Write, Read, or Fix? Exploring Alternative Methods for Secure Development Studies

Kelsey R. Fulton* (Colorado School of Mines), Joseph Lewis (University of Maryland), Nathan Malkin* (New Jersey Institute of Technology), and Michelle L. Mazurek (University of Maryland)

* - Work performed while at University of Maryland

Developers struggle with security

- NVD reported 28,831 vulnerabilities in 2023 [1]
  - 25,081 in 2022
- Often caused by developers:
  - Making mistakes
  - Misunderstanding security
- Addressing this requires understanding it
  - Studying developers as they build code

[1] - https://nvd.nist.gov

How do we study developers?

- Interview studies
- Surveys
- Code writing studies

Challenges with code writing tasks

- Code writing is time consuming
- Tasks are difficult to scope
- It is hard to effectively design studies
- Developers are hard to recruit and retain:
  - Hard to find
  - Participate outside of work hours
  - Participate for less money than they are paid at work

↑ **Dropouts**

↓ **Samples**

Are there alternative approaches that will yield similar results while reducing stress?

Using code review

- In 2021, Danilova et al. explored the use of code review [1]
- Participants wrote code reviews about snippets from a prior study
- Code review is potentially useful in place of long programming tasks
  - Able to identify issue developers faced

Expand on this by directly comparing a ***Read*** and ***Fix*** condition

[1] - Danilova et al. Code Reviewing as Methodology for Online Security Studies with Developers – A Case Study with Freelancers on Password Storage. In SOUPS 2021

Write, *Read*, and *Fix*

Write

- ***Write*** code to complete spec
- Provided tests

Read

- ***Read*** completed code
- Identify any bugs/vulns
- Describe fixes
- Do not actually alter code
- Cannot run code

Fix

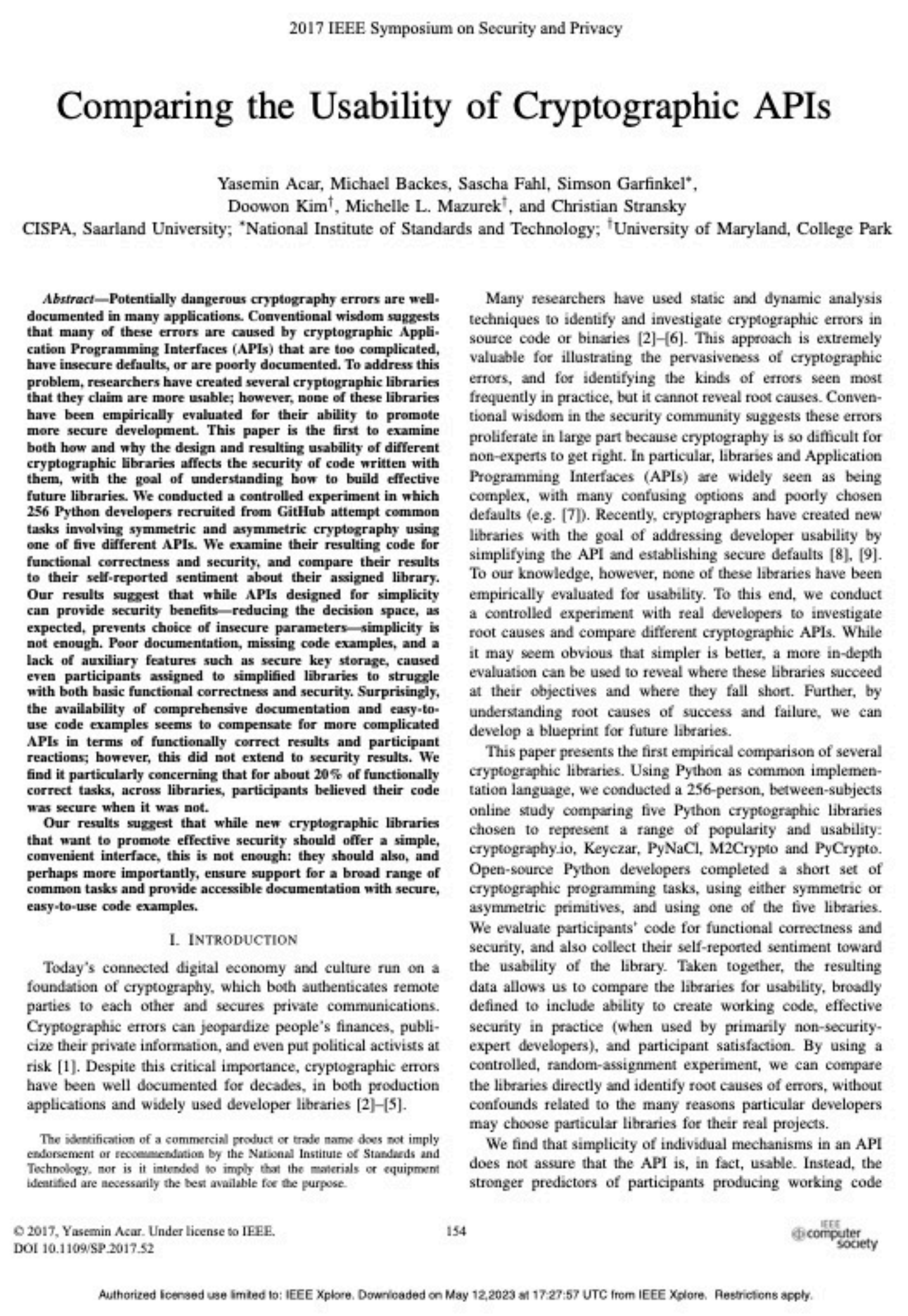
- Read completed code
- Identify any bugs/vulns
- ***Fix*** bugs/vulns
- Provided tests

Research questions

- Do the ***Read*** and ***Fix*** conditions provide the same results as ***Write***?
  - Functionality and security
- Do participants in ***Read*** and ***Fix*** experience fewer negative effects?
  - Drop-out rate
  - Frustration
  - Time spent

Study design

- Partially replicated prior study [1]
  - Participants completed self-contained, short ***Write*** tasks
  - Utilized 1 of 5 Python libraries
  - Tasks were focused on (a)symmetric encryption
- Allowed us to compare our ***Write*** results
- While allowing us to compare ***Write***, ***Read***, ***Fix***



2017 IEEE Symposium on Security and Privacy

Comparing the Usability of Cryptographic APIs

Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel*,
Doowon Kim†, Michelle L. Mazurek†, and Christian Stransky
CISPA, Saarland University; *National Institute of Standards and Technology; †University of Maryland, College Park

***Abstract*—Potentially dangerous cryptography errors are well-documented in many applications. Conventional wisdom suggests that many of these errors are caused by cryptographic Application Programming Interfaces (APIs) that are too complicated, have insecure defaults, or are poorly documented. To address this problem, researchers have created several cryptographic libraries that they claim are more usable; however, none of these libraries have been empirically evaluated for their ability to promote more secure development. This paper is the first to examine both how and why the design and resulting usability of different cryptographic libraries affects the security of code written with them, with the goal of understanding how to build effective future libraries. We conducted a controlled experiment in which 256 Python developers recruited from GitHub attempt common tasks involving symmetric and asymmetric cryptography using one of five different APIs. We examine their resulting code for functional correctness and security, and compare their results to their self-reported sentiment about their assigned library. Our results suggest that while APIs designed for simplicity can provide security benefits—reducing the decision space, as expected, prevents choice of insecure parameters—simplicity is not enough. Poor documentation, missing code examples, and a lack of auxiliary features such as secure key storage, caused even participants assigned to simplified libraries to struggle with both basic functional correctness and security. Surprisingly, the availability of comprehensive documentation and easy-to-use code examples seems to compensate for more complicated APIs in terms of functionally correct results and participant reactions; however, this did not extend to security results. We find it particularly concerning that for about 20% of functionally correct tasks, across libraries, participants believed their code was secure when it was not.**

Our results suggest that while new cryptographic libraries that want to promote effective security should offer a simple, convenient interface, this is not enough: they should also, and perhaps more importantly, ensure support for a broad range of common tasks and provide accessible documentation with secure, easy-to-use code examples.

I. Introduction

Today's connected digital economy and culture run on a foundation of cryptography, which both authenticates remote parties to each other and secures private communications. Cryptographic errors can jeopardize people's finances, publicize their private information, and even put political activists at risk [1]. Despite this critical importance, cryptographic errors have been well documented for decades, in both production applications and widely used developer libraries [2]–[5].

Many researchers have used static and dynamic analysis techniques to identify and investigate cryptographic errors in source code or binaries [2]–[6]. This approach is extremely valuable for illustrating the pervasiveness of cryptographic errors, and for identifying the kinds of errors seen most frequently in practice, but it cannot reveal root causes. Conventional wisdom in the security community suggests these errors proliferate in large part because cryptography is so difficult for non-experts to get right. In particular, libraries and Application Programming Interfaces (APIs) are widely seen as being complex, with many confusing options and poorly chosen defaults (e.g. [7]). Recently, cryptographers have created new libraries with the goal of addressing developer usability by simplifying the API and establishing secure defaults [8], [9]. To our knowledge, however, none of these libraries have been empirically evaluated for usability. To this end, we conduct a controlled experiment with real developers to investigate root causes and compare different cryptographic APIs. While it may seem obvious that simpler is better, a more in-depth evaluation can be used to reveal where these libraries succeed at their objectives and where they fall short. Further, by understanding root causes of success and failure, we can develop a blueprint for future libraries.

This paper presents the first empirical comparison of several cryptographic libraries. Using Python as common implementation language, we conducted a 256-person, between-subjects online study comparing five Python cryptographic libraries chosen to represent a range of popularity and usability: cryptography.io, Keyczar, PyNaCl, M2Crypto and PyCrypto. Open-source Python developers completed a short set of cryptographic programming tasks, using either symmetric or asymmetric primitives, and using one of the five libraries. We evaluate participants' code for functional correctness and security, and also collect their self-reported sentiment toward the usability of the library. Taken together, the resulting data allows us to compare the libraries for usability, broadly defined to include ability to create working code, effective security in practice (when used by primarily non-security-expert developers), and participant satisfaction. By using a controlled, random-assignment experiment, we can compare the libraries directly and identify root causes of errors, without confounds related to the many reasons particular developers may choose particular libraries for their real projects.

We find that simplicity of individual mechanisms in an API does not assure that the API is, in fact, usable. Instead, the stronger predictors of participants producing working code

The identification of a commercial product or trade name does not imply endorsement or recommendation by the National Institute of Standards and Technology, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

DOI 10.1109/SP.2017.52

[1] - Acar et al. Comparing the Usability of Cryptographic APIs. In S&P 2017.

Study flow

Consent → Condition assignment → Tasks → Final survey

Condition assignment

- Write
- Read
- Fix

\+

- PyCrypto
- Crypto.io

Tasks

- Encrypt/ decrypt data
- Generate and store a key

Final survey

- Performance on tasks
- Frustration and fun
- Background

Data analysis

- Manually reviewed code for bugs/vulnerabilities
  - Leveraging the vulns/bugs from [1] and our known list
- To compare results among conditions:
  - Ran various regressions for impact of library and condition

Recruitment and participants

- Recruited 112 valid participants from Upwork and CS student mailing lists
    - ***Write***: 35 participants
    - ***Read***: 37 participants
    - ***Fix***: 40 participants
- Our participants were fairly experienced, but not in security:
    - Avg 6.8 years programming experience
    - Avg 4 years Python experience
    - Avg 1.2 years security experience

Research questions

- Do the ***Read*** and ***Fix*** conditions provide the same results as ***Write***?
  - Functionality and security
- Do participants in ***Read*** and ***Fix*** experience fewer negative effects?
  - Drop-out rate
  - Frustration
  - Time spent

Takeaway #1: Use *Write* to measure the efficacy of code writing tools

- ***Write*** was able to reveal important differences between crypto APIs
    - Specifically, in the security of solutions participants produced
- Also revealed documentation issues
- These differences were substantially less visible in ***Read*** and ***Fix***
- Security APIs are designed to prevent developers from making security mistakes
    - Rather than identifying or fixing them

Takeaway #2: Use *Read* to measure developers' knowledge

- ***Read*** participants pay close attention to the code
  - Identified fewer, but more diverse bugs than ***Fix*** participants
  - Identified more vulns than ***Fix***, even identifying 8 out-of-scope vulns
- Making ***Read*** useful for identifying overall security awareness and knowledge

Takeaway #3: Use *Fix* to measure quick fixes

- ***Fix*** participants heavily focused on passing provided tests
    - All of our ***Fix*** participants started by running the code
    - Causing them to miss bugs and vulnerabilities
- ***Fix*** may be useful for identifying vulns and bugs developers can quickly find
    - Offer lower bound on their abilities

Takeaway #4: Use *Read* and *Fix* to minimize time, frustration

- ***Read*** and ***Fix*** participants spent less time than ***Write*** participants
    - And had fewer dropouts
- ***Read*** and ***Fix*** participants actually enjoyed their tasks
- ***Read*** and ***Fix*** may offer an appropriate option when recruitment is a concern

- We explored two alternatives (***Read*** and ***Fix***) to code writing studies (***Write***)
- ***Write*** more clearly identifies security differences between security APIs
- ***Read*** participants paid close attention to the code
- ***Fix*** participants focused on passing tests, missing key vulns
- Participants felt fewer negative effects (frustration, time spent) in ***Read*** and ***Fix***
  - Possibly helping in retention and recruitment

Questions?
kelsey.fulton@mines.edu

NSF
SPAM Security & People At Mines
School of Mines 1874 Colorado
SPUR
NJIT New Jersey Institute of Technology
SP2 SECURITY. PRIVACY. PEOPLE