Developers Deserve Security Warnings, Too: On the Effect of Integrated Security Advice on Cryptographic API Misuse

Peter Leo Gorski and Luigi Lo Iacono, ***Cologne University of Applied Sciences;***
Dominik Wermke and Christian Stransky, ***Leibniz University Hannover;***
Sebastian Möller, ***Technical University Berlin;*** **Yasemin Acar,** ***Leibniz University Hannover;***
Sascha Fahl, ***Ruhr-University Bochum***

https://www.usenix.org/conference/soups2018/presentation/gorski

This paper is included in the Proceedings of the Fourteenth Symposium on Usable Privacy and Security.

August 12–14, 2018 • Baltimore, MD, USA

ISBN 978-1-931971-45-4

Open access to the Proceedings of the Fourteenth Symposium on Usable Privacy and Security is sponsored by USENIX.

Developers Deserve Security Warnings, Too

On the Effect of Integrated Security Advice on Cryptographic API Misuse

Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke*,
Christian Stransky*, Sebastian Moeller†, Yasemin Acar*, Sascha Fahl**
Cologne University of Applied Sciences, *Leibniz University Hannover,
†Quality and Usability Lab, Technical University Berlin, **Ruhr-University Bochum

ABSTRACT

Cryptographic API misuse is responsible for a large number of software vulnerabilities. In many cases developers are overburdened by the complex set of programming choices and their security implications. Past studies have identified significant challenges when using cryptographic APIs that lack a certain set of usability features (e. g. easy-to-use documentation or meaningful warning and error messages) leading to an especially high likelihood of writing functionally correct but insecure code.

To support software developers in writing more secure code, this work investigates a novel approach aimed at these hard-to-use cryptographic APIs. In a controlled online experiment with 53 participants, we study the effectiveness of API-integrated security advice which informs about an API misuse and places secure programming hints as guidance close to the developer. This allows us to address insecure cryptographic choices including encryption algorithms, key sizes, modes of operation and hashing algorithms with helpful documentation in the guise of warnings. Whenever possible, the security advice proposes code changes to fix the responsible security issues. We find that our approach significantly improves code security. 73% of the participants who received the security advice fixed their insecure code.

We evaluate the opportunities and challenges of adopting API-integrated security advice and illustrate the potential to reduce the negative implications of cryptographic API misuse and help developers write more secure code.

1 Introduction

A large number of software vulnerabilities are caused by developers who misuse security APIs [12,19,36]. Previous work identified multiple trouble spots including secure network connections [15], the use of permissions in mobile apps [17] and the use of cryptographic APIs [1,32]. Some of the most serious data breaches in recent history were caused by not properly using TLS to secure data in transit or not securely storing data in rest [12,19]. Such incidents affect millions or even billions of users worldwide and jeopardize their security and privacy.

In this work we focus on the challenges of using cryptographic APIs securely. Using cryptographic APIs correctly in many cases requires detailed knowledge and overburdens non-security expert developers on a regular basis. Acar et al. conducted several studies and investigated the usability of cryptographic APIs and the impact of information resources developers use to solve programming questions on code security [1, 2]. They find that the design of cryptographic APIs and the quality of available developer documentation amongst other factors have a significant impact on code security. In particular, the availability of easy-to-understand documentation and ready-to-use and functional code snippets helped participants in their studies to produce more secure results. Motivated by their findings and results of measurement studies of real world software repositories [2], we design and implement a novel approach to help software developers write more secure cryptographic code.

While there is previous work that tries to improve code security by enhancing API simplicity [23] or by providing IDE plugins [29, 34], we propose a different and novel approach that allows providers of existing and future cryptographic APIs to improve code security. Therefore, they do not have to change their programming interfaces, rely on the development and integration of plugins for integrated development environments (IDEs) or hope that security of unsafe information sources such as Stack Overflow becomes better. Instead, we propose the integration of effective security advice directly into cryptographic APIs. We develop an API-integrated security advice concept that provides context sensitive help and offers ready-to-use and secure code snippets to fix security issues. We implement our approach for Python and the PyCrypto cryptographic API and conduct a between-subjects online study with 53 experienced Python developers. In the course of this study we try to answer the following research questions:

RQ1: *Does API-integrated security advice have a significant effect on code security?* With this research question we try to assess the ability of our approach to improve code security. We analyze all changes made to the code after security advice has been shown. We find that our approach had a significant positive impact on 73% of our participants who left their code insecure at the first place: They upgraded bad cryptographic choices to secure ones.

USENIX Symposium on Usable Privacy and Security (SOUPS) 2018.
August 12–14, 2018, Baltimore, MD, USA.

RQ2: *Does API-integrated security advice have a significant impact on perceived API usability?* We were interested in whether providing context sensitive security advice affects the perceived usability of the PyCrypto API. We find that while security significantly improved, the security advice had no statistical significant impact on the perceived usability.

RQ3: *How does our approach compare to other approaches?* Previous work by Acar et al. [1, 2] found that interfaces and supported use cases of cryptographic APIs and the types of information resources developers use have a significant impact on code security. Nguyen et al. [34] tested their Android studio plugin and found that their approach has a significant contribution to code security.

We find that similar to high quality developer documentation, good API design and helpful IDE plugins, our approach has a significantly positive effect on code security, but allows API providers to improve code security for existing cryptographic APIs in a low-level approach without having to change API design or relying on third party tools.

Our work makes the following contributions:

1. We design a security advice concept that is directly integrated into an API, drawing on guidelines, suggestions and research on human factors on security APIs and warning messages.
2. We implement our concept for the PyCrypto API.
3. We conduct a between-subjects online controlled experiment with experienced Python developers to test the effectiveness of our approach.
4. We assess the real world applicability, limitations and potential of API-integrated security advice, and conclude with lessons learned from our experiment.

2 Related Work

We discuss related work in two key areas: research on human factors on security APIs and tools for developers; research on security warning messages.

Research on Security APIs and Tools: Researchers have investigated challenges developers have when interacting with security APIs and tools.

Both Wurster and Oorschot [41] and Green and Smith [22] analyze the developers' roles in writing secure software and come to the conclusion that security is often only of secondary or tertiary concern for developers and that (security) APIs and libraries need to be designed with usability in mind. Lo Iacono and Gorski [30] present a classification of security APIs according to their abstraction level. They evaluate their approach by investigating a set of popular software development kits and conducting an online study with developers. They find that developers prefer APIs that provide comfortable abstractions and enable them to take full control as required by the specific programming task. Gorski and Lo Iacono propose a set of eleven characteristics to evaluate security API usability as they find that security API usability goes beyond general API usability [21].

Nadi et al. manually examined the top 100 Java cryptography posts on Stack Overflow and found that a majority of problems were related to API complexity rather than a lack of domain knowledge [32]. Relatedly, Acar et al. investigated how the use of different documentation resources affects developers' security decisions, including decisions about certificate validation. They report that good usability and the availability of ready-to-use and functional code snippets as part of documentation significantly impacts code security [2]. Barik et al. [6] conducted an eye-tracking study to investigate the use of Java compiler error messages finding that the difficulty of reading error messages is comparable to reading source code. Acar et al. conducted a controlled experiment online and compared the usability of different cryptographic libraries for Python [1]. They found that in addition to safe defaults, the number of supported use cases and the availability of good documentation have a significant impact on code security. Naiakshina et al. conducted a qualitative developer study and investigated how computer science students implemented secure password storage [33]. We develop and test a novel approach that supports developers using a security warning that presents context-sensitive documentation and code snippets as part of an API.

Nguyen et al. present a plugin for the Android Studio IDE called FixDroid which helps developers write more secure code by highlighting insecure code and providing quick fixes. In a user study they find that FixDroid users write significantly more secure code than participants without FixDroid [34]. Similarly, Krueger et al. present the CogniCrypt tool which is an Eclipse IDE plugin for the Java programming language that helps developers to securely use cryptographic APIs by auto-generating secure code for common tasks [29]. Xie et al. present and evaluate an Eclipse IDE called ASIDE that interactively reminds programmers of secure programming practices [42]. Johnson et al. conducted a user study and investigated why developers do not use static analysis tools to find bugs and report that too many false positives and complicated errors messages were significant hurdles for their participants [25]. We propose an IDE-agnostic approach that allows API and library providers to improve code security without having to rely on third parties such as IDE plugins or static code analysis tools.

To the best of our knowledge, in contrast to previous work our paper is the first to introduce and study security advice as part of an API.

Research on Security Warnings: Researchers have investigated challenges in designing usable security warnings. Due to a lack of related work for software developers we limit the following presentation to previous work for warnings for end-users.

Sunshine et al. conducted multiple studies to investigate the effectiveness of SSL warnings and found while they could improve warning message effectiveness still many participants clicked-through a warning. In addition to further improve warnings, they recommend to reduce their occurrence [38]. Felt et al. experimented with SSL warnings for Google Chrome and found that while they could not improve the rate of comprehension of the warnings' text significantly, opinionated design drastically improved the warnings' adherence rate [4, 16, 18].

Weinberger and Felt run a field study to investigate how long the Chrome browser should store users' decisions for SSL warnings to minimize the effect of habituation [40]. Sim-

ilarly, Vance et al. conduct an fMRI experiment to study warning message habituation [39]. Both studies conclude that the risk of habituation decreases after one week.

Almuhimedi et al. investigate factors that contribute to why Chrome users click-through their malware warnings and find that familiarity with a website had significant impact on users' click-through behavior [5]. Egelman et al. investigated the difference between passive and active warnings against phishing attacks and found that active warnings were more successful [13]. Bravo-Lillo et al. designed and tested multiple attractors for security warnings [8].

Bauer et al. present and discuss a set of design guidelines for warning messages [7]. Our approach follows their guidelines and includes lessons learned from other related work presented above.

In contrast to end-users, our work is the first to investigate a novel security warning concept targeted at software developers.

3 API Level Advantages

Making security advice part of the API has multiple advantages over other approaches:

Environment Agnostic: Integrating security advice into an API instead of providing extensions or plugins for integrated development environments (IDEs) or editors (e. g. [29, 34]) makes the security warnings agnostic to developers' programming environments. Instead of having to provide multiple extensions or plugins for different programming environments only one implementation for a particular API is needed. API integration is not just agnostic to the IDE or editor used but also to the way developers use programming language interpreters or compilers. Security warnings that are part of an API can provide helpful information in terminal as well as in IDE environments.

Immediate Feedback: Making security advice part of an API can provide context sensitive and secure information (e. g. secure code snippets or targeted information) as immediate feedback. Such an approach has the potential to prevent developers from falling back on insecure information resources online such as Stack Overflow [2].

A Bottom Up Approach: An integrated feedback mechanism gives API providers the power to provide very specific and context sensitive security advice and make it available through the regular distribution channels of an API. API users immediately benefit from feedback integration after using an updated API version. Instead of having to install or update external third party tools such as plugins or extensions, relying on the regular update channels of an API has the potential to speed up distribution of feedback mechanisms.

4 Security Advice Design

Below we discuss design decisions for our security warning.

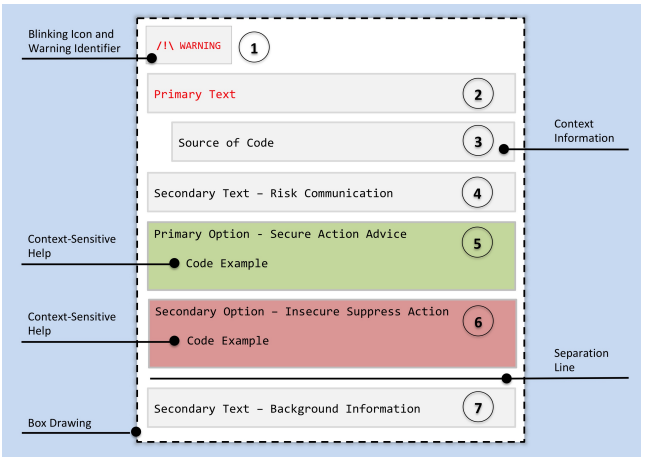

Figure 1: Design concept of our security feedback mechanism.

4.1 Design Decisions

The main goal of our approach is to help users of security APIs to avoid insecure programming choices whenever possible. However, due to the complex nature of security decisions and the fact that information security is not a top priority [30] for many developers, we expected this to be challenging.

There is no previous work on designing security warnings with a specific focus on making secure programming choices. Therefore, we based our work on previous research and lessons learned from warning message design for end-users. Especially, the warning design guidelines by Bauer et al. [7] provide a comprehensive list of principles for designing security warnings with a focus on end-users. We rely on their general and abstract principles for designing developer centered security warnings. Thus we adapted design goals from their guidelines and applied them to an API-integrated security advice concept. Additionally, we considered lessons learned from previous secure programming studies with software developers [1–3, 6].

Figure 1 illustrates the design concept of the security feedback mechanism we present below. Although we contribute a design concept based on existing principles and lessons learned, we do not comparatively evaluate different design approaches in our work. We expect this to be future work.

Goal 1: *Follow a Consistent Layout.* In contrast to security warnings for graphical user interfaces, an API-integrated security feedback mechanism that relies on terminal and console output only allows for limited interactions with users. Interface control elements such as buttons are hardly available in such an environment.

However, we still aimed to provide a consistent look and feel and layout concept for security warnings in different scenarios. Figure 1 illustrates all seven sections relevant for our security warning. The upper left corner (1) is used to indicate a dangerous situation. We use section (2) to give a brief description of the security warning's root cause and section (3) to point the developer to the file and line number that triggered the warning. Section (4) is used to communicate

consequences of using the insecure API calls that were responsible for the security warning. We use sections (5) and (6) to provide context sensitive and actionable advice for the developer. Section (5) provides actionable advice to improve code security while section (6) shows information that allows developers to turn off future security warnings. Section (7) provides links to further background information.

Goal 2: *Describe the Risk Comprehensively.* We aim to clearly and comprehensively communicate the underlying risk to the developer. In contrast to TLS warnings [15] or Android permission dialogs [17], our security warning does not have to deal with false positives. Even in cases when developers made insecure choices intentionally, e.g., for backward compatibility requirements of legacy systems, a security warning is still a true positive.

We rely on sections (1), (2) and (4) to communicate the respective risk to the developer. Section (1) uses a red flashing text icon "/!\", indicating a warning sign. Additionally, we integrated "WARNING" text in capital letters and red color in section (1). Section (2) uses red colored text explaining the root causes of the warning, e. g.: "You are using the weak encryption algorithm RC4 (aka ARC4 or ARCFOUR)". Additional details to communicate the existing risk and its potential consequences are provided in section (4). In case of an RC4 warning, e.g., "The use of ARC4 puts the processed data's confidentiality at risk and may lead to data disclosure."

Goal 3: *Present Relevant Contextual Information.* We aim to present relevant contextual information including the specific location in the source code that triggered the security advice. This helps developers to identify the insecure API use that needs to be fixed. In addition to the filename and line number, section (3) includes a snippet of the source code that triggered the warning.

Goal 4: *Offer Meaningful Options.* The most crucial aspect of a security warning is to offer meaningful options to get out of the situation that triggered the warning. In our case we expect the developer to modify code to either fix a security issue or suppress the warning message for future runs (i. e. click through the warning). In section (5) we provide a secure code snippet to turn the insecure code into secure code and offer an insecure option in section (6) which disables this specific security warning in future runs. Additionally, we provide links to more background information such as OWASP or NIST guidelines for secure programming.

Goal 5: *Be Concise and Accurate.* The guideline of Bauer et al. [7] focused on the design of end-user warnings and recommends to refrain from technical jargon. However, since we target software developers, we do not adopt this recommendation. Technical jargon from the software development domain such as specific names, locations and values of source codes are common elements for developers. Thus, we made such terms part of the warning message. Terms, concepts, technologies and standards from the cryptography domain, however, can not be expected to be general knowledge of a developer [1]. Hence, we omitted cryptographic jargon as much as possible.

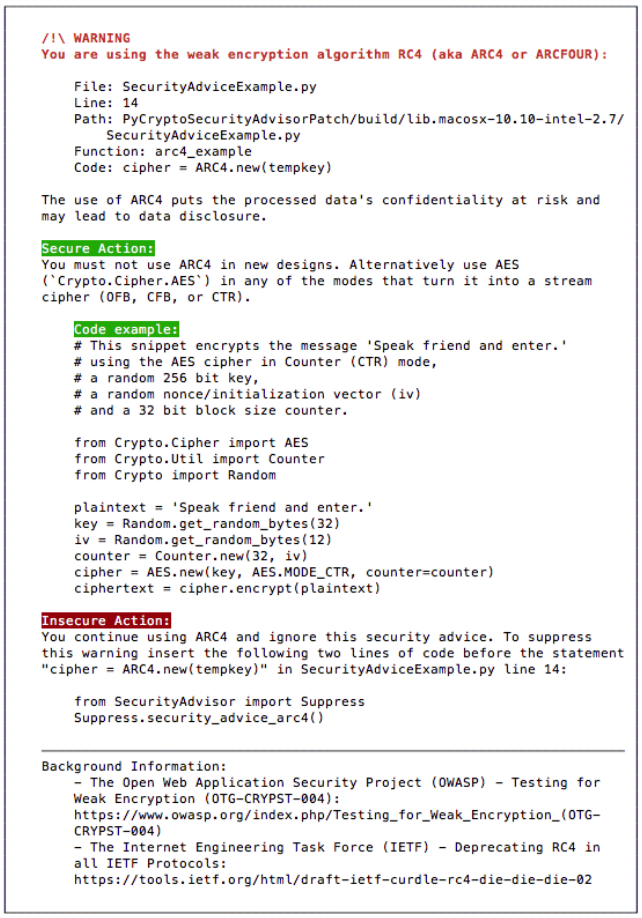

Figure 2: Security advice design of the patched version of PyCrypto triggered by an RC4 usage and displayed in a terminal running python code.

5 Implementation

We implemented the security warning concept from above for a subset of the PyCrypto API for the Python programming language. Figure 2 shows a sample security warning for the insecure RC4 algorithm for symmetric encryption. To assess API call security, we followed the classification provided by Acar et al. [1].

Selecting Python and PyCrypto: We chose to use Python for our experiment because it is very popular, used across many communities and supports many different fields of application. Since Python is easy to read and write and has a large user base [20] we reasoned that recruiting Python developers for our study would be straightforward.

The Python cryptographic PyCrypto [35] API is widely used amongst Python developers. The API provides low level interfaces for cryptographic functionalities, features symmetric as well as asymmetric encryption and supports multiple hashing algorithms as well as some utility features.

PyCrypto comes with primarily auto-generated documentation that includes minimal code examples. The documentation recommends the Advanced Encryption Standard (AES) and provides an example, but also describes the weaker

Data Encryption Standard (DES) as cryptographically secure.[1] The documentation warns against weak exclusive-or (XOR) encryption. However, the documentation does not warn against using the default Electronic Code Book (ECB) mode, or the default empty IV, neither of which is secure. [11, 31]

We chose this API as Acar et al. [1] had identified that developers using this API are likely to produce functionally correct but insecure code. This indicates in general a high potential for improvement. Furthermore, more than 30% of 307 participants in another study [3] preferred PyCrypto over other cryptographic APIs for Python.

5.1 How our Patch works

The PyCrypto patch hooks specific API calls that create instances of weak cryptographic objects such as the call to `Crypto.Cipher.ARC2.new()` which creates a new cipher object that uses the insecure ARC2 [27] algorithm. Whenever an insecure cryptographic object is created, our patch calls an advice method that uses contextual information to show a security warning. To fetch contextual information, the advice method relies on Python's `inspect` module and accesses the cryptographic object's stack frame. The stack frame is used to add information about the responsible file, the line number in that file and the name of the method that triggered a new security warning. Using the respective stack frame information and information about the cryptographic object instantiation that called the security advice method is then used to compile a context specific security warning (cf. Section 4).

5.2 Covered API Calls

For the security warnings, we focused on aspects that we wanted to test in a developer study later on (cf. Section 6). Table 1 gives an overview of both the API calls the security advice does cover and the API calls for which we did not implement security warnings.

In particular, we addressed weak symmetric encryption algorithms (cf. Table 1) and recommended the use of the Advanced Encrytion Standard (AES) as a secure alternative. This is in line with the recommendation of the official developer documentation of PyCrypto. The security warning also recommended an upgrade from the insecure Electronic Code Book (ECB) mode of operation to the secure counter mode (CTR) streaming cipher. In general, we recommended the counter mode (CTR) as a secure mode of operation in all symmetric security warnings. CTR is considered a secure mode of operation and is recommended by the official PyCrypto documentation.

In addition to security warnings for insecure symmetric encryption algorithms, we triggered security warnings for weak hash algorithms (cf. Table 1) and recommended the use of the SHA-512 hash function as a secure alternative.

In general, all security warnings we provided adhered to the documentation to not confuse participants in case they looked up programming questions.

[1]This might be due to the fact that the library has last been updated on 20 Jun 2014.

5.3 Not Implemented

We did not implement a security warning for every insecure cryptographic choice PyCrypto users can make. While we implemented all features that affected the programming tasks in our developer study (cf. Section 5.2), our patch does not cover the PyCrypto API calls below.[2]

We did not implement security warnings for any of the public key and digital signature schemes provided by PyCrypto (cf. Table 1).

6 Developer Study

We used an online, between-subjects study to compare how effectively developers could write correct, secure code using either PyCrypto as a control, or our patched version of PyCrypto with the security intervention. We recruited developers with demonstrated Python experience (on GitHub) for an online study; we also recruited via mailing lists and developer forums.

Participants were assigned to complete a short set of programming tasks; they were randomly assigned either the PyCrypto control condition, or the PyCrypto patch condition, where we tested our security warning.

Within each condition, task order was randomized. All participants were given a symmetric encryption task and a symmetric key generation and storage task.

After finishing the tasks, participants completed a brief exit survey about the experience. We examined participants' submitted code for functional correctness and security.

Ethics and Pre-testing: Due to the location of our universities, there was no formal IRB process. We did, however, model our study material and procedures after an IRB-approved study and adhered to the strict German data and privacy protection laws.

We conducted expert reviews for the design and implementation of our security advice. Therefore, we asked experienced human computer interaction researchers to walk through the warnings and give us feedback. Additionally, we pre-tested the functionality of our PyCrypto patch extensively with participants we excluded from the study later on.

6.1 Study Design

Our study has two conditions; it is modeled closely after the Acar et al. 2017 study on cryptographic Python APIs, which compared the usability of five cryptographic APIs for Python, namely PyCrypto, cryptography.io, M2Crypto, Keyczar and PyNaCl [1], in a between-subjects study for symmetric and asymmetric encryption via three symmetric or four asymmetric programming tasks: (a) a key generation and storage task, (b) an encryption and decryption task, (c) key derivation (symmetric condition only), (d) certificate validation (asymmetric condition only). They find that usability varies wildly across libraries and tasks, with poor usability contributing to insecure code.

In our study, we compare the PyCrypto library to our patched version of PyCrypto. The PyCrypto condition

[2]However, extending the patch to cover a more comprehensive list of features is possible.

Triggers a Security Warning	Security Advice
Crypto.Cipher	
↳ AES.new(k, AES.MODE_ECB, iv)	→ AES.new(k, AES.MODE_CTR, iv)
↳ CAST.new(k, CAST.MODE_ECB, iv)	→ CAST.new(k, CAST.MODE_CTR, iv)
↳ CAST.new(length(k) < 128 bit, mode, iv)	→ AES.new(k, AES.MODE_CTR, iv)
↳ ARC2.new(k, mode, iv)	
↳ DES.new(k, mode, iv)	
↳ DES3.new(k, mode, iv)	
↳ Blowfish.new(k, mode, iv)	
↳ XOR.new(k, mode, iv)	
↳ ARC4.new(k, mode, iv)	
Crypto.Hash	
↳ MD2.new()	→ SHA512.new()
↳ MD4.new()	
↳ MD5.new()	
↳ RIPEMD.new()	
↳ SHA.new()	

Not Implemented
Crypto.Cipher
↳ PKCS1_OAEP
↳ PKCS1_v1_5
Crypto.Protocol
↳ AllOrNothing.isInt
Crypto.PublicKey
↳ DSA
↳ ElGamal
↳ RSA
Crypto.Signature
↳ PKCS1_PSS
↳ PKCS1_v1_5
Crypto.Util
↳ RFC1751
↳ strxor

Table 1: All implemented/not implemented PyCrypto API calls. Implemented calls trigger security warnings; `k` is the key parameter; `iv` is the initialization vector parameter. Not implemented calls did not affect our study results, as they were not useful for our selection of study tasks.

serves as a control. In this study, we focus on symmetric encryption only, and only assign two tasks in random order: In an online Python coding environment [37], our participants were asked to solve two randomly ordered tasks – a symmetric encryption and key generation and storage task – after which they were asked to complete an exit survey that asked usability questions about the library they used in their condition, familiarity with programming in general and Python in particular, and demographic information. In the PyCrypto patch condition, participants were asked to solve both programming tasks with the pre-installed patched PyCrypto version that showed our security advice when triggered according to Section 5.2. Participants could then take the advice for their final solution; ignoring or bypassing the security advice was also possible. In the PyCrypto condition, participants were asked to solve the same two tasks without the support of security advice. Our control condition replicates the 2017 PyCrypto condition for a subset of two out of three of the original tasks [1]. We can therefore not only compare our results across our conditions, but also to the 2017 study.

6.2 Recruitment and Framing

Our study reuses most of the infrastructure of the publicly available Developer Observatory [1,37], and our recruitment strategy closely resembles that described in past studies with the same framework. To gain meaningful, ecologically valid results, we aimed to recruit developers familiar with Python.

We sent a total of 38,533 email invites to randomly sampled contributors from 100,000 publicly available Python repositories. We additionally posted invitations in Python forums and sent emails to our personal network.

In our invitation, we asked Python developers to participate in a Python study via an online code editor. Our invitations did not mention a security or cryptography context to avoid biasing potential participants. The invitation email included links to learn more about the study and to blacklist the recipient email from any further communication related to our research, a request which we honored. The participation link contained a unique pseudonymous identifier (ID), which allowed us to assign study results and GitHub statistics to the invited email addresses.

Recipients who clicked the link to participate in the study were sent to a landing page containing a consent form. Once they confirmed their legal age, consented to the study and were comfortable with participating in the study in English, they were introduced to the study framing previously used by Acar et al. [1]. We asked participants to imagine they were developing code for an app called CitizenMeasure, "a new global monitoring system that will allow citizen-scientists to travel to remote locations and make measurements about such issues as water pollution, deforestation, child labor, and human trafficking. Please keep in mind that our citizen-scientists may be operating in locations that are potentially dangerous, collecting information that powerful interests want kept secret. Our citizen scientists may have their devices confiscated and hacked." We hoped that this framing would both engage participants' interest and nudge them to attempt to write secure code. We also gave instructions for the study infrastructure, which we describe next.

6.3 Experiment Infrastructure

Our online developer study uses our publicly available framework (cf. [37]). The framework allows participants to write and test cryptographic code in their browser, is based on a Jupyter Notebook environment [26] and was hosted on our server. This allowed us to control the development environment including available libraries (PyCrypto in this study) and to retrieve written code and corresponding metadata (e.g., copy&paste events).

As our security advice implementation (cf. Section 5) uses ANSI ESCAPE sequences [24] to colorize text in diverse terminals on various platforms, we had to update Jupyter Notebook to the latest version 4.4.0 in order to be able to display our warning appropriately (ANSI colors were not processed correctly by Jupyter until version 4.1.0). Due to API changes we had to adjust some parts of the Developer Observatories implementation. Depending on conditions, the original version of PyCrypto or the patched version of PyCrypto were used by a participant. Because both share

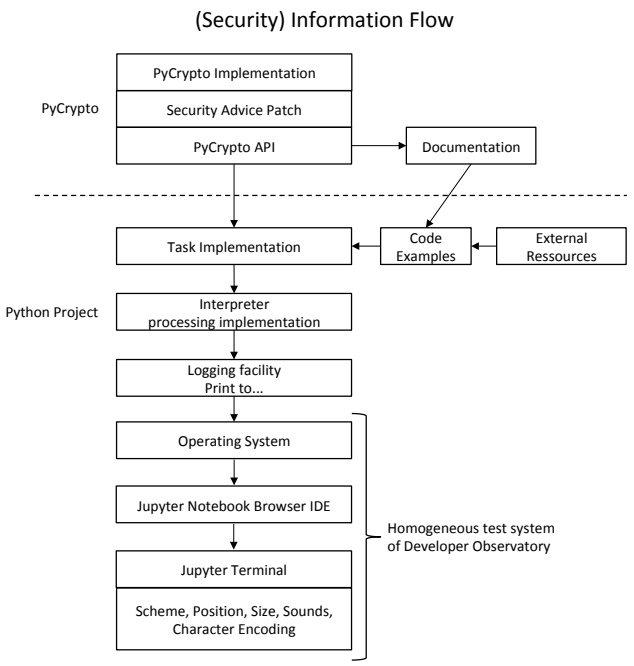

Figure 3: Security Information Flows in development environments through the example of the cryptographic Python API PyCrypto and Developer Observatory [37]

exactly the same name space and the identical API, we installed each library in a virtual Python 2.7.12 environment of which only one was used as kernel in Jupyter.

Figure 3 illustrates how the information of our security advice is technically transferred from the patched PyCrypto API via the Python intepreter and Python logging facility to the participants homogeneous test environment of Developer Observatory.

To prevent interference between participants, each participant was assigned to a Notebook running on a separate Amazon Web Service (AWS) instance. We maintained a pool of prepared instances so that each new participant could begin without waiting for an instance to boot. Instances were shut down when each participant finished, to avoid between-subjects contamination.

Tasks were shown one at a time, with a progress indicator showing that the participant had completed, e. g., 1 of 2 tasks. For each task, participants were given buttons to "Run and test" their code, and to move on using "Solved, next task" or "Not solved, but next task." After each button press, we stored the participant's current code, along with metadata like timing, in a remote database.

Allowing participants to write and execute Python code presents serious security concerns. To mitigate this, we removed all unnecessary software packages from the AWS image. We used the AWS firewall to restrict incoming traffic to port 80 and prevent outgoing traffic other than to our study database, which was password protected and restricted to sanitized insert commands. All instances were shut down within 4 hours of the last observed participant activity.

6.4 Task Design

To be able to compare our results not only to our own control, but also to past results both in functionality outcome, security outcome and usability, we re-used a subset of tasks from the Acar et al. study on the usability of cryptographic APIs [1]. These tasks had previously been chosen to be "short enough so that the uncompensated participants would be likely to complete them before losing interest, but still complex enough to be interesting and allow for some mistakes" and designed to "model real world problems that Python developers could reasonably be expected to encounter in their professional career." We chose two symmetric encryption tasks: generating an encryption key and storing it securely in a password-protected file, and using the key to encrypt some plain text.

For both tasks, participants were provided with stub code and some commented instructions. These stubs were designed to make the task clear and ensure the results could be easily evaluated. We also provided a main method pre-filled with code to test the provided stubs. This helped orient participants and saved time, but it did prevent us from learning how participants might have designed their own tests.

We also asked participants to please use only the PyCrypto documentation, if at all possible, and to report (in comments) any additional documentation resources they consulted. Task order was randomized between participants.

Replication: In the control group, participants were asked to solve the tasks using PyCrypto as-is. Except for a change in task design (i.e., removing the decryption task), this condition is identical with the Acar et al. study PyCrypto condition for their set of symmetric tasks.

Security Advice Condition: Participants in the PyCrypto patch condition were asked to solve the same set of tasks using PyCrypto; they were not alerted that they were using a patched version of PyCrypto. If they successfully executed functional code that was insecure according to the classification in Table 1, and the insecure programming choice was covered by the patched version of PyCrypto, the respective warning message was shown.

6.5 Exit Survey

Once both tasks had been completed or abandoned, the participants were directed to a short exit survey. We asked for their opinions about the tasks they had completed and the PyCrypto API, including the Acar et al. usability questionnaire for security APIs [1]. We also collected their demographics and programming experience. The participant's code for each task was displayed (imported from our database) for their reference with each question about that task. We were also interested in whether participants perceived the security warning at all, if it was helpful and if participants could recall the security warning's content. The Exit survey can be found in the Appendix D.

6.6 Evaluating Solutions

We based our analysis on the code submitted for each task by our participants. Submitted solutions were evaluated both for functional correctness and security. We evaluated each

task independently with two coders based on a subset of the codebook provided by [1]. Disagreements between the two coders were adjudicated by a third coder allowing us to solve all conflicts.

Functionality: For each programming task, we assigned a participant a functionality score of 1 if the code ran without errors, passed the tests and completed the assigned task, or 0 if not.

Security: We assigned security scores only to those solutions which were graded as functional. To determine a security score, we considered several different security parameters. Our scoring followed the relevant parts of the security scoring in [1]. Still we give a brief summary of the security scoring we applied.

For key generation, we checked key size and randomness. For key storage we checked if encryption keys were actually encrypted and if a proper encryption key was derived from the password we provided. For key derivation, we scored use of a static or empty salt, HMAC-SHA1 or below as the pseudorandom function, and less than 10,000 iterations as insecure. For the symmetric encryption task, participants had to select encryption parameters. Therefore, we scored the security of the chosen encryption algorithm, mode of operation, and initialization vector. We scored ARC2, ARC4, Blowfish, (3)DES, and XOR as insecure, and AES as secure. We scored the ECB as an insecure mode of operation and scored CBC, CTR and CFB as secure. Static, zero or empty initialization vectors were scored insecure.

We calculated Krippendorff' alpha [28] for the initial coding by two coders across all security codes; $\alpha = 0.764$, which is within reasonable bounds for agreement [14]. Conflicts were resolved afterwards.

Participant Stories: In addition to our assessment of code functionality and security, we analyzed participants' code in detail, qualitatively, based on the recorded code and console output that we automatically stored for each test run of code. We recreated the sequence of task solutions that each participant executed, the *participant story*, where we could see whether they were shown our security advice and which version was shown, whether or not they subsequently adapted their code to incorporate our suggestions, and whether or not this reaction lead to a secure version of their solution. We additionally see whether they reported having seen a warning in the exit survey, and whether or not they perceived it as useful. We use these participant stories to give insight into four questions: (1) did the developers see the warning?, (2) did they react by modifying their code? (3) did they use our examples in their code? and (4) did this consideration lead to improved code security?

7 Data Analysis

In our data analysis, we use the non-parametric Mann-Whitney-U test (MWU) to compare two groups with continuous responses, compare categorical responses with Person's chi-squared test (χ^2) or instead with Fisher's exact test where applicable, and fit regression models to our results.

For each regression analysis, we consider a set of candidate models and select the model with the lowest Akaike Information Criterion (AIC) [9]. In cases when we consider results on a per-task rather than a per-participant basis, we use a mixed model that adds a random intercept to account for multiple tasks from the same participant. We consider candidate models consisting of the required factors "Task" and "Warning displayed", as well as (where applicable) the participant random intercept, plus every possible combination of the optional variables. Required factors, optional factors, and corresponding baseline values are described in Table 2.

We present the outcome of our regressions in tables where each row contains a factor and the corresponding change of the analyzed outcome in relation to the baseline of the given factor. For logistic regressions, the odds ratio (O.R.) measures change in likelihood of the targeted outcome in relation to the baseline factor O.R. of one. Linear regression models measure change from baseline factors with a coefficient (Coef.) of zero for the value of the outcome. For each factor of a model, we also list a 95% confidence interval (C.I.) and a p-value indicating statistical significance.

8 Results

We present the results for our study based on 53 valid participants. Participants were generally successful in functionally solving the tasks, while security results varied across conditions, the patched condition being an improvement over PyCrypto where applicable. This improvement was pronounced: participants who wrote code that triggered a warning message were 15× as likely to convert it to a secure condition as opposed to participants who wrote similar insecure code in the PyCrypto condition. However, the effectiveness of our PyCrypto patch was negatively impacted by the limited applicability of the warnings.

8.1 Participants

We recruited participants for our study by sending email invitations to GitHub developers (cf. Figure 4) and by advertising the study in developer forums. Of 38,533 sent invitation emails, 3,422 (8.9%) bounced and 65 (0.2%) recipients requested to be removed from our mailing list.

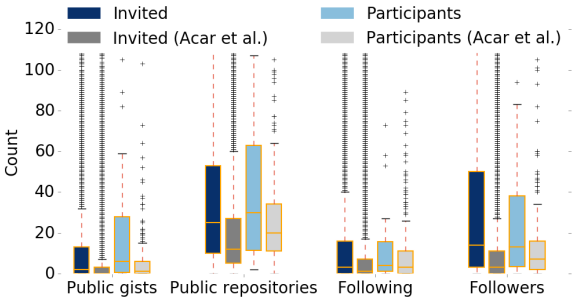

Figure 4: Boxplots comparing invited participants with valid participants and participants from Acar et al. [1]. The center line indicates the median; the boxes indicate the first and third quartiles. The whiskers extend to ±1.5 times the interquartile range. Outliers greater than 150 were truncated for space.

Factor	Description	Baseline
Required		
Task	Performed task (Storage or Encryption).	Encryption
Warning	True or False, whether a warning was displayed.	False
Participant	Random effect accounting for repeated measures.	n/a
Optional		
Used documentation	True or False, used API documentation, self-reported.	False
Development experience	Development experience in years, self-reported.	n/a
Python experience	Python programming experience in years, self-reported.	n/a
PyCrypto experience	Previous experience with the PyCrypto library (used, seen, none), self-reported.	None
Security task experience	Previous experience in solving security tasks (written, seen, none), self-reported.	None

Table 2: Factors used in regression models. Model candidates were defined using all possible combinations of optional factors, with the required factors included in every candidate. Final models were selected by minimum AIC. Categorical factors are individually compared to the baseline.

We received no reports of technical errors with the survey infrastructure; one participant refused to participate in the study, because our Amazon AWS instances were not accessible via HTTPS. One participant refused to participate because he perceived our invitation email to be dubious.

272 people agreed to the consent form and 177 started working on the tasks. Of those, 70 finished the tasks and 68 completed the exit survey. We excluded 15 participants since results indicated a lack of serious answers (4) or were the result of curious clicking-through (3). Unless stated otherwise, we report results for the remaining 53 valid participants which finished the tasks and completed our exit survey.

The majority of our 53 valid participants reported being male (49, 92.5%) while the remaining participants reported being female (1), other (1), or preferred to not answer (2). The reported age was between 20 and 60 (Mean 34.9, SD 8.1). 44 of our participants received invitation emails as GitHub developers, while the remaining 9 were recruited on developer forums. Our participants reported a mean developer experience of 15.8 years (SD 8.2, prefer not to answer: 3) and a mean Python experience of 8.44 years (SD 4.7, prefer not to answer: 3). 48 reported their occupation as being professionals and 3 reported being students (Both: 1, prefer not to answer: 1).

8.2 Dropouts

95 did not continue to the study while 177 started the first programming tasks by clicking the begin button. 57 participants stopped in the key storage task, additional 44 in the content encryption task and 4 in the final test routine before finishing the online programming part of the study. 29 had written code to solve a task in contrast to 76 who did not modify any text in the Jupyter notebook. 5 dropped out of our PyCrypto patch condition after having triggered security advice. 70 proceeded to the exit survey. 68 participants finished the exit survey of which we had to exclude 15 persons due to non serious participation and technical issues in our infrastructure.

We saw that out of 115 participants in the PyCrypto patch condition, 90 participants dropped out of whom 5 were shown a warning. However, the 26 who finished the study were shown 11 warnings, so we assume that seeing a warning was not a strong reason to drop out of the study. We compare this to 34 dropouts out of 62 who started the PyCrypto condition. The increased count of starting participants was due to an effort to counterbalance for the limited applicability of the warnings.

8.3 Results for Functionality

Generally, participants were well able to solve tasks: 87.8% of attempted tasks were functional (89.7% functional in the PyCrypto condition, 85.9% in the PyCrypto patch condition).

We were unable to observe a significant impact, positive or negative, of our warning messages on results, as shown in Table 3. Since the warning message was only presented after functionally correct code was executed, this is to be expected. However, the interruption caused by the warning message did not cause developers to break their code.

Factor	O.R.	C.I.	p-value
Storage Task	0.00	[0, ∞]	0.972
Warning displayed	0.22	[0.03, 1.9]	0.169
Development experience	0.95	[0.84, 1.07]	0.369
Python experience	1.19	[0.93, 1.52]	0.169

Table 3: Results of the final logistic regression model examining whether displayed warnings affect task functionality. Odds ratio (O.R.) indicates relative likelihood of a task being functional. Some trends are observable but no results are statistically significant. See Table 2 for further details.

8.4 Results for Security

For security, we observed 26.9% secure solutions in the PyCrypto condition; compared with 50.7% in the PyCrypto patch condition. We were not able to obtain a meaningful regression model (cf. Appendix B), caused by the small number of tasks that triggered and ended up with insecure code in the PyCrypto patch condition (11), as well as the small number of tasks that would have triggered a warning but were not modified to be secure in the PyCrypto condition (22). We followed this inconclusive model up with Fisher's exact test (cf. Table 4, which was significant ($p<0.01$), with an odds ratio of 56. The warning messages were noticed by participants who saw them, which was clear both from self-reported memory of them as well as changes in their code:

		Secure	
		F	T
Warning	F	21	1
	T	3	8

Table 4: Contingency table for secure task solutions and triggered warnings used in our Fisher's exact test.

Factor	Coef.	C.I.	*p*-value
Warning displayed	0.00	[0, 112.51]	0.271
Development experience	0.67	[0.35, 1.27]	0.229
Python experience	0.73	[0.23, 2.3]	0.595

Table 5: Linear regression model examining usability perceived by participants. See Table 2 for further details.

the warning message lead to a change from initial insecure code to a secure solution in most cases (8 out of 11). Generally, the applicability of the warning message was limited; it applied to 24 of 44 insecure solutions across conditions, and was shown in 11 of 22 insecure cases in the PyCrypto patch condition.

Impact of Intervention on Perceived Usability: API usability as interpreted by answers to questionnaire by Acar et al. [1] based on the Cognitive Dimensions framework [10] did not change for better or worse with the warning (cf. Table 5). This is to be expected, as only one of our 10 questions that are calculated into the usability score focus on meaningful warning/error messages. We investigate in detail the answers to the following questions:

W1 The security warnings displayed in the console helped to solve this task.

W2 When I made a mistake, I got a meaningful error message/exception.

W3 Using the information from the error message/exception, it was easy to fix my mistake.

We transform agreement on a 5-point likert-scale as follows: neutral is represented by 0, while strong disagreement is represented by -2 and strong agreement is represented by $+2$. The mean agreement to W1 was 1 (median = 1) in the PyCrypto condition compared with 0.76 (median = 1) in the PyCrypto patch condition (MWU-test; U=32.5; p=0.4205). Participants gave a mean agreement of 0.593 (median = 1) to W2 in the PyCrypto condition compared with 0.833 (median = 1) in the PyCrypto patch condition (MWU-test; U=384; p=0.2167), and a mean agreement of of 0.846 (median = 1) to W3 in the PyCrypto condition compared with 0.917 (median = 1) in the PyCrypto patch condition (MWU-test; U=296; p=0.7484). We interpret this as a generally positive impression of our warning, despite our preliminary fear of annoying or overwhelming developers. However, even in these specific cases, perceptions were not significantly more positive or negative than in the control condition.

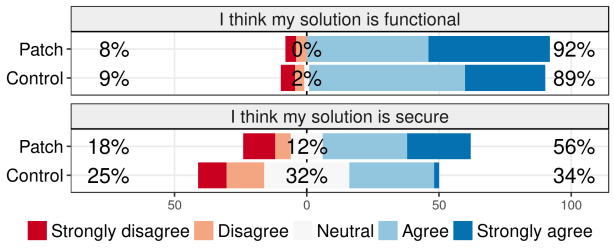

Figure 5: Likert-plot showing our participants' perceptions regarding functionality and security of their solutions. "I don't know" answers were omitted.

8.5 Detailed Task Analysis

Participants were asked to rate their functional and security success after completing the tasks (cf. Figure 5). Interestingly, we found that for the encryption tasks, all participants who saw the warning message were correct in assessing their solution's security. We compare this to the control condition, where, for the encryption task, only 66% task security ratings were correct in cases where the warning would have applied. In the key storage task in the patched condition, 73% of assessments were correct, while all assessments were correct in the control group.

Participant Stories: From the collected participants' stories we derived further qualitative results. When focusing on the content encryption task, 7 participants were shown a security warning. All of them saw and remembered it, as they reported in the exit survey. 2 of the 7 participants did not choose to use our guidance to improve their code. One tried to suppress the advice, another one ignored it. The remaining 5 participants accepted the advice and modified their code: 2 of them adopted the example code provided by the advice; they later stated their satisfaction: "*The warning helpfully directed me towards an improved solution, and provided example code*" and "*The warning explained clearly that DES was considered as insecure, and provided an example to use AES instead. This helped me solving this task in a more secure manner*". The remaining 3 participants partially followed the advice: they did adapt their code in response to the warning, but chose a different mode of operation than was suggested in the warning. The proposed solution recommended the use of standard encryption algorithm AES in counter (CTR) mode. The 3 participant instantiated AES in cipher block chaining (CBC) mode instead. A closer look at their code revealed that 2 of them appeared to have problems in transferring the suggested code snippet into running code. While this points to a usability problem with the warning/advice, we were able to observe that 4 out of 5 participants who modified their code in reaction to our warning at least attempted to adhere to our suggestion. Altogether, 5 out of 7 participants who saw the warning for the encryption task modified their code into a secure solution.

We could observe similar behavior for the key generation and storage task. Here, 4 participants were shown security advice; all of them noticed the warning. One ignored the warning; the remaining 3 modified their code. One adopted the suggested code snippet as-is; the other 2 chose CBC mode instead of CTR mode.

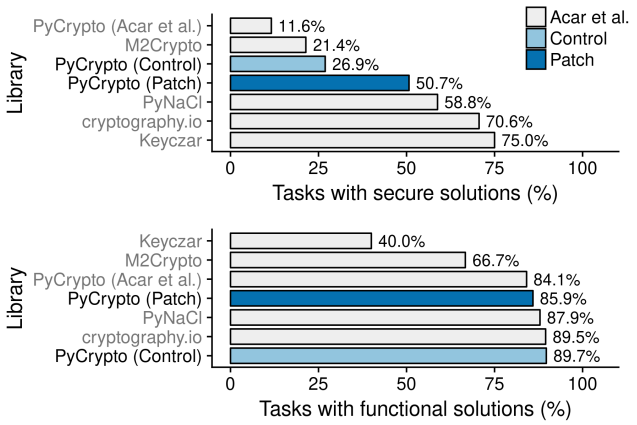

Figure 6: Comparison between secure solution percentages of libraries from Acar et al. [1] and our PyCrypto Control/Patch data. To match our tasks, only symmetric encryption tasks are considered for libraries from Acar et al.

Limited applicability of warning message: The programming tasks in our study were designed in a way participants had to only use symmetric cryptography. Thus we did not cover insecure asymmetric API features (cf. Table 1). However, 4 participants in the patch condition solved tasks by using asymmetric methods. They implemented key derivation for asymmetric RSA keys or applied RSA to encrypt keys and messages. Our security advice implementation were not able to help these participants using symmetric cryptography since it is not possible to give task sensitive advice at this position. For this reason we had to exclude these tasks from the detailed analysis.

8.6 Replication Results

Our study is based on the study that compared the usability of different cryptographic APIs conducted by Acar et al. [1]. This section discusses the aspects we replicated and the replication results.

Our participants created a similar level of functional tasks as compared to the 2017 study (cf. Figure 6). However, our control group achieved better security results than the original study.

Participants in the PyCrypto patch condition of our study achieved a higher level of security than our own control group, which places the PyCrypto patch condition among the better-performing of libraries. While the effect of more experience applies here, too, it is interesting to see that this result was achieved without changes to the API abstraction level, learnability, documentation, Stack Overflow or the study design. Additional details about common errors of our participants compared to PyCrypto library users from Acar et al. [1] can be found in Appendix C.

9 Limitations

We address multiple limitations below:

Security Advice Design: The design of our security advice is based on heuristics defined by previous research from warning message design for end-users. Additionally, we considered lessons learned from previous work on secure programming studies. After manual pre-testing and expert reviews, we opted for a solution shown in Figure 1. However, there might be more effective designs we did not consider (e. g. following an opinionated design approach might provide better results). Although this is a limitation of our current approach, results for our solution show a significant positive impact on code security. Hence, we leave changes to the design and comparing different versions to future work.

Security Advice Implementation: The implementation of our security advice does not cover all possible insecure choices PyCrypto users can make, e.g. we did not implement security warnings for PyCrypto's asymmetric API (cf. Section 5), however, these were not included in our study design. We address participants using APIs not covered by our security advice, as well as cases where we failed to show security advice (e.g., non-random IVs for symmetric tasks) in our data analysis (cf. Section 7).

User Study: We decided to conduct an online study over a laboratory study because it is difficult to recruit software developers (rather than students) at a reasonable cost. This design decision allowed us less control over the study environment. On the other side, we were able to recruit a geographically diverse set of participants. Sadly, we could not simply recruit participants from an online service such as Amazon Mechanical Turk for end-user focused studies. Since it is difficult to manage participants compensations outside such infrastructures, we did not offer our participants compensation. Due to the combination of unsolicited email invites and no compensation we expected a strong self-selection bias and are aware of the fact that our results might not necessarily be representative for all developers but in particular for those who are interested and motivated enough to participate. Our participants seem to be more active than average GitHub users (cf. Appendix A). However, these limitations apply across both conditions. In any online study, some participants may not provide full effort, or may answer haphazardly. We attempted to remove any obviously low-quality data before analysis, but cannot discriminate perfectly. Additionally, we tested a simple and limited scenario, which may have limited applicability to complex real world code.

Real-world applicability: Critically, a real-world roll-out of our advice is contingent on buy-in from library developers. While this requirement severely limits employment across all libraries, several cryptographic library developers have reached out after the 2017 study and showed commitment to improve their libraries' usability. We therefore hope that our study is not only of academic relevance, but can and will be applied to libraries with a large userbase.

10 Discussion

Overall, we found that our API-integrated security advice had a significantly positive effect on code security. However, we only tested a first implementation of our approach. Changing parameters such as text or advice design might result in even more secure code. We leave this to future

	Functionality	Security	Usability
Information Source [2]	✓	✓	—
Cryptographic Library [1]	✓	✓	✓
FixDroid [34]	✗	✓	—
Security Advice	✗	✓	✗

Table 6: Comparison of the impact of our security warning compared to previous investigations of the impact of other factors on code security.

work. The majority of the participants who were shown a security warning, fixed their code. Interestingly, showing participants a security warning had no effect on the functionality of participant solutions. Also, the perceived usability of PyCrypto as a cryptographic API was not affected by the security warning. Only one participant who received the advice, suppressed security warnings for future runs and two participants copied secure code snippets from a warning into their code.

Other Approaches: Comparing our security advice approach to previous work yields interesting results. Similar to high quality developer documentation, simple programming interfaces or IDE plugins our approach has a positive impact on code security. However, in contrast we could not find a positive effect on functionality (cf. Table 6). Also, in contrast to API design, our warning did not have a positive impact on perceived API usability.

However, our approach has multiple advantages in terms of deployability (cf. Section 4) and allows API providers to improve code security for existing cryptographic APIs in a bottom-up approach without having to change API design or relying on third party tools.

Lessons Learned: Most importantly we learned that API-integrated security advice can have a significant impact on code security. The majority of the participants who received security advice turned insecure code into secure code. Also, the adherence rate to our security advice (73%) was similar to adherence rates for browser warnings reported in previous work [15]. However, additionally we learned that designing and implementing effective security advice is challenging and has its limitations. Providing context sensitive information and secure and ready-to-use code snippets is complex and requires future work.

Future Work: Our work leaves room for future work in multiple directions.

While we evaluated API-integrated security advice for Python's PyCrypto API and reported a significantly positive effect on code security, it is unclear to which extent our concept can be applied to other security APIs. Hence, we aim to implement and test similar security warning concepts for a number of other security APIs such as for secure networking (e. g. TLS and HTTPS) or authentication (e. g. OAuth) as suggested by [30].

We followed security warning design guidelines by Bauer et al. [7] and considered lessons learned from related work on developer usable security research (cf. [1, 2, 34]) to design a first attempt at security advice. However, we only chose one specific design to test, and did not conduct any testing against other designs. Likely, the concrete design, content and presentation of the security advice can be improved. Future work could investigate the effect of an opinionated design approach or other security indicators. Warning message research for end-users showed significant impact of such factors on security (cf. [15]). Also, the integration of our approach in an integrated development environment (IDE) needs to be considered.

We conducted a between-subjects first contact study. In future work we plan to conduct a large scale in-situ field experiment to investigate the impact of habituation and fatigue on our approach.

11 Conclusion

In this paper, we evaluate the first API-integrated security advice for cryptographic APIs. We follow design guidelines by Bauer et al. [7] and consider lessons learned from previous work on human factors research for software developers. We implement a first design approach for Python's PyCrypto API and use the Developer Observatory framework [37] to conduct a between-subjects online controlled experiment. We evaluate the impact of our security advice on code security and perceived API usability and put our results in perspective of other approaches that try to support developers to write more secure cryptographic code.

Overall, we find that our security advice had a significantly positive impact on code security (**RQ1**) and did not affect the perceived API usability of our participants (**RQ2**). Similar to other approaches in previous work, the presented security advice helps to improve code security. Differently from other work, our approach allows API providers themselves to fix security and usability shortcomings of their interfaces without having to change programming interfaces or relying on resources outside their sphere of influence, such as third party information resources, IDE plugins or static code analysis tools (**RQ3**).

12 Acknowledgments

The authors would like to thank Joe Calandrino and the anonymous reviewers for providing feedback; and all participants of this study for their voluntary participation. This work has been partially funded by the German Federal Ministry of Education and Research within the funding program "Forschung an Fachhochschulen" (contract no. 13FH016IX6).

13 References

[1] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky. Comparing the usability of cryptographic APIs. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 154–171, 2017.

[2] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky. You get where you're looking for: The impact of information sources on code security. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 289–305, May 2016.

[3] Y. Acar, C. Stransky, D. Wermke, M. L. Mazurek, and S. Fahl. Security developer studies with github users: Exploring a convenience sample. In *Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pages 81–95, Santa Clara, CA, 2017. USENIX Association.

[4] D. Akhawe and A. P. Felt. Alice in warningland: A large-scale field study of browser security warning effectiveness. In *Proceedings of the 22Nd USENIX Conference on Security*, SEC'13, pages 257–272, Berkeley, CA, USA, 2013. USENIX Association.

[5] H. Almuhimedi, A. P. Felt, R. W. Reeder, and S. Consolvo. Your reputation precedes you: History, reputation, and the chrome malware warning. In *10th Symposium On Usable Privacy and Security (SOUPS 2014)*, pages 113–128, Menlo Park, CA, 2014. USENIX Association.

[6] T. Barik, J. Smith, K. Lubick, E. Holmes, J. Feng, E. Murphy-Hill, and C. Parnin. Do developers read compiler error messages? In *39th IEEE/ACM International Conference on Software Engineering (ICSE)*, pages 575–585. IEEE, 2017.

[7] L. Bauer, C. Bravo-Lillo, L. Cranor, and E. Fragkaki. Warning design guidelines. Technical Report CMU-CyLab-13-002, CyLab, Carnegie Mellon University, 2013.

[8] C. Bravo-Lillo, S. Komanduri, L. F. Cranor, R. W. Reeder, M. Sleeper, J. Downs, and S. Schechter. Your attention please: Designing security-decision uis to make genuine risks harder to ignore. In *Proceedings of the Ninth Symposium on Usable Privacy and Security*, SOUPS '13, pages 6:1–6:12, New York, NY, USA, 2013. ACM.

[9] K. P. Burnham. Multimodel Inference: Understanding AIC and BIC in Model Selection. *Sociological Methods & Research*, 33(2):261–304, 2004.

[10] S. Clarke. How usable are your APIs? In A. Oram and G. Wilson, editors, *Making software: what really works, and why we believe it*, Theory in practice, pages 545 – 565. O'Reilly, Beijing, 1 edition, 2010.

[11] Cwe-329: Not using a random iv with cbc mode. [Online]. Available: http://cwe.mitre.org/data/definitions/329.html, 2018.

[12] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel. An empirical study of cryptographic misuse in android applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 73–84, New York, NY, USA, 2013. ACM.

[13] S. Egelman, L. F. Cranor, and J. Hong. You've been warned: An empirical study of the effectiveness of web browser phishing warnings. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 1065–1074, New York, NY, USA, 2008. ACM.

[14] P. J. Fahy. Addressing some common problems in transcript analysis. *The International Review of Research in Open and Distributed Learning*, 1(2), 2001.

[15] A. P. Felt, A. Ainslie, R. W. Reeder, S. Consolvo, S. Thyagaraja, A. Bettes, H. Harris, and J. Grimes. Improving ssl warnings: Comprehension and adherence. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 2893–2902, New York, NY, USA, 2015. ACM.

[16] A. P. Felt, A. Ainslie, R. W. Reeder, S. Consolvo, S. Thyagaraja, A. Bettes, H. Harris, and J. Grimes. Improving SSL warnings: Comprehension and adherence. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 2893–2902. ACM, 2015.

[17] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, pages 3:1–3:14. ACM, 2012.

[18] A. P. Felt, R. W. Reeder, H. Almuhimedi, and S. Consolvo. Experimenting at scale with google chrome's ssl warning. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 2667–2670, New York, NY, USA, 2014. ACM.

[19] M. Georgiev, S. Iyengar, S. Jana, R. Anubhai, D. Boneh, and V. Shmatikov. 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, New York, NY, USA, 2012. ACM.

[20] Githut - programming languages and github. [Online]. Available: http://githut.info/, 2018.

[21] P. L. Gorski and L. Lo Iacono. Towards the Usability Evaluation of Security APIs. In *10th International Symposium on Human Aspects of Information Security and Assurance (HAISA)*, 2016.

[22] M. Green and M. Smith. Developers are not the enemy!: The need for usable security apis. *IEEE Security & Privacy*, 14(5):40–46, 2016.

[23] S. Indela, M. Kulkarni, K. Nayak, and T. Dumitraş. Helping johnny encrypt: Toward semantic interfaces for cryptographic frameworks. In *Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2016, pages 180–196, New York, NY, USA, 2016. ACM.

[24] ISO/IEC 6429. Information technology – control functions for coded character sets. [Online]. Available: https://www.iso.org/standard/12782.html, 1992. Third edition.

[25] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge. Why don't software developers use static analysis tools to find bugs? In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 672–681, Piscataway, NJ, USA, 2013. IEEE Press.

[26] Jupyter notebook. [Online]. Available: http://jupyter.org/, 2018.

[27] J. Kelsey, B. Schneier, and D. Wagner. Related-key cryptanalysis of 3-way, biham-des,cast, des-x, newdes, rc2, and tea. In Y. Han, T. Okamoto, and S. Qing, editors, *Information and Communications Security*, pages 233–246, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[28] K. Krippendorff. *Content Analysis: An Introduction to*

Its Methodology (2nd ed.). SAGE Publications, 2004.

[29] S. Krüger, S. Nadi, M. Reif, K. Ali, M. Mezini, E. Bodden, F. Göpfert, F. Günther, C. Weinert, D. Demmler, and R. Kamath. Cognicrypt: Supporting developers in using cryptography. In *Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering*, ASE 2017. IEEE Press, 2017.

[30] L. Lo Iacono and P. L. Gorski. I Do and I Understand. Not Yet True for Security APIs. So Sad. In *The 2nd European Workshop on Usable Security*, EuroUSEC '17, 2017. doi: 10.14722/eurousec.2017.23015.

[31] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[32] S. Nadi, S. Krüger, M. Mezini, and E. Bodden. "Jumping Through Hoops": Why do Java Developers Struggle With Cryptography APIs? In *Proceedings of the 37th International Conference on Software Engineering (ICSE 2016)*, 2016.

[33] A. Naiakshina, A. Danilova, C. Tiefenau, M. Herzog, S. Dechand, and M. Smith. Why do developers get password storage wrong?: A qualitative usability study. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17. ACM, 2017.

[34] D. C. Nguyen, D. Wermke, Y. Acar, M. Backes, C. Weir, and S. Fahl. A stitch in time: Supporting android developers in writing secure code. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17. ACM, 2017.

[35] Pycrypto - the python cryptography toolkit. [Online]. Available: https://www.dlitz.net/software/pycrypto/, 2018.

[36] B. Reaves, N. Scaife, A. Bates, P. Traynor, and K. R. Butler. Mo(bile) money, mo(bile) problems: Analysis of branchless banking applications in the developing world. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 17–32, Washington, D.C., 2015. USENIX Association.

[37] C. Stransky, Y. Acar, D. C. Nguyen, D. Wermke, D. Kim, E. M. Redmiles, M. Backes, S. Garfinkel, M. L. Mazurek, and S. Fahl. Lessons learned from using an online platform to conduct large-scale, online controlled security experiments with software developers. In *10th USENIX Workshop on Cyber Security Experimentation and Test (CSET 17)*. USENIX Association, 2017.

[38] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor. Crying wolf: An empirical study of ssl warning effectiveness. In *Proceedings of the 18th Conference on USENIX Security Symposium*, SSYM'09. USENIX Association, 2009.

[39] A. Vance, B. Kirwan, D. Bjornn, J. Jenkins, and B. B. Anderson. What do we really know about how habituation to warnings occurs over time?: A longitudinal fmri study of habituation and polymorphic warnings. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 2215–2227, New York, NY, USA, 2017. ACM.

[40] J. Weinberger and A. P. Felt. A week to remember: The impact of browser warning storage policies. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, pages 15–25, Denver, CO, 2016. USENIX Association.

[41] G. Wurster and P. C. van Oorschot. The developer is the enemy. In *Proceedings of the 2008 New Security Paradigms Workshop*, NSPW '08, pages 89–97, New York, NY, USA, 2008. ACM.

[42] J. Xie, B. Chu, H. R. Lipford, and J. T. Melton. ASIDE: IDE support for web application security. In *Twenty-Seventh Annual Computer Security Applications Conference, ACSAC 2011, Orlando, FL, USA, 5-9 December 2011*, pages 267–276, 2011.

APPENDIX

A Participants

Age	Youngest, Oldest	20, 60
	Prefer not to answer	3
	Mean years (SD)	34.9 (8.1)
Sex	Male	49
	Female	1
	Other	1
	Prefer not to answer	2
Recruitment	GitHub	44
	Other	9
Experience	Mean development years (SD)	15.8 (8.2)
	Mean Python years (SD)	8.44 (4.7)
	Prefer not to answer	3
Occupation	Pro	48
	Student	3
	Both	1
	Prefer not to answer	1

Demographic	**Invited**	**Valid**
Hireable	20.7%	13.0%
Company listed	41.4%	30.4%
URL to Blog	49.4%	47.8%
Biography added	19.1%	21.7%
Location provided	63.9%	65.2%
Public gists (median)	2.0	6.0
Public repositories (median)	25.0	30.0
Following (users, median)	3.0	4.0
Followers (users, median)	14.0	13.0
GitHub profile creation (days ago, median)	2431.0	2589.0
GitHub profile last update (days ago, median)	30.0	30.0

Table 7: GitHub-related demographics for invited users and valid GitHub participants.

Error	Our Study	Acar et al.
No Encryption	0 (0.00%)	0 (0.00%)
Weak Algorithm	10 (35.71%)	17 (41.46%)
Weak Mode	9 (32.14%)	23 (56.10%)
Static IV	11 (39.29%)	29 (70.73%)
Participants	53 (100%)	41 (100%)

Table 10: Common errors in the encryption task of our participants compared to PyCrypto library users from Acar et al. [1].

B Regression Model

Factor	O.R.	C.I.	p-value
Warning displayed	4.70	[0.04, 492.14]	0.515
Storage Task	0.13	[0.01, 1.25]	0.078
Development experience	1.11	[0.88, 1.39]	0.378

Table 8: Results of the final logistic regression model examining whether displayed warnings improve task security in cases where a warning would have been triggered. Odds ratio (O.R.) indicates relative likelihood of a task being secure. Some trends are observable but not results are statistically significant. See Table 2 for further details.

C Replication

Error	Our Study	Acar et al.
Key In Plain	1 (3.57%)	4 (9.76%)
Weak Cipher	9 (32.14%)	11 (26.83%)
Weak Mode	7 (25.00%)	14 (34.15%)
Static IV	9 (32.14%)	3 (7.31%)
No KDF	16 (57.14%)	15 (36.59%)
Custom KDF	16 (57.15%)	11 (26.83%)
KDF Salt	1 (3.57%)	1 (2.44%)
KDF Algorithm	3 (10.71%)	1 (2.44%)
KDF Iterations	1 (3.57%)	2 (4.88%)
Participants	53 (100%)	41 (100%)

Table 9: Common errors in the key file task of our participants compared to PyCrypto library users from Acar et al. [1].

D Exit Survey Questions

D.1 Task-specific questions: Asked about each task

Please rate your agreement to the following statements:

I think I solved this task correctly.

- strongly agree
- agree
- neutral
- disagree
- strongly disagree
- I don't know

I think I solved this task securely.

- strongly agree
- agree
- neutral
- disagree
- strongly disagree
- I don't know

Did you use the PyCrypto API documentation to solve this task?

- Yes
- No

If Yes: Please rate your agreement to the following statements:

The documentation was helpful in solving this task.

- strongly agree
- agree
- neutral
- disagree
- strongly disagree
- I don't know

Which parts of the documentation did you use?

Did you see any security warnings while working on this task?

- Yes
- No

If Yes: Please rate your agreement to the following statements:

The security warnings displayed in the console helped to solve this task.

- strongly agree
- agree
- neutral
- disagree
- strongly disagree
- I don't know

Please explain why the security warnings were helpful or rather unhelpful.

- freetext answer

D.2 General questions about previous experience

Have you used the PyCrypto library before? For example, maybe you worked on a project that used PyCrypto, but someone else wrote that portion of the code.

- I have used PyCrypto before
- I have seen PyCrypto used but have not used it myself
- No, neither
- I don't know

Have you used or seen code for tasks similar to the tasks given in the study before? For example, maybe you worked on a project that included a similar task, but someone else wrote that portion of the code.

- I have written similar code
- I have seen similar code but have not written it myself
- No, neither
- I don't know

D.3 Usability perception

Please rate your agreement to the following questions on a scale from 'strongly agree' to 'strongly disagree.' (strongly agree; agree; neutral; disagree; strongly disagree; does not apply)

- I had to understand how most of the assigned library works in order to complete the tasks.
- It would be easy and require only small changes to change parameters or configuration later without breaking my code.
- After doing these tasks, I think I have a good understanding of the assigned library overall.
- I only had to read a little of the documentation for the assigned library to understand the concepts that I needed for these tasks.
- The names of classes and methods in the assigned library corresponded well to the functions they provided.
- It was straightforward and easy to implement the given tasks using the assigned library.
- When I accessed the assigned library documentation, it was easy to find useful help.
- In the documentation, I found helpful explanations
- In the documentation, I found helpful code examples.
- When I made a mistake, I got a meaningful error message/exception.
- Using the information from the error message/exception, it was easy to fix my mistake.

D.4 Message design assessment

Please rate your agreement to the following statements concerning this console warning:

[Example security advice figure]

How helpful would you rate... (not helpful at all; somewhat unhelpful; neutral; somewhat helpful; very helpful; I don't know)

- ...the risk explanation?
- ...the recommendation for secure action?
- ...the given code example?
- ...the described option for insecure action?
- ...the given background information?
- ...the structure of this security advice?
- ...the amount of information in the message?
- ...the appearance of this kind of messages when using the PyCrypto Library?

What aspects of the warning could be improved, in your opinion?

- free text

D.5 Development Environment

Please tell us some details about your usual Python software development tool chain.

Which console do you use?

- free text

Which text editor do you use?

- free text

What IDE do you use?

- free text

Do you use other tools for software development?

- free text

D.6 Demographic Questions

What type(s) of software do you develop?

- Web Applications
- Mobile Applications
- Desktop Applications
- Embedded Applications
- Enterprise Applications
- Other:

How many years of development experience do you have?

- Number field 0-100
- Prefer not to answer

How many years have you been programming in Python?

- Number field 0-100
- Prefer not to answer

What is your current occupation?

- Freelance developer
- Industrial developer
- Industrial researcher
- Academic researcher
- Graduate student
- Undergraduate student
- Prefer not to answer
- Other:

What is your gender?

- Female
- Male
- Prefer not to answer
- Other:

What country do you live in?

- Please choose... (Dropdown)

How old are you?

- Free text for number of years
- Prefer not to answer