Tracing Cryptographic Agility in Android and iOS Apps

Kris Heid[a], Jens Heider[b], Matthias Ritscher and Jan-Peter Stotz
ATHENE - National Research Center for Applied Cybersecurity,
Fraunhofer Institute for Secure Information Technology SIT, Rheinstraße 75, 64295 Darmstadt, Germany

Keywords: Cryptography, Agility, Android, Static Analysis.

Abstract: Cryptography algorithms are applicable in many use cases such as for example encryption, hashing, signing. Cryptography has been used since centuries, however some cryptography algorithms have been proven to be easily breakable (under certain configurations or conditions) and should thus be avoided. It is not easy for a developer with little cryptographic background to choose secure algorithms and configurations from the plenitude of options. Several publications already proved the disastrous cryptographic quality in mobile apps in the past. In this publication we research how cryptography of the top 2000 Android and iOS applications evolved over the past three years. We analyze at the example of the weak AES/ECB mode how and why apps changed from an insecure to a secure configuration and vice versa.

1 INTRODUCTION

Cryptography already accompanies humankind since centuries. First well known Caesar cipher dates back to 100BC and was then used to secure communication. This encryption method and many newer methods are nowadays crackable. Such cryptographic methods are either broken through mathematical workarounds or through the constantly increasing processing power of modern computers, which allow many complex mathematical operations in shorter time. Thus, from time to time cryptographic algorithms have to be updated to maintain their security today and in the near future. This is where *cryptographic agility* comes into play, which describes a practice paradigm where interfaces support multiple cryptographic algorithms which in turn allows an easy upgrade to new, secure standards.

For programming languages, this means that there is an API given which is constant and cryptographic algorithms are easily exchangeable during initialization.

Previous works(Chatzikonstantinou et al., 2016; Wickert et al., 2021) have shown that it is not easy for developers to choose secure cryptographic configurations. Oftentimes, boilerplate code from developer forums(Braga and Dahab, 2016a) is copied and therewith outdated cryptographic configurations. However, outdated cryptography can not simply be banned in the APIs since legacy code must be supported. It is also sometimes complex to detect a cryptographic misuse. For example, initializing a cryptographic algorithm once is fine, however using it a second time with the same initialization vector is not secure.

As Section 2 will point out, a plenitude of works already exist which highlight common insecure configuration and pitfalls. Other works also monitor current situation for misconfigured cryptography in projects or apps. In this work we would like to focus on the change of cryptography algorithms from insecure configurations to secure configurations and sometimes even vice versa over time. Especially we will focus on AES/ECB mode since it is a very common insecure algorithm. We analyze the used cryptography of more than 4000 iOS and Android apps. Test apps' cryptography is analyzed in their current version as well as the cryptography used three years ago in 2019.

The remainder of this paper is structured as follows. Section 2 highlights related work in this area and the unique selling points of our work. The used analysis environment for iOS and Android apps is described in Section 3 along with the selection process for analyzed apps. Section 4 shows cryptography misuse in context of time. Different flaws are identified along with common causes for misconfiguration. The last section concludes the paper and proposes possible future work.

[a] https://orcid.org/0000-0001-7739-224X
[b] https://orcid.org/0000-0001-8343-6608

Heid, K., Heider, J., Ritscher, M. and Stotz, J.
Tracing Cryptographic Agility in Android and iOS Apps.
DOI: 10.5220/0011620000003405
In *Proceedings of the 9th International Conference on Information Systems Security and Privacy (ICISSP 2023)*, pages 38-45
ISBN: 978-989-758-624-8; ISSN: 2184-4356

2 RELATED WORK

Related works in this area are separated into two main categories. On the one hand, works that capture the current situation in cryptography misuse by analyzing existing apps. On the other hand, there are publications covering where the heavy cryptography misuse comes from and how the problem can be tackled and eliminated.

Starting off in 2013, first publications focused on the topic of cryptography misuse in mobile applications. One of the first and also well known publication by Egele et al.(Egele et al., 2013) covers the topic very well. More than 11.000 Android apps were statically analyzed with their CryptoLint tool. Results revealed that around 87% of these apps violate the cryptographic algorithm's security due to misuse or misconfiguration. This work also proposes a list of rules or common pitfalls with cryptography which is used to check the apps for misconfiguration. Chatzikonstantinou (Chatzikonstantinou et al., 2016) and Lazar (Lazar et al., 2014) confirm previous results though a static and dynamic analysis. In 2018, the proposed rules and pitfalls are projected on iOS applications by Feichtner et al.(Feichtner et al., 2018). Even though iOS apps are programmed in another language, namely Objective-C and Swift and not Java like in Android, the results are similar. Around 82% of all checked apps contained misuse. These publications proved the disastrous cryptography state on the mobile platforms.

The aftermath of these results were several publications that tried to identify the cause of the problem. Hazhirpasand et al.(Hazhirpasand et al., 2019) tried to correlate developer's experience with the cryptography quality, but could not see a relation. Acar et al.(Acar et al., 2017) compared different python APIs through a field study by assigning developers cryptography tasks with different libraries. They concluded that simple APIs avoid misuse through a narrow decision space, but the main factor for secure cryptography is identified as a good documentation with code examples. Wickert et al.(Wickert et al., 2021) investigated python's cryptographic API and came to the conclusion, that in contrast to Java with 87%, only 52% of the statically analyzed projects contained cryptography misuse. They concluded that the python API seems to be easier to use.

Different publications (Braga and Dahab, 2016b; Fischer et al., 2017; Meng et al., 2018) identified bad code quality of code snippets in online forums, which were copied and pasted to projects as a severe source of cryptography misuse. Fischer et al.(Fischer et al., 2017) identified that almost all (98%) of all copied code snippets contained cryptography misconfiguration or misuse.

Gao et al.(Gao et al., 2019) was the first work to look at the time flow on how apps changed from misconfigured cryptography to secure configurations. Their initial theory, that app developers update cryptographic API usage to fix misuse, proved to be mostly wrong. In their observations, updates fixing cryptographic misuse seemed unintentionally, only to be re-introduced by later updates. Their work comes very close to our approach.

2.1 Contribution

Our work differentiates from previous work by:

- Observe the change in cryptography misuse in a larger time span of three years instead of making only a current snap-shot like other works(Egele et al., 2013; Chatzikonstantinou et al., 2016; Feichtner et al., 2018).
- We first give an overview and then focus on one specific misuse to find detailed traces of its origin, instead of a very broad analysis (Gao et al., 2019).
- Comparing numbers of outdated cryptography modes in own/custom app code versus third party library code for a representative example.

Through such a focused process we are able to specifically identify causes for misconfigured cryptography. The majority of the discovered weaknesses should be solvable by only a few responsible developers as stated in future work.

3 ANALYSIS ENVIRONMENT

We use static analysis methods to discover the used cryptographic algorithms, modes and key lengths for iOS and Android apps. The Android environment is based on soot(Lam et al., 2011; Vallée-Rai et al., 1999) and the iOS analysis uses angr(Shoshitaishvili et al., 2016; Stephens et al., 2016; Shoshitaishvili et al., 2015) in combination with IDA Pro's[1] FindCrypt2 plugin[2].

3.1 Android Analysis

The analysis environment loads the app with soot and searches for usage of `Cipher.getInstance(String transformation, ...)` to find the cryptographic

[1]https://hex-rays.com/ida-pro/

[2]https://hex-rays.com/blog/findcrypt2/

algorithm and `KeyGenerator.init(int size)` to find the encryption key size. Respective methods to find used hashing and signature algorithms were used as well. Third party security providers like the commonly used Bouncy Castle[3] are registered in the app but used over the same API as the shipped security providers. This makes recognizing third party library algorithms also well recognizable with our approach. The used soot tool also offers the technique of *symbolic execution* to resolve method arguments that are not directly provided but reside inside a field or are dynamically built.

3.2 iOS Analysis

Used Cryptography in iOS apps are on the one hand discovered via IDA Pro's FindCrypt2 plugin. FindCrypt2 has a large list of magic constants which almost all cryptographic algorithms use in the program body. Occurrences of these constants are found and an association with the respective cryptographic algorithm can be made. However, this approach does not provide information about the used cryptographic configuration and does not find calls to cryptographic APIs. Therefore, apps are additionally loaded into the angr analysis framework for an investigation of calls to cryptographic API calls, such as calls to *Common Crypto* library and *CryptoKit*. Parameters of cryptographic API calls are then resolved with symbolic execution.

3.3 App Selection Process

We select the most popular 2000 apps (top apps) from each Google Play and Apple's App Store as representatives to be tested since they have widespread usage among consumers. From this group, we select apps which already existed three years ago, since it is our goal to find changes throughout the years. We chose three years to have a sufficiently large time span for new app versions containing fundamental changes. Also, the existence and continuous updates of an app in the stores for at least three years are hints for some professionalism during development. However, going further back than three years would shrink the app selection to a point which would not allow well-founded statements. 64.1% of the top 2000 iOS apps existed three years ago and 31.4% of the Android top 2000 apps have a three-year-old version. One can see, that there is a much higher fluctuation in Android top apps. We assume that it is easier for apps to rise to the top apps in Google Play, but we can't verify our assumption since the exact calculation metric for both stores remains secret. In the following we refer to those apps as consumer apps since we also have a secondary group called business apps.

We run an automated app test service where customers are able to manually upload custom apps to be tested. These business apps can either be public apps also available in public app stores or apps exclusively available to a customer. Companies using such services usually aim to secure their employee's mobile devices through tests for security vulnerabilities. We assume that such business apps meet higher security and therewith cryptography standards and would very well reflect apps often used in a business environment. In total, we have 142 Android and 478 iOS business apps with a 2022 version and a three years old counterpart. The larger number of iOS business apps might be since iOS is widely used in business environments.

4 EVALUATION

We focus on hashing and encryption algorithms in this evaluation. For both methods, we evaluate the past usage of cryptographic methods against today's usage. Especially, we are interested in the change from outdated methods such as MD5 or AES/ECB towards state-of-the-art methods such as SHA-256 or AES/CBC.

4.1 Hashing Functions

Hashing functions such as MD5[4] and its predecessors as well as SHA1(Wang et al., 2005) are long known to be insecure and prone to collision attacks. It is advised[5] to move to more secure alternatives like SHA224 or up to SHA512.

We analyze the used hashing functions in business apps and the top apps for iOS and Android to see the current situation. We compare the 2019 with the 2022 versions of the apps to visualize if a trend for cryptographic agility can be traced. The results are displayed in Figure 1.

To our surprise, outdated SHA1 and MD5 hashing is still found in 70% to 80% of the analyzed apps and thus the most used hashing algorithms in iOS and Android in both, the top and business app groups. Even the long outdated MD2 algorithm is still used in 5% to 10% of all apps. These are alarming news with respect to security. SHA256 is the only used, yet secure algorithm which is as widespread as MD5 and SHA1.

[3]https://www.bouncycastle.org/

[4]https://www.kb.cert.org/vuls/id/836068

[5]http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-131a.pdf

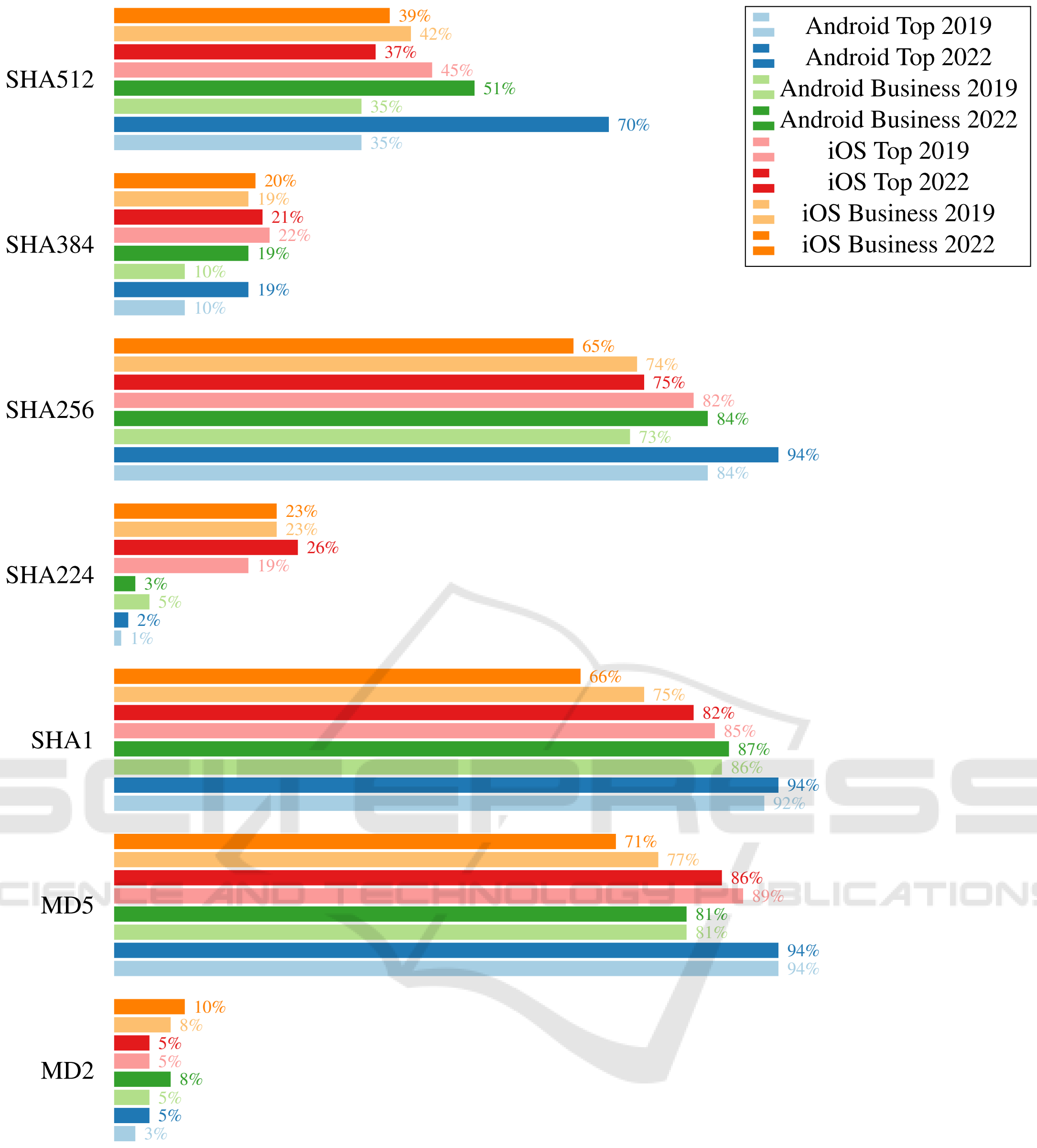


Figure 1: Percentage of apps using the specified **hashing** functions in top and business apps (2019 VS 2022).

If we now compare hashing in top and business apps, one can see that business apps use more up to date hashing algorithms. However, also fewer business apps use secure hashing functions. This leads to the conclusion that business apps use less hashing functionality in general.

If we now look at the evolution of the top apps on Android and iOS from 2019 to 2022, we can see that the usage of MD5 and SHA1 mostly remains constant with only slight variations. On Android, the SHA2 family and especially SHA512 usage increased. In the case of SHA512, the usage in apps doubled, which is at first sight a positive trend. However, since the usage of outdated algorithms remains constant, one must say, that only more hashing algorithms are used and secure algorithms are not replacing the outdated ones. On iOS, the situation is vice versa: The usage of the SHA2 family even declines which leads to the assumption that less hashing is used on iOS.

In conclusion, one can say that even though Android developers embracing the SHA2 family, outdated hashing functions constantly and heavily remains in Android and iOS apps.

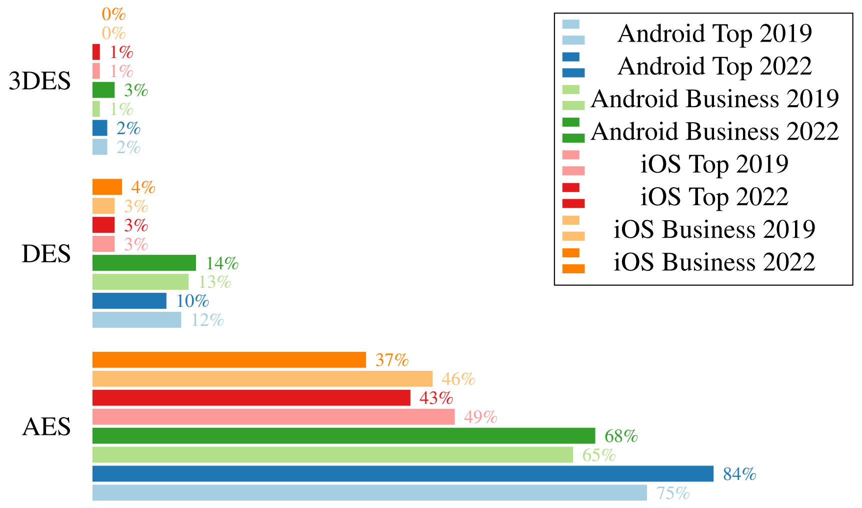


Figure 2: Percentage of apps using the specified **encryption** functions in top and business apps (2019 VS 2022).

4.2 Symmetric Encryption Algorithms

In this section we research the usage of symmetric encryption algorithms. Figure 2 shows which encryption algorithms are used in apps. As one would expect, AES is the most widely used symmetric encryption algorithm. DES and 3DES are mostly used in around 3% of the analyzed apps. One exception is DES in Android which still seems very popular with a usage in around 12% of the tested apps. Throughout the years, DES and 3DES usage remains mostly constant. However, looking at AES usage over time, one can see that the usage in Android increases in the latest app versions, while at the same time the AES encryption in iOS decreases. The decrease of cryptography usage in iOS matches with the observations in Section 4.1. Especially on Android, a discrepancy between business and top apps becomes obvious. Business apps seem to use less AES and slightly more DES encryption.

4.3 Deepdive: ECB Usage

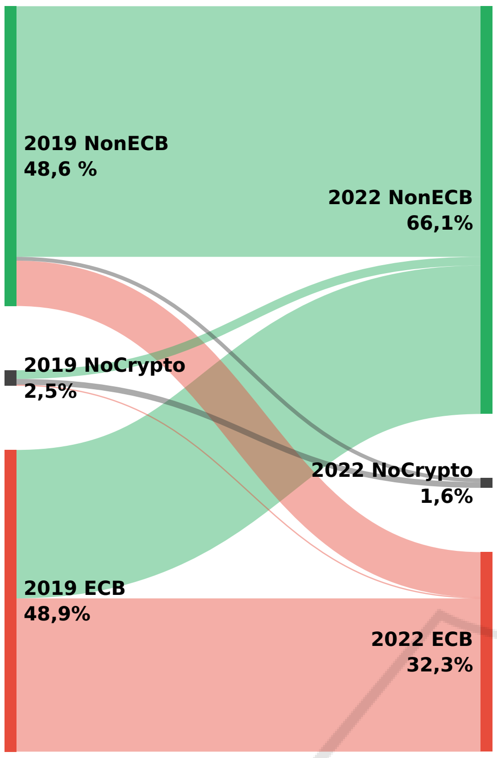
Usage of the ECB mode is a very common weakness when applying cryptography. ECB mode outputs the same ciphertext for the same plaintext (when the same key is used). This means that pattern are not hidden very well and one could draw conclusions on the plaintext. With other techniques like CBC or CTR mode, succeeding block's encryption depend on one another, which introduces randomness and hides pattern. We are aware that under certain conditions the usage of ECB mode is fine, but we advise against using it since secure conditions might easily become insecure during app upgrades, code restructuring and new requirements.

We visualize the usage of insecure ECB mode versus other modes in Figure 3. In the visualization we lay focus on the explicit transition of used secure and insecure modes from 2019 to 2022. Figures 3a to 3d represent the different app groups for Android, iOS and business or top apps.

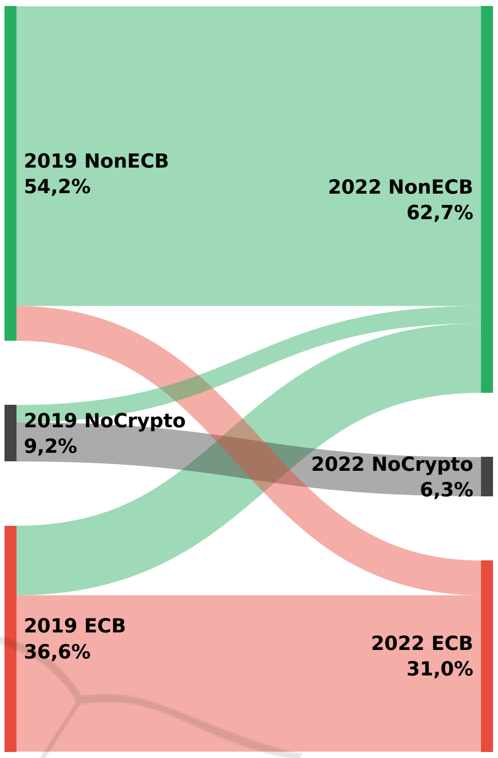
Looking at the transitions for Android top apps in Figure 3a, we see that 48.9% ECB mode usage in 2019 shrinks to 32.3% in 2022, which is very positive. One can also see that many apps shift from ECB mode to other secure modes. However, a small percentage of apps used secure cryptography in 2019, are now using ECB in 2022. Android business apps in Figure 3b used much less ECB in 2019 compared to top apps and the ECB usage didn't decrease as drastically as with the Android top apps. We also find a flow from 2019 ECB to 2022 non-ECB as well as from 2019 ECB to 2022 non-ECB. The flow from 2019 ECB to 2022 non-ECB is much smaller compared to the Android top apps group.

The situation on iOS looks much different. Figure 3c shows that out of the top apps on iOS, only 12% use ECB mode in 2019 and the majority uses secure alternatives. These numbers are similar for iOS business apps. However, after three years, things didn't turn out well for iOS apps. With 16% for top apps and 12% on business apps in 2022, more apps are using insecure ECB mode compared to 2019. Even though numbers increased on iOS, ECB usage on Android is still far more widespread, but decreases.

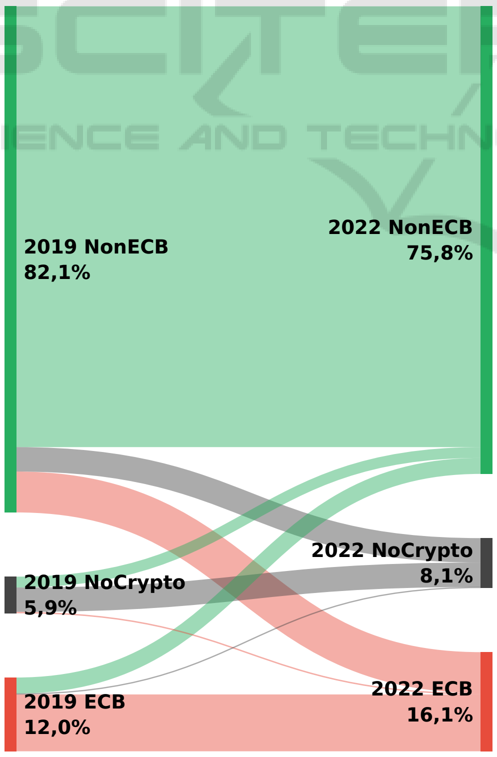
The total usage of cryptography is the highest for Android top apps, with only 2.5% of all apps using no cryptography. The other groups like Android business, iOS busines and iOS top apps contain 10% apps



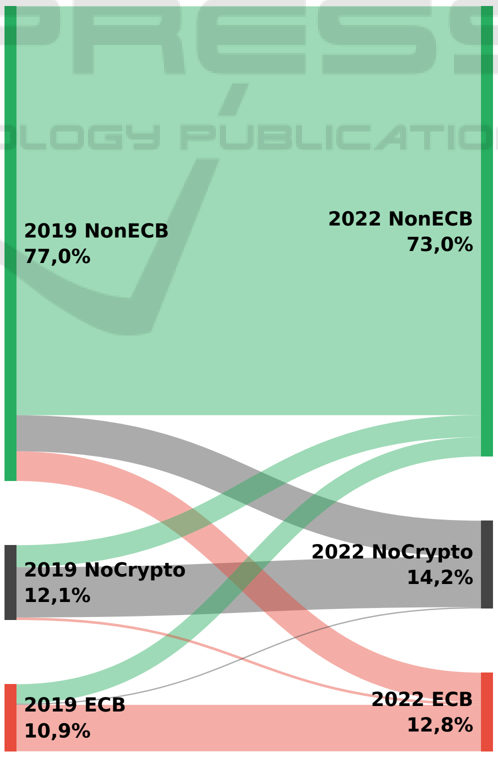
(a) Android top apps.



(b) Android business apps.



(c) iOS top apps.



(d) iOS business apps.

Figure 3: ECB mode usage transitions from 2019 to 2022.

without cryptography. Especially for iOS, we see a trend that many apps that used secure cryptography in 2019 don't use cryptography any more in 2022.

From all observations in Figure 3 we find the transitions from secure (non-ECB) to insecure (ECB) cryptography and vice versa very interesting. Understanding reasons for the transitions could give hints on how developers could be lead towards better cryptography standards. The transitions from ECB to non-ECB and non-ECB to ECB is significantly strong in Android top apps in Figure 3a, thus we analyze these apps deeper. For this reason, we manually decompile the respective Android apps. Then we analyze in which class or library cryptography with ECB was used and what changed in the respective class or library to not use ECB anymore.

To our surprise, it is often not directly the app developer who's to blame. In 95% of all cases, the transition from ECB to non-ECB is triggered by an included third party library on Android. However, if we now analyze the transition from non-ECB to ECB, we observe that with only 13% the transition is triggered through own app code and in 87% of the cases through a third party library. However, what is positive is that the total number of own code changes from ECB to non-ECB (40 occurrences) is much larger than vice versa (10 occurrences). Thus, app developers rather introduce secure cryptography than insecure cryptography configurations.

In the next step, we analyze ECB to non-ECB transitions in iOS top apps. Libraries are identified on iOS by finding duplicate method calls in the compiled executable among all apps. Those method calls consist of method name and class name by convention. This method is very primitive and by design only works for popular libraries and is thus rather an estimation. With 32%, own code changes and with 68% library changes are the transition cause towards secure cryptography (ECB $\rightarrow$ non-ECB). The transition from non-ECB towards ECB are with 30% and 70% in the same range.

Unfortunately, identifying libraries and library versions on iOS (swift/Objective-C) apps is much more complex than on Android (Java) and worth an own publication for itself. Also, we didn't find existing, good tools for this job. Thus, for the library analysis, we stick to Android apps, where a simple decompilation reveals the package names of the included libraries.

We looked at the different Android third party libraries which cause the transition from an insecure to a secure cryptography mode and vice versa. The results are shown in Table 1. The transition from ECB to non-ECB in Table 1b is pretty clear, 98% of the apps discontinued using ECB due to not using Google GMS Advertisement library anymore. In 2% of the apps, the respective library was not identifiable due to obfuscation. Table 1a shows which libraries triggered the ECB usage in 2022 apps. Leading with 29% is Google GMS Advertisement library followed by Icelink (12%), Microsoft Identity (10%) and Apache Commons (10%). We analyzed respective apps deeper in order to see if ECB was introduced through a third party library update or just by adding a new third party library with ECB usage. In fact, in 92% of the cases libraries with ECB usage were added and only in 8% of the cases a third party library update introduced ECB.

Table 1: Libraries triggering transition non-ECB/ECB on Android.

(a) 2019 non-ECB to 2022 ECB.

percentage	library
29%	Google GMS Ads
19%	unknown
12%	fm.icelink
10%	com.microsoft.identity
10%	org.apache.commons
7%	com.instabug
5%	com.google.crypto.tink
8%	various others

(b) 2019 ECB to 2022 non-ECB.

percentage	library
98%	Google GMS
2%	unknown

In conclusion one could say that in case of ECB usage, mostly libraries contain or introduce weak cryptography. This indicates how important the selection of suitable libraries can be for the development of secure software. However, own code changes outside common libraries are often more related to the key functionality of the app and in the category of own code we have definitely seen an improvement in secure cryptographic configurations.

5 CONCLUSION & FUTURE WORK

More than 4000 iOS and Android apps were analyzed for misconfigured cryptography from 2019 and 2022 in this publication to research aspects of their cryptographic agility. We have shown that the majority of apps still use insecure cryptography. The trend over the past years unfortunately shows no significant drift towards secure algorithms on the broad front. Some single aspects like ECB usage on Android point into the right direction. Our detailed analysis in finding causes of the ECB usage on Android brought us to the conclusion that this flaw is mostly introduced through the usage of third party libraries during app development. The app developers themselves are mostly not directly responsible for misconfigured cryptography. The trend shows that on custom code insecure cryptographic modes have been removed more often than they were newly introduced, which is a positive development.

As future work, we see two actions to further improve the situation. As a short term action, we'll try to reach out to developers of widely used libraries with flaws to update their libraries. But this does not tackle the problem on the long term. Developers currently have little means to get insights to the security and privacy aspects of their used third party libraries. Third party library sites like `mavencentral.com` only show CVEs for direct vulnerabilities but not lighter security or privacy issues. We aim to deliver such data for common libraries and provide such information to developers. App stores and development IDEs could process such information and highlight flaws to developers during app development. We believe that with such tight integration most flaws can easily be avoided.

ACKNOWLEDGEMENTS

This research work was supported by the National Research Center for Applied Cybersecurity ATHENE and the Hessian Ministry of the Interior and Sports.

REFERENCES

Acar, Y., Backes, M., Fahl, S., Garfinkel, S., Kim, D., Mazurek, M. L., and Stransky, C. (2017). Comparing the usability of cryptographic apis. In *IEEE Symposium on Security and Privacy (SP)*.

Braga, A. and Dahab, R. (2016a). Mining cryptography misuse in online forums. In *IEEE International Conference on Software Quality, Reliability and Security Companion*.

Braga, A. and Dahab, R. (2016b). Mining cryptography misuse in online forums. In *2016 IEEE International Conference on Software Quality, Reliability and Security Companion*.

Chatzikonstantinou, A., Ntantogian, C., Karopoulos, G., and Xenakis, C. (2016). Evaluation of cryptography usage in android applications. In *Proceedings of the 9th EAI International Conference on Bio-Inspired Information and Communications Technologies (Formerly BIONETICS)*. ICST.

Egele, M., Brumley, D., Fratantonio, Y., and Kruegel, C. (2013). An empirical study of cryptographic misuse in android applications. CCS. ACM.

Feichtner, J., Missmann, D., and Spreitzer, R. (2018). Automated binary analysis on ios: A case study on cryptographic misuse in ios applications. WiSec '18. ACM.

Fischer, F., Böttinger, K., Xiao, H., Stransky, C., Acar, Y., Backes, M., and Fahl, S. (2017). Stack overflow considered harmful? the impact of copy&paste on android application security. In *2017 IEEE Symposium on Security and Privacy (SP)*.

Gao, J., Kong, P., Li, L., Bissyandé, T. F., and Klein, J. (2019). Negative results on mining crypto-api usage rules in android apps. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories*.

Hazhirpasand, M., Ghafari, M., Krüger, S., Bodden, E., and Nierstrasz, O. (2019). The impact of developer experience in using java cryptography. In *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*.

Lam, P., Bodden, E., Lhoták, O., and Hendren, L. (2011). The soot framework for java program analysis: a retrospective.

Lazar, D., Chen, H., Wang, X., and 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. ACM.

Meng, N., Nagy, S., Yao, D. D., Zhuang, W., and Argoty, G. A. (2018). Secure coding practices in java: Challenges and vulnerabilities. ICSE '18. ACM.

Shoshitaishvili, Y., Wang, R., Hauser, C., Kruegel, C., and Vigna, G. (2015). Firmalice - automatic detection of authentication bypass vulnerabilities in binary firmware.

Shoshitaishvili, Y., Wang, R., Salls, C., Stephens, N., Polino, M., Dutcher, A., Grosen, J., Feng, S., Hauser, C., Kruegel, C., and Vigna, G. (2016). Sok: (state of) the art of war: Offensive techniques in binary analysis.

Stephens, N., Grosen, J., Salls, C., Dutcher, A., Wang, R., Corbetta, J., Shoshitaishvili, Y., Kruegel, C., and Vigna, G. (2016). Driller: Augmenting fuzzing through selective symbolic execution.

Vallée-Rai, R., Co, P., Gagnon, E., Hendren, L., Lam, P., and Sundaresan, V. (1999). Soot - a java bytecode optimization framework. CASCON '99, page 13. IBM Press.

Wang, X., Yin, Y., and Yu, H. (2005). Finding collisions in the full sha-1.

Wickert, A.-K., Baumgärtner, L., Breitfelder, F., and Mezini, M. (2021). *Python Crypto Misuses in the Wild*. ACM.