other non-best practice padding schemas. Of the applications with properly implemented modes, only one application properly implemented one of these padding schema.

4.2.4 Other Confidentiality Parameters

Other best practice parameters to consider during the encryption processes are employing cryptographically random numbers as initialization vectors (IVs) (i.e., nonces) [28,29]. Sheth [26] advised to make sure only a small number of plaintexts are encrypted with the same key and IV pair. The one application which properly implemented the symmetric cipher transformations string did not initialize the cipher with a cryptographically random IV therefore not implementing cryptography correctly.

4.3 Proper Integrity Implementations

We examined all 203 applications for proper data integrity implementations. Integrity risks can be mitigated through the use of cryptographic hash algorithms. In cases where data is changing on a regular basis and cannot be properly compared against a known duplicate, non-deprecated hash algorithms are essential to protect data integrity. In the case of these applications which do not contain data snapshot cryptographic hashes, non-deprecated algorithms are essential. In addition, when storing cryptographic password hashes, best practice mandates minimum generation iterations (e.g. NIST [30] standards identify a minimum of 10,000 iterations based on computing resources), certain creation algorithms, and adding randomness via salting.

4.3.1 Proper Message Digest Algorithms

Current best-practices mandate SHA2 (with SHA-512 or higher) or SHA3 family of cryptographic hash algorithms. We identified only 11 applications employing cryptographic hash algorithms, but all using MD5, SHA1, or other SHA algorithms below SHA-512 for integrity needs. Therefore, we were unable to identify any applications with proper cryptographic message digest algorithms.

4.3.2 Other Integrity Parameters

Other integrity parameters to consider are needed when storing cryptographic password hashes as reported by CMU SEI in Figure 7. In such cases, the salting and iterations are essential along with proper algorithms. We identified only one application attempting to store passwords correctly. The application applied a secure password generation algorithm of PBKDF2, but it was applied with only 100 iterations, which is below industry best practice standards such as NIST's recommendation of at least 10,000 iterations [30].

4.4. Other Cryptographic Issues

During application analysis we identified some other ancillary cryptographic issues within the mobile application source code. These findings may more appropriately belong in the OWASP secure storage knowledge-graph discussion, but since they include cryptographic techniques, we will reference the issues. We identified two applications which had upgraded their internal database from the standard SQLite database that comes with the device to instead implement a more secure version, the *net.sqlcipher* SQLiteDatabase [31]. This particular implementation claims to add cryptography to the database so that stored information is not stored in plaintext as it is in the standard SQLite database. The library analysis of the *sqlcipher* database cryptography usage and respective key management is outside the scope of this research. A knowledge graph is ideal for representing hybrid security concerns where mitigations overlap such as cryptographically secure storage implementations, where if any part of the secure implementation contains weaknesses, the overall concept will be of high risk.

5. Future Work

Best practice cryptographic implementations require community effort in maintain. Ancillary concerns such as cryptographic key generation and management and cryptography in commonly imported libraries are areas of future research. Similarly, cryptography encompasses certain aspects of data-in-motion however, there remains a vague distinction between the OWASP top ten threat of insufficient cryptography and other OWASP top ten threats, such as that of insecure communications. Knowledge-graphs can be useful to show longitudinal relationships between security concerns. These are other areas of future research. Lastly, the case study shows the importance of building

new tools and techniques to aid the secure software development lifecycle to identify weaknesses in cryptographic implementations and provide secure software training—whether developer or penetration testing. Currently, for example, penetration testing remains an art rather than science since the field lacks standardization. The creation of knowledge-graphs can be useful to provide standardization and to add risk ratings to inform sector-wide risk likelihoods. There remains a lot of further research to develop such standardized security ontologies.

6. Conclusions

This research examined the OWASP Top Ten mobile device security threats focusing on the OWASP Mobile Application Threat of insufficient cryptography. We first contributed the development of mobile device specific knowledge graph for insufficient cryptography. From the knowledge-graph we analyzed 203 mobile device applications source code uploaded to GitHub. The analyzed applications where healthcare applications that collect sensitive human information such mental health, exercise routines, pregnancy indicators, skin photographs, and other important body information needed for health. We were unable to identify any application that properly implemented confidentiality and integrity needs.

As our world becomes more interconnected, it is essential that we build more robust tools to identify privacy and security weaknesses. Many different software and software developers at large, such as developers of free healthcare applications, are neither required by regulations to implement security features nor have access, awareness, or training for such security features. The industry need has become dire for creating access to security training and tools to develop more secure applications especially when applications store extremely sensitive information about humans greatly affecting both their own lives and those of their family.

Conflict of Interest

The authors declare no conflict of interest.

References

[1] Curry, D., 2021. Android Statistics (2021) https://www.businessofapps.com/data/android-statistics

[2] Braga, A.M., Dahab, R., 2016. Towards a Methodology for the Development of Secure Cryptographic Software. 2016 International Conference on Software Security and Assurance (ICSSA). pp. 25-30. DOI: https://doi.org/10.1109/ICSSA.2016.12

[3] Haney, J.M., Garfinkel, S.L., Theofanos, M.F., 2017. Organizational practices in cryptographic development and testing. 2017 IEEE Conference on Communications and Network Security (CNS). pp. 1-9. DOI: https://doi.org/10.1109/CNS.2017.8228643

[4] Nanisura Damanik, V.N., Sunaringtyas, S.U., 2020. Secure Code Recommendation Based on Code Review Result Using OWASP Code Review Guide. 2020 International Workshop on Big Data and Information Security (IWBIS). pp. 153-158. DOI: https://doi.org/10.1109/IWBIS50925.2020.9255559

[5] Bojanova, I., Black, P.E., Yesha, Y., September 25-28, 2017. Cryptography Classes in Bugs Framework (BF): Encryption Bugs (ENC), Verification Bugs (VRF), and Key Management Bugs (KMN). IEEE Software Technology Conference (STC 2017), NIST, Gaithersburg, USA.

[6] MITRE, 2021. CWE-780 Use of RSA Algorithm without OAEP. https://cwe.mitre.org/data/definitions/780.html

[7] Lazar, D., Chen, H.G., Wang, X., Zeldovich, N., 2014. Why does cryptographic software fail? a case study and open problems. In Proceedings of 5th Asia-Pacific Workshop on Systems (APSys '14). Association for Computing Machinery, New York, NY, USA. Article 7, 1-7. DOI: https://doi.org/10.1145/2637166.2637237

[8] Egele, M., Brumley, D., Fratantonio, Y., Kruegel, Ch., 2013. An empirical study of cryptographic misuse in android applications. In Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security (CCS '13). Association for Computing Machinery, New York, NY, USA. pp. 73-84. DOI: https://doi.org/10.1145/2508859.2516693

[9] Shuai, S., Guowei, D., Tao, G., Tianchang, Y., Chenjie, S., 2014. Modelling Analysis and Auto-detection of Cryptographic Misuse in Android Applications. 2014 IEEE 12th International Conference on Dependable, Autonomic and Secure Computing. pp. 75-80. DOI: https://doi.org/10.1109/DASC.2014.22

[10] Gao, J., Kong, P., Li, L., Bissyandé, T.F., Klein, J., 2019. Negative Results on Mining Crypto-API Usage Rules in Android Apps. 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). pp. 388-398. DOI: https://doi.org/10.1109/MSR.2019.00065

[11] Singleton, L., Zhao, R., Song, M., Siy, H., 2019. FireBugs: Finding and Repairing Bugs with Security Patterns. 2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft). pp. 30-34. DOI: https://doi.org/10.1109/MOBILESoft.2019.00014

[12] Gajrani, J., Tripathi, M., Laxmi, V., Gaur, M.S., Conti, M., Rajarajan, M., 2017. sPECTRA: A precise framework for analyzing cryptographic vulnerabilities in Android apps. 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC). pp. 854-860.
DOI: https://doi.org/ 10.1109/CCNC.2017.7983245

[13] CMU SEI, 2021. MSC61-J. Do not use insecure or weak cryptographic algorithms https://wiki.sei.cmu.edu/confluence/display/java/MSC61-J.+Do+not+use+insecure+or+weak+cryptographic+algorithms

[14] Sabt, M., Traore, J., 2016. Breaking Into the KeyStore: A Practical Forgery Attack Against Android KeyStore. in 21st European Symposium on Research in Computer Security (ESORICS), Heraklion, Greece.

[15] Sincerbox, C., March/April 2014. Security Sessions: Exploring Weak Ciphers. [Online]. Available: https://electricenergyonline.com/energy/magazine/779/article/Security-Sessions-Exploring-Weak-Ciphers.htm

[16] CMU SEI, 2021. MSC62-J. Store passwords using a hash function https://wiki.sei.cmu.edu/confluence/display/java/MSC62-J.+Store+passwords+using+a+hash+function

[17] OWASP, 2021. Mobile Top 10 2016-M5-Insufficient Cryptography. [Online]. Available: https://www.owasp.org/index.php/Mobile_Top_10_2016-M5-Insufficient_Cryptography

[18] Cole, S., October 30 2018. New Study Suggests People Are Keeping Their Phones Longer Because There's Not Much Reason to Upgrade.

[19] Henry, J., 3 August 2018. 3DES is Officially Being Retired. [Online]. Available: https://www.cryptomathic.com/news-events/blog/3des-is-officially-being-retired

[20] Google, 26 January 2019. Full-Disk Encryption. [Online]. Available: https://source.android.com/security/encryption/full-disk

[21] Google, 26 January 2019. Encryption. [Online]. Available: https://source.android.com/security/encryption

[22] Google, 1 January 2019. File-Based Encryption. [Online]. Available: https://source.android.com/security/encryption/file-based

[23] HHS, 2021. Covered Entities and Business Associates. https://www.hhs.gov/hipaa/for-professionAls/covered-entities/index.html

[24] FTC, 2021. Health Breach Notification Rule. https://www.ftc.gov/enforcement/rules/rulemaking-regulatory-reform-proceedings/health-breach-notification-rule

[25] Oracle, 2021. Java SE 14 Security Developer's Guide. https://docs.oracle.com/en/java/javase/14/security/java-cryptography-architecture-jca-reference-guide.html

[26] Mansi Sheth, 2017. Encryption and Decryption in Java Cryptography. https://www.veracode.com/blog/research/encryption-and-decryption-java-cryptography

[27] CMU SEI, 2021. DRD17-J. Do not use the Android cryptographic security provider encryption default for AES.

[28] CMU SEI, 2021. MSC63-J. Ensure that SecureRandom is properly seeded https://wiki.sei.cmu.edu/confluence/display/java/MSC63-J.+Ensure+that+SecureRandom+is+properly+seeded

[29] CMU SEI, 2021. MSC02-J. Generate strong random numbers.

[30] Grassi, P., Fenton, J., Newton, E., Perlner, R., Regensheid, A., Burr, W., Richer, J., 2017. National Institute of Standards and Technology (NIST) Special Publication 800-63B https://pages.nist.gov/800-63-3/sp800-63b.html

[31] Zetetic LLC, 2021. android-database-sqlcipher https://github.com/sqlcipher/android-database-sqlcipher