A Longitudinal and Retrospective Study on How Developers Misuse Cryptography in Online Communities

Alexandre Braga[1,2], **Ricardo Dahab**[2]

[1] Fundação CPqD Centro de Pesquisa e Desenvolvimento em Telecomunicações
R. Dr. Ricardo Benetton Martins, 1.000, Parque II do Polo de Alta Tecnologia
Campinas, SP, Brazil, Zip Code 13086-510

[2]Institute of Computing, State University of Campinas
Av. Albert Einstein, 1251, Cidade Universitária Zeferino Vaz
Campinas, SP, Brazil, Zip Code 13083-852

ambraga@cpqd.com.br,rdahab@ic.unicamp.br

Abstract. *Software developers participating in online communities benefit from quick solutions to technology specific issues and, eventually, get better in troubleshooting technology malfunctioning. In this work, we investigate whether developers who are part of online communities for cryptography programming are getting better in using cryptography with time. This is a crucial issue nowadays, when "real-world crypto" is becoming a topic of serious investigation, not only academically but in security management as a whole: cryptographic programming handled by non-specialists is an important and often invisible source of vulnerabilities [RWC]. We performed a retrospective and longitudinal study, tracking developers' answers about cryptography programming in two online communities. We found that cryptography misuse is not only common in online communities, but also recurrent in developer's discussions, suggesting that developers can learn how to use crypto APIs without actually learning cryptography. In fact, we could not identify significant improvements in cryptography learning in many daily tasks such as avoiding obsolete cryptography. We conclude that the most active users of online communities for cryptography APIs are not learning the tricky details of applied cryptography, a quite worrisome state of affairs.*

Resumo. *Desenvolvedores de software, participantes de comunidades on-line, costumam se beneficiar de soluções rápidas para problemas tecnológicos e, eventualmente, melhoram suas habilidades na resolução de problemas de mau funcionamento da tecnologia. Neste trabalho, investigamos se desenvolvedores de software criptográfico que participam de comunidades on-line se tornam melhores no uso de criptografia com o tempo. Esse é um aspecto de segurança crucial nos dias de hoje, em que "real-world crypto" se tornou um tópico de interesse sério, não só academicamente, mas no gerenciamento de segurança como um todo: programação criptográfica feita por não-especialistas é uma fonte frequente e muitas vezes invisivel de vulnerabilidades [RWC]. Realizamos um estudo retrospectivo e longitudinal para rastrear as respostas dos desenvolvedores sobre programação de criptografia em duas comunidades on-line. Descobrimos que o uso indevido da criptografia é não apenas comum em comunidades on-line, mas também é recorrente nas discussões, sugerindo que os desenvolvedores aprendem a usar as APIs criptográficas sem realmente aprender criptografia. Não conseguimos identificar melhora alguma na percepção e aprendizado de vulnerabilidades criptográficas, mesmo em tarefas simples como a de evitar o uso de criptografia obsoleta. Concluimos, assim, que os usuários ativos nas comunidades on-line para APIs criptográficas não tem evoluído no seu aprendizado dos detalhes e armadilhas do uso da criptograpfia, um estado de coisas muito preocupante.*

1. Introduction

Software developers are regular users of security mechanisms (e.g., security APIs, protocols, and tools), but are, by no means, security experts. They, however, do make security decisions that have a huge impact on end-user and system security. Developers are also frequent users of online communities for programming. The agility in problem solving provided by many question-and-answer communities brings benefits to ordinary programmers lacking knowledge in specific topics, such as secure coding. In this work, we investigate whether developers participating in online communities for cryptography programming are getting better in using cryptography with time. Also, we investigate whether cryptography misuse is persistent in posts of specific developers.

Cryptography misuse is a programming bad practice frequently found in misuse cases of cryptographic software, ultimately leading to vulnerabilities, but also associated to design flaws and insecure architectural choices [Braga and Dahab 2016]. Improvement in cryptography knowledge can be evidenced by a time series showing a steady decrease in the number of cryptography misuses. Also, persistence of a specific cryptography misuse can be illustrated by the repetition by developers of the same misuse over and over.

We tracked users (i.e. developers) of two online communities for programming, along with their questions and answers, from posts in a data set provided by a previous study [Braga and Dahab 2016]. We recorded the occurrence of known cryptography misuses for selected developers over a period of five years. We then computed statistics of cryptography misuse associated to specific moments in time for each tracked developer.

We found that the use of weak cryptography (e.g., broken algorithms or misconfigured implementations of standards) is not only common in online communities, but also recurrent in developers' discussions, suggesting that they learn how to use crypto APIs without actually learning cryptography. We also found that the lack of knowledge in cryptography is a recurrent source of coding bugs in API usage and does not depend on how long developers use cryptography APIs. We observed that platform issues dominate design concerns and obfuscate many complex cryptography misuses, which go unnoticed by developers in long lifespans. In summary, we conclude that users of online communities are not actually learning cryptography, despite their immediate gains in solving current programming issues related to cryptographic APIs.

This study is longitudinal, i.e., it performed repeated observations of the same developers over a period of time. Also, it is retrospective because it looks back in time using existing data. As far as we know, this is the first such study of cryptography misuse in online communities. The main contributions of this work are the following: *(i)* a method for clustering developers' posts from their asynchronous lifespans in online communities; *(ii)* a longitudinal study of selected developers of two communities, showing similarities and differences of these communities concerning how developers misuse cryptography; *(iii)* evidence that cryptography misuse is persistent across communities and developer lifespans; and *(iv)* evidence that developers learn how to use cryptography APIs without learning cryptography.

This text is organized as follows. Section 2 analyses related work and Section 3 details our research method. Section 4 explains our results and findings, while Section 5 details two users' lifespans. Section 6 discusses our findings and Section 7 shows our conclusions.

2. Related Work

[Fahl et al. 2012] investigated the SSL/TLS protocol usage in Android apps from Google Play and discovered security threats posed by misuses of that protocol. [Egele et al. 2013] were among the first to perform large-scale experiments in Google Play App Store to measure cryptographic

misuse in Android with the standard Crypto API. Their main contribution was a broad view of the prevalence of misused cryptographic functionality in Android apps. These two pioneering works were followed by others on related topics (e.g., [Lazar et al. 2014, Shuai et al. 2014, Georgiev et al. 2012, Chatzikonstantinou et al. 2015]).

[Wang and Godfrey 2013] were among the first to analyze API-related posts from a questions-and-answers (Q&A) website for mobile app development. They discovered repetitive scenarios with obstacles in API usage to developers, not specifically related to security. In a recent work [Wang et al. 2015], they investigated methods and proposed a methodology to distill and rank Q&A posts with API-related issues that would be valuable to API designers.

[Nadi et al. 2016] performed an empirical investigation into the obstacles developers face while using the Java cryptography APIs and the programming tasks they perform (e.g., authenticate users, store login data, establish secure connections, and encrypt data). By triangulating data from Stack Overflow posts, GitHub repositories, and developers' surveys, they found that developers find it difficult to use cryptographic algorithms correctly, despite being confident with cryptography concepts. They also found that cryptographic APIs are generally perceived as too low-level and not task-oriented.

[Acar et al. 2016b, Acar et al. 2016a] systematically analyzed the impact to code security of information resources commonly used by developers. They surveyed app developers who have published in the Google Play market, conducted a lab study with Android developers, analyzed 139 Stack Overflow threads accessed by developers during the lab study, and statically analyzed a random sample of Google Play apps. They concluded that real-world developers use Q&A communities as a major resource for solving programming problems, including security problems, suggesting that those online communities help developers to arrive at functional solutions more quickly than other resources. However, because online communities contain many insecure answers, developers who rely on this resource are likely to create less secure code. Also, access to quick solutions via a Q&A community may also inhibit developers' security thinking or reduce their focus on security.

[Braga and Dahab 2016] performed a transversal study to analyze how developers misuse cryptography in two online communities: Oracle Java Cryptography (OJC) and Google Android Developers (GAD). That work showed not only the most frequent cryptography misuses (e.g., weak cryptography, coding bugs, etc.), but also relationships among misuses through strong associations of double or triple misuses that appear together with non-negligible probabilities.

Most of the above-mentioned studies focus on the same online community or app store, with little variation. Also, none of these works study the behavior of frequent users over time, in order to examine whether developers are getting better in using cryptography with time.

3. Methodology

We analyzed data collected by a previous study [Braga and Dahab 2016] and observed that repeated measures were made for some developers. This fact motivated us to perform a retrospective, longitudinal study to analyze developers' behavior from a series of observations already made about them. This study is longitudinal because it performed repeated observations of the same developers (and their posts) over a period of time. Also, it is retrospective because it looks back in time using existing data.

Roughly speaking, our method segments the set of posts (collected for specific developers with determined lifespans) into a predefined number of clusters. When ordered chronologically, these clusters determine the phases a developer is supposed to pass for learning cryptography. These phases are then analyzed for the occurrence of cryptography misuses which are well-known in secure software development and secure coding.

The following subsections detail our methodology in four topics: classification of cryptography misuse, selection of communities and posts, selection of developers to evaluate, and method for clustering posts from a developer's lifespan.

3.1. Classification of Cryptography Misuse

The original study [Braga and Dahab 2016] introduced a classification of cryptography misuses in order to capture how software developers actually misuse cryptography in practice. The classification has nine categories: Weak Cryptography (WC), Bad Randomness (BR), Coding and Implementation Bugs (CIB), Program Design Flaws (PDF), Improper Certificate Validation (ICV), Public-Key Cryptography (PKC) issues, Poor Key Management (PKM), Cryptography Architecture Issues (CAI), and IV/Nonce Management (IVM) issues. Table 1 details categories in descriptive subsets.

The classification collected cryptography misuse from various sources, including software security books (e.g., [Viega and McGraw 2001, Howard and LeBlanc 2003, Chess and West 2007, Howard et al. 2009, Howard and Lipner 2006, Shostack 2014]), studies on cryptography misuse (e.g, [Lazar et al. 2014, Chatzikonstantinou et al. 2015, Egele et al. 2013, Shuai et al. 2014, Braga and Dahab 2015, Georgiev et al. 2012, Fahl et al. 2012]), newly discovered misuses (e.g., [Alashwali 2013, Bos et al. 2014, Mart and Hern 2013, Adrian et al. 2015]), and industry initiatives for software security (e.g., [Safecode 2011, OWASP , CYBSI 2014]). Table 1 shows the grouping of misuse categories, misuse main categories, and subsets.

Cryptography misuses are not all equally difficult to avoid [Braga and Dahab 2016]: some are easier to find and correct than others, depending on the involved complexity to identify and fix misuses. There are three complexity groupings for the nine misuse categories (in Table 1):

1. **Low complexity** misuses are related to coding activities and issues in APIs, and could be easily found by simple code reviews and skilled developers (supported by tools). This group includes Weak Crypto (WC), Coding Bugs (CIB), and Bad Randomness (BR).
2. **Medium complexity** misuses are related to flaws in program design affecting a few programs and may be difficult to identify due to feature distribution across programs. This group includes Improper Certificate Validation (ICV) issues, Program Design Flaws (PDF), and Public-Key Crypto (PKC) issues.
3. **High complexity** is related to flaws in system design and architecture, and requires understanding of system architecture to analyze underlying cryptosystems. This group includes Poor Key Management (PKM), IV and Nonce Management (IVM) issues, and Crypto Architecture Issues (CAI).

3.2. Selection of Communities and Posts

The original study [Braga and Dahab 2016] selected two programming communities possibly supported by experts in applied cryptography: Oracle Java Cryptography (OJC) [OJC], a forum aimed at programming with Java Cryptographic Architecture (JCA), and Google Android Developers (GAD) [GAD], a forum for Android programming.

The reasons to choose these two communities follows. Both OJC and GAD share the same Java-based API for the Java Cryptographic Architecture (JCA) [Oracle], thus limiting the knowledge required by a code reviewer to four aspects: Java programming, JCA, Android security, and applied cryptography. Also, JCA offers a stable and generic API, which has been used **for a long time** by a large number of developers for both server-side applications and mobile devices. Furthermore, JCA was adopted by the Android platform as its main API for cryptographic services. These two communities together reach a large number of ordinary developers, most

Table 1. Classification of cryptography misuse from a developer's viewpoint.

Low complexity		Medium complexity		High complexity	
Cat.	**Misuse subtype**	**Cat.**	**Misuse subtype**	**Cat.**	**Misuse subtype**
WC	-Risky/broken crypto -Proprietary crypto -Determin. symm. enc. -Risky/broken hash -Risky/broken MAC -Custom implement.	PDF	-Insec. default behavior -Insecure key handling -Streamcipher:insec. use -Insecure combo encr./auth -Insecure combo encr./hash -Side-channel attacks	IVM	-CBC w/ non-random IV -CTR with static counter -Hard-coded or const. IV -Reused nonce in encrypt.
CIB	-Wrong configs for PBE -Common coding errors -Buggy IV generation -Null cryptography -Leak/Print of keys	ICV	-No validation of certs -Broken SSL/TLS channel -Incomplete cert. valid. -Improper valid. host/user -Wildcards certs -Self-signed certs	PKM	-Short/improper key size -Hard-coded/const. keys -Hard-coded PBE passw. -Streamcipher:reused key -Use of expired keys -Key distrib. issues
BR	-Use of Statistic PRNG -Predictible seeds -Low-entropy seeds -Static, fixed seeds -Reused seeds	PKC	-Determ. encryp. RSA -Insec. padding RSA enc. -Weak configs RSA enc. -Insec. padding RSA sign. -Weak RSA sign. -Weak ECDSA sign -Key agr.: DH/ECDH -ECC: insecure curves	CAI	-Crypto agility issues -API misunderstanding -Multiple access points -Randomness issues -PKI and CA issues

of them are supposed to be non experts in cryptography. These assumptions may not hold for specialized communities with other APIs, such as openssl [OpenSSL] or bouncy castle [BC].

Collected posts comprised a time period of five years, from January 2011 to December 2015. Posts were listed by date (newest first) and manually saved as PDF files.

OJC was the most active community, with the most posts in the selected time period. GAD is very active in general, but showed less activity for cryptographic matters. For OJC, 310 posts were collected, and the 155 most viewed were selected for further analysis (50% of total). In GAD, a pre-analysis showed that specific keywords, such as "encryption", "hash" and "sign", were covered by the more general keywords "cryptography" and "encryption", which were used to select posts. For GAD, 170 posts were collected and the 100 most viewed were selected for analysis.

The manual inspection with code review was the method to analyze each single post. Posts were inspected by a cryptography expert with the skills mentioned above. Each post was inspected for occurrences of misuse. Many posts were discarded for not being related to cryptography programming, showing only discussions about threats or attacks. After discarding, OJC data set was reduced to 140 posts and GAD achieved 71 posts.

A few topics related to environment and platform specific issues were identified: configuration and installation issues, key storage and recovery issues, bug found or reported, tool misuse or misunderstanding, interoperation issues (e.g., platforms, versions, etc.), and hardware integration issues.

3.3. Selection of Developers to Evaluate

We found that most developers just ask one question to the community and never return. On the other hand, a few developers answered most questions. This fact made it possible to track users' answers and determine whether they had learned cryptography with time. These developers not only failed in giving good answers to questions related to cryptography; sometimes, they also omitted information that could prevent cryptography misuse.

In OJC, we counted 43 distinct developers who answered at least one question. Only 8 of these

developers answered 3 or more questions, corresponding to 74.5% of all answers. One developer asked 11 questions, totalling 9 evaluated developers in OJC. In GAD, we counted 97 distinct developers who answered at least one question. Only 14 of them answered 4 or more questions, corresponding to 52% of all answers. One developer asked 6 questions, totalling 15 evaluated developers in GAD. In summary, this study was conducted for two cohorts: one for a cryptography-specific forum (OJC) with 11 subjects and the other for a general-purpose forum (GAD) with 15 subjects.

The average lifespan for OJC developers was about 22 months with a standard deviation of around 9 months. The average lifespan for GAD developers was about 20 months with a standard deviation around 11 months. Table 2 shows lifespans and number of posts for both OJC and GAD. OJC developers are identified by J# and GAD developers are identified by G#.

Table 2. Selected developers, their lifespans (in months) and number of posts.

OJC Developers			GAD Developers					
OJC#	**Lifespan**	**# of posts**	**GAD#**	**Lifespan**	**# of posts**	**GAD#**	**Lifespan**	**# of posts**
J#1	8.4	7	**G#1**	14.4	6	**G#10**	8.1	6
J#2	30.5	14	**G#2**	22.5	6	**G#11**	10.9	4
J#3	26.4	33	**G#3**	25.1	6	**G#12**	17.2	4
J#4	19.9	6	**G#4**	17.1	5	**G#13**	7.3	4
J#5	27.1	13	**G#5**	36.0	8	**G#14**	7.1	5
J#6	29.6	36	**G#6**	22.3	11	**G#15**	11.9	5
J#7	31.1	11	**G#7**	28.8	13	**-**	-	-
J#8	8.3	5	**G#8**	45.4	29	**-**	-	-
J#9	16.1	3	**G#9**	23.9	11	**-**	-	-

3.4. Method for Clustering Developer's Lifespan

In online communities, interaction among users has a chronological order, but does not have to follow simultaneous events for synchronization of activities. For instance, a developer can show a very active participation for a few months and never return, while another can have a consistent participation for a couple of years.

We noticed that users participate in communities within different lifespans, diverse in length (duration) and number of posts. A user's lifespan is counted from the first to the last participation found in the period of study. In order to capture developers' distinct lifespans within the studied time period, we adopted a simple clustering technique to split the activity in each developer's lifespan (e.g., all posts for a user) into a defined number of clusters, as described next.

Clustering is a technique for combining observed objects into groups, segments, or clusters [Murthy 2015]. Its goal is to partition the observations into groups ("clusters") so that the differences among elements assigned to the same cluster tend to be smaller than among elements in different clusters [Friedman et al. 2009].

Clustering results need to be tied to specific semantic interpretations and applications [Murthy 2015]. Therefore, it is important to utilize expert knowledge to identify clusters [Murthy 2015]. We observed a natural fit between clustering methods and the life cycle presented next.

We devised a method to normalize lifespans and compare them. Our method consists in associating a life cycle to a lifespan. A life cycle is a qualitative sequence of phases. Lifespans use absolute time scales and are quantitative, while life cycles are relative and subjective, being qualitative in nature. A life cycle is divided into five phases according to the progress the user in evaluation is supposed to have had in his lifespan. The appropriate number of phases was apparent from prior knowledge about the data set. The five phases are the following:

1. an **entrant** or newbie is a new member or an inexperienced newcomer;
2. a beginner or **novice** is starting to learn crypto and taking part in communities;
3. an accustomed **fellow** is a regular user involved in the same activities of others;
4. an **expert** has knowledge or skills in cryptography gained over a period of time;
5. a **veteran** has long experience in cryptography and is considered knowledgeable.

This is an optimistic life cycle because it supposes developers improve their skills in cryptography with time. In fact, this is a strong assumption that may not hold for most developers.

The general idea of our clustering method is to apply a divisive clustering technique in order to distribute posts through five phases according to that user lifespan. The segmentation starts with all posts in one cluster (the whole lifespan) and iteratively splits existing clusters into two smaller clusters, until it satisfies a termination condition, e.g. the desired number of clusters. Roughly speaking, the lifespan is the bigger cluster, which we divide in five slices (life-cycle phases) in three iterations of the clustering method.

The distribution of posts through phases (i.e. cluster assignments, e.g. the assignment of posts to clusters) were refined by repeatedly attempting subdivision, and keeping the best resulting splits. In our method, a short lifespan lasts less than one year and a long lifespan lasts more than one year. We managed to distribute posts uniformly whenever it is possible, according to the following rules:

1. when there were less than five posts in the lifespan, we managed to put two at the ends and, optionally, two (or one) near the middle;
2. posts within the same month (a small time lapse) were put in the same phase (cluster);
3. when there was a large time lapse (bigger than 1 or 2 months in short lifespans or bigger than 4 to 6 months in long lifespans) between two consecutive posts, we adopted this gap as a phase separator (split point).

In general, divisive clustering methods, as the one we adopted, encounter difficulties regarding the selection of split points, leading to low-quality clusters [Murthy 2015]. We achieved a balanced distribution in general and a similar amount of posts into clusters, for both communities, with slightly more posts found at middle phases (Novice, Fellow, and Expert), and fewer posts found in first and last phases (Entrant and Veteran). Figure 1 shows the distribution of posts throughout the five phases for OJC and GAD.

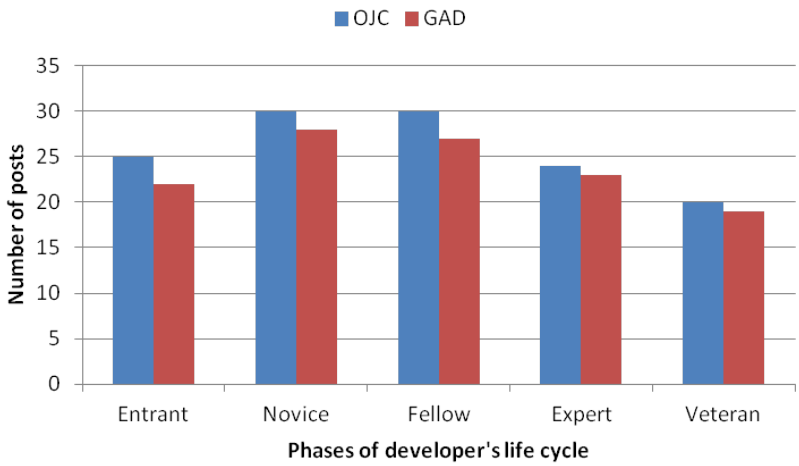

Figure 1. Distribution of posts through life cycle phases.

4. Results and Findings

This section analyses results for both communities in two distinct measures: misuse count and misuse density. Also, misuse density is analyzed in the context of the complexity groupings in Table 1. Before, we show general statistics about cryptography misuse in Table 3, that uses data from [Braga and Dahab 2016].

Table 3. Crypto misuse in online communities, from [Braga and Dahab 2016].

Categories of Cryptography Misuse	Communities	
	OJC	GAD
Weak Cryptography (WC)	26%	21%
Coding and Implementation Bugs (CIB)	17%	17%
Bad Randomness (BR)	0%	1%
Program Design Flaws (PDF)	6%	8%
Improper Certificate Validation (ICV)	4%	3%
Public-Key Cryptography (PKC) issues	16%	10%
Poor Key Management (PKM)	11%	4%
IV/Nonce Management (IVM) issues	5%	6%
Crypto Architecture Issues (CAI)	20%	1%
Platform Specific Issues (PSI)	60%	32%

The occurrence of cryptography misuse for each community is summarized in Table 3, which shows that weak cryptography (WC) is the most common misuse in both OJC and GAD. Also, both communities suffer negative influence from platform-specific issues (PSI). Besides specific issues, OJC suffers the most influence from weak cryptography (WC, 26%), architectural issues (CAI, 20%), coding bugs (CIB, 17%), public-key issues (PKC, 16%), and poor key management (PKM, 11%). These numbers are due to API misuse, lack of knowledge in applied cryptography, and complexity of JCA.

GAD suffers most from weak cryptography (WC, 21%), coding bugs (CIB, 17%), and public-key issues (PKC, 10%). These numbers are due to API misuse and lack of knowledge in cryptography programming. Despite preserving the Java API, Android has its own architecture for enabling cryptographic libraries, which simplifies installation and configuration, but brings new interoperational issues. Also, misuse of Pseudo-Random Number Generators (PRNGs) was barely mentioned in both communities, suggesting that developers have no doubts about simple uses of Java's SecureRandom API.

4.1. Cryptography Misuse Count per Life Cycle Phase

Figure 2 shows misuse counts for OJC (left) and GAD (right), for all misuse categories, distributed along the five phases of developer's life cycle. These communities have distinct behavior.

In OJC (left), developers start with a shy participation with relatively few misuses in Entrant phase. It is possible to observe that Novice is the phase with most misuses, with a notable presence of categories CAI, PKC, and WC. Also, OJC developers seem to be improving their skills in cryptography, because the total count of misuses gradually decreases from Novice to Veteran phase. However, the numbers for simple misuses (e.g., WC and CIB) are relatively stable (not decreasing), suggesting that simple misuses are recurrent and developers are not getting better at them. PKM and PKC are common issues in early phases (Novice and Fellow), with higher values, suggesting that developers are improving in these categories. Also, the number of CAI issues decreases from Novice to Veteran, suggesting that knowledge about Java's crypto API increases with time.

In GAD (right side), developers start in the Entrant phase with an expressive participation, having most misuses in WC and CIB. The notable decrease in crypto misuse for Novice and

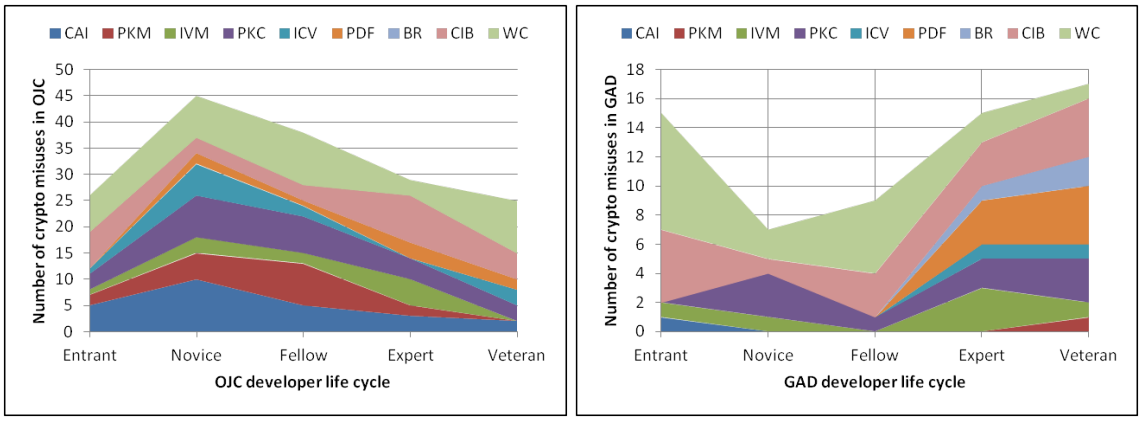

Figure 2. Crypto misuse counts for life cycle phases: OJC (left) and GAD (right).

Fellow phases suggests developers have a fast learning curve for those misuses less influenced by platform specific issues. However, in the Expert and Veteran phases the number of misuses increases again due to the influence of platform issues associated to sophisticate and complex misuses (e.g., PDF, PKC, ICV, and IVM). Four misuse categories (WC, CIB, PKC, and IVM) are recurrent in all phases, suggesting Android's diversity of hardware and software makes it difficult to learn cryptography not only for simple misuses, but also for complex misuses.

4.2. Cryptography Misuse Density per Life Cycle Phase

In order to measure crypto misuse density, we adapted the traditional metric for issue density per unit of size, which is also named defect density [Pandian 2003], bug density [Hutcheson 2003], and fault density [Bourque and Fairley 2014]. In order to measure how developers misuse cryptography, we counted the number of misuses that have been detected in posts for a developer and normalized this measure by the total number of posts for the developer in question, obtaining a value for misuse density. A straightforward analysis on misuse density over time was used to evidence a learning curve for developers as well as to show misuse reduction (or growing) as a trend.

Figure 3 shows the misuse density for OJC throughout the life cycle in two charts. On the left, crypto misuse per post (c.m.p.p) is compared to platform issues per post (p.i.p.p). In this chart, it is possible to observe that misuse density is relatively stable over time, despite a gradual reduction in density for platform issues. The chart on the right shows misuse density for low-, medium- and high-complexity misuses. In this chart, it is possible to observe that density of simple misuses (WC, CIB, and BR) increases over time, while density of moderate (PDF, PKC, and ICV) misuses has a small decrease, and density for high-complexity misuses (CAI, PKM, and IVM) shows a gradual decrease.

This behavior suggests that simple misuses are recurrent in OJC and do not depend on the actual knowledge of Java's crypto API. On the other hand, medium-complexity misuses are the most influenced by platform issues, closely following p.i.p.p behavior. Then, complex misuses (related to system design and architecture) decreases over time, suggesting a gradual improvement in developer's knowledge about Java's crypto API.

Similarly, Figure 4 shows the misuse density for GAD throughout the life cycle in two charts. On the left, it is possible to observe that misuse density increases over time (the opposite behavior of OJC), despite a relatively stable density of platform issues. The chart on the right shows that density for complex misuses is relatively stable, while density for low- and medium-complexity misuses grows over time. In fact, GAD developers start with a high density for simple misuses and,

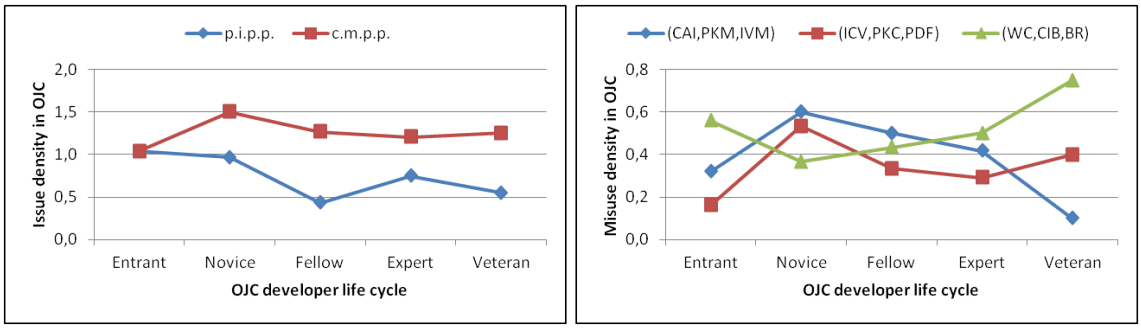

Figure 3. Misuse density in OJC compared to issue density and complexity groups.

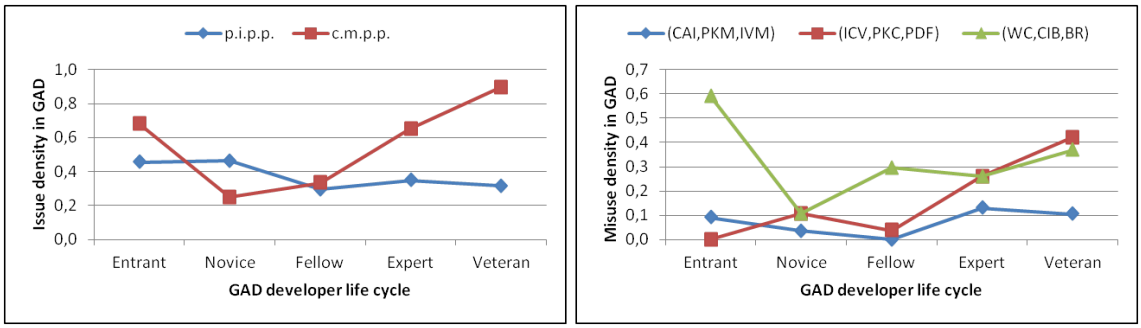

Figure 4. Misuse density in GAD compared to issue density and complexity groups.

after a sharp drop, this density gradually grows. This behavior suggests that Android developers are not getting better in cryptography over time.

5. Two Life Cycles in Detail

This section details the life cycles of two developers. The objective is to illustrate with real cases the recurrence of crypto misuse in developers' lifespans. We selected two users with more posts and longer lifespans, being good representatives of their communities: J#6 from OJC with a lifespan of 29.6 months and 36 posts, and G#8 from GAD with a lifespan of 45.4 months and 29 posts. These developers not only failed to give good answers, but sometimes omitted information that could prevent crypto misuse. Also, we could not identify improved cryptography skills for these developers.

5.1. Life Cycle for a Java Developer (J#6)

In the entrant phase, J#6 contributed to posts related to incompatible crypto providers, buggy hardware modules, cross-platform verification of certificates, and cross-language (e.g., from C++ to Java) encryption. Misuses were associated to hard-coded keys and IVs, wrong ciphertext encoding, weak cryptography with custom implementations, short keys, misconfigured RSA, and deterministic encryption with RSA.

In the novice phase, J#6 experienced misuses associated to deterministic symmetric encryption (AES/ECB), broken cryptography (DES), proprietary cryptography with custom key agreement, and coding errors. Other misuses were associated to unsafe defaults, insecure padding for RSA, deterministic encryption or short keys for RSA, and issues for DH and ECC. Complex misuses were associated to insufficient length and key distribution issues, as well as IV misconfiguration and design flaws in cryptographic architecture.

In the fellow phase, J#6 contributed to posts related to fails in TLS authentication, RSA encryption, digital signature verification, and proprietary encryption. Misuses were associated

to buggy PKI software, proprietary cryptography, risky cryptography (3DES in ECB), coding errors in insecure key handling, insecure padding or inadequate key length for RSA.

In the expert phase, J#6 still talked about coding errors when using AES with password-based encryption, hard-coded IVs and deterministic encryption, misunderstanding digital certification, deterministic signatures with RSA, and insecure key derivation. Platform issues led to misunderstand of PKI functions. Other misuses were associated to flawed IV generation, broken hash function, unsafe default, insecure padding for RSA, non-random or constant IVs, and reuse of keys with stream ciphers.

In the veteran phase, J#6 could not give correct answers to posts related to MAC with broken hash (e.g., MD5), encoding of keys, issues in key generation, and misconfigured PKI software. Other misuses were associated to public-key issues (insecure padding for RSA, and misconfigured DH), unsafe defaults, improper certificate validation (non-validated hostname and self-signed certificates), coding errors disabling cryptography, and deterministic symmetric encryption.

5.2. Life Cycle for an Android Developer (G#8)

In the entrant phase, G#8 was involved in several discussions about password-based encryption, errors when decrypting data from strings, use of SHA1 to generate keys from passwords, use of AES in CBC mode to encrypt files, and Android's full encryption. Misuses were related to broken encryption and hashes, misconfigured PBE, insecure deterministic encryption, ciphertext encoding errors, custom implementation of PBE, and constant IVs.

In the novice phase, G#8 was involved in posts related to buggy implementations of ECC in SSLv2 and signature verification on crypto libraries, proprietary implementation of SSL, and use of RSA encryption. Several misuses were associated to insecure padding and deterministic encryption for RSA, custom implementation of SSL, and attempts to use buggy implementations of ECC.

In the fellow phase, G#8 discussed cryptography adopted by Google Drive, errors when using the wrap method for protection of keys with PBE in specific versions of Android, and cross-platform verification of signatures (Java and dotNET). Misuses involved misconfigured PBE with small parameters, use of risky hashes and broken encryption, errors in ciphertext encodings, and insecure padding and deterministic encryption for RSA.

In the expert phase, G#8 was involved in discussions about several errors related to bad padding in encryption with AES, parsing keys from certificates for RSA encryption, and backward incompatibility of encryption algorithms in Android. Misuses associated to improper certificate validation, insecure defaults, deterministic encryption with RSA, non-random or constant IVs, and misconfigured PBE.

In the veteran phase, G#8 was involved in posts related to cryptography issues in Android, such as storage and recovery of keys from the device's keystore, cross-version decryption of files, and encoding ciphertext as integers. Misuses were associated to improper certificate validation with self-signed root certificates, misconfigured PBE with small parameters, insecure defaults for AES, non-random IVs, and ciphertext encoding errors.

6. Discussion

We are aware that our analysis have to be put in context and is restricted to the main subject of the two communities evaluated. That said, we tried to generalize our conclusions.

For developers, as much as for end-users, security is a secondary concern. Developers usually have priorities (e.g., functional correctness, time to market, maintainability, economics, compliance

with other corporate policies) that often appear to conflict with security. Frequently, developers look for quick, but insecure solutions and online communities favor this behavior.

Ideally, developers should not be forced to learn cryptography in order to correctly use cryptographic APIs, specially for simple use cases. However, in practice, crypto APIs are unable to foster their correct use without domain knowledge obtained from elsewhere but online communities.

Java has a stable API, with a very predictable behavior, favoring developers with enough time to understand its particularities. In general, developers improve their skills in misuse categories affected by platform issues, but this does not happen for simple misuses. On the other hand, Android uses the same crypto API of the Java platform, but this fact alone is not enough to promote a positive learning curve for cryptography. Many issues related to diversity of both hardware and software negatively affect how developers learn cryptography in Android.

Java cryptographic architecture presents issues with installation and configuration that divert developers from actual tasks of cryptographic programming. Also, specific issues showed up when integrating Java programs to cryptographic hardware or communicating with applications in other platforms. These troublesome issues frequently obfuscate crypto misuses in the same code. For instance, in Java, a worst-case scenario occurs when developers write buggy code for encrypting data and use weak cryptography by accidentally adopting insecure defaults.

Android solved many issues faced by Java developers, but brought to daily troubleshooting several interoperation issues due to the diversity of both hardware and software in that platform. Developers, confused by these issues, are distracted from actual cryptographic pitfalls. For instance, developers suffering from interoperation issues among devices used weak cryptography to protect stored passwords, and derived keys directly from password hashes.

Developers learn how to make APIs work, but this does not mean cryptography was used correctly. In fact, coding bugs are persistent issues when using general-purpose (function-based) crypto APIs to implement application-specific use cases, because developers are forced to make insecure choices without actually understanding the whole situation. This suggests developers would benefit from high-level cryptographic frameworks (oriented toward use cases) or task-based APIs that could avoid simple misuses and insecure design decisions.

The overabundance of complex options for security leads to disengagement when confronted by other concerns. We have noticed that complex architectures distract developers from actual cryptographic misuse and contribute to perpetuate issues in cryptographic programming. For instance, one curious reason developers gave to use homemade code for cryptographic algorithms is to avoid dependencies to external libraries.

Finally, we did not see in developers' lifespans any posts concerned with the insecure combination of encryption and authentication, padding-oracle attacks, or selection of insecure elliptic curves. This suggests that these developers never learned about design flaws for authenticated encryption, side-channel attacks or obsolete implementations for elliptic curve cryptography. Another remarkable absence in lifespans was the concern with weak parameters in public-key cryptography (e.g., RSA, DH), suggesting that developers always take for granted the quality of parameters generated by tools.

7. Concluding Remarks

Ordinary software developers are used to obtain quick solutions to daily problems from fellows in online communities. Those communities associated to cryptographic programming are good for finding solutions to platform-specific issues (e.g., implementation bugs, incompatible hardware, and misconfigured software) as well as to clarify obscured aspects of API usage. On the other

hand, developers willing to learn cryptography from online communities, if lucky, receive only shallow advice and usually do not have a positive learning curve over time. Our study shows that cryptography misuse is perpetuated in online communities and frequently reappear in recurrent issues, because these communities favor quick, but insecure solutions and even active developers (of those communities) take security as a secondary concern.

We believe this longitudinal study effectively contributes to better understanding how cryptography is handled by ordinary developers of two online communities, bringing to light their attitudes and priorities concerning cryptography-based security over time. It is paramount to improve APIs to increase usability and to foster best practices. Also, there are opportunities for future work in behavioral experiments of cryptographic programming, surveys with actual developers, as well as replication studies focusing on other communities and crypto APIs.

Acknowledgements

We thank Intel and CNPq for the financial support, as well as CPqD and UNICAMP for the institutional support.

References

Acar, Y., Backes, M., Fahl, S., Kim, D., Mazurek, M. L., and Stransky, C. (2016a). You Get Where You're Looking For: The Impact Of Information Sources on Code Security. In *Security and Privacy (SP), 2016 IEEE Symposium on*, pages 289–305. IEEE.

Acar, Y., Fahl, S., and Mazurek, M. L. (2016b). You Are Not Your Developer, Either: A Research Agenda for Usable Security and Privacy Research Beyond End Users.

Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J. A., Heninger, N., Springall, D., Thomé, E., Valenta, L., and Others (2015). Imperfect forward secrecy: How Diffie-Hellman fails in practice. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 5–17. ACM.

Alashwali, E. S. (2013). Cryptographic vulnerabilities in real-life web servers. In *Third International Conference on Communications and Information Technology (ICCIT)*, pages 6–11. Ieee.

BC. The Legion of the Bouncy Castle.

Bos, J. W., Halderman, J. A., Heninger, N., Moore, J., Naehrig, M., and Wustrow, E. (2014). Elliptic curve cryptography in practice. In *Financial Cryptography and Data Security*, pages 157–175. Springer.

Bourque, P. and Fairley, R., editors (2014). *Guide to the Software Engineering Body of Knowledge (SWEBOK)*. IEEE Computer Society, version 3. edition.

Braga, A. and Dahab, R. (2015). Introdução à Criptografia para Programadores: Evitando Maus Usos da Criptografia em Sistemas de Software. In *Caderno de minicursos do XV Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais — SBSeg 2015*, pages 1–50.

Braga, A. and Dahab, R. (2016). Mining Cryptography Misuse in Online Forums. In *2nd International Workshop on Human and Social Aspect of Software Quality*.

Chatzikonstantinou, A., Ntantogian, C., Xenakis, C., and Karopoulos, G. (2015). Evaluation of Cryptography Usage in Android Applications. *9th EAI International Conference on Bio-inspired Information and Communications Technologies*.

Chess, B. and West, J. (2007). *Secure programming with static analysis*.

CYBSI (2014). Avoiding The Top 10 Software Security Design Flaws.

Egele, M., Brumley, D., Fratantonio, Y., and Kruegel, C. (2013). An empirical study of cryptographic misuse in android applications. *ACM SIGSAC conference on Computer & comm. security (CCS'13)*, pages 73–84.

Fahl, S., Harbach, M., and Muders, T. (2012). Why Eve and Mallory love Android: An analysis of Android SSL (in) security. In *ACM conference on Computer and communications security*, pages 50–61.

Friedman, J., Hastie, T., and Tibshirani, R. (2009). *The elements of statistical learning*, volume 2. Springer-Verlag.

GAD. Google Android Developers.

Georgiev, M., Iyengar, S., and Jana, S. (2012). The most dangerous code in the world: validating SSL certificates in non-browser software. In *Proceedings of the 2012 ACM conference on Computer and communications security - CCS '12*, pages 38–49.

Howard, M. and LeBlanc, D. (2003). *Writing secure code*.

Howard, M., LeBlanc, D., and Viega, J. (2009). *24 Deadly Sins of Software Security: Programming Flaws and How to Fix Them*. McGraw-Hill Education.

Howard, M. and Lipner, S. (2006). *The Security Development Lifecycle*. Microsoft Press, Redmond, WA, USA.

Hutcheson, M. L. (2003). *Software testing fundamentals: methods and metrics*. John Wiley & Sons.

Lazar, D., Chen, H., Wang, X., and Zeldovich, N. (2014). Why Does Cryptographic Software Fail?: A Case Study and Open Problems. In *5th Asia-Pacific Workshop on Systems*, APSys '14, pages 7:1—–7:7, New York, NY, USA. ACM.

Mart, V. G. and Hern, L. (2013). Implementing ECC with Java Standard Edition 7. *International Journal of Computer Science and Artificial Intelligence*, 3(4):134–142.

Murthy, M. R. (2015). *Introduction to Data Mining and Soft Computing Techniques*. Laxmi Publications.

Nadi, S., Krüger, S., Mezini, M., and Bodden, E. (2016). "Jumping Through Hoops": Why do Java Developers Struggle With Cryptography APIs? *The 38th International Conference on Software Engineering*.

OJC. Oracle Java Cryptography.

OpenSSL. OpenSSL Cryptography and SSL/TLS toolkit.

Oracle. Java Cryptography Architecture (JCA) Reference Guide.

OWASP. Cryptographic Storage Cheat Sheet.

Pandian, C. R. (2003). *Software metrics: A guide to planning, analysis, and application*. CRC Press.

RWC. Real World Crypto Symposium.

Safecode (2011). Fundamental Practices for Secure Software Development.

Shostack, A. (2014). *Threat modeling: Designing for security*. John Wiley & Sons.

Shuai, S., Guowei, D., Tao, G., Tianchang, Y., and Chenjie, S. (2014). Modelling Analysis and Auto-detection of Cryptographic Misuse in Android Applications. In *IEEE 12th International Conference on Dependable, Autonomic and Secure Computing (DASC)*, pages 75–80.

Viega, J. and McGraw, G. (2001). *Building Secure Software: How to Avoid Security Problems the Right Way*.

Wang, W. and Godfrey, M. W. (2013). Detecting API Usage Obstacles : A Study of iOS and Android Developer Questions. pages 61–64.

Wang, W., Malik, H., and Godfrey, M. W. (2015). Recommending posts concerning api issues in developer q&a sites. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 224–234. IEEE Press.

