"I'm pretty expert and I still screw it up": Qualitative Insights into Experiences and Challenges of Designing and Implementing Cryptographic Library APIs

Juliane Schmüser, Philip Klostermeyer, Kay Friedrich, and Sascha Fahl
CISPA Helmholtz Center for Information Security, Germany
{juliane.schmueser,philip.klostermeyer,kay.friedrich,fahl}@cispa.de

***Abstract*—Cryptographic libraries are a vital security component of software systems, yet their misuse has caused several incidents. Prior work has established that misuse of cryptographic libraries is common, and developers struggle to use their APIs correctly. However, it is currently unknown how the design and implementation decisions that shape cryptographic library APIs are made. To investigate these decisions and associated challenges in the design and implementation process of cryptographic library APIs, we conducted 21 semi-structured interviews with experienced developers of cryptographic libraries and used thematic analysis to identify overarching topics and challenges they encountered. We find that design decisions span a spectrum of abstraction levels and are heavily influenced by cryptographic standards, other libraries, legacy code, and developers' intuitions. Developers are challenged by the optimal level of abstraction for cryptographic APIs to balance security, usability, and flexibility. They lack systematic knowledge on defining usability and achieving such balance. Consequently, developers rely on usability self-tests, personal experiences, and opinions. Based on our findings, we make detailed recommendations to tailor future research toward better empirically validated support of cryptographic library API design and implementation decisions. Further, we advocate for integrating research-based usability guidance into cryptographic standardization to foster community discussion early on and better support secure, usable, and flexible cryptographic library APIs.**

1. Introduction

Whether as low-level components in operating systems, as external dependencies in application development performing cryptographic tasks, or for securing network communication, cryptographic libraries are vital components of modern soft- and hardware. While being fundamental security providers for the software products that use them, they can also introduce vulnerabilities and create a false sense of security when misused. In 2021, cryptographic failures rose to rank two of the OWASP top ten security risks to web applications [1]. Vulnerable implementations, such as the recently patched email-related buffer overflows in OpenSSL [2], can pose severe risks to client software. Due to rapidly increasing user numbers during the 2020 pandemic, numerous vulnerabilities were identified in the video conference software Zoom, which offered many attack opportunities for hackers based on encryption with a single AES-128 key in Electronic Code Book (ECB) mode [3]. Vulnerabilities due to the incorrect generation of keys continue to occur [4]. Prior research has shown that developers often struggle to use cryptographic libraries securely. Empirical analyses found that misuse of cryptographic libraries, such as using short key lengths, insecure modes of encryption like ECB, or deprecated algorithms like SHA-1, is widespread across languages and development communities [5]–[7].

The usability of cryptographic libraries has been examined as a significant factor in developers' success with implementing secure software. Identified areas for improvement include simplicity, level of abstraction, secure defaults, extensive and comprehensible documentation with secure usage examples, and the range of covered functionality and use cases [8]. However, it is so far unknown how such design and implementation choices are made in the development process of cryptographic libraries, precisely their consumer-facing library elements, the application programming interfaces (APIs). In addition, it is unclear if and how recent research insights are considered. To improve software security, we argue that it is essential to investigate these processes and their unique challenges.

In this work, we aim to shed light on the design and implementation decision processes of cryptographic library APIs affecting their security and usability. We explore libraries' behind-the-scene processes and the guidelines and standards they follow. We are especially interested in processes often not directly visible from the libraries' repository data, e.g., decisions on supported primitives and level of abstraction, challenges encountered by developers, and guiding factors of design decisions. We conducted 21 in-depth, semi-structured interviews with contributors and maintainers of a diverse set of cryptographic libraries based on the following research questions:

RQ1. *"How are design and implementation decisions for cryptographic library APIs made?"* Cryptographic libraries include structures and decision processes that are not inherently visible to the public. We are interested in why and how API design decisions are made, especially in the context of the security and usability of cryptographic libraries.

RQ2. *"What challenges do cryptographic API developers face in their design and decision processes?"* We aim to

identify challenges in the design and decision processes by capturing the developers' perspectives.

RQ3. *"How can cryptographic library designers and implementers be better supported to improve library security and usability?"* Cryptographic libraries face unique security and usability challenges for developers, often complicating their use. We seek to identify opportunities to better support cryptographic library designers and implementers with creating secure, usable APIs to improve overall software security.

In this paper, we make the following contributions:

Insights from Experienced Cryptographic API Developers. We report insights from 21 semi-structured interviews with experienced developers of cryptographic library APIs, including their opinions on API design and strategies for decision processes. We find that levels of abstraction varied across libraries, and decisions were influenced by standards, other libraries, legacy code, and developers' intuitions.

Key Challenges of Cryptographic API Design. We identify critical challenges in the design of cryptographic library APIs, such as limited resources for usability engineering, difficulty determining usability, balancing usability, security, and flexibility, and a lack of specific, empirically validated guidance in research and standards.

Recommendations for Usability Research and API Design Guidance in Cryptographic Standards. Based on our findings, we identify open research questions and give detailed recommendations for future work on usable cryptographic APIs. We argue that cryptographic standardization should include API design and usability considerations for multiple levels of misuse resistance and flexibility to help cryptographic library developers make informed decisions.

2. Related Work

We discuss previous work in three key areas: API design principles and processes, the usability of cryptographic libraries, and the impact of misusing cryptographic libraries on software security. We put our work into context and illustrate its novelty.

API Design Principles and Processes. Developers make many design decisions to create an API [9]. Previous research has emphasized the importance of good API design [10], and developed various guidelines for the creation of usable APIs. Henning offers general API design recommendations [11], while Tello-Rodríguez *et al.* focus on guidelines for web APIs [12]. Myers *et al.* argue for the use of human-centered design approaches in API design [13]. Stylos *et al.* describe their work on a user-centered redesign to improve the usability of an existing API [14]. Macvean *et al.* discusses part of the web API design process at Google, which includes an expert review of a proposed API design [15]. Stylos *et al.* suggest user studies to validate recommendations for API design [16]. Further, Weber *et al.* investigate the effect of immutability on API usability and security, and call for further empirical evaluation of API design guidelines [17]. Complementing the work on design guidelines, studies aim to understand developers' current design decisions, processes, and issues in API development [18], [19]. In interviews with API developers, Zhang *et al.* found a need for better tool support, as well as possibilities to collect feedback from developers using the API [20]. For tooling to assist API developers in implementing the recommended design guidelines, Santos *et al.* implemented Dacite, a tool developed to help API designers improve API discoverability and documentation through design annotations [21]. In light of the positive influence of good documentation on API usability [22], the Jadeite tool provides documentation in the style of Javadoc that facilitates API learning and navigation through the documentation [23]. Myers *et al.* state that API usability should not only be considered during the design phase but also the evaluation of the API [13]. Researchers have proposed HCI methods to improve the evaluation of API usability [24], as well as metrics to automatically measure API usability [25]. Oliveira *et al.* suggest developer-centric piloting and testing of libraries [26], and Ahasanuzzaman *et al.* present a classification of online forum posts as a way to collect user feedback [27]. Related work provides an understanding of and guidelines for the general API design process. Our work adds qualitative insights into the specific case of cryptographic APIs.

Usability of Cryptographic Libraries. The identification of misuse by developers as a key issue of cryptographic libraries led to research into their usability. Patnaik *et al.* conducted a review of 45 years of security API usability, discovering that there has always been a small focus on documentation, code quality assessment, and organizational practices when designing security APIs [28]. This aligns with Votipka *et al.* calling for simpler APIs and more precise documentation including multiple use and edge cases [29]. Green *et al.* suggested 10 principles for usable cryptographic libraries in 2016 [30]. Furthermore, Gorski *et al.* conducted extensive literature research on security APIs and identified eleven crucial usability attributes that can be used to evaluate security APIs in general [31]. Patnaik *et al.* analyzed issues on Stack Overflow to identify violations of these principles, finding a high correlation to struggles of developers [32], and Wijayarathna *et al.* used them to design a cryptographic library evaluation questionnaire [33]. Based on a comparison study with developers including the questionnaire, Acar *et al.* recommend simplicity, high-level abstraction, safe defaults, better documentation, code examples, and developer warnings as measures to improve usability [8]. These recommendations are echoed by other API analyses [34], [35], an issue analysis and survey [36], and developer studies [37], [38]. Based on the identified issues and recommendations there have been efforts to improve the usability of cryptographic libraries by designing and introducing warnings [39], [40], external code examples [41], semantic interfaces [42], [43], and usability focused library design and development [44], [45]. We expand on the above prior work by leveraging in-depth interviews with experienced cryptographic library designers and implementers to inves-

tigate their design and implementation processes, practices, and challenges and better understand cryptographic library security and usability.

Impact of the Misuse of Cryptographic Libraries on Software Security. Multiple studies illustrated the negative impact on software security when misusing cryptographic libraries. An analysis of faulty cryptography implementation in software attributed approximately 83% of root causes to misuse of cryptographic libraries [46]. Further studies found incorrect usage of libraries in 88% of 11,748 analyzed Android applications [6], 65% of 98 scanned iOS applications [47], 78% of open source Java projects [5], 52% of open source Python projects [7], and as a common reason for vulnerable SSL connections in critical software [48]. Gao *et al.* found that updates to commonly used Android apps often do not fix cryptographic library misuses and, in many cases, even reintroduce security issues that were previously fixed [49]. Tools for the automated detection and repair of cryptographic library misuse have since evolved [50], [51] and performed well on benchmarks [52]. Paletov *et al.* infer safe API usage rules from code changes on GitHub under the assumption that these fix issues rather than introduce them and recommend these rules to developers as security checkers [53]. However, a recent evaluation of such tools identified several flaws that impede the discovery of misuse in practice [54]. Another study found a significant discrepancy between the capabilities of various tools for misuse detection and developers' expectations of these tools [55]. Through our interviews, we seek to gain insights to help overcome challenges of cryptographic API design and reduce cryptographic library misuse.

3. Methodology

In this section, we illustrate our methodology for interviewing 21 experienced designers and implementers of cryptographic libraries. We describe our analysis approach, highlight ethical considerations, and discuss limitations.

3.1. Recruitment and Demographics

Developers of cryptographic library APIs are a small expert population that is challenging to recruit. We aimed for a diverse sample that includes foundational libraries that support the ecosystems of different programming languages as we consider their influence and thus, their challenges to be of particular importance. To achieve such a sample, we employed various recruitment strategies, including recruitment from our professional networks, snowball sampling [56], invitations to public email addresses of developers and maintainers of well-known cryptographic libraries, and posts on mailing lists of cryptographic libraries and the IETF cfrg mailing list [57]. Of these strategies, snowball sampling and posts to mailing lists were most successful, recruiting 14 of our 21 participants. With 21 interviews, we had reached thematic saturation [58] and stopped recruitment. While our approach led to a convenience sample, we are confident that we have interviewed a diverse and meaningful group of developers of cryptographic library APIs. Our sample includes open-source (17) and proprietary (4) cryptographic library developers, and many of our participants have worked on several cryptographic libraries over the years. Two participants maintained the same library, all other projects were unique. Prior to scheduling an interview, we asked participants to fill out a short demographics survey, which we provide in our replication package (cf. Availability). To protect our participants' anonymity, we only provide aggregated demographics in Table 1. For context, we also provide some project information in Table 2.

3.2. Interview Procedure

We conducted and recorded the 21 interviews between July 2023 and February 2024 online using an online conference tool of the participants' choice. We developed the interview guide based on our research questions, and evaluated it in a pilot interview. We made adaptations based on the participants' feedback and excluded the pilot from our dataset. Figure 1 and below illustrate the final interview structure. Appendix A contains the interview questions, and we provide the full interview guide in our replication package (cf. Availability). We split our interview guide into an introduction, seven question sections with open-ended questions and associated follow-up questions, and a debriefing. The interviews took an average of 62 minutes.

Introduction. We thanked the participants for their time and explained the interview structure and the procedure for our analysis. We gave brief information about the involved institutions and our research and obtained consent for recording.

Personal Background. Starting with introductory questions, we aimed to learn about the participants' backgrounds and experiences with working on cryptographic libraries.

Design Decisions. We asked the participants about the current design of their library and its API. That includes its purpose, differences from other cryptographic libraries, their algorithms, and functions. Moreover, we asked the participants how they designed functions, chose defaults, wrote documentation, and addressed usability considerations. Finally, we asked about the potential challenges of API design decisions our participants had experienced.

Project Processes and Tooling. We asked about the participants' organizational structure and decision-making process to get an impression of their current project's development processes and workflow, tooling, and knowledge transfer.

Initial Design Process. Further, we asked participants about the initial design decisions of the cryptographic library and its API. We encouraged them to explain the early implementation process and inquired about the information sources they considered. Lastly, we requested participants to discuss any challenges they faced in the initial design phase.

Updates. To gain insights into update procedures, we requested the participants to describe update processes and reasons for updates. Moreover, we asked participants if they stay informed on the latest cryptographic advancements and

TABLE 1. AGGREGATED DEMOGRAPHICS OF OUR PARTICIPANTS

Number of Participants	**21**
Years of Experience in the Design and Implementation of Cryptographic Libraries:	
Mean	11.97
Median	10
std	7.98
Min	2
Max	32
Crypto Engagement:	
Main Occupation	12
Secondary Occupation	6
Something I do in my free time	3
Cryptography Learning Resources*:	
University courses	11
Textbooks	14
Research papers	20
Tutorial books	4
Online courses (e.g., coursera)	10
Informal community discussion (Discord, Slack)	10
Forum-based platforms (Reddit)	6
QA platforms (Kaggle, Stackoverflow)	6
Company-provided trainings	1
Conferences	16
Cryptographic Standards	18
Other: Reading/Writing Cryptographic Code	5
Education:	
Secondary school (e.g. American high school, German Gymnasium, Spanish or French Baccalaureate, British A-Levels)	2
Some college/university study without earning a degree	1
Bachelor's degree (B.A., B.S., B.Eng., etc.)	3
Master's degree (M.A., M.S., M.Eng., MBA, etc.)	9
Other doctoral degrees (Ph.D., Ed.D., etc.)	5
Prefer not to disclose	1
Field of Degree*†:	
Computer Science	7
Cyber/Computer Security	3
Cryptography	3
Mathematics	3
Electrical and Computer Engineering	2
Other	2
Employment Status:	
Employed full-time	13
Independent contractor, freelancer, or self-employed	7
Employed part-time	1
Student	1
Programming Language*†:	
C/C++	11
Rust	4
Golang	3
Python	2
Jasmin	1
WebAssembly	1
Zig	1
Java	1
Assembly	1
Location:	
North America	11
Europe	9
Oceania	1

* multiple answers allowed † open-ended answers

TABLE 2. PARTICIPANTS' PRIMARY PROJECTS AND ROLES.

Alias[Role]	**Library Usage**	**Library Access**
P1[1]	General Purpose Library	Open Source
P2[1]	TLS for Embedded Devices	Open Source
P3[2]	General Purpose Library	Open Source◇
P4[1]	Language Default Library	Open Source
P5[2]	Wrapper for OS Crypto Lib	Company Internal
P6[1]	Language Default Library	Open Source
P7[1]	Micro Library	Open Source
P8[1]	General Purpose Library	Commercial Product▽
P9[1]	General Purpose Library	Open Source
P10[1]	API for New Standard	Open Source
P11[1]	General Purpose Library	Open Source
P12[1]	Crypto for Security Software	Open Source
P13[1]	Micro Library	Open Source
P14[1]	TLS for Embedded Devices	Commercial Product▽
P15[2]	General Purpose Library	Open Source◇
P16[1]	Research Library	Open Source
P17[1]	Web Signature Library	Open Source
P18[2]	Crypto for Firmware	Company Internal
P19[1]	Research Library	Open Source
P20[2]	General Purpose Library	Open Source◇
P21[2]	General Purpose Library	Open Source◇

[Role] *1: Maintainer, 2: Member of Development Team*
▽ Open Source License Available ◇ Developed in Company

new API design developments. Finally, we inquired about challenges they faced regarding update decisions.

User Interactions and Feedback. This section focused on the communication and interaction between the designers, implementers, and users of cryptographic libraries. We asked about communication and feedback channels, and the impact of user feedback on API design and implementation.

Challenges and Wishes for the Future. In the final section, we asked about the biggest challenges when designing, developing, and implementing APIs for cryptographic libraries. We also encouraged the participants to mention ideas and wishes for improving future cryptographic library design and implementation.

Debriefing. At the end of the interviews, we asked the participants if they had any further comments. We stopped recording and offered the participants a compensation of $60 and a preprint of the paper once we finished the study and before publication. Lastly, we thanked all participants for their valuable time and discussed final comments and recommendations off the record.

3.3. Qualitative Analysis

We transcribed all interview recordings using the GDPR-compliant manual *Amberscript* transcription service [59]. We then conducted thematic analysis following the process outlined by Braun *et al.* [60]. We used a semi-open coding approach, developing an initial set of codes based on our research, interview questions, and impressions from the interviews. Additionally, we allowed new codes to emerge from the data and iteratively refined our codebook during analysis. The first author coded all interviews, and each was independently coded by at least one of two more coders.

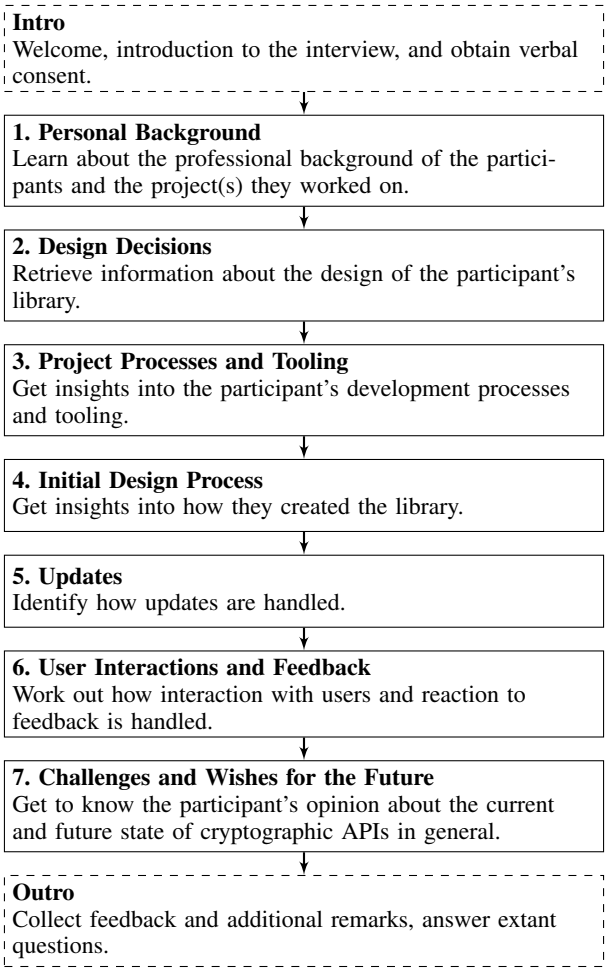


Figure 1. Overview of the sections of our interview guide.

Coders met to discuss and resolve any disagreements. Thus, we refrain from reporting inter-rater reliability [61]–[63]. All coders met and used affinity diagramming [64] on the final set of codes to identify themes. Our final codebook is available in our replication package (cf. Availability).

3.4. Ethical Considerations and Data Protection

Our institution's Ethical Review Board approved this interview study. Throughout the study design and execution, we prioritized ethical practices described in the Menlo report [65], ensuring voluntary participation. During the recruitment process, we provided potential participants with details about the purpose of the study. Before each interview, we informed the participants of the structure and estimated duration of the interviews and reassured them that their reported practices were not being judged. Instead, the focus was solely on their opinions and experiences. We obtained informed consent from all participants to record audio and emphasized that they could refuse to answer any questions or stop the interview at any time. To compensate participants for their time, we offered $60.

We complied with national data protection laws and the general data protection regulation (GDPR) in our entire process. We stored the collected data encrypted in an internally hosted cloud and commissioned transcripts with a service based in the EU and compliant with the GDPR.

3.5. Limitations

Some limitations and biases typical for interview studies apply to our work, such as self-report bias, under- or over-reporting bias, recall bias, and social desirability bias. We attempted to mitigate these by carefully probing for elaborate answers, and assuring participants that we do not judge their practices. Our study examines an expert population that is small and difficult to recruit. Hence, most of our participants were highly educated and had many years of experience developing cryptographic libraries. In addition, our sample may be influenced by sampling biases resulting from snowball sampling and self-selection. We mentioned API design and usability during our recruitment process, thus the cryptographic library designers and developers in our sample may be especially interested or knowledgeable in usability and API design. Therefore, our results do not necessarily generalize to the overall population of cryptographic library developers. Instead, our analysis is qualitative in nature, and the insights we provide into cryptographic library APIs' design and development processes must be interpreted in context.

4. Results

We report the results of 21 semi-structured interviews grouped by the themes we identified below. The analysis and all results are qualitative in nature and should not be interpreted as quantitative or representative findings. To avoid presenting numbers while preserving a general sense of prevalence and weight, we use the quantifiers depicted in Figure 2 in our reporting, following the example of prior interview studies in our field [62], [66]–[69].

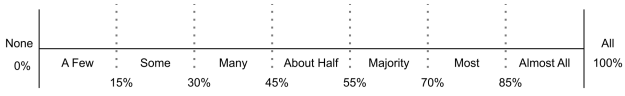


Figure 2. Overview of quantifiers and corresponding shares of the 21 participants as used in result reporting.

We report on learning resources, API designs, influences and decision factors, and usability challenges including evaluation and tensions with other design goals. Figure 3 depicts an overview of how these topics interact, influence each other, and determine cryptographic API design.

4.1. Acquiring Knowledge on Cryptography and Library Design

Our data initially pinpoints two areas from which our participants drew their cryptographic experiences. Firstly,

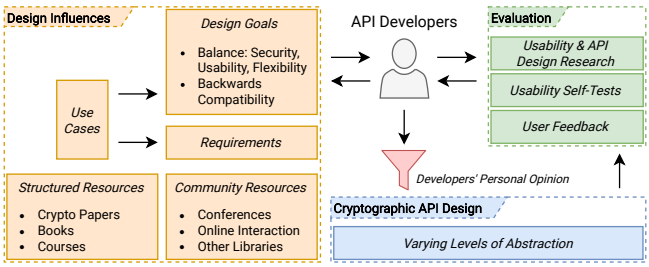


Figure 3. Overview of the connection between design influences, API design, and usability evaluation.

we identified structured learning resources, and secondly, our participants also fostered much of their experience through access to various community resources. About half of our participants learned cryptography through learning by doing or self-teaching, e.g., by writing cryptographic code themselves and by *"playing with it and learning little bits as I went along"* (P3), drawing on both structured and community resources.

Structured Resources. Almost all participants stated they used or had used some form of structured resource influencing their experience with cryptography. A few participants had been exposed to basic maths and sometimes cryptography in earlier school years, e.g., in high school, and had developed their interest from there. Most participants, however, reported that their most influential exposure to cryptography was in some form of higher education institution, such as maths, computer science, or explicit cryptography courses at universities or colleges. In addition, some participants reported that they actively drew some of their knowledge from academic research papers. Another type of structured learning we encountered was books. Moreover, some participants attended structured online courses on cryptography.

Community Resources. Further, most described community resources influencing their cryptographic knowledge. The majority of our participants had gained cryptographic experience in further steps of learning cryptography on their job, both in a corporate and a general project context. In relation, reading and working with source code of other cryptographic libraries has been mentioned by some participants, *"trying to understand how everything was working."* (P10) One of these also mentioned the documentation of projects as a source of knowledge. Furthermore, some participants mentioned social interaction spaces where they exchanged ideas and shared knowledge. A few participants mentioned scientific conferences or social collaboration tools like Slack or Discord.

> **Cryptography Knowledge Stems from Structured and Community Resources.** Our participants did not rely upon a uniform resource to gain cryptographic knowledge. Both structured, e.g., books and courses, and community resources, e.g., documentation and source code of other libraries, contributed to their education on the subject.

4.2. Reported API Designs

Discussed API design aspects include the libraries' primary purpose, design goals, primitive selection, level of abstraction in function design, and additional information sources such as warning or error messages and the libraries' documentation. Our participants reported on diverse libraries, and API designs covered a wide spectrum across these design aspects.

Library Purpose. The primary purpose of our participants' libraries played a major role in determining design goals and influenced many design decisions. Our interviewees developed libraries for a wide variety of purposes. Some participants worked on libraries with specific purposes, such as optimized implementations for embedded systems, supporting cryptography research, or implementing one specific primitive, i.e., providing a microlibrary. The majority, however, considered their libraries general purpose, seeking to provide a range of cryptographic functionality that supports many use cases. Some considered TLS connections to be their main use case, and some aimed to be the primary cryptographic library for their specific programming language. Other participants noted that the programming language was a design choice, and some wished to change it, typically to increase memory safety. Next to standalone libraries, many participants reported that their libraries wrapped an underlying cryptographic library, sometimes to make it available in a different language, or to provide a more usable interface. In contrast, a few interviewees also intended their library as a foundation for others to write wrappers in other languages or to use it in higher-level cryptography like protocols: *"We assume that the typical consumer [...] would be somebody who is implementing a higher-level crypto library, and they will know which algorithm they need."* (P19) This quote also demonstrates how the library purpose connects to the intended users of a library. Most of our participants generally considered their users' expertise when making API design decisions, and only a few explicitly stated that they do not consider it. Some participants developed their libraries for specific user groups, and a few mentioned challenges when their library was used by people not part of this designated group. One developer of a research library described how it was used in commercial products despite not being intended for use in production systems, and another participant elaborated how they felt about users ignoring the library's security disclaimers: *"I tend to feel it is buyer beware, I told you this is unsafe and you used it anyway. It's on you."* (P17)

Design Goals. Our participants derived a number of design goals from their libraries' purpose, intended use cases, and users. Almost all participants considered good usability an essential design goal for their library. They most frequently named the reduction of misuse potential as a motive, often in conjunction with perceived low cryptography expertise of users:

> *"It should be idiot proof, meaning it should be as difficult as possible to implement insecure encryption using the library. [...] It should tell you what you've done wrong if you make a mistake."* — P8

About half of the participants wanted their library to work out of the box to ease adoption and prevent user frustration. In addition to low expertise, they considered that users may have little time and resources for cryptography, one stating how people *"want to copy and paste stuff, especially since users of crypto are not people who are interested in crypto most of the time."* (P10) Security was another major design goal for which participants formulated several subgoals. Many of our interviewees considered the memory safety of their libraries, either through implicit guarantees of the programming language or by designing their API to control the necessary memory allocations tightly:

> *"There can be very little risk of causing a buffer overflow or anything because these have static sizes. If you look at the [LIBRARY'S] APIs, they have the same key, which is over 32 bytes. The nonce is always 24 bytes and so on."* — P13

In addition, most sought to reduce their library's and API's complexity to help with misuse resistance and several security goals, including minimizing the attack surface and reducing the maintenance burden, as well as supporting auditability: *"My day job is code auditing. I've been a security auditor for the last 10 years. For me, [...] the code auditability is extremely important and also a limited attack surface."* (P13) To what extend they considered it possible to reduce complexity depended largely on the library's purpose, as it stands in direct conflict with some general purpose libraries' goal to offer as many primitives as possible. Still, a few participants prioritized complexity reduction:

> *"I think we have an unusual allergy for complexity. We will gladly take 80% less code if it has 20% lower performance or implements a subset of a spec if that's what 90% of our users will ever use. That will expose less attack surface, and we don't consider completeness a goal."* — P4

Additional design goals included performance and interoperability with other libraries for a majority of our participants, and flexibility e.g., to support cryptographic agility and accommodate new algorithms, for about half of the participants.

Primitive Selection. All libraries except the two micro libraries included and exposed multiple cryptographic primitives. Several factors and design goals contributed to our participants' decision to include or exclude primitives from their library. A majority considered standardization an inclusion criteria, indicating that it implies security, relevance, and interoperability. However, the majority also mentioned relevance and popularity as a separate criterion, detailing that not all standardized primitives are (still) popular or relevant, and that not all popular or relevant primitives are standardized yet. Thus, some participants excluded primitives due to limited use, and in order to reduce complexity and maintenance costs. Others deemed newer, experimental primitives worthy of inclusion to cover more use cases, or to

> *"be able to ship a library that has at least two options. One is conservative and well-adopted, and one is more experimental but believed to maybe have a longer crypto life period where it's not going to be broken."* — P17

Many participants also included primitives that are part of government regulations such as FedRAMP [70]. Finally, some participants excluded primitives due to known vulnerabilities.

Perception and Level of Abstraction Vary. The design of the API itself, i.e., the available functions and their signatures, steer an API's level of abstraction by determining users' access to the underlying primitives. A higher-level design that limits user options both reduces the potential for misuse, but also limits the API's flexibility, while a design with low level access has the opposite effect. Our participants' desired level of abstraction and their perception of what constitutes high- or low-level access varied widely. Participants had an inconsistent understanding of when an API provided high- or low-level access, depending on whether they compared it to libraries that allow detailed configuration of cryptographic algorithms or to libraries that accept no parameters other than the plaintext for encryption. They described abstraction levels ranging from *"very low-level"* (P16) to *"very high-level"* (P19). However, most of our participants provided both higher and lower-level access to the included primitives through multiple functions, while only a few restricted themselves to either one. Even those who considered their API mainly higher-level still often exposed lower-level options to maintain flexibility and avoid blocking users from using the library. About half of our participants divided their API into multiple layers for the user to choose from, and a few wished for such a separation. For individual function signatures, many participants stayed close to the respective standards they were implementing, and about half employed speaking or consistent names for functions and parameters to foster overall consistency and usability.

The spectrum of abstraction levels then became especially apparent in participants design choices on what parameters to expose, and whether to provide defaults for these parameters. Many participants reported that they did not restrict the parameters in any way, but most of our participants limited the choice given to their users by either deciding to not expose certain parameters, making them unconfigurable, or by providing a range or enumeration of possible values. About half took this principle further and denied any choice besides parameters they considered necessary. While this can generally be considered high-level access to the primitive, there was still disagreement on what constituted a necessity, with one participant stating: *"Necessary parameters are keys and additional data sizes of keys."* (P11) while another mentioned: *"The default is just all the keys are automatically generated and [...] managed."* (P12) Many participants further used abstract parameter types to hide complexity, support multiple different representations of keys, or to account for better backward compatibility:

> *"We always return an opaque object. If we return different types of opaque objects for new keys, the code still works*

also. Due to that, we can easily add algorithms and change things without changing, without breaking the API." — P3

Parameter defaults provide an opportunity to enable high-level usage without setting parameters while at the same time allowing for low-level configuration if desired by the user. While many participants set defaults wherever possible, others used defaults only sparingly or not at all, explaining that it is hard to choose appropriate values without knowing the use case. One participant discussed how defaults impaired auditability, requiring auditors to investigate which version of a library was used and provided what default instead of seeing the value of cryptographic parameters directly in the code. When selecting default values, the majority of participants strove for safe defaults, and many chose values they expected to work in most cases. Our participants also considered standards, regulations, or the anticipated hardware when choosing defaults.

To summarize, our participants did not agree on any ideal level of abstraction, and the level of abstraction a potential library user is exposed to varied on a broad spectrum:

> *"[LIBRARY-0] lets you do only very specific things. You've basically no choice at all. Whereas [LIBRARY-1] exposes basically everything, you can fiddle with things at very low levels. […] I don't think there's going to be one right answer, horses for courses, as they say."* — P21

Additional Information in Warnings and Documentation. Our participants used errors, warnings, and documentation to communicate correct and incorrect usage of their API to users, attempting to increase usability and misuse resistance. Almost all of our participants used error messages and exceptions to clarify failures or inform their users about invalid parameters. However, many also stated that this can conflict with security, and that they needed to be careful about potential information leaks when using errors or exceptions: *"One example […] is concerning decryption failure. Understanding which type of decryption failure happened, […] can be used to leverage these certain types of attacks."* (P6) In addition to errors, many libraries issued warnings to inform their users about less secure options, e.g., when they provide more secure mechanisms but also *"the weaker ones for backward compatibility, but with warnings and caveats to the developer when choosing to use these."* (P6)

All participants reported documentation that provided examples or explained options and use cases for the API, while many also highlighted potential pitfalls. Some used the documentation to recommend high-level functionality or parameters over low-level interfaces to avoid misuse. The majority of our participants wrote their documentation manually, and about half were dissatisfied with the current state of the documentation, e.g., seeing a mismatch with the goal to inform users on safe usage: *"[T]he documentation should strive to explain what it does and how to use it safely. […][J]ust because of the complexity of [LIBRARY] and its interfaces, there is a big gap."* (P15) Some participants attributed less importance to documentation, regarding the test cases of their library as good examples.

> **Levels and Perceptions of Abstraction Varied.** Different library purposes resulted in a variety of design goals, which in turn guided primitive inclusion, function design, and information for users. They led to varying abstraction levels, and there was no consensus about which API designs provided high- or low-level access.

4.3. Legacy, Standards, Community and Background Influence Design Decisions

Our participants discussed the decision making processes for their API design, and reported many factors that influenced their individual API design decisions. All our participants were involved in at least one library's decision-making process. The majority were involved in cryptographic libraries with a small team of core maintainers making design decisions. About half were part of a library in which the main maintainer, usually the participant, made the final API design decisions, and a few participants named a committee, a management position, or a security department as decision makers. About half of the participants explained that design decisions were typically made after reaching a consensus in discussion, requiring no authoritative final say.

Our participants described how the existing APIs affected their design choices because of backward compatibility and consistency concerns. In addition, they turned to standards and community resources for reference. However, almost all of our participants stated that some decisions ultimately depended on their personal tastes and opinions.

Initial Design Decisions. A majority of our participants made many important design decisions in the initial phase of creating a new cryptographic library. These decisions included the library's primary purpose and design goals, but also specific technical aspects of the API design such as how to structure functions and objects. Participants often made these decisions based on their reasons to create a new library. Most participants saw problems or gaps in existing libraries at the time. First, some wanted to implement new functionalities and cryptography designs that did not exist in a language or environment before, such as P5 asking themselves *"do we already have algorithms or cryptography that can accomplish this in a sound way? Then usually it becomes, no, we don't yet. We need to go make something."* (P5) Second, some were unhappy with available solutions. They started rewrites, forks, or new projects with consciously different design decisions to achieve design goals such as improved usability, better performance, or less resource requirements, catering to embedded devices: *"It should be possible to do something which is a lot lighter."* (P2) Some participants also began libraries with a concrete use case or user base in mind, informing their design decisions. This was the case for most of the proprietary and micro libraries. It also applied to some open-source projects, such as one inviting potential users to take part in the initial design process: *"We have meetings every week or so that anybody can join. […] there are some actual users of these APIs."* (P10)

Maintaining Consistency and Backward Compatibility. Existing API design influenced the design decisions of all of our participants, and about half of them directly described how initial design decisions continued to shape their current API. For example, one participant explained that their concept was *"initially meant for symmetric stream ciphers, and then everything else got bolted on."* (P11) For some participants, this resulted in persisting problems: *"We are still dealing with choices we made back then, which is that we bound everything we could rather than actually only binding the things we required."* (P1) Past design decisions typically became problematic because of the cryptography field's continuous growth and development, which a majority of our participants identified as a challenge for consistent API design. They described how new algorithms and designs emerged, and how new attacks and problems made old implementations dangerous or obsolete. Several participants regretted some of their design decisions in hindsight, such as one pointing out how post-quantum cryptography developments had highlighted their designs deficiencies in terms of flexibility and extensibility:

> *"It just assumes that Diffie-Hellman is the only key establishment mechanism in existence. Of course, that's no true. It wasn't true then, but now it's relevant and not true because we have actual post-quantum cryptography things."* — P7

Participants attempted to address these problems while trying to maintain consistency across their library's API and ensuring backward compatibility. They introduced new APIs with a different design, or changed or deprecated existing ones, with all approaches presenting challenges. Deviating from past design in new APIs often meant a loss of consistency across the API, which participants considered a difficult decision to make *"because that puts consistency with existing APIs, as much as they might be bad, [...] up against the benefits of whatever the change would be."* (P12)

A major problem with changing or deprecating existing APIs was preserving backward compatibility, which almost all participants were concerned about, and a majority perceived as challenging to maintain. Many participants reported having backward compatibility commitments or guarantees for their libraries, and almost all carefully weighted breaking changes against the potential workload they would cause for all of the library's users. In addition, many participants pointed out that breakage should align with the library users' expectations: *"If we introduce a breaking change at any point in time, it's not going to break just one application. It's going to break absolutely everything."* (P10) Reasons to accept breaking changes such as deprecation included vulnerabilities, APIs being easy to misuse, and a general lack of use. While many participants removed deprecated functionality from their code, some did not to avoid breakage. Instead, they pressured users to abandon deprecated API features by providing no support or removing them from the documentation. A few also had direct access to upstream projects and implemented the necessary changes following a deprecation themselves. However, about half of the participants identified problems in the deprecation process or wished for a better one, citing warning fatigue and slow user adoption rates. In addition, many decided not to deprecate at all to preserve backward compatibility and continue support for consumers with long update cycles: *"We won't be deprecating really almost any of our API because a user or customer might be using it, so we try to support it. That's why it's set in stone once it's written."* (P14)

In summary, existing API design was a large influence on design decisions, especially since reworking API design while maintaining consistency and backwards compatibility was perceived as a major challenge. This led some participants to feel stuck with prior design decisions. One participant stated that API designers should therefore *"try to do it very right at the first time"* (P20) to avoid such challenges.

Standard Guidance Leaves Room For Personal Choice. Cryptography standards often guided our participants design decisions, but participants did not always agree with them, and most standards left room for interpretation and personal choice. Most participants followed the cryptography standard that they were implementing with their API design. The majority of the participants selected primitives based on standards, many modeled their function design following the standard documents, and a few said they took parameters and defaults from a standard. Often, participants considered it mandatory to adopt the standard specification for their API design to ensure compliance, consistency, and interoperability: *"When you're building something that is not just a primitive, you have to base the API on the RFC."* (P7) However, a majority of our participants reported that not all design decisions were specified in the standards, or wished for fewer options to choose from. They felt that consistency and interoperability were threatened by a lack of specificity, and thought that standards could improve developer guidance for API design by being more specific:

> *"That's an important part of writing a standard, which is to give clear support to the developer to make the right API choices. Oftentimes the advice is more generic than would fit the particular use case."* — P6

In contrast, one participant described how they disagreed with a specific design choice in a standard, and felt that it hindered their goal of having tight restrictions against misuse: *"Sometimes the standard will say things like the algorithm field is optional in the key, which is obviously not helpful if I intend to always make it mandatory."* (P17) Such disagreement illustrates one reason why standards leave options for individual decisions by implementers and API designers. Standards are developed in a collaborative process, and some of our participants pointed out that achieving a consensus in the cryptography community about which option, design, or parameter set is best was very difficult. Different design philosophies exist, and use cases and environments cause varying requirements. Thus, picking the best option is far from a trivial problem. One participant believed that an ideal goal would be a perfect match of cryptographic properties and developer expectations, but

conceded that there was no clear path to achieve this given the diversity of expectations:

> *"I think the thing to be wished for is a perfect alignment between cryptographic algorithms and the properties they provide and developers' expectations of them, which is even more than a money and people problem."* — P21

In addition to these fundamental problems, some participants thought that the standardization process does not necessarily equally reflect the opinions of the entire cryptography community. They cited difficult access for less-known community members and an imbalance between large tech companies and open-source maintainers.

Overall, our participants considered standards to be important and binding guides for their implementation that tended to lack specificity. The standards did not provide our participants with conclusive information on how to make all of their API design choices, and almost all of them reported at least one decision they had made based on their personal opinion of what achieved the best balance of their design goals such as security, usability, flexibility, and performance. As one participant said, *"We went our own way as best we saw fit."* (P1)

Community Influences Personal Choice. Informal community resources played a significant role in forming our participants' opinions on cryptography and API design, and thus affected their design decisions. About half of them referred to programming language or community best practices for some of their design decisions, and all looked at the source code of other libraries to learn from existing implementations. Most mentioned one or more libraries that inspired their design choices or that they wanted to be consistent with. In line with the initial goal to fix a problem or fill a gap in existing libraries as described above, many participants attempted to learn from other libraries and improve on them. They evaluated what did and did not work in other libraries, and adopted or avoided certain design choices accordingly. In addition, about half of our participants received feedback from cryptographers not associated with their library, who either tried to use their libraries, or whom they met at community venues like conferences:

> *"In terms of people going, "Oh, wouldn't it be nice if this was more ergonomic or I think it would be harder to misuse if you did X, Y, and Z." [...] Those are conversations we get in the hallway track with other cryptographic developers."* — P1

Typically, our participants evaluated multiple community resources to inform their own opinions.

> **Legacy Code, Standards, Other Libraries, and Developer Intuition Influenced Design Decisions.** Maintainers, committees, or management staff made cryptographic API design decisions strongly influenced by legacy APIs due to consistency and backward compatibility concerns. Standards, best practices, and other cryptographic libraries guided decisions, but left open questions that required a decision based on personal opinion.

4.4. The Challenge of Finding Usable API Design

Usability was a goal for all our participants and an essential aspect of the API design. However, many participants noted the high complexity of cryptography being in conflict with designing usable API access. Participants had limited developer hours and funding to spend on usability, and struggled to identify usable API design. Furthermore, the resources for decision making described above made either no or few clear recommendations regarding usability, leaving especially much room for developer decisions in this area. To compensate, our participants attempted to make the best of the situation through personal choices based on feedback and their own experience, but they could only measure their success to a limited extent.

Finding Usable API Design Is Hard. Our participants found it difficult to identify what API design achieved good usability, and used different approaches to help them with this task. Many actively tried to keep up-to-date with new knowledge in API usability best practices. For formalized knowledge and scientific insights on usable APIs, only some of our participants turned to academic publications. They were aware that research in this area exists, and felt that it could help them: *"There are also some examples that you can rely on from academia, [...] there are lessons you can take from these."* (P6) However, about half of the participants saw a gap between literature and practice, and explained that it was hard to find results that were directly applicable to their work. They criticized the quality of example code and functions, if any was available, as too far from practice. As one participant put it: *"The discussions of API design in formal literature have always felt fairly removed from the realities of maintaining a [...] project."* (P4) A few participants wished for the inclusion of additional guidance on API design in cryptography publications, with one elaborating that they meant *"not providing APIs, but at least providing the set of functions that would make sense for an API to expose [...]."* (P10) Some participants also suggested that more usability and user research studies on cryptographic APIs specifically could be beneficial: *"Also, usability researchers getting involved in setting these best common standards and practices, would be useful."* (P6) As a concrete research gap, some participants said they lacked a systematization of best practices for the usable and secure design of cryptographic APIs, emphasizing that the knowledge appears to be scattered and hard to access:

> *"Looking at other libraries, figuring out what their experiences were, what do they regret, what would they do differently, how successful was each decision is something that would probably benefit from the systematization of knowledge."* — P4

Informally, our participants used evaluation of other libraries and discussions in community channels in an attempt to identify usable cryptographic API design, but lacked accessible scientific backup for their efforts.

Utilizing User Feedback Is Helpful But Resource-Intensive. To complement the insights on API usability our

participants gained from research and community resources, they looked at feedback they received from their users. Most participants received feedback on user needs and mistakes. While a majority considered large parts of the feedback unqualified, most participants still utilized it to identify API usability problems and try to improve their API. They achieved this, e.g., by asking themselves *"can I make the code easier to use or can I make a change to the manual to solve this problem for them [...] even if it's, let's say, it's a stupid question."* (P8) About half stated that they actively sought user feedback to gather usability information, indicating that they valued their users' perspective on API usability. The majority used GitHub issues or pull requests to exchange feedback over bug reports or feature requests, about half mentioned exchange via email or mailing lists, and many mentioned community exchange platforms like Slack or Stack Exchange. A few, however, also stated not receiving any input.

While feedback provided valuable insights, processing it also entailed some challenges. The majority of our participants considered and addressed feedback as it came in, indicating no particular strategy, and some participants explicitly noted that they were not employing a systematic evaluation. In combination with the large amount of feedback that participants perceived as unqualified, this meant that some of them struggled to handle all their feedback. One noted how they had to take care *"to not burn out just from the weight of stupidity of your users."* (P1) Several participants reported receiving a lot of basic questions that showed a general lack of understanding of cryptography, and sometimes even programming, by the users. And even from users with such basic understanding, not all feedback was perceived as helpful, and there were several reasons for participants to deny user requests. On one hand, participants made content based decisions. They denied requests that focused on individual needs rather than generally improving the API, or requests that they deemed insecure: *"Sometimes it's looking at the person saying, we can't do this, this is insecure. No, we will not support symmetrically encrypted data that lacks integrity."* (P3) On the other hand, social effects played a role in accepting or denying user requests.

Some participants reported unreasonable expectations or even threats in the feedback they received, especially in the open source space. Depending on the developers' personality and confidence in their own and their project's standing in the community, responses to such feedback varied. One participant reported that they did not consider it on principle: *"Lots of these requests remain unanswered just because they are formulated where 'you must do this now, otherwise, I stop using your library'."* (P11) In contrast, another participant felt uncomfortably pressured by user expectations and threats. They described how they sometimes gave in to user demands, especially in the earlier years of their project when they were still looking to establish themselves in the community and the project depended a lot on early user adoption:

> *"In some cases, I think we just got bullied into doing it. There are social effects there. [...] In like 2010, I was much more easily influenced by people being shouty on the internet than perhaps I should have been."* — P12

To summarize, participants indicated that the feedback they received, while informative, required effort to sift through and personal judgment that withstands unreasonable expectations and threats to identify useful insights.

Lack of Usability Evaluation Tools. Similar to the lack of strategy in user feedback evaluation, our participants lacked the tools to perform a formal usability evaluation, and most of our participants did not have a process to test for usability. While many participants thus did not test for usability at all, about half described that they imagined themselves as a user of their library to evaluate usability. *"If I can use it, it's already a victory. Most of the way I design the API, I build on my experience of reading my own code from several months or years before."* (P2) This approach meant that the API developers' perspective strongly influenced decisions on usability, and a few participants reflected on how their perspective may differ from that of actual users, especially regarding cryptographic expertise: *"I've spent a lot of time as a developer, I asked myself, what would I want? Okay, I have more crypto experience than most, which makes it hard to judge."* (P3) One participant tried to diversify the user perspective by creating personas of user groups they thought used their library, and envision their point of view. However, none of our participants mentioned any formalized framework or tool that would support them with usability evaluation.

> **Resources to Determine Usability Are Missing.** Our participants lacked sources and approaches to determine what achieved good usability. They used different strategies to find usable API design, filling a gap in empirical data, systematized knowledge, and evaluation tools with user feedback and usability self-tests.

4.5. Tensions Between Usability, Security, and Flexibility

In addition to the general challenge of finding a usable API design, our participants needed to balance usability with other design goals. Many reported conflicts between security and usability, and a majority perceived making trade-offs between usability, especially misuse resistance, and flexibility as a major design challenge.

Security–Usability–Trade-off. Reducing the misuse potential of cryptographic APIs is a usability aspect that almost all of our participants considered, and that helps to improve security for consumer applications. However, many participants still reported having to make trade-offs between usability and security in their API design, typically choosing security over usability. They reported two main factors that caused tension between the security and usability of an API, forcing them to make trade-offs.

First, complex cryptography properties inhibited usability measures. Most common among these properties were constant time execution and resistance against side-channel attacks, which many participants said restricted the space

for user-friendly error messages. Another such property was formal verification. One participant described how code that was automatically generated based on a formal verification specification had to satisfy lower requirements than manually written code:

> *"We gave some license to that code because of the formal verification. That code is not readable. [...] but knowing that there's formal proofs of specific parts of it being correct makes us worry less."* — P4

While the majority of our participants did not employ formal verification methods, or only to some small degree, about half of them wished for more formal verification in their development process. Finally, complex schemes that provided strong security if used correctly were sometimes difficult to harden against misuse, and thus included despite containing pitfalls for developers: *"There are some times when we put in some stuff that definitely had sharp edges that could be used unsafely because the potential benefit for using it correctly was very high."* (P12)

The second major cause of security-usability trade-offs were security measures that prevented misuse by, e.g., enforcing type and memory safety, but contradicted other usability aspects such as API ergonomics. Adding security guards that prevent misuse often restricted user options or required a more complex implementation:

> *"It's been challenging because it's really hard to balance making this harder to screw up [...] and also not making people hate working with it. [...] Even if it makes the API more cumbersome to an extent, we really want to push for you can't mess this up in a way that will be harmful."* — P5

To force users too accept and use a more complex, but also more secure option, one participant did not implement easier-to-use solutions:

> *"Sometimes we'll not add something because it will be easier than an alternative but less secure. We are worried about that cannibalizing the usage of the thing we actually want users to use."* — P4

This shows that even though increasing misuse resistance of cryptographic APIs combines the design goals security and usability, our participants still encountered scenarios which they perceived as choices between the two, and usually opted for increased security.

Misuse Resistance–Flexibility–Trade-off. Reduced misuse potential was a major aspect of usability for our participants, and in many cases combined two important design goals with usability and security. A majority of our participants identified trade-offs between misuse resistance and flexibility, another important design goal, as a major design challenge. Most of our participants associated increased misuse resistance with a higher level of abstraction and restricting the user's access and ability to configure low-level cryptographic parameters, which limits flexibility. They explained that at a certain level of flexible, low-level access, misuse resistance was very hard to realize with API design:

> *"I'm pretty expert in this stuff and I still screw it up. That's not really something API design can defend against if you're going to expose signatures as a thing. Maybe if you expose some even more abstracted thing like NaCl does and then maybe you could put some guards there."* — P21

One participant described how both extremes on the spectrum between misuse resistance and flexibility had unfavorable outcomes:

> *"If it's too flexible, developers can glue it together in horrible ways and get everything wrong, and the API won't guide developers in the right direction. However, if it's too inflexible, any sort of change won't work."* — P3

Participants elaborated that giving choices was important to address different use cases and requirements, yet too many choices were also problematic.

> *"It's hard to make everybody happy. It's even impossible because some people want something super expensive. Some people want something minimal. If you try to address every possible case, then you end up with a huge API surface, which people don't like."* — P10

These tensions between misuse resistance and flexibility show that usability optimization needs to be seen in context, and is thus a hard problem with no one size fits all solution. One participant summarized:

> *"I think there's basically one challenge which is, how much flexibility do you give people to innovate and do new stuff versus locking stuff down to make things safe? There's libraries who have made choices all along that spectrum."* — P21

> **Trade-offs Between Security, Flexibility and Security Are Major Design Challenges.** Our participants struggled to balance usability, security, and flexibility in their API design. Despite misuse resistance increasing both security and usability, participants still saw tensions between these two design goals. In addition, they described the level of abstraction as a non-trivial trade-off between misuse resistance and flexibility.

5. Discussion

Below, we discuss our results in the context of our research questions and related work. We then provide recommendations for the research community and standardization organizations for cryptographic primitives and protocols. We encourage developers of cryptographic APIs to carefully consider existing literature on general and cryptographic API design to guide their design decisions. In addition, design decisions should be made with a clear target audience in mind, and expected cryptography expertise should be communicated to potential users as clearly as possible. However, given that many developers of cryptographic APIs volunteer their time to contribute to open-source projects or struggle with a lack of funding and developer hours, we argue that it is unreasonable to expect everyone to become experts in usable API design on top of the required cryptography expertise, and solve these challenges by themselves. Therefore, we primarily direct our recommendations to develop supportive guidelines and tooling at researchers of usable cryptography and standardization organizations. Still, we hope that cryptographic API developers will actively participate in the research and standardization efforts

outlined below. Their expertise, experience, and perspective are paramount for creating actionable guidelines that align with cryptographic API development.

5.1. Contextualizing Findings with our RQs and Results from Prior Work

To answer each of our research questions, we discuss our findings in the context of results from related work.

Cryptographic API Design Decisions (RQ1). Reported API designs followed diverse purposes and design goals, varied widely, and led to a spectrum of abstraction levels ranging from very high to very low (cf. Section 4.2). Congruence with API design guidelines from the literature was inconsistent. Only a few participants indicated awareness of existing general API design guidelines [11], [13], [18], or recommendations specifically for security and cryptography [8], [28]. Still, several reported designs were in line with recommendations such as consistent naming, clear documentation with examples, or providing safe defaults [8], [13], [18]. In contrast, participants also described deviating designs. Some were unintentional, e.g., incomplete documentation that participants were dissatisfied with, while others were conscious decisions such as participants avoiding defaults to increase code readability and auditability of consumer code by forcing explicit setting of parameters. There was a similar split for the recommendation to tailor APIs to users and use cases [18]. User expertise was considered, but not by all participants, and had a varying influence on the API design. Problems arose when users who did not match these considerations tried to use the APIs. Participants did not always communicate targeted users and use cases, and some deemed their current practice of a notice in the documentation insufficient. Prior work on how users select third-party packages indicates that they look at a library's documentation before adoption. However, it does not explicitly indicate that a user's expertise influences their decision [71]. Thus, documentation can be a good place to note use cases and required user expertise, but further research is needed on communicating the risk of cryptography misuse to users.

API design decisions were guided by standards, community best practices, other libraries, and legacy APIs, but often ultimately came down to the developers personal taste and opinion (cf. Section 4.3) shaped by their background in cryptography (cf. Section 4.1). Our results show that cryptographic standards do not sufficiently specify API design guidelines. Further, related work depicts standards as a product of many influences that struggle to represent a broader community consensus and lack usability [72]. We also found that our participants used source code from other cryptographic libraries as learning materials for cryptography and inspiration for library API design. This propagation of code has also been found in the general open source ecosystem [73]–[75], and led our participants to pick and choose API designs based on which projects were generally considered successful and which they personally liked.

Challenges in Cryptographic API Design (RQ2). We identified several key challenges our participants faced in designing cryptographic APIs, including limited developer hours and funding, establishing and measuring usability (cf. Section 4.4), and handling the trade-off between usability, security, and flexibility (cf. Section 4.5).

Widespread problems in software engineering regarding a lack of time or, especially in the open source sector, a lack of funding [76], [77] applied to some of our participants. It limited their opportunities to invest time and effort in API usability. Further, it is a challenge to assess the usability of specific API designs, identify options with good usability, and balance usability with security and flexibility. We found that while our participants read academic publications, it was more likely to be cryptography papers than research on API design. In addition, usable cryptographic API design is not yet comprehensively studied, and most existing recommendations lack empirical validation [28]. Concerning the relationship between programmatic levels of API abstraction, functionality, and usability, current research does not provide empirical data on the effect of different options for specific design choices and thus cannot provide specific recommendations. As a result, the research did not sufficiently support our participants in identifying a specific API design that provides a good balance of usability, security, and flexibility.

Participants also reported challenges in systematically evaluating user feedback to derive usability problems and identify areas for improvement, e.g., due to an overwhelming amount of low-quality or inappropriate requests. Related work considers analyzing user feedback a viable strategy to find usability issues in general software engineering [78], [79], and dedicated tooling to collect user feedback from various sources could help to address the lack of systematic strategies for evaluation [80]. However, our participants were unaware of such research or tooling, which does not cater to the specific usability challenges of cryptographic APIs. While there are some efforts to create tools to identify misuse of cryptographic APIs [50], [51], these still have flaws that impede the discovery of misuse in practice [54]. In addition, they are designed to detect and repair API misuse rather than systematically evaluate user mistakes to identify opportunities for improvement in API design. Finally, measuring cryptographic APIs' usability was a significant challenge and remains an unsolved problem in research. None of our participants performed formal usability tests, but they attempted to put themselves in a user's place to assess the usability of their API. Research in software engineering suggests this approach can be helpful, mainly if performed within a systematic framework. However, it also highlights the importance of tests with people that represent the eventual users [18], and there can be a significant discrepancy in cryptographic expertise between the developers and users of cryptographic APIs.

Support for Cryptographic API Designers (RQ3). Our participants recognized the challenges they described as complex problems that require further research, community discussion, and design work. They described a significant

gap between the practical problems they faced in API design and the guidance they received through the cryptographic standards and research literature they consumed. The unavailability of a scientifically grounded community consensus required them to choose what they believed to be the best option based on their own experiences (cf. sections 4.3 & 4.4). Related work on the continued widespread misuse of cryptographic APIs across programming languages and software ecosystems [5]–[7], [46]–[48] highlights the consequences of this lack of support. To better support future design decisions for cryptographic APIs, we outline open research questions and detailed recommendations for usable security researchers and standardization processes for cryptographic primitives and protocols below.

5.2. Recommendations for Human-Centered Cryptography Research

We point out open research questions, provide recommendations based on the need for support to overcome challenges identified in our results and relate them to recommendations from prior work. We base our recommendations for the usable security research community on the finding that our participants missed a scientific foundation for secure, usable, and flexible cryptographic library API design. Overall, research should expand prior work to give more specific, empirically validated recommendations. In addition, an effort is needed to develop better tooling for usability testing and user feedback evaluation and to make it accessible to developers of cryptographic APIs without needing to conduct full-fledged user studies. We recommend to research cryptographic API design that balances security, usability, and flexibility at different layers of abstraction. Finally, it should address how to best steer users towards the API that is best suited for their use case and expertise. Our participants had close ties to academia, and many read cryptography research papers, but this rarely extended to usability or API design research. Cryptography research venues should consider publishing human-centered cryptography papers to increase the visibility of this research in the cryptography community.

Validate Specific Recommendations And Provide Evaluation Tools. In prior work, researchers gave recommendations to improve the usability of cryptographic APIs and evaluated the usability of some of the libraries available at the time (cf. Usability of Cryptographic Libraries). In a systematic literature review, Patnaik *et al.* find that the recommendations focus on code construction and user understanding and often lack empirical validation [28]. In addition, we find related work primarily investigated and compared problems of existing systems. Experimental evaluation of specific design choices, such as which parameters should be configurable, which values should be chosen for defaults and non-configurable parameters, or how functionality should be broken up into functions, is missing. Many recommendations from prior work thus provide desirable design properties rather than specific recommendations for design choices and did not comprehensively address the challenges described by our participants.

Complementary to the empirical evaluation of specific design choices and recommendations, research should develop methods and tooling to enable the developers of cryptographic APIs to evaluate their usability effectively and efficiently. Our participants lacked a systematic, tool-supported approach to test their APIs for usability and evaluate user feedback. For example, researchers could look into specific usability questionnaires for capturing user experiences with cryptographic APIs such as suggested by Wijayarathna *et al.* [33]. However, since most developers likely cannot easily access appropriate user samples, such methods may be more suitable for research purposes. Given the prevalence of usability self-tests and user feedback, we argue for developing frameworks and tools that support usability tests for cryptographic APIs based on empirical research results and structured feedback evaluation. Where possible, tooling should utilize automation and be integrated into existing test suites and continuous integration solutions.

Consider Layers of Abstraction. Related work commonly recommends to make APIs simple or high level [29], [30], [34], or at the highest level possible [81]. However, it does not specify what exactly constitutes high-level API design, and our findings show that cryptographic API developers disagree on what API designs are considered high-level. For example, our participants disagreed on which parameters were necessary for the user to set, and they did not perceive the decision on how high-level an abstraction was possible without losing the flexibility required to support different use cases, as evident. To what degree security mechanisms should or need to be hidden from their users is an open discussion on the broader field of usable security research [82]. Prior work suggests that workarounds introduced by users if an inflexible security mechanism does not fit their use case, can become a severe security problem [83]. An equivalent for such insecure workarounds in the context of cryptographic APIs would be switching to a lower-level API that supports the use case but is less fortified against misuse. Our participants described similar considerations and pointed out the trade-off between either making decisions with a high level of cryptographic expertise, but little insight into the specific use cases or leaving the decisions to users who lack basic knowledge of cryptography, but are familiar with the use case.

In general, not all use cases can be supported by APIs with a very high level of abstraction, and related work by Lo Iacono *et al.* shows that there is demand for low and medium-level cryptographic APIs among developers [84]. Thus, future research should consider different layers of abstraction and study how many provide the best balance between matching use cases and not overloading users with available choices and unnecessary complexity. Our participants typically spoke of low and high-level abstraction as two layers. Still, the desire for medium-level cryptography APIs identified by Lo Iacono *et al.* might indicate that three layers better match user needs. Specific API design choices should then be evaluated for their trade-off between usabil-

ity, reduced misuse potential, and flexibility. Future research should make appropriate design recommendations for each layer. To ensure that the different layers match their users' expectations and that users pick a layer that aligns with their expertise and use case, research should investigate criteria that developers use to choose a cryptographic API, and establish effective ways to communicate API capabilities and expertise expectations to users. This includes how to separate different layers of abstraction, i.e., whether to have various (micro) libraries for specific use cases or multiple layers as parts of a general purpose cryptographic library.

5.3. Recommendations for Human-Centered Cryptography Standardization

In addition to human-centered publications at cryptography venues, another way of disseminating results may be needed for insights from research on usable, secure API design to be adopted in practice. In contrast to other areas of software engineering, cryptographic libraries are heavily influenced by cryptographic standardization processes and organizations. Standards were an essential resource for our participants and considered binding by most, but often did not specify API design choices.

Increase Visibility and Adoption. Following empirical analysis of the effects of specific API design choices, we believe the derived recommendations should become part of the respective standards for cryptographic primitives and protocols. We argue that standards are a great place to disseminate guidelines specific to a protocol or algorithm, given that most of our participants consulted standards and held them in high regard. Similar to a reference implementation, recommendations for API design could be provided as additional material alongside the standard, and help developers who implement the standard in their libraries. Standards should adopt the approach of multiple abstraction layers, as outlined above, and make API design recommendations for each to provide options for varying degrees of misuse protection and flexibility. This would help to ensure the standards provide useful guidance for developers of libraries with different purposes and intended user groups. Including definitions for layers of abstraction and their associated degree of flexibility and misuse potential would help developers to select options accordingly. A definition of layers of abstraction could also provide a reference for communicating a library's position on the misuse-resistance-flexibility-spectrum to their users. Overall, we argue that adding API design recommendations based on empirical research can significantly boost their visibility and adoption.

Foster Interdisciplinary Collaboration. Software engineering research has established a need for interdisciplinary collaboration and roles in API design [18]. Integrating API design considerations in the standardization process would provide a context for usability researchers and API designers to collaborate with the broader cryptography community. Our participants described a disconnect between cryptography and usability, and prior work found misunderstandings between theoretical and applied cryptographers and software engineers were a significant challenge in the overall adoption of cryptography [72]. The discussion of research results, design approaches, and practical experience required to include API design recommendations in standards could help bridge this gap and ensure that usability is considered early when designing cryptographic primitives and protocols. We support Fischer *et al.*'s call for a more transparent and open standardization process to foster such discussion and ensure it includes various community perspectives. Our participants confirmed challenges they identified with an imbalance between individual developers of open source projects and big tech companies and difficulties encountered when wanting to impact standardization processes as outsiders. We conclude that standardization organizations should not only improve outreach towards academia [72], but also developers of cryptographic libraries to collaboratively include API design guidelines for multiple abstraction levels into cryptographic standards.

6. Conclusion

We conducted 21 semi-structured interviews with experienced developers of cryptographic libraries. We reported their views on API design, decision factors, and processes, and the challenges they faced. The API designs covered a variety of abstraction levels. Cryptography standards, other libraries, and legacy code influenced the design decisions. Developers found it challenging to determine usability and balance security, usability, and flexibility through an appropriate level of abstraction. The lack of systematic resources available to the developers on the topic coincided with a lack of consensus and highlights the need for further systematic research in this area. We propose to conduct human-factors research to empirically evaluate specific design choices, help define API abstraction levels that cater to different cryptography expertise and usability needs, and develop tooling to test usability. We suggest integrating knowledge gained from this research into cryptographic standards in an open, transparent process that encourages community discussion and feedback loops between academia and practical application. We thus hope to provide actionable guidance to developers of cryptographic libraries to consistently achieve usable API design at multiple levels of abstraction and flexibility.

Availability

To support the replication and transparency of our work, we make our study material, including our study invitation, consent form, demographics survey, interview guide, and codebook, available at: https://osf.io/t9q2p/?view_only=9f7f7105c86742a2a05003227c38996c.

References

[1] OWASP. *"A02:2021 – Cryptographic Failures*." (2021), [Online]. Available: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/ (visited on 05/10/2023).

[2] O. S. Team, *CVE-2022-3786 and CVE-2022-3602: X.509 Email Address Buffer Overflows*, https://www.openssl.org/blog/blog/2022/11/01/email-address-overflows/, Accessed: 2022-11-23, 2022.

[3] B. Marczak and J. Scott-Railton. *"Move Fast and Roll Your Own Crypto-A Quick Look at the Confidentiality of Zoom Meetings."* (2020), [Online]. Available: https://citizenlab.ca/2020/04/move-fast-roll-your-own-crypto-a-quick-look-at-the-confidentiality-of-zoom-meetings/ (visited on 04/30/2023).

[4] R. Naraine. *"Zoom Patches High Risk Flaws on Windows, MacOS Platforms."* (2023), [Online]. Available: https://www.securityweek.com/zoom-patches-high-risk-flaws-windows-macos-platforms/ (visited on 04/30/2023).

[5] M. Hazhirpasand, M. Ghafari, and O. Nierstrasz, *"Java Cryptography Uses in the Wild,"* in *Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Association for Computing Machinery, 2020.

[6] M. Egele, D. Brumley, Y. Fratantonio, and C. Kruegel, *"An empirical study of cryptographic misuse in android applications,"* in *Proc. 20th ACM Conference on Computer and Communication Security (CCS'13)*, ACM, 2013.

[7] A.-K. Wickert, L. Baumgärtner, F. Breitfelder, and M. Mezini, *"Python Crypto Misuses in the Wild,"* in *Proceedings of the 15th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Association for Computing Machinery, 2021.

[8] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky, *"Comparing the Usability of Cryptographic APIs,"* in *Proc. 38th IEEE Symposium on Security and Privacy (SP'17)*, IEEE, 2017.

[9] J. Stylos and B. Myers, *"Mapping the Space of API Design Decisions,"* in *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)*, 2007, pp. 50–60.

[10] J. Bloch, *"How to design a good API and why it matters,"* in *Proc. 21st Conference on Object-oriented Programming Systems Languages and Applications (OOPSLA'06)*, ACM, 2006.

[11] M. Henning, *"API: Design Matters: Why Changing APIs Might Become a Criminal Offense.,"* *Queue*, vol. 5, no. 4, pp. 24–36, 2007.

[12] M. Tello-Rodríguez, J. O. Ocharán-Hernández, J. Pérez-Arriaga, X. Limón, and Á. J. Sánchez-García, *"A design guide for usable web APIs,"* *Programming and Computer Software*, vol. 46, pp. 584–593, 2020.

[13] B. A. Myers and J. Stylos, *"Improving API Usability,"* *Communications of the ACM*, vol. 59, no. 6, pp. 62–69, 2016.

[14] J. Stylos, B. Graf, D. K. Busse, C. Ziegler, R. Ehret, and J. Karstens, *"A case study of API redesign for improved usability,"* in *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*, IEEE, 2008, pp. 189–192.

[15] A. Macvean, M. Maly, and J. Daughtry, *"API Design Reviews at Scale,"* Association for Computing Machinery, 2016, pp. 849–858.

[16] J. Stylos, S. Clarke, and B. A. Myers, *"Comparing API Design Choices with Usability Studies: A Case Study and Future Directions.,"* in *Proceedings of the 18th Annual Workshop of the Psychology of Programming Interest Group, PPIG 2006, Brighton, UK, September 7-8, 2006*, Psychology of Programming Interest Group, 2006, p. 17.

[17] S. Weber, M. Coblenz, B. Myers, J. Aldrich, and J. Sunshine, *"Empirical Studies on the Security and Usability Impact of Immutability,"* in *2017 IEEE Cybersecurity Development (SecDev)*, 2017, pp. 50–53.

[18] L. Murphy, M. B. Kery, O. Alliyu, A. Macvean, and B. A. Myers, *"API Designers in the Field: Design Practices and Challenges for Creating Usable APIs,"* in *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2018, pp. 249–258.

[19] J. Ofoeda, R. Boateng, and J. Effah, *"Application programming interface (API) research: A review of the past to inform the future,"* *International Journal of Enterprise Information Systems (IJEIS)*, vol. 15, no. 3, pp. 76–95, 2019.

[20] T. Zhang, B. Hartmann, M. Kim, and E. L. Glassman, *"Enabling Data-Driven API Design with Community Usage Data: A Need-Finding Study,"* Association for Computing Machinery, 2020, pp. 1–13.

[21] A. L. Santos and B. A. Myers, *"Design annotations to improve API discoverability,"* *Journal of Systems and Software*, vol. 126, pp. 17–33, 2017.

[22] S. Y. Jeong, Y. Xie, J. Beaton, B. A. Myers, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and D. K. Busse, *"Improving Documentation for eSOA APIs through User Studies,"* in *End-User Development*, V. Pipek, M. B. Rosson, B. de Ruyter, and V. Wulf, Eds., Springer Berlin Heidelberg, 2009, pp. 86–105.

[23] J. Stylos, A. Faulring, Z. Yang, and B. A. Myers, *"Improving API documentation using API usage information,"* in *2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, IEEE, 2009, pp. 119–126.

[24] T. Grill, O. Polacek, and M. Tscheligi, *"Methods towards API Usability: A Structural Analysis of Usability Problem Categories,"* in *Human-Centered Software Engineering*, Springer Berlin Heidelberg, 2012, pp. 164–180.

[25] T. Scheller and E. Kühn, *"Automated measurement of API usability: The API Concepts Framework,"* *Information and Software Technology*, vol. 61, pp. 145–162, 2015.

[26] D. S. Oliveira, T. Lin, M. S. Rahman, R. Akefirad, D. Ellis, E. Perez, R. Bobhate, L. A. DeLong, J. Cappos, and Y. Brun, *"API Blindspots: Why Experienced Developers Write Vulnerable Code,"* in *Proc. 14th Symposium on Usable Privacy and Security (SOUPS'18)*, USENIX, 2018.

[27] M. Ahasanuzzaman, M. Asaduzzaman, C. K. Roy, and K. A. Schneider, *"Classifying stack overflow posts on API issues,"* in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 244–254.

[28] N. Patnaik, A. Dwyer, J. Hallett, and A. Rashid, *"Slr: From saltzer and schroeder to 2021... 47 years of research on the development and validation of security api recommendations,"* *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 3, Apr. 2023.

[29] D. Votipka, K. R. Fulton, J. Parker, M. Hou, M. L. Mazurek, and M. Hicks, *"Understanding security mistakes developers make: Qualitative analysis from Build It, Break It, Fix It,"* in *Proc. 29th Usenix Security Symposium (SEC'20)*, USENIX, 2020.

[30] M. Green and M. Smith, *"Developers are Not the Enemy!: The Need for Usable Security APIs,"* *IEEE Security & Privacy*, vol. 14, no. 5, pp. 40–46, 2016.

[31] P. Gorski and L. L. Iacono, *"Towards the Usability Evaluation of Security APIs,"* in *Proc. Tenth International Symposium on Human Aspects of Information Security & Assurance (HAISA 2016)*, School of Computing & Mathematics Plymouth University, 2016.

[32] N. Patnaik, J. Hallett, and A. Rashid, *"Usability Smells: An Analysis of Developers' Struggle With Crypto Libraries,"* in *Proc. 15th Symposium on Usable Privacy and Security (SOUPS'19)*, USENIX, 2019.

[33] C. Wijayarathna, N. Arachchilage, and J. Slay, *"Using Cognitive Dimensions Questionnaire to Evaluate the Usability of Security APIs,"* in *Proceedings of the 19th International Conference on Human-Computer Interaction*, 2017.

[34] K. Mindermann, P. Keck, and S. Wagner, *"How Usable are Rust Cryptography APIs?"* In *Proc. 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS'18)*, IEEE, 2018.

[35] J. Luo, X. Yang, X. Yi, F. Han, I. Gondal, and G.-B. Huang, *"A Comparative Study on Design and Usability of Cryptographic Libraries,"* in *Proceedings of the 2023 Australasian Computer Science Week*, Association for Computing Machinery, 2023.

[36] S. Nadi, S. Krüger, M. Mezini, and E. Bodden, *""Jumping Through Hoops": Why do Java Developers Struggle With Cryptography APIs?"* In *Proc. 38th IEEE/ACM International Conference on Software Engineering (ICSE'16)*, ACM, 2016.

[37] 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 *Proc. 24th ACM*

Conference on Computer and Communication Security (CCS'17), ACM, 2017.

[38] C. Wijayarathna and N. A. G. Arachchilage, *"Why Johnny Can't Store Passwords Securely? A Usability Evaluation of Bouncycastle Password Hashing*," in *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, Association for Computing Machinery, 2018, pp. 205–210.

[39] P. L. Gorski, Y. Acar, L. Lo Iacono, and S. Fahl, *"Listen to Developers! A Participatory Design Study on Security Warnings for Cryptographic APIs*," in *Proc. CHI Conference on Human Factors in Computing Systems (CHI'20)*, ACM, 2020.

[40] P. L. Gorski, L. L. Iacono, D. Wermke, C. Stransky, S. Möller, Y. Acar, and S. Fahl, *"Developers Deserve Security Warnings, Too: On the Effect of Integrated Security Advice on Cryptographic API Misuse*," in *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, USENIX Association, 2018, pp. 265–281.

[41] K. Mindermann and S. Wagner, *"Usability and Security Effects of Code Examples on Crypto APIs*," in *2018 16th Annual Conference on Privacy, Security and Trust (PST)*, 2018, pp. 1–2.

[42] S. Indela, M. Kulkarni, K. Nayak, and T. Dumitraş, *"Toward Semantic Cryptography APIs*," in *2016 IEEE Cybersecurity Development (SecDev)*, 2016, pp. 9–14.

[43] S. Indela, M. Kulkarni, K. Nayak, and T. Dumitraş, *"Helping Johnny Encrypt: Toward Semantic Interfaces for Cryptographic Frameworks*," in *Proc. 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! 2016)*, ACM, 2016.

[44] D. J. Bernstein, T. Lange, and P. Schwabe, *"The Security Impact of a New Cryptographic Library*," in *Proc. 2nd International Conference on Cryptology and Information Security in Latin America (LATINCRYPT'12)*, Springer, 2012.

[45] A. Zeier, A. Wiesmaier, and A. Heinemann, *"API Usability of Stateful Signature Schemes*," in *Proc. 14th International Workshop on Security (IWSEC'19)*, Springer, 2019.

[46] D. Lazar, H. Chen, X. Wang, and N. Zeldovich, *"Why does cryptographic software fail?: a case study and open problems*," in *Proc. 5th Asia-Pacific Workshop on Systems (APSys'14)*, ACM, 2014.

[47] Y. Li, Y. Zhang, J. Li, and D. Gu, *"iCryptoTracer: Dynamic Analysis on Misuse of Cryptography Functions in iOS Applications*," in *Network and System Security*, M. H. Au, B. Carminati, and C.-C. J. Kuo, Eds., Springer International Publishing, 2014, pp. 349–362.

[48] 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 *Proc. 19th ACM Conference on Computer and Communication Security (CCS'12)*, ACM, 2012.

[49] J. Gao, P. Kong, L. Li, T. F. Bissyandé, and J. Klein, *"Negative Results on Mining Crypto-API Usage Rules in Android Apps*," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, 2019, pp. 388–398.

[50] S. Ma, D. Lo, T. Li, and R. H. Deng, *"CDRep: Automatic Repair of Cryptographic Misuses in Android Applications*," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, Association for Computing Machinery, 2016, pp. 711–722.

[51] 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*, IEEE Press, 2017, pp. 931–936.

[52] S. Afrose, S. Rahaman, and D. Yao, *"CryptoAPI-Bench: A Comprehensive Benchmark on Java Cryptographic API Misuses*," in *2019 IEEE Cybersecurity Development (SecDev)*, 2019, pp. 49–61.

[53] R. Paletov, P. Tsankov, V. Raychev, and M. Vechev, *"Inferring Crypto API Rules from Code Changes*," in *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, Association for Computing Machinery, 2018, pp. 450–464.

[54] A. S. Ami, N. Cooper, K. Kafle, K. Moran, D. Poshyvanyk, and A. Nadkarni, *"Why Crypto-detectors Fail: A Systematic Evaluation of Cryptographic Misuse Detection Techniques*," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 614–631.

[55] Y. Zhang, M. Kabir, Y. Xiao, D. Yao, and N. Meng, *"Automatic Detection of Java Cryptographic API Misuses: Are We There Yet?*" *IEEE Transactions on Software Engineering*, vol. 49, no. 01, pp. 288–303, 2023.

[56] L. A. Goodman, *"Snowball Sampling*," *The Annals of Mathematical Statistics*, vol. 32, no. 1, pp. 148–170, 1961.

[57] *"Crypto Forum Research Group*." (2024), [Online]. Available: https : / / mailman . irtf . org / mailman / listinfo / cfrg/ (visited on 02/29/2024).

[58] B. Saunders, J. Sim, T. Kingstone, S. Baker, J. Waterfield, B. Bartlam, H. Burroughs, and C. Jinks, *"Saturation in qualitative research: exploring its conceptualization and operationalization*," *Quality & Quantity*, vol. 52, pp. 1893–1907, 2017.

[59] *"Human-Made Transcription*." (2024), [Online]. Available: https://www.amberscript.com/en/products/manual-transcription/ (visited on 02/08/2024).

[60] V. Braun and V. Clarke, *"Using thematic analysis in psychology*," *Qualitative Research in Psychology*, vol. 3, no. 2, pp. 77–101, 2006.

[61] N. McDonald, S. Schoenebeck, and A. Forte, *"Reliability and Inter-Rater Reliability in Qualitative Research: Norms and Guidelines for CSCW and HCI Practice*," *ACM on Human-Computer Interaction*, vol. 3, no. CSCW, 72, pp. 1–23, 2019.

[62] S. Amft, S. Höltervennhoff, R. Panskus, K. Marky, and S. Fahl, *"Everyone for Themselves? A Qualitative Study about Individual Security Setups of Open Source Software Contributors*," in *In 45th IEEE Symposium on Security and Privacy, IEEE S&P 2024, May 20-23, 2024*, IEEE Computer Society, 2024.

[63] A. McDonald, C. Barwulor, M. L. Mazurek, F. Schaub, and E. M. Redmiles, *""It's stressful having all these phones": Investigating Sex Workers' Safety Goals, Risks, and Practices Online*," in *USENIX Security Symposium*, 2021.

[64] H. Beyer and K. Holtzblatt, *Contextual Design: Defining Customer-Centered Systems*. Morgan Kaufmann Publishers Inc., 1997.

[65] E. Kenneally and D. Dittrich, *"The Menlo Report: Ethical principles guiding information and communication technology research*," *SSRN Electronic Journal*, 2012.

[66] W. Usman, J. Hu, M. Wilson, and D. Zappala, *"Distrust of big tech and a desire for privacy: Understanding the motivations of people who have voluntarily adopted secure email*," in *Nineteenth Symposium on Usable Privacy and Security (SOUPS 2023)*, USENIX Association, 2023, pp. 473–490.

[67] S. Zhang, Y. Feng, Y. Yao, L. F. Cranor, and N. Sadeh, *"How usable are ios app privacy labels?*" *Proceedings on Privacy Enhancing Technologies*, 2022.

[68] P. Emami-Naeini, H. Dixon, Y. Agarwal, and L. F. Cranor, *"Exploring How Privacy and Security Factor into IoT Device Purchase Behavior*," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, Association for Computing Machinery, 2019, pp. 1–12.

[69] H. Habib, S. Pearman, J. Wang, Y. Zou, A. Acquisti, L. F. Cranor, N. Sadeh, and F. Schaub, *""It's a scavenger hunt": Usability of Websites' Opt-Out and Data Deletion Choices*," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, Association for Computing Machinery, 2020, pp. 1–12.

[70] G. T. T. Services. *"Fedramp*." (2024), [Online]. Available: https://www.fedramp.gov (visited on 06/06/2024).

[71] E. Larios Vargas, M. Aniche, C. Treude, M. Bruntink, and G. Gousios, *"Selecting Third-Party Libraries: The Practitioners' Perspective*," in *Proc. 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ACM, 2020.

[72] K. Fischer, I. Trummová, P. Gajland, Y. Acar, S. Fahl, and A. Sasse, *"On The Challenges of Bringing Cryptography from Papers to Products: Results from an Interview Study with Experts*," in *In 33rd USENIX Security Symposium, USENIX Security '24, Philadelphia, PA, USA, August 14-16, 2024*, USENIX Association, 2024.

[73] S. Haefliger, G. von Krogh, and S. Spaeth, *"Code Reuse in Open Source Software*," *Management Science*, vol. 54, no. 1, pp. 180–193, 2008.
[74] M. Sojer and J. Henkel, *"Code Reuse in Open Source Software Development: Quantitative Evidence, Drivers, and Impediments*," *Computing Technologies eJournal*, 2010.
[75] M. Jahanshahi and A. Mockus, *"Dataset: Copy-based Reuse in Open Source Software*," *MSR 2024 Data and Tool Showcase Track*, 2024.
[76] R. D. Austin, *"The Effects of Time Pressure on Quality in Software Development: An Agency Model*," *Inf. Syst. Res.*, vol. 12, pp. 195–207, 2001.
[77] N. Nan and D. E. Harter, *"Impact of Budget and Schedule Pressure on Software Development Cycle Time and Effort*," *IEEE Transactions on Software Engineering*, vol. 35, pp. 624–637, 2009.
[78] M. Stade, F. Fotrousi, N. Seyff, and O. Albrecht, *"Feedback Gathering from an Industrial Point of View*," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*, 2017, pp. 71–79.
[79] R. T. Høegh, *""Software Development and Feedback from Usability Evaluations"*," in *Interdisciplinary Aspects of Information Systems Studies*, Physica-Verlag HD, 2008, pp. 47–53.
[80] S. van Oordt and E. Guzman, *"On the Role of User Feedback in Software Evolution: a Practitioners' Perspective*," in *2021 IEEE 29th International Requirements Engineering Conference (RE)*, 2021, pp. 221–232.
[81] P. Gutmann, *"Lessons Learned in Implementing and Deploying Crypto Software*," in *11th USENIX Security Symposium (USENIX Security 02)*, USENIX Association, 2002.
[82] V. Distler, M. Zollinger, C. Lallemand, P. B. Rønne, P. Y. A. Ryan, and V. Koenig, *"Security - Visible, Yet Unseen?"* In *Proc. CHI Conference on Human Factors in Computing Systems (CHI'19)*, ACM, 2019.
[83] D. A. Norman, *"When security gets in the way*," *Interactions*, vol. 16, pp. 60–63, 2009.
[84] L. Lo Iacono and P. Gorski, *"I Do and I Understand. Not Yet True for Security APIs. So Sad*," Internet Society, 2017.
[85] J. Corbin and A. Strauss, *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*, 4th. Thousand Oaks, CA: Sage Publications, 2015.
[86] M. B. Miles, A. M. Huberman, and J. Saldaña, *Qualitative data analysis : a methods sourcebook*. Los Angeles: Sage, 2014.
[87] J. Hielscher and S. Parkin, *""what keeps people secure is that they met the security team": Deconstructing drivers and goals of organizational security awareness*," in *33rd USENIX Security Symposium (USENIX Security 24)*, Philadelphia, PA: USENIX Association, Aug. 2024, pp. 3295–3312.
[88] P. Badva, K. M. Ramokapane, E. Pantano, and A. Rashid, *"Unveiling the Hunter-Gatherers: Exploring threat hunting practices and challenges in cyber defense*," in *33rd USENIX Security Symposium (USENIX Security 24)*, Philadelphia, PA: USENIX Association, Aug. 2024, pp. 3313–3330.

Appendix A: Interview Questions

Background and Experience.
1. How did you get into cryptography?
1.1. Do you have any formal education in computer science or cryptography?
2. For how long have you been working on cryptographic libraries?
3. Can you briefly tell me what cryptographic library project(s) you are or have been working on?
3.1. [*If: multiple projects*] Which project of yours would be most interesting to talk about in the context of API design decisions and processes?

Design Decisions.
1. In what programming language is the API written?
2. What is the primary purpose or premise of your library?
3. What differentiates it from other cryptographic libraries?
4. How do you decide which cryptographic algorithms and primitives to include in your library?
4.1. What are inclusion and exclusion criteria?
4.2. Is formal verification considered?
5. What types of functions does your API expose to its users?
5.1. Is there low-level access to the underlying cryptographic primitives or are functions more high-level and abstract?
5.2. How do you decide what functions to expose?
6. What types of functions does your API expose to its users?
6.1. Would you say they provide more of a high or low-level access?
6.2. How do you decide what functions to expose?
7. Looking at individual functions, do you follow any design principles for them?
7.1. How are functions named?
7.2. What kinds of parameters do the functions allow users to set?
7.3. What are typical return values of the functions?
7.4. Are there any warning or error messages for the users?
8. Does the API have default configurations?
8.1. How were they selected?
9. What does the documentation of the project look like?
9.1. Do you use any tools to create documentation?
9.2. Do you provide usage examples?
9.3. What are the goals for the documentation?
9.4. Are you satisfied with the documentation?
10. Is usability considered in the design and development of the API?
10.1. [*If: Yes*] What do you do to ensure good usability?
10.2. [*If: Yes*] Did you consider the user's expertise in cryptography during the design process?
10.3. [*If: Yes*] Does your testing process include tests for usability?
11. We talked a lot about different design decisions for the current API now - do you face any challenges with any of them?

Project Processes.
1. How is collaboration organized in the project?
1.1. How many people work on it?
1.2. Which roles are there?
2. How are design or development decisions made?
2.1. Who makes the final decision?

Initial Design Process.

Applicability: For the following block of questions, I'd like to know if you have been part of the initial design decisions when the library and its API(s) were first created? [*If: No - Skip this section*]
1. Can you describe what the initial design process of your library's API was like?
1.1. What were the initial design goals and requirements?
1.2. Was there a design phase?
1.3. Did you have a project vision from the beginning, or did it grow over time?
1.4. Were past experiences considered?
2. What sources of information or knowledge did you consider in the design process?
3. Did you follow any guidelines, standards, or policies during the design process?
3.1. [*If: Yes*] Which ones?
3.2. [*If: Yes*] How did you choose them?
4. Did you face any challenges during the initial design and development process?

Updating.
1. How do you update your API?
1.1. How often does an API update happen?
1.2. What triggers such an API update?
1.3. Is there a roadmap of upcoming features or changes for the project?
2. Do you stay informed about new developments in API design or usability?
2.1. How do you stay up to date?
3. Is there a deprecation process for parts of the API?
3.1. When and why do you deprecate things?
3.2. How is it done?
3.3. What are your thoughts on the process?
4. What challenges do you face concerning update decisions?

User Interactions & Feedback.
1. Do you know what kind of people and projects use your API?
1.1. Do you know what they use it for?
1.2. Do you know what level of cryptographic expertise they have?
2. Do you get user feedback on your API design?
2.1. What communication channels are used for that?
2.2. How useful is the feedback in your opinion?
2.3. Who do you get feedback from?
2.4. Are you actively looking for feedback?
2.5. Do you have a strategy for evaluating user feedback?
3. How do you react to user feedback?
3.1. Do you reply?
3.2. Do you consider user feedback in future plans for the API?

Challenges and Wishes for the Future.
1. To summarize, what do you think are the biggest challenges when designing, developing and implementing APIs for cryptographic libraries?
2. Is there anything you wish the crypto community or crypto researchers would do or do differently?
3. Assuming no constraints at all (in terms of money, time etc.), what would you like to do differently or to improve in the design or development process?

Appendix B: Meta-Review

The following meta-review was prepared by the program committee for the 2025 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

B.1. Summary

This paper presents a qualitative study investigating how cryptographic libraries' API design and implementation decisions are made.

B.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field
- Addresses a Long-Known Issue

B.3. Reasons for Acceptance

Cryptographic APIs of libraries have been of significant importance historically because of their role in designing secure software. However, software security can be compromised by misusing cryptographic APIs, which can happen for several reasons, such as lack of simplicity, level of abstraction, insecure defaults, and lack of comprehensive documentation.

The paper addresses this long-known issue by investigating how design and implementation choices are made in cryptographic API libraries, thus providing a valuable step toward understanding existing challenges in ensuring the usability of cryptographic library APIs.

B.4. Noteworthy Concerns

One reviewer expressed concern about the generalizability of the findings due to the sample size being 21 for a highly specialized field.

Appendix C: Response to the Meta-Review

Regarding our results' generalizability, we would like to point out that establishing generalizability is not a goal of qualitative interview studies. Rather, qualitative research aims to promote diversity, explore new research areas, and understand participants' inner experiences [85]. Specifically, semi-structured interviews allow for exploring topics and problems relevant to individual participants previously unknown to the researchers while ensuring that a foundational core of topics is covered in each interview [85]. To achieve this, qualitative research typically studies a comparatively small sample in-depth [86]. Following standard best practices for qualitative interview studies, we determined the sample size required to provide such insights and support our findings and conclusions through reaching thematic saturation [58]. Our sample size is comparable to similar studies that employed this method and were published in the security research community (e.g. [62], [87], [88]).