"That's my perspective from 30 years of doing this": An Interview Study on Practices, Experiences, and Challenges of Updating Cryptographic Code

Alexander Krause
alexander.krause@cispa.de

Harjot Kaur
harjot.kaur@cispa.de

Jan H. Klemmer
jan.klemmer@cispa.de

Oliver Wiese
oliver.wiese@cispa.de

Sascha Fahl
sascha.fahl@cispa.de

CISPA Helmholtz Center for Information Security, Germany

Abstract

Keeping cryptographic code up to date and free of vulnerabilities is critical for overall software security. Updating algorithms (e.g., SHA-1 to SHA-512), key sizes (e.g., 2048 to 4096 bits), protocols (e.g., TLS 1.2 to 1.3), or transitioning to post-quantum cryptography (PQC) are common objectives of cryptographic updates. However, previous work and recent incidents illustrate developers' struggle with cryptographic updates. The research illustrates that many software products include outdated and insecure cryptographic code and libraries. However, the security community lacks a solid understanding of cryptographic updates. We conducted an interview study with 21 experienced software developers to address this research gap. We wanted to learn about their experiences, approaches, challenges, and needs. Our participants updated for security and non-security reasons and generally perceived cryptographic updates as challenging and tedious. They lacked structured processes and faced significant challenges, such as insufficient cryptographic knowledge, legacy support hindering cryptographic transition, and a lack of helpful guidance. Participants desired the assistance of cryptographic experts and understandable resources for successful updates. We conclude with recommendations for developers, academia, standards organizations, and the upcoming transition to PQC.

1 Introduction

Cryptography is constantly evolving and is crucial for overall software security. Unlike software in general, crypto[1] implementations—while keeping their functionality—can lose their security guarantees over time, e.g., weaknesses in cryptographic primitives or protocols might decrease security when a vulnerability is discovered. In the past, there was often a need for transitions, meaning that prior solutions had to be deprecated and replaced by new, (more) secure, state-of-the-art crypto [1, 2]. Crypto code is particularly susceptible to weaknesses that emerge over time, often due to the deprecation of primitives or the discovery of vulnerabilities in previously secure algorithms. This differs from general software vulnerabilities, which are often due to implementation errors, e.g., the *log4j* incident [3] in 2021. In contrast, crypto weaknesses often require fundamental changes to the underlying algorithms or protocols, making updates more complex and far-reaching. Crypto transitions and vulnerabilities require updating crypto implementations; we use the term *crypto update* for this.

For example, key length recommendations for RSA and AES increased, and hashing algorithms evolved from MD4 to MD5, later to SHA-1, and finally to SHA-2 and SHA-3 [4]. The SHA-1 deprecation, which took more than a decade, illustrates the long-term nature of crypto transitions compared to typical software patches. Even in the younger history, post-quantum cryptography (PQC) algorithms have, in part, replaced classical crypto to counter the risks associated with the emergence of quantum computers [5]. Such crypto updates are necessary to ensure security in the future and proactively counter "store now, decrypt later" attacks, as well as to mitigate newly discovered vulnerabilities in crypto.

While updating software in general can be challenging, we argue that crypto updates, in particular, are complex and require even more consideration. "Never implement your own crypto" is a common phrase to prevent non-experts from making serious mistakes that put users and their data at risk. For example, developers might choose wrong algorithms, keys that are too short, or weak hashing functions [6–8]. Making the right decisions requires expert knowledge. In 2008, Bernstein et al. introduced the *NaCl* crypto library to improve usability, e.g., by secure defaults that prevent non-experts from having to choose parameters [9]. Another challenge in implementing crypto is that it can be tedious, e.g., when specific standards have to be met in order to pass audits. Last but not least, crypto updates might impact compatibility between different versions of the same software or protocol, which is

[1] For brevity, we use the term *crypto* throughout the paper instead of words like cryptography or cryptographic.

often spread over a complex distributed system. Ott et al. also acknowledge similar challenges referring to *crypto transition and agility* and found this to be a major research gap, especially when considering the upcoming transition to PQC [1]. LaMacchia highlighted that the PQC "*migration will be more involved and complicated than any we have faced in the past*" and that early awareness, education, and planning are essential for a successful update [2].

Crypto transitions might take a long time: Currently, the industry faces the transition to PQC [10]. In 2024, Apple announced the upgrade of *iMessage's* crypto protocol to PQC to protect users from "Harvest Now, Decrypt Later" [11]. Until the first PQC standards from the U.S. National Institute of Standards and Technology (NIST) and the Internet Engineering Task Force (IETF) became final after eight years [12–14], Google tested and implemented an early draft of the X25519Kyber768 PQC algorithm in Chrome. They now have to update again to comply with the final standards [15].

Much research investigates developers' and companies' mindsets, challenges, and experiences regarding updating software components or how developers deal with deprecated APIs [7, 16–19]. We investigate developers' strategies to plan and perform crypto updates, as well as needed improvements to mitigate current challenges. To this end, we explore the following research questions:

RQ1 *[Awareness]* How do developers become aware of (potential) updates of crypto implementations?

RQ2 *[Objectives]* Why do developers update crypto implementations?

RQ3 *[Processes]* What are the developer's processes when planning and performing crypto updates?

RQ4 *[Challenges]* What are developers' experiences and challenges when performing an update of a crypto implementation?

Contributions and Key Findings. We conducted 21 in-depth semi-structured interviews with experienced developers of crypto products who had experience updating crypto implementations. We find crypto updates to be unique and more complex than regular software updates, e.g., outdated hardware plays a role in algorithm choice. While most participants updated their crypto implementation for security reasons, external factors or non-security reasons such as performance also played a role. However, participants lacked structured processes to update their crypto implementations. They experienced several challenges and blockers during the update process, such as legacy backward compatibility, insufficient crypto knowledge, or lack of helpful documentation. We finally identified participants' needs for improving the crypto update process, e.g., the need for crypto experts or resources that non-crypto developers can easily understand. Based on our insights, we conclude with recommendations for different stakeholders, such as developers of crypto libraries, developers using those libraries, standardization organizations, and academic research.

2 Related Work

We discuss related work and highlight our novel contributions in three key areas: (1) updates of software components and crypto, (2) deprecation of libraries and application programming interfaces (APIs), and (3) misuse of crypto APIs.

Updating Software Components and Cryptography. Previous work highlighted the difficulties and implications associated with out-of-date dependencies [20–22]. For example, Derr et al. found that 85.6% of libraries from analyzed Android apps were outdated but could be upgraded by at least one version without modifying the app code [22]. This also applied to the upkeep and updating cryptographic APIs. In 2015, Cox et al. concluded that continuously measuring dependencies increases developers' awareness and that utilizing outdated dependencies increases the likelihood of encountering security issues up to four times [23]. Previous research has concentrated on the evolution of software, outlining the difficulties that developers encounter, like insufficient documentation (which is particularly challenging when there is a high rate of employee turnover), the lack of consistent development practices among teams, and the struggles in predicting and assessing the consequences of alterations [24–26]. Haney et al. identified maintaining backward compatibility with older cryptographic algorithms as the main problem after updating [27]. The organizations studied in this research believed that providing excellent product security and robust cryptographic implementations is fundamental [27, 28]. While previous work presented different insights on updates as part of a larger organizational focus, we focus on cryptographic updates specifically and qualitatively explore them in depth.

Cryptographic Deprecation. The deprecation of crypto is one reason for the need to update cryptographic implementations. Deprecating non-security-related APIs in libraries turned out to be a challenge for library developers as well as for users of libraries [17, 29–31]. Gorski et al. investigated developers' suggestions for good deprecation warnings in cryptographic APIs and identified which security information is considered helpful in avoiding insecure cryptographic API use during development [32]. Ott et al. reported that many cryptographic workshop participants experienced problems with deprecating algorithms that are in active use [33]. Deprecations, such as those of SHA-1, MD4, or MD5, are usually suggested by institutions that define or recommend standards, like the IETF, NIST, or German Federal Office for Information Security (BSI). These deprecations typically stem from known weaknesses in cryptographic algorithms or the recommended key length being too short to increase security preventively. Developers often implement crypto suggested by researchers before a new cryptographic algorithm becomes a standard, which can lead to additional challenges [34]. However, even if a standard exists, it poses challenges for developers, including standard deprecation and updates [35].

In general, the literature indicates that cryptographic deprecation is a difficult problem that necessitates clear and helpful deprecation messages and documentation to assist developers in making informed choices.

Misuse of Cryptographic APIs. Previous research has investigated the misuse of cryptographic APIs. Rahaman et al. developed a code screening tool and found that in Android applications, 95% of the cryptographic vulnerabilities originate from misused third-party libraries [36]. Similarly, Egele et al. found that 88% of Android applications suffered from a misused cryptographic API [37]. Gao et al. used a data-driven approach and found that Android developers who tried to fix API misuses were often unable to make the correct fixes. Additionally, misuse was often reintroduced by later updates, indicating that the initial fix may have been unintentional [38]. Mistakes related to cryptographic APIs did not only occur for Android; other languages were affected as well. For example, Wickert et al. conducted an empirical study by statically analyzing Python projects, revealing that over 50% misused cryptographic APIs [39]. Furthermore, Georgiev et al. found many non-browser SSL implementations to be wholly broken and identified cryptographic APIs as the reason [40]. The security not only depends on the underlying cryptographic algorithm but also on the design and usability of the particular cryptographic libraries. Past studies have focused on the design and usability of cryptographic libraries and their impact on code security. Acar et al. found that simplicity, support of auxiliary tasks, and thorough documentation are critical aspects of secure cryptographic libraries [7]. In a related study, Patnaik et al. identified 16 usability issues and four areas where these libraries fail to implement usability principles: documentation, confusion, postmortem, and compatibility issues [16]. Overall, using crypto presents difficulties due to a lack of user-friendly documentation and implementations.

3 Methodology

To collect in-depth insights into updating crypto implementations and the challenges for developers and other involved professionals, we conducted 21 semi-structured interviews. Interview studies are designed to explore complex phenomena and individual narratives in detail, focusing on rich, contextual data rather than quantitative metrics or statistical generalizations. Figure 1 shows an overview of our study process. We provide comprehensive study materials in a replication package (Section Open Science).

3.1 Recruitment Process

We recruited software developers who had experience in updating crypto implementations from companies, open-source projects, and a freelancer platform. To ensure interviewing developers with different perspectives, we used a mix of the following recruitment strategies [41–45]:

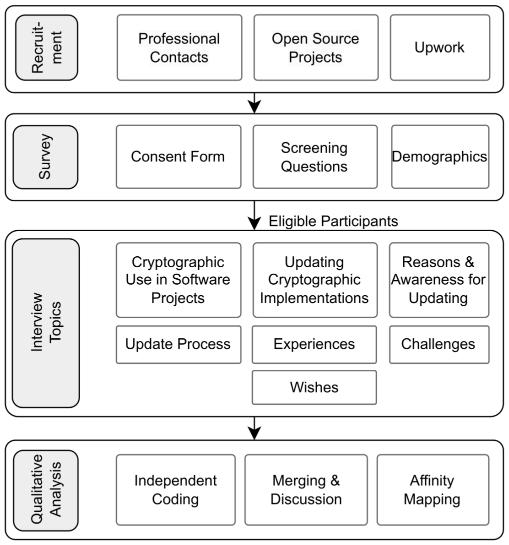

Figure 1: Methodology of our 21 semi-sturctured interviews and their qualitative analysis.

- **Professional Contacts:** We used our professional contacts to recruit experienced developers from companies and freelancers. Some of these participants were active open-source contributors. We used snowball sampling [46] and asked professional contacts to refer us to potential participants meeting our inclusion criteria.
- **Email:** We sent email invites to two groups: (1) Developers of crypto products who made documented contributions to crypto projects: We chose those developers based on an online list of encryption products from a survey conducted through Bruce Schneier's blog [47]. Participants recruited through this list included professionals from companies that provide crypto products and open-source developers, of which we were able to find contact information on their personal publicly available websites. (2) Open-source project developers: To select suitable projects, we collected GitHub projects with activities from January until October 2023 that had the "cryptography" tag. We excluded repositories that did not contain any source code, e.g., collections of documents. We randomly selected projects and searched for active contributors. To adhere to GitHub's *Acceptable Use Policies* [48], we refrained from using contact information from GitHub projects but contacted contributors via public websites external to GitHub.
- **Upwork:** To recruit freelance developers, we posted our job on the freelance platform *upwork.com*, which is recommended for research projects [41].

We asked interested invitees to fill out a short screening survey. We collected demographics about the potential interviewee and background information on their experiences with cryptography and updates of crypto code. Based on this information, we verified participants' answers about experiences with updating crypto code by reviewing their contributions to crypto projects. We thus made sure that, we only invited participants with experience updating crypto code and excluded everyone who did not. The survey explained our study's purpose, and we obtained consent for participation, data processing, and interview recording. We offered each participant a compensation of $60 via PayPal or Upwork, which four participants declined.

3.2 Interview Guide Design

Our interview guide consists of two parts. The first part contains general instructions for the interviewer about conducting an interview and helpful phrases inspired by Rader et al. [49]. The semi-structured interview guide contains 71 questions related to our research questions. These allowed us to ask the same questions and provide the same information to all interviewees. Based on their answers, some participants did not encounter specific follow-up questions. The interview guide defined crypto implementations as crypto protocols, primitives, algorithms, and libraries. We also included a definition of software supply chain security [50].

Piloting and Refinement. After creating the initial interview guide, we contacted experts who had experienced crypto code updates. We further adopted an iterative approach to refine the initial version of our interview guide. This procedure involved conducting three walkthroughs with experienced researchers from our group. To improve the quality of the interview guide, we gathered feedback on the clarity and comprehensiveness of our questions based on the researchers' experiences. Additionally, we used the first two interviews with our professional contacts as pilots. We included these two interviews in our final data analysis since we made only minor changes to the interview guide.

Interview Guide Structure. Below, we outline the structure of our interview guide with brief descriptions of all sections. Appendix B provides the entire interview guide.

The first two sections of the interview include an introduction, a briefing, and questions about the use of crypto in software projects. In these sections, consent to record the interview is obtained again, and we mentioned the possibility of skipping questions and stopping the interview at any time. The first questions contribute to a better understanding of crypto, the tasks, and the participants' roles, and they help in building a friendly, open-minded interview atmosphere.

Before diving into the crypto-update-related questions, we introduce Google Chrome's deprecation process for SHA-1 certificates (Section 1) as an example of updating crypto software. This example aims to remind participants of their own previous experiences with crypto updates. We continue with questions about their **awareness** (RQ1) and **reasons for updating** (RQ2). The next part aims to identify how developers' crypto **update processes** (RQ3) look like. The following questions give insights into the **challenges** (RQ4) and developers' needs and required support. Lastly, we ask for feedback, comments, and information our questions might have missed.

3.3 Interview Procedure

Between July 2023 and April 2024, we conducted 21 interviews with participants from various recruitment channels (Section 3.1). We recruited five participants through professional contacts, four freelancers through Upwork, one participant through snowball sampling, and eleven through sending out emails, resulting in a total of 21 participants.

We conducted most interviews with one researcher leading and the other supporting the interviewing process. The supporting interviewer ensured that the main interviewer did not miss any interview questions and asked follow-up questions when necessary. We conducted the interviews in English[2] via *Zoom* [51] or *BigBlueButton* [52]. We advertised interviews to last about 60 minutes, while the actual interviews lasted a median of 51:14 minutes.

3.4 Qualitative Data Analysis

We used the GDPR-compliant service Amberscript [53] to transcribe the interviews and deleted the recordings after manually reviewing the transcripts for accuracy. We took notes during the interviews and deleted parts from the recording that contained PII or otherwise sensitive information in case our participants requested deletion.

Three researchers leveraged an iterative, semi-open coding approach [54, p. 130] to analyze the transcribed interviews. To begin, two researchers created an initial codebook based on the interview guide and our expert knowledge from related work and conducting the interviews. Two researchers independently coded each interview transcript and merged and discussed their codings after each interview, resolving any conflicts, and iteratively added new codes to the codebook when necessary [54, 55]. We obtained the final codebook by axial and selective coding to reduce the number of codes, and we used thematic analysis [56], [57, pp. 72–75] to identify common themes and topics in our data following established practices in our community [44, 58–60]. All researchers met for a discussion session to identify key themes using affinity diagramming [61]. Due to the exploratory nature of our data analysis [62] and established practices in our field [42–44, 58, 63–65], we decided not to calculate inter-rater reliability. The coding reached inductive thematic saturation after 17 interviews [66]. To ensure data saturation and validate our findings, we conducted four supplementary interviews. Overall, we assigned 1402 codes, with a median of 67 codes per interview transcript. We provide the final codebook online (Section Open Science).

[2]We conducted one German pilot interview due to participant preference.

3.5 Limitations

Our study has limitations typical of qualitative research methods and should be interpreted in context. This includes interview study biases, like self-selection, self-reporting, social desirability, and potential over- or under-reporting of individual experiences. To counter these biases, we stressed that we do not judge participants' answers and allowed them to skip questions at any time, as they preferred. As is typical for qualitative research, the above biases can influence results and might not be generalized to broader populations. We recruited a sample with different backgrounds and experiences and are confident we have gained high-quality insights that explore common practices and challenges around crypto updates in depth. During the interview, we used the SHA-1 Google Chrome deprecation process as an example to set the context when we started talking about participants' experiences with updating crypto code. We may have influenced participants' responses. Although participants discussed their experiences independently, we cannot completely rule out potential bias from using this specific case.

3.6 Demographics

We present a detailed overview of the participants' demographics in Table 1. While we recruited 21 participants with different backgrounds in terms of experience and contributions, we identified two participant groups: 13 participants updated self-implemented crypto implementations (group *S*), e.g., self-designed crypto protocols or primitives in crypto libraries. Twelve primarily used and updated existing crypto implementations in their own software (group *U*), e.g., using a crypto library. Four participants belonged to both groups.

Our participants were software engineers, open-source developers, administrators, and project leaders. They dealt with various crypto aspects in their projects, like encryption, signing, implementing low-level crypto primitives or APIs, crypto libraries, and hashing. They used and implemented crypto protocols, reviewed and maintained crypto implementations, and improved the security of existing crypto implementations. Participants mainly dealt with the following crypto implementations: Advanced Encryption Standard (AES), elliptic curve cryptography (ECC), RSA, HTTPS and TLS protocols, diverse SHA algorithms such as SHA-1, SHA-2, or SHA-256, and PQC. Participants who implemented crypto by themselves generally had a deeper understanding and knowledge of crypto. Those who used existing crypto implementations often had no dedicated background in crypto, nor did they volunteer to handle crypto implementations. Instead, their mindset was that somebody had to deal with crypto, and even their limited knowledge was better than neglecting it.

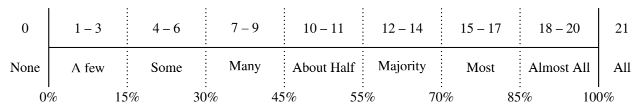

Figure 2: Overview of the quantifiers and percentages used throughout our results. Please note that each quantifier refers to the same number of participants throughout the study.

4 Results

We report the findings of our semi-structured interview study with 21 experienced software developers about their experiences, approaches, and challenges when updating crypto code in software and hardware products. We follow established practices of prior interview studies [42, 67–70] by using the quantifiers from Figure 2 to illustrate the prevalence of our findings and underline the qualitative nature, instead of reporting exact counts.

4.1 Crypto Update Process

Depending on their projects and backgrounds, participants experienced various update scenarios. While developers, who implemented crypto algorithms and primitives, less often updated crypto code, it was a recurring task for those who relied on existing crypto implementations. In contrast, some solo developers only worked on crypto projects as side projects and, therefore, needed more time.

> "*It has taken ten years. However, at that time, because the project was mostly an open-source side project, for sure, there was no full-time work on this. Maybe it's half a year in total.*" — P9

However, the update frequencies varied widely among participants, from once a week, which applied to those who implemented and updated crypto primitives and algorithms in libraries, to once every two or three years for those who worked with other software. Updating used crypto libraries was often perceived as easier and faster than implementing new algorithms or primitives. Crypto transitions took the longest. We also found that those participants who frequently updated crypto implementations did this because of vulnerabilities or as a preventive measure.

Figure 3 gives an overview of the crypto update process we distilled from the interview analysis. Below, we report our findings following the phases of this update process.

4.1.1 Crypto Update Triggers

Our participants considered crypto updates, especially those related to vulnerabilities, critical and timely. For this purpose, they had various information sources and followed different strategies to learn about potential updates. They also reported various consequences of becoming aware, such as rolling back to a previous version because an update was unavailable.

Table 1: Overview of interviewed contributors, their project background, and project metadata. We only report aggregated project metrics to preserve both our participants' and their projects' privacy.

ID	Interview		Demographics						
	Duration	Recruitment Channel	Occupation	Involved Projects	Contributions	Dev. Exp.[1]	Sec. Exp.[2]	Highest Degree	Group[3]
P1	0:35:46	Prof. Network	Software Engineer	11–15	Company, Open Source	14	8	High School	S
P2	1:55:34	Snowball	Software Engineer	>20	Solo, Company, Open Source	35	30	Master	S,U
P3	1:12:02	Prof. Network	Software Engineer, Manager	6–10	Company, Open Source	12	12	Master	U
P4	0:49:11	Email	Software Engineer	>20	Open Source	40	37	Bachelor	U
P5	0:49:05	Email	Head of Research, CISO	6–10	Company, Open Source	20	35	PhD	U
P6	0:34:58	Upwork	Software Engineer	16–20	Company	20	12	Bachelor	U
P7	0:45:46	Upwork	Software Engineer	1–5	Company	20	12	Bachelor	U
P8	1:01:56	Email	Software Engineer	6–10	Solo, Company	7	10	Bachelor	S
P9	0:35:06	Email	Software Engineer	>20	Solo, Company, Open Source	35	23	Master	S,U
P10	1:04:50	Email	Software Engineer	>20	Solo, Company, Open Source	20	5	High School	S
P11	0:47:19	Upwork	CTO	>20	Open Source	20	10	Master	S
P12	0:55:06	Prof. Network	Software Engineer	1–5	Company, Open Source	8	13	Bachelor	S
P13	0:35:06	Prof. Network	Head of Security, Software Engineer	6–10	Solo, Company, Open Source	25	10	High School	U
P14	1:02:33	Upwork	Software Engineer	>20	Solo, Company	20	4	Master	U
P15	1:04:01	Upwork	CTO	16–20	Company	12	3	Master	U
P16	0:49:19	Prof. Network	Researcher	6–10	Solo, Open Source	26	18	PhD	S
P17	0:52:01	Email	PhD Student	6–10	Open Source	9	7	Master	S,U
P18	0:31:12	Email	Researcher	1–5	Solo, Open Source	40	35	PhD	S
P19	0:57:10	Email	PhD Student	>20	Solo, Open Source	6	2	Master	S,U
P20	0:27:28	Email	Research Engineer	1–5	Solo	5	5	Master	S
P21	0:57:17	Email	Student	1–5	Solo	3	0	High School	S

[1] Developer experience in years. [2] Security experience in years. [3] We identified two participant groups: (S) developers who updated self-implemented cryptography, e.g., libraries or primitives; (U) developers using cryptography for their own projects/products.

The majority of our participants learned about a crypto update through colleagues, including new ones joining their team and pointing out security issues, developers, or other experts, e.g., on social media in general or GitHub users in particular. In addition to GitHub issues, one participant said that "*for some reason, quite a lot of people prefer to send emails rather than raise issues*" (P10). A few were notified through reports in their organizations' bug bounty programs or a confidential mailing list. For example, one participant reported being under non-disclosure because their company got "early access information" on vulnerabilities through mailing lists from vendors or security-critical projects.

Specifically for social media, one participant mentioned that this requires "*following the right people*" (P13), i.e., crypto experts, on Mastodon, X, LinkedIn, Slack, or blogs. However, it might be challenging to identify those experts.

While many participants used Common Vulnerabilities and Exposures (CVE) trackers to be informed about vulnerabilities, a few participants noticed that CVE trackers may not cover all vulnerabilities and signed up for mailing lists, e.g., of a crypto library they used, to receive critical information on time. Participants also learned about an update through their Integrated Development Environments (IDEs) (e.g., by getting notified of using an outdated crypto dependency) or other software developing tools they used, like deprecation warnings in a web browser's developer console. During a benchmark test, one participant discovered that their existing crypto implementation was performing well below expectations, resulting in an update (Section 4.1.2).

More generally, a few participants were aware of the problem that today's state-of-the-art crypto might be considered insecure in the future and will, therefore, require an update at some point. P3 stated:

> "*I think, in most cases, [updating crypto is] a known thing that needs to be done. That's in the zeitgeist. No specific notification or event, but it's just like everybody knows SHA-1 is weak, so we must get rid of it at some point.*" — P3

> Key Takeaways: Crypto Update Triggers.
>
> - Peer developers, social media, and GitHub issues were primary sources for crypto-update-related information.
> - Participants relied on CVE trackers and (confidential) mailing lists to receive crypto-related vulnerability notifications.

4.1.2 Objectives of Updating Cryptography

Participants updated their crypto implementations for different reasons. While security is a central consideration for them, they also mentioned other objectives not related to security.

Preventive Security Upgrades. Many developers updated their code to improve security. Some intentionally updated

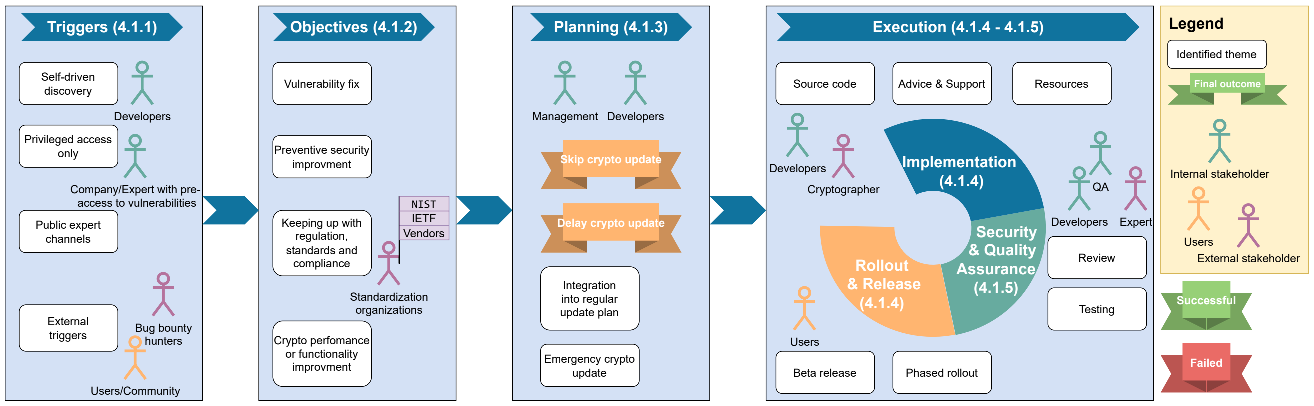

Figure 3: Illustration of the process of a crypto update, divided into six distinct phases: triggers, the objective of updating, planning, implementation, security & quality assurance, and rollout & release, including the involved stakeholders. The outcome of a crypto update process can be either successful, failed, skipped, or delayed.

for preventive reasons, e.g., by increasing the key length or switching to a more secure algorithm, e.g., updating from SHA-2 to SHA-3. One participant fittingly summarized: "*We try to be ahead of the curve.*" (P10). We note that preventive crypto updates are distinct from vulnerability fixing, as the participants' motivation in these cases was to maintain a security margin, e.g., preventively updating to PQC before powerful quantum computers become available.

Vulnerabilities. Besides preventive crypto updates, the other main motivation for a crypto update was to remediate actual security issues. About half of our participants deployed crypto updates to fix security vulnerabilities, e.g., "*We had security issues on a protocol two times, where we had to immediately fix it because it was [vulnerable].*" (P9). Another example is a participant who reported constant attacks due to vulnerable crypto that required remediation through a crypto update:

> "*They were continually attacked by people who reverse-engineered the [...] encryption scheme and managed to use the tools that allowed them to cheat.*" — P8

Crypto Standards. External factors forced many participants to update their crypto implementation. For example, they responded to an implementation's deprecation, e.g., increasing key length due to new recommendations, removing methods from a crypto library, or updating primitives or reference implementations because of changed standards. A few had trouble with crypto standards that changed or with incompatible implementations of a standard, e.g., "*Every now and then, you run into a new implementation that interprets the standards differently from everyone else.*" (P18). Besides changing standards, the participant was concerned about the influence of big industry players that might enforce a *de facto* standard that requires updating crypto implementations: "*If Google decides to update everything in Chrome, [...] the entire world will be updated.*" (P18).

Non-Security Reasons. Non-security reasons for a crypto update were critical for some participants. For example, one participant mentioned performance as their main obstacle and motivation to perform a crypto update for improved performance: "*If there are significant performance upgrades on the library themselves*" (P8). Some other developers updated because new crypto algorithms became available or libraries introduced new features they wanted to use, e.g., "*use of new classes and APIs*" (P12). This includes higher library usability for the developer or a better fit for their use case.

> Key Takeaways: Objectives of Updating Cryptography.
>
> - Security improvements are a central consideration for updates.
> - Preventive updates aim to maintain a security margin for future threats.
> - Vulnerabilities often require immediate fixes through updates.

4.1.3 Planning a Cryptographic Update

Before implementing and rolling out the actual crypto update, participants had various aspects in planning how to approach the crypto update. This mainly concerns the urgency and, therefore, the timeframe in which the update needs to be completed. A possible decision in this step might also be to skip or delay an update.

Many developers systematically and routinely analyzed the urgency of an upcoming crypto update to determine its priority and further steps. The implementation of such updates, however, faces significant constraints and complexities in various scenarios (Section 4.3). In urgent cases, like vulnerabilities, a few participants tried to update their code as quickly as possible, e.g., by deploying emergency updates:

> "*If we have updates that fix critical vulnerabilities [...] we have emergency updates [...] where the fix is done and published immediately.*" — P8

One participant explicitly tried to estimate potential negative consequences, e.g., becoming vulnerable or causing negative effects for customers due to unavailability or incompatibility if the update is not done immediately and decided to update in the regular release cycle instead. More often, some developers incorporated the necessary crypto code changes into their roadmap or upcoming releases. One participant described a strategy depending on breaking code changes as follows:

> "*[We] look at the breaking changes, depending on that, do the [update] [...]. Depending on the size and the differences, [I read] the history and diffs of the actual code changes and then do [the update].*" — P11

In some situations, developers explicitly highlighted the importance of fully understanding the crypto issues in order to prevent future crypto issues:

> "*We were first shocked to see that our product is so vulnerable and we are lacking a lot of security aspects [...]. We started educating and training ourselves about the different security violations that can occur.*" — P6

Decision-Making Process. While some participants mentioned having a pre-defined decision-making process, such as release meetings, to discuss crypto updates, the majority had no dedicated decision-making process for crypto updates.

However, the existing decision-making processes varied widely. Some participants reported that the management would be in charge of this process, and some made update decisions alongside other involved stakeholders, e.g., other developers or management. In contrast, only a few mentioned that a single developer was responsible for the decision-making process. A few specifically stated that a crypto expert (someone with dedicated crypto expertise and education) was responsible.

Without a decision-making process, we found that decisions were made mainly by an individual stakeholder, e.g., developers, without syncing and discussing update decisions with the larger team. This was particularly challenging for solo developers:

> "*There wasn't an actual concrete plan that I was given. I just [...] started by doing research on my own, [...][and] to see which library implements these secure algorithms. [...][I looked] at those libraries to see if they are flagged as deprecated or if they don't have a lot of downloads. If many people are using it, chances are it can't be that bad.*" — P7

Key Takeaways: Planning a Cryptographic Update.

- Many developers lacked a structured process for crypto updates.
- Crypto updates' urgency and timeframe varied based on developers' project-related factors, such as team size or used crypto algorithms.
- Decision-making processes ranged from individual choices to team discussions.

4.1.4 Implementing a Cryptographic Update

Participants followed diverse update implementation processes, mostly due to the different nature of their projects, e.g., some are rolled out to customers and end-users, some are crypto libraries, and some have requirements in their companies. Some participants reported quality assurance processes (Section 4.1.5), including testing their update in development environments before deploying it.

Ultimately, a few updates included switching crypto libraries if a library does not implement fixes—even if participants generally tried to avoid this additional work:

> "*If there's a bug that's found in a library like a timing attack or some stuff is left hanging around in memory, and we find it and the developers are not really responding to fix it or apply it to their upstream. In that case, we try to move to a different library that's more reliable, but we try in the first place to vet at the start if the library is reliable or not.*" — P17

Involved Stakeholders. Almost all participants were in charge of the crypto update process and implemented the update independently of others. About half could rely on other software developers on their team to support them in performing the crypto update. While many participants reported their projects or companies consulted external experts, some were supported by internal experts, e.g.,

> "*I'll rely on experts for our companies as internal experts for crypto, and external experts for really hard math or practical crypto analysis.*" — P12

In a few cases, the management was involved, mostly in approving major crypto changes. Solo developers generally receive no support and have to take on all the roles themselves.

Information Resources. Many participants who relied on resources besides experts mostly searched online for guidance on planning and implementing a crypto update. A few exchanged experiences and engaged in discussions in "*developer discussion forums or other places [like] Reddit [or] StackOverflow*" (P15), not necessarily to discuss implementation details. Others reported considering academic publications, but only when implementing novel crypto primitives when other information resources are not available. However, scientific publications were often reported to be complicated, hard to understand, and time-consuming to read:

> "*I also tried to check out several academic [...][papers]. I only hold a bachelor's degree in computer science and haven't specialized in [...][crypto]. Well, it was a bit more time-consuming to understand the actual research papers.*" — P7

Although participants used standards for the crypto update decision, no participant mentioned consulting those standards when implementing the actual crypto update. Potential explanations might be that participants did not mention them because it was too obvious or because standards provide little actionable guidance. For developers using crypto libraries, a few participants mentioned reading the libraries' changelogs to get update guidance (e.g., on breaking changes that might impact compatibility). One developer contacted a processor manufacturer to understand why their crypto implementation did not work on a specific processor.

Required Time and Effort. We observed differences regarding the duration and required effort when performing crypto updates. Some update processes took over a decade, while others took more than a year, e.g., "*the RSA deprecation [...] took five years*" (P4). Many participants reported that updates took multiple months or weeks; some mentioned that fixes could be implemented in several days or less. The latter is typically the case for small and less complicated crypto updates, as P2 explained: "*Oftentimes, the fix is obvious. It's one or two lines. It's an easy change to make, deploy, and test internally.*" (P2). A few participants reported on more user-oriented methods, including beta phases or *phased rollouts*, which allowed developers to collect detailed feedback on the crypto update's functionality and security before rolling out the update to a larger group of users. Regarding technical effort, participants mentioned the need for code reviews (Section 4.1.5), working with release managers on emergency updates, and performing major rewrites of their code to ensure compatibility whenever the dependent crypto library was updated. Another contributing factor is the extent of necessary code changes and several challenges in the update process (Section 4.3).

> Key Takeaways: Implementing a Cryptographic Update.
>
> - Implementation processes varied widely based on project's nature and requirements.
> - Most participants managed the update process on their own.
> - About half could rely on other team members for support.

4.1.5 Crypto Update Security & Quality Assurance

Almost all participants reported some code security and quality measures and considered supply chain security when implementing crypto updates. However, the extent of these measures varied largely. Only a few participants reported the absence of any security and quality management practices:

> "*With that project where I'm co-developing, it's full trust. There's no review because we trust each other that the code that we are committing is good enough.*" — P16

Testing and Analysis. Even if a QA department was not present, participants used various techniques and measures to ensure crypto update quality. The majority used automated testing, including unit tests. In a few cases, the participants reported utilizing static code analysis, vulnerability scans, fuzz testing, or formal verification as specific security measures. Additionally, some participants engaged in penetration testing, either internally or externally. Some tested for non-security metrics like performance, e.g., execution time. Others relied on external software audits, compatibility assessments, or dependency checks. However, not all participants had structured QA. Especially in those cases, many participants mentioned conducting basic functionality testing, e.g., by running the software and manually debugging it.

Code Review. Many participants engaged in code reviews for their crypto updates with varying strategies. However, the strategies varied in how strict reviewing policies were. A few participants adhered to a six-eye rule, as they argue that only a second person reviewing code is not sufficient for security-critical aspects like crypto:

> "*We have a six-eye rule because we are a security company. If I make any changes, at least two other people have to review and approve the changes, always.*" — P12

Similarly, to increase the thoroughness of their review, a few participants reviewed crypto source code printed on paper. Generally, our impression was that participants put extra scrutiny and time into reviewing crypto code.

Supply Chain Security. Developers had different levels of awareness and employed various strategies to manage and mitigate risks associated with third-party components from their software supply chains. About half of the participants generally used third-party dependencies, with some specifically mentioning third-party crypto libraries.

About half of the participants knew of supply chain security. They followed various strategies to mitigate risks introduced through their software supply chains. While a few used package managers to directly include third-party crypto libraries, many participants reported forking existing crypto libraries for their own projects or compiling them from source to prevent issues with the update, e.g., by waiting before using the novel upstream version:

> "*We make a fork of the same version of the snapshot of that library. Then we compile it ourselves, and we integrate it into our projects. We always have a mirror-reflected version on our end.*" — P8

A few participants modified third-party libraries, primarily due to compatibility issues with their software. Additionally, a few participants verified third-party components before use, e.g., by checking the dependencies that libraries relied on or required signed packages or certified manufacturers.

> Key Takeaways: Security & Quality Assurance.
>
> - Almost all participants reported some code security and quality measures, e.g., automated testing, including unit tests.
> - Many engaged in code reviews, with some following strict policies like a six-eye rule.
> - Supply chain security awareness and mitigation strategies varied among participants.

4.2 Uniqueness of Cryptographic Updates

An overarching theme across interviews is that participants generally consider crypto updates unique in several aspects compared to general software updates. Participants' update processes varied, as they performed crypto updates in various contexts, such as updating primitives, a library they maintain, or a crypto dependency. Only a few participants reported handling crypto updates and regular software updates the same. Some participants reported that updating crypto implementations requires more effort than regular software updates and is more critical:

> "*If you don't update, for example, your PDF library, you may have a logo positioned wrongly. But if you don't update a crypto library, you may have your data leaked.*" — P13

Some participants highlighted the need for detailed review, while others relied on a crypto library that first needed to be reviewed and approved by a governmental department. A few also found crypto updates more challenging, e.g., because of having to consider legacy support:

> "*We have to narrow down to decide what those legacy devices support. [...] Eventually, we go and say, "We're not going to support that crypto algorithm because it's forcing that vendor to actually fix and replace that software.""* — P4

> Key Takeaways: Why Cryptographic Updates Stand Out.
>
> - Participants considered crypto updates unique, more critical, and more demanding than general software updates.
> - Crypto updates required more effort and detailed review.
> - Legacy support considerations and backward compatibility complicated crypto updates.

4.3 Challenges of Updating Cryptography

Developers faced several challenges in multiple phases when attempting crypto updates. We identified 13 challenges that we illustrate in Table 2 and describe below in detail. Due to these challenges, many participants reported failing to update their crypto code at least once.

Problems Complicating Updates. Overall, we find that unstructured crypto update processes (C6) are an overarching issue that complicates crypto updates. The majority of developers mentioned backward compatibility issues (C4). Specifically, a few participants highlighted problems with legacy code (C5), crypto implementations, and systems. As a reason for having to consider legacy support, a few participants noted the long lifetime of embedded systems, which complicates updates (C4):

> "*One problem is that a lot of the stuff is used in embedded systems and basically systems that don't change much. You've got 10 or 15-year-old implementations.*" — P18

Many participants reported that implementing crypto updates led to non-functional software (C11), e.g., due to implementation bugs, incompatibilities between client and server, or changes of supported algorithms. A few participants mentioned that updates affected other applications, leading to additional complications (e.g., a crypto library used by multiple internal software products led to breakage). Additionally, a few pointed out a lack of information (C2) as a significant challenge. A few participants were concerned that crypto primitives or protocols might contain backdoors (C10):

> "*When updating crypto, my main concern is to be sure that the crypto primitives or protocols do not contain backdoors for us, which is very important.*" — P5

Others mentioned problems with staying up to date (C1) with CVE trackers, noting that it required too much effort to filter out which CVEs affected the project's security.

One participant mentioned that the crypto update changed the workflow in the application and, therefore, affected the user interface for password changes, which annoyed customers (C12).

Delaying Updates. Participants reported various reasons for delaying crypto updates. Some were unsure about the update process or responsible parties due to a lack of organizational structures within the company or team, while others postponed the crypto update until absolutely necessary. A few expressed concerns about backward compatibility (C4), making them hesitant to proceed. One participant cited the task's magnitude and lack of time or priority, particularly in open-source or hobby projects. This issue also stemmed from resource constraints within companies, e.g., missing support through available experts. Finally, delaying a crypto update could also result in skipping it, because an even newer version, algorithm, or primitive became available.

Skipping Updates. A widespread reluctance to implement crypto updates emerged among developers, driven by concerns about customer pushback, security trade-offs, and performance impacts. Some cited customer resistance to upgrades, while others only updated when necessary for security, especially regarding third-party libraries. Performance concerns (C13) also played a role, with one participant noting worse performance and a lack of algorithm support in a new version. Another participant mentioned skipping standard updates to maintain long-term stability for users, given the frequent changes in standards.

Blockers Preventing Updates. Besides general challenges, participants also encountered several hard blockers that could prevent crypto updates entirely. Many participants mentioned difficulties in understanding and learning crypto (C7), e.g., due to the complexity of crypto:

> "*While Bruce Schneier's book, the* Introduction to Cryptography, *made it seem like there are prepackaged blocks you can just connect together like Lego, this is certainly not true. [...][For example, there are] edge cases you have to be aware of.*" — P9

Some reported that the absence of crypto experts (C8) in their teams posed significant challenges: "*As a small company, you don't have the [...][crypto people] to do crypto ourselves.*" (P10). One participant shared that their implementation was vulnerable, but the client did not have the funding to commission a crypto update. The frequent deprecation of crypto standards and their rapid evolution (C3) created significant obstacles for many developers trying to maintain up-to-date implementations. Furthermore, many were frustrated by documentation problems (C9), with some noting inadequate or missing documentation. Academic publications, though recognized for their high quality, were often viewed as too complex for practical implementation, while *Request for Comments (RFCs)* received mixed reviews—generally more accessible than academic publications but varying significantly in their developer-friendliness. One participant mentioned that documentation often was not tailored for inexperienced developers, while another found multiple library options and configurations confusing without crypto expertise (C8).

Table 2: Identified challenges from the interviews mapped to the cryptographic update process phases and their relative prevalence.

Challenge	**Description**	**Prevalence**
Crypto Update Triggers		
C1: Staying Up to Date	Difficulty in keeping track of and filtering relevant CVEs and how to follow the right cryptographic experts.	●○○
C2: Lack of Information*	Insufficient or unclear information about updates and their implications.	●●●
Crypto Update Objectives		
C3: Changing Crypto Standards	Changes in existing crypto standards and the emergence of new standards make it hard to keep up.	●○○
Planning		
C4: Backward Compatibility of Cryptography	Maintaining compatibility with old systems (e.g., embedded devices) complicates crypto updates and agility.	●●●
C5: Legacy Support	Dealing with legacy code, cryptographic implementations, and systems makes updates difficult.	●○○
C6: Lack of Structured Processes*	Absence of well-defined, structured processes to approach cryptographic updates.	●●●
Implemention		
C7: Understanding Cryptography*	Difficulty in learning and understanding complex cryptographic concepts.	●●●
C8: Lack of Cryptographic Expertise*	Absence of cryptography experts in teams, especially in smaller companies.	●●○
C9: Documentation Issues	Poor, missing, or hard-to-understand documentation, including academic publications.	●●●
Security & Quality Assurance Measures		
C10: Trust in Third-Party Cryptography*	Challenges in verifying and trusting third-party cryptography and its implementations.	●●●
Rollout & Release		
C11: Technical Issues	Cryptographic updates can cause breakages in the current application or affect other applications.	●●○
C12: User Interface Changes	Cryptographic updates can affect user interfaces, potentially annoying customers.	●○○
C13: Performance Issues of Cryptography	Cryptographic updates sometimes result in worse performance, leading to skipped updates.	●○○

Challenges marked with * were also present in other phases of the crypto update process to a lower degree.

Failing Updates. Participants described failed crypto updates as changes to cryptosystems that lead to issues during or after deployment. These failures stemmed from various factors, including implementation errors (C11), compatibility problems (C4), and security vulnerabilities. Some participants reported incompatibilities with updated libraries, while others discovered bugs post-release.

Failed crypto updates invariably required the update's rollback, triggering time-consuming incident response processes for many participants. For example, one participant noted that they did not sufficiently test (Section 4.1.5) their updated certificate validation, which contained a bug (C11), and therefore rolled it back before releasing an improved update later:

> "*We just said, no, no, it's easiest to back it out now; we will fix this and just do it again on the next release cycle. [. . .] We got to roll that back. When something has been found out after release that we did not catch beforehand.*" — P2

Some updates also failed due to performance issues (C13) or compatibility problems (C4) with older devices.

> Key Takeaways: Challenges of Updating Cryptography.
>
> - Backward compatibility was a major challenge.
> - A of lack of crypto expertise and structured processes added to the complexity of crypto updates.
> - Technical issues, a negative performance impact, and an impact on end-user usability can lead to failed or delayed updates.

4.4 Wishes for Cryptographic Updates

In the interviews, we asked developers about their wishes and suggestions for improving crypto code updates.

Awareness, Education, and Practice. Many participants wished for more awareness, education, and practice to update crypto implementations, especially those who updated crypto for the first time. For example, one participant reported that new employees get a crypto onboarding. Similarly, a few developers emphasized the importance of specialized training to ensure code safety and security. They also stressed the need for continuous education to stay updated, recognizing the field's constant evolution: "*Crypto implementations are getting enhanced and refined every passing day, so [developers] need to be aware of new challenges.*" (P6).

Need for Cryptographic Expertise. Besides raising awareness and providing education, some participants identified the need for dedicated crypto experts or staff within their teams having the needed background in crypto. These experts should provide specialized knowledge and support, ensuring crypto updates are implemented correctly and efficiently. Overall, many participants often felt insufficiently prepared to implement and deploy crypto updates.

Resources and Documentation. Many participants suggested extending and improving documentation, for example, by adding more examples or more precise error messages, making them developer-friendly. Moreover, a few participants expressed the need for more examples and resources on the internet, such as better access to fixes and information on vulnerabilities. Finally, a few participants suggested that standards should provide accessible change logs.

API and Library Improvements. A few developers advocated streamlining crypto libraries to better support non-experts, emphasizing the criticality of stable APIs, and also complained about usability:

> "*Any crypto primitives and libraries already have a very high bar to entry, both due to complexity and because programmers have been told for decades that they shouldn't touch these things unless they know exactly what they're doing.*" — P11

Consequently, they suggested that algorithm, standard, or library updates should have minimal impact on developer usability. Initially, selecting a reliable crypto library is crucial, as participants reported that poor choices lead to challenges during the update process, including breakage and backward incompatibility (Section 4.3).

Testing and Verification. Other participants mentioned the need for easier testing and verification of crypto updates. A few participants indicated that cryptanalysis and formal verification should be more accessible or stressed the criticality of penetration testing of crypto implementations and updates. Some participants emphasized the need for increased automation in testing and vulnerability detection:

> "*An eternal wish for every security developer is a better-automated analysis of the code to try and detect vulnerabilities.*" — P18

Best Practices. Regarding approaching crypto updates in general, many participants emphasized the need for well-defined best practices for a structured crypto update process due to the problems they encountered and lacking best practices: "*We didn't have any real best practices regarding security audits or security updates.*" (P7). However, while participants desired best practices, they did not propose what such best practices should entail, except that these would need to be clearly specified and communicated.

> Key Takeaways: Wishes for Cryptography Updates.
>
> - Many participants wished for better awareness, education, and practice with updating crypto.
> - Dedicated team members with crypto expertise and improved documentation and resources were desired.
> - Easy-to-use, up-to-date, and well-documented crypto libraries and enhanced testing tools were suggested.

5 Discussion

Below, we discuss our results in the context of our RQs, provide recommendations, put our work in context with related work, and compare crypto with security and general software updates to illustrate their distinct characteristics.

Awareness (RQ1). While crypto updates can be highly security-critical, especially for vulnerabilities whose fixes are time-critical, most participants lacked information sources to stay up to date with crypto updates. They tried to utilize various channels. However, many participants were mainly informed by peers, e.g., through mailing lists or colleagues and community members (Section 4.1.1). One participant mentioned to "*follow the right people*" (P13), which may be hard to find. Even though national agencies or standardization organizations, such as NIST, provide information about upcoming cryptographic updates [71], e.g., deprecations and key lengths, our participants did not consider those. This confirms the limited relevance of national agencies and standardization organizations for software development practices [72]. Despite security shortcomings [73], our participants relied on social media and online forums (e.g., *StackOverflow* or *Reddit*) instead. Our findings suggest that many developers have limited awareness of the cryptographic lifecycle. Hence, we suggest better tool support, such as static code analysis for cryptographic implementations, to identify and notify developers in case their cryptographic code is vulnerable or outdated [74]. Additionally, the wider adoption of existing solutions, for example, IDEs that warn developers when they use deprecated crypto defaults or deprecated and insecure algorithms [75, 76], seems promising.

Objectives of Cryptographic Updates (RQ2). We identified several objectives for crypto updates (Section 4.1.2). Vulnerability fixes and preventive updates were the most important. Vulnerability fixes require immediate action by updating the implementation and releasing the fix. Preventive updates aim to improve security for future security issues (e.g., advances in crypto analysis or quantum computers [77]). Besides that, changes in crypto standards and other non-security reasons, like crypto library performance, features, or usability, are reasons for a crypto update. While all these are reasons to update crypto implementations, not all updates are deployed, e.g., P8 did not perform a crypto update because it did not improve security. Finally, the question remains whether developers and organizations see the need for a crypto update or not. With this question in mind, several challenges (RQ4) and a lack of incentives might prevent potential crypto updates.

Cryptographic Update Processes (RQ3). While all participants described their approach to crypto updates, we could not identify structured update processes. Due to the perceived criticality of crypto updates, what seems to be structured—but also pertains to the general software development lifecycle—are security code reviews and software testing (Section 4.1.5). Hence, our results align with Haney et al.'s who found "rigorous development and testing practices" among organizations that develop cryptographic products [28].

Given the many challenges (Section 4.3) and especially the criticality of crypto updates, structured processes are essential for improved security. A structured process could also ensure preventively updating crypto implementations ahead of time, e.g., increasing key sizes or switching to PQC, before becoming urgent security issues [6, 78]. However, our participants described highly individual crypto update experiences and challenges. Hence, one process might not fit all needs.

The involvement of stakeholders was highly individual for the participants' projects. While some participants worked in a larger team, others were solo-developers. Commonly, participants called for team members with dedicated crypto expertise—often finding their personal crypto knowledge insufficient [6, 7]. Unfortunately, this might be unrealistic for solo developers or projects with limited human or monetary resources [79]. A typical pattern reported was having a security or crypto advocate (often our interviewee) who usually initiated and performed the crypto update [28]. Without someone taking this role, crypto code updates did not seem to happen. Therefore, we recommend assigning this role explicitly within a team. Fischer et al. identified similar issues for the crypto ecosystem stakeholders [34].

Challenges (RQ4). Our participants described many challenges when updating crypto implementations (Section 4.3). Backward compatibility (C4) was the major challenge for our participants; consequently, some decided to delay the update due to compatibility considerations. In case of preventive updates, developers depend on their customers and (legacy) hardware, having to consider both and backward compatibility. However, it is even more challenging that their customers will only notice an update if it fails because otherwise (without an update), the software continues to run. Another challenge is trust in code and third parties (C10). One participant mentioned that developers must reconsider every update of a third-party library, as it might contain a backdoor [80, 81] or vulnerability [82]. One strategy is to skip updates if they are not (security) critical. Consequently, developers might skip multiple updates, potentially making necessary future updates more complex. Trust in the software supply chain is a general challenge [83]. When updating a crypto library, developers should be aware of software provenance, e.g., code from new developers or new standards for parameters or algorithms.

Participants developing crypto libraries have delayed updating to implement standard changes because they believe that standards change too frequently to incorporate every modification. The IETF publishes hundreds of RFCs and thousands of drafts every year [84]. This makes keeping up hard for developers and libraries, so we can support similar findings on challenges surrounding crypto standards [35]. In that sense, challenges through crypto standard updates seem to translate to challenges in updating crypto implementations.

5.1 Recommendations

Our results imply that external factors primarily trigger crypto updates. Identifying the right information sources for crypto updates is challenging, and there seems to be no single information source for all developers. However, we recommend that junior developers (especially solo developers) working with cryptography find and follow crypto experts that are critical for their projects, e.g., on blogs, social media, GitHub, or mailing lists. For those with more crypto expertise, we recommend sharing their rationales for and experiences with crypto updates. Established developers could share their professional network such that new developers can "*follow the right people*" (P13) more easily. Since software projects are diverse, suggesting an overall update process is difficult; however, we encourage developers to establish processes for long-term preventive updates, e.g., crypto transitions, and to provide update roadmaps for affected stakeholders.

Consider Cryptographic Updates Periodically. As discussed before, cryptographic advances require deprecating crypto algorithms, primitives, and protocols regularly and updating them with state-of-the-art crypto for improved security. We recommend regularly revisiting crypto implementations to check whether they need to be updated and not miss a necessary transition (e.g., from SHA-1 to SHA-3). Besides newly discovered vulnerabilities, crypto and related standards evolve over years or decades (Section 4.1). Checking for updates every few months or years is likely sufficient when updating a crypto standard. We recommend that crypto is a permanent technical liability that developers should treat as a recurring task.

For Developers Using Cryptographic Implementations. As participants experienced major challenges with backward compatibility and breakages after a crypto update (Section 4.3), we suggest considering potential crypto updates in the initial project design phase. Ideally, developers create a plan for approaching future updates to reduce the effort required and enhance flexibility, i.e., crypto agility [85]. When starting a new project, developers should select reliable crypto libraries with robust APIs. They should examine different libraries and review their histories and any issues reported by other developers to avoid challenges when updating the crypto implementation in the future (Section 4.3). By incorporating this into the initial design, developers can ensure their software's crypto remains updatable, reducing the time and resources needed for future crypto updates and allowing swift responses to emerging threats. Finally, as crypto updates are perceived as challenging and tedious, developers should initially use state-of-the-art and no outdated crypto in their projects to prevent having to update in the near future.

For Developers Providing Cryptographic Implemenations. Secure defaults for crypto libraries are critical, as they support non-experts to use crypto securely (Section 4.4), e.g., without having to choose parameters. While this is desirable, the defaults and deprecated algorithms also might need to be updated at some point. Changing defaults or deprecating algorithms also affects the crypto libraries' code bases, which can be an indicator of up-to-date crypto libraries, e.g., test cases with deprecated certificates:

> "*We wanted to remove support for MD5, and all old tests had been written using MD5 15 years ago. We had to regenerate all the tests so that we could get rid of MD5.*" — P2

To prevent this, we recommend leveraging test suites before performing a crypto update. Otherwise, the update might cause issues for downstream developers using the library. This example outlines that updates should be considered early in a library's cryptographic lifecycle and communicated to its users. We recommend that library developers document changes and provide actionable update recommendations, e.g., in changelogs, as participants suggested. However, manually checking changelogs might be too tedious for downstream library users. Instead, automatic recognition and performing updates (semi-)automatically is desirable. For example, a large-scale tool similar to GitHub's *Dependabot* that recognizes outdated crypto and proposes an update.

For Upcoming PQC Updates. Updating to post-quantum crypto poses significant challenges due to the generally limited experiences with PQC. However, experts expect the transition to be time-consuming and complex. Multiple updates and adaptations are likely as PQC standards evolve. Consequently, developers should plan for significant resources to accommodate the PQC transition by, e.g., reading and applying documents from NIST [86], because those are often complicated and unclear when it comes to actionable recommendations.

For Cryptographic Standards. Participants requested change logs for standards, which would be highly beneficial for developers who create reference implementations or maintain crypto libraries. Hence, we support a similar recommendation by Huaman et al. [35]. This would help developers stay informed about updates and better support their work by clearly showing what changes have occurred. For example, the IETF Datatracker provides a diff-style view for corrected errata. For *updated* or *outdated standards*, such a diff does not exist and might not make sense as they are different RFCs, but they require detailed update notes to summarize changes. Additionally, participants considered *StackOverflow* rather than crypto standards. For instance, P18 said:

> "*one of the problems with a lot of these standards is that there might be things the latest thesis was like 130 pages long, including things that no one will ever use.*" — P18

Others mentioned a lack of education to follow and comprehend crypto standards (Section 4.3). We therefore recommend that crypto standard documents should be shipped with actionable guidelines, simplified explanations, and implementation examples to prevent developers from using *StackOverflow*, increasing the risk of vulnerable implementations.

5.2 Comparing Cryptographic Updates to Regular and Security Updates

Overall, crypto updates can be considered a special case of security updates and share some commonalities with general and security software updates. Nonetheless, updating crypto presents extra obstacles that must be taken into account in comparison to security and general software updates [24–26] (Section 2).

First, crypto updates need more review and testing to identify and prevent subtle vulnerabilities (Section 4.2, Section 4.1.5). Especially for crypto libraries, misuse is a significant challenge and is known to weaken security [7], and it might also happen in updates. Introducing bugs in regular software updates might have no or less severe security consequences.

Second, crypto implementations likely require updates due to future deprecation of current state-of-the-art crypto. Such updates are mainly long-term, e.g., transition to PQC, and require skill and knowledge, e.g., to implement new cryptographic approaches. This also means to plan cryptographic updates carefully. Estimating consequences of crypto and security updates requires specialized knowledge. In both cases, developers' fears of breaking software can lead to delayed or skipped updates [87].

Third, updating crypto means deprecating crypto, while legacy support might be critical at the same time (C4 & C5 in Table 2). Backward compatibility is crucial for ensuring functionality and a challenge for all types of updates—including crypto updates, as our participants mentioned (Section 4.3). For crypto specifically, hardware constraints like limited CPU power may force the continued use of older, weaker algorithms when updates are impossible. Legacy systems often hinder timely crypto updates. Early planning and crypto agility are crucial for smooth transitions during device replacements or upgrades [1].

6 Conclusion

We conducted 21 semi-structured interviews with experienced developers to study their experiences and challenges around updating crypto implementations. They reported that updating crypto code was complicated due to their limited crypto expertise, legacy support issues, and missing helpful and actionable documentation. A cornerstone for a successful crypto update was the availability of crypto experts and documents comprehensible for non-crypto-experts.

Our results illustrate technical challenges and challenges in the broader development life cycle. The benefits of crypto updates were hard to measure and unclear, especially for long-term precautionary crypto updates (e.g., the transition to PQC). Hence, many participants reported that organizations or developers might skip or delay crypto updates. Some participants consult academic publications for updates, despite the process being time-consuming. Making crypto research more accessible could improve its integration into products. Many crypto implementations will eventually become insecure, requiring projects to transition to PQC. This will affect many developers and users—it will likely be the most significant precautionary crypto update in the next decade. Our results demonstrate that updating crypto implementations is a regular and challenging task. Hence, better developer support for the PQC transition is essential for improving future crypto implementations and aids broader software security.

Acknowledgments

We sincerely thank all interviewees for their valuable support and contributions to our research. We appreciate the constructive feedback from the anonymous reviewers and our shepherd.

This research was funded by VolkswagenStiftung Niedersächsisches Vorab (ZN3695). Any findings and opinions expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

Ethics Considerations

Our study was approved by the ethical review board (ERB) (IRB equivalent) of our organization. Our research is consistent with the standards defined in the *Menlo Report* [88]. All collected data was processed according to the GDPR. We explicitly stated the information about data handling to our participants. In addition, we obtained informed consent from all participants in the pre-survey and reconfirmed their permission for both the interview and the audio recording before starting the interview. We always allowed the interviewees to skip questions or end the interview anytime for any reason. If participants did not read the consent form initially, we repeated this information directly before the interview study. The pre-survey consent form, which also served as the written consent form for the complete study, is part of the replication package (Section Open Science). If interview recordings contained personally identifiable or sensible information, we manually cut out the part of the interview that contained it before handing the recording over to the GDPR-compliant transcription service.

Open Science

In accordance with ethical research practices and data protection regulations, we have chosen not to share the full interview transcripts as part of our dataset. This decision is rooted in our commitment to safeguarding participants' privacy and upholding their right to data protection. The sensitive nature of the information shared during interviews, which often includes personal experiences and opinions, necessitates a cautious approach to data dissemination. Maintaining participant confidentiality is paramount in qualitative research, as it fosters trust and encourages open, honest responses. By withholding raw transcripts, we mitigate the risk of participant identification through contextual details or unique phrasing, even if names and obvious identifiers are removed. Rather than providing entire transcripts, we offer our findings via carefully curated, anonymized quotes and thematic analyses. This approach enables us to effectively communicate the core of our research while honoring the privacy and dignity of our participants.

Availability. To support transparency, replication, and meta-research, we provide the following research artifacts in an online replication package: (1) The consent form, (2) the pre-questionnaire, (3) the interview guide, (4) the codebook, and (5) the recruitment materials. The artifacts can be found at: https://doi.org/10.6084/m9.figshare.25975120

References

[1] David Ott, Kenny Paterson, and Dennis Moreau. "Where Is the Research on Cryptographic Transition and Agility?" In: *Communications of the ACM* 66.4 (Mar. 2023), pp. 29–32.

[2] Brian LaMacchia. "The long road ahead to transition to post-quantum cryptography". In: *Communications of the ACM* 65.1 (Dec. 2021), pp. 28–30.

[3] *CVE-2021-44228*. https://nvd.nist.gov/vuln/detail/CVE-2021-44228, as of January 30, 2025. 2021.

[4] *SHA-3 standard:: permutation-based hash and extendable-output functions*. Federal Information Processing Standards Publications (FIPS PUBS) 202. National Institute of Standards and Technology (U.S.), 2015.

[5] Emily Grumbling and Mark Horowitz, eds. *Quantum Computing: Progress and Prospects*. National Academies Press, Mar. 2019.

[6] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel von Zezschwitz, and Matthew Smith. “"If you want, I can store the encrypted password": A Password-Storage Field Study with Freelance Developers”. In: *Proc. 2019 CHI Conference on Human Factors in Computing Systems*. CHI '19. Glasgow, Scotland, UK: ACM, 2019, pp. 1–12.

[7] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L Mazurek, and Christian Stransky. "Comparing the usability of cryptographic apis". In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2017, pp. 154–171.

[8] Yasemin Acar, Christian Stransky, Dominik Wermke, Charles Weir, Michelle L Mazurek, and Sascha Fahl. "Developers need support, too: A survey of security advice for software developers". In: *2017 IEEE Cybersecurity Development (SecDev)*. IEEE. 2017, pp. 22–26.

[9] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. *NaCl: Networking and Cryptography library*. Version 2016.03.15. 2008. URL: https://nacl.cr.yp.to/index.html (visited on 05/10/2024).

[10] US Communications Sector Coordinating Council. *The Engineer Who Cried Quantum*. https://www.comms-scc.org/wp-content/uploads/2023/08/The-Engineer-Who-Cried-Quantum2.pdf, as of January 30, 2025. July 2023.

[11] Apple Security Engineering and Architecture (SEAR). *iMessage with PQ3: The new state of the art in quantum-secure messaging at scale*. https://security.apple.com/blog/imessage-pq3/, as of January 30, 2025. Feb. 2024.

[12] *Module-lattice-based key-encapsulation mechanism standard*. Federal Information Processing Standards Publications (FIPS PUBS) 203. National Institute of Standards and Technology (U.S.), Aug. 2024.

[13] *Module-lattice-based digital signature standard*. Federal Information Processing Standards Publications (FIPS PUBS) 204. National Institute of Standards and Technology (U.S.), Aug. 2024.

[14] *Stateless hash-based digital signature standard*. Federal Information Processing Standards Publications (FIPS PUBS) 205. National Institute of Standards and Technology (U.S.), Aug. 2024.

[15] David Adrian, Bob Beck, David Benjamin, and Devon O'Brien. *Advancing Our Amazing Bet on Asymmetric Cryptography*. https://blog.chromium.org/2024/05/advancing-our-amazing-bet-on-asymmetric.html, as of January 30, 2025. 2024.

[16] Nikhil Patnaik, Joseph Hallett, and Awais Rashid. "Usability Smells: An Analysis of Developers' Struggle With Crypto Libraries." In: *SOUPS@ USENIX Security Symposium*. 2019.

[17] Romain Robbes, Mircea Lungu, and David Röthlisberger. "How do developers react to API deprecation? The case of a Smalltalk ecosystem". In: *Proc. ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. 2012, pp. 1–11.

[18] Raula Gaikovina Kula, Daniel M German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. "Do developers update their library dependencies? An empirical study on the impact of security advisories on library migration". In: *Empirical Software Engineering* 23 (2018), pp. 384–417.

[19] Anand Ashok Sawant, Mauricio Aniche, Arie van Deursen, and Alberto Bacchelli. “Understanding developers’ needs on deprecation as a language feature”. In: *Proc. 40th International Conference on Software Engineering*. 2018, pp. 561–571.

[20] Abbas Javan Jafari, Diego Elias Costa, Emad Shihab, and Rabe Abdalkareem. “Dependency Update Strategies and Package Characteristics”. In: *ACM Trans. Softw. Eng. Methodol.* 32.6 (Sept. 2023).

[21] Xiaoxing Ma, Tianxiao Gu, and Wei Song. “Software Is Not Soft”. In: *Engineering Trustworthy Software Systems*. Ed. by Jonathan P. Bowen, Zhiming Liu, and Zili Zhang. Cham: Springer International Publishing, 2018, pp. 143–175.

[22] Erik Derr, Sven Bugiel, Sascha Fahl, Yasemin Acar, and Michael Backes. “Keep me Updated: An Empirical Study of Third-Party Library Updatability on Android”. In: *Proc. 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2017, pp. 2187–2200.

[23] Joël Cox, Eric Bouwers, Marko Van Eekelen, and Joost Visser. “Measuring dependency freshness in software systems”. In: *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. Vol. 2. IEEE. 2015, pp. 109–118.

[24] T. Mens, M. Wermelinger, S. Ducasse, S. Demeyer, R. Hirschfeld, and M. Jazayeri. “Challenges in software evolution”. In: *Eighth International Workshop on Principles of Software Evolution (IWPSE’05)*. 2005, pp. 13–22.

[25] Iulian Neamtiu, Guowu Xie, and Jianbo Chen. “Towards a better understanding of software evolution: an empirical study on open-source software”. In: *Journal of Software: Evolution and Process* 25.3 (2013), pp. 193–218.

[26] Mívian M. Ferreira, Mariza Andrade da Silva Bigonha, and Kecia A. M. Ferreira. “On The Gap Between Software Maintenance Theory and Practitioners’ Approaches”. In: *CoRR* abs/2104.03824 (2021).

[27] Julie M. Haney, Simson L Garfinkel, and Mary F Theofanos. “Organizational practices in cryptographic development and testing”. In: *2017 IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2017, pp. 1–9.

[28] Julie M. Haney, Mary Theofanos, Yasemin Acar, and Sandra Spickard Prettyman. “"We make it a big deal in the company": Security Mindsets in Organizations that Develop Cryptographic Products.” In: *SOUPS@ USENIX Security Symposium*. 2018, pp. 357–373.

[29] Anand Ashok Sawant, Romain Robbes, and Alberto Bacchelli. “To react, or not to react: Patterns of reaction to API deprecation”. In: *Empirical Software Engineering* 24 (2019), pp. 3824–3870.

[30] Jiawei Wang, Li Li, Kui Liu, and Haipeng Cai. “Exploring How Deprecated Python Library APIs Are (Not) Handled”. In: *Proc. 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ESEC/FSE 2020. Virtual Event, USA: ACM, 2020, pp. 233–244.

[31] Anand Ashok Sawant, Romain Robbes, and Alberto Bacchelli. “On the reaction to deprecation of clients of 4 + 1 popular Java APIs and the JDK”. In: *Empirical Software Engineering* 23 (2018), pp. 2158–2197.

[32] Peter Leo Gorski, Yasemin Acar, Luigi Lo Iacono, and Sascha Fahl. “Listen to Developers! A Participatory Design Study on Security Warnings for Cryptographic APIs”. In: *Proc. 2020 CHI Conference on Human Factors in Computing Systems*. CHI ’20. New York, NY, USA: ACM, 2020, pp. 1–13.

[33] David Ott, Christopher Peikert, and other workshop participants. *Identifying Research Challenges in Post Quantum Cryptography Migration and Cryptographic Agility*. 2019.

[34] Konstantin Fischer, Ivana Trummová, Phillip Gajland, Yasemin Acar, Sascha Fahl, and Angela Sasse. “On The Challenges of Bringing Cryptography from Papers to Products: Results from an Interview Study with Experts”. In: *33rd USENIX Security Symposium, USENIX Security ’24, Philadelphia, PA, USA, August 14-16, 2024*. USENIX, Aug. 2024.

[35] Nicolas Huaman, Jacques Suray, Jan H. Klemmer, Marcel Fourné, Sabrina Klivan, Ivana Trummová, Yasemin Acar, and Sascha Fahl. ““You have to read 50 different RFCs that contradict each other”: An Interview Study on the Experiences of Implementing Cryptographic Standards”. In: *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX, Aug. 2024, pp. 7249–7266.

[36] Sazzadur Rahaman, Ya Xiao, Sharmin Afrose, Ke Tian, Miles Frantz, Na Meng, Barton P Miller, Fahad Shaon, Murat Kantarcioglu, and Danfeng Yao. “Deployment-quality and Accessible Solutions for Cryptography Code Development”. In: *Proc. Tenth ACM Conference on Data and Application Security and Privacy*. 2020, pp. 174–176.

[37] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. “An empirical study of cryptographic misuse in android applications”. In: *Proc. 2013 ACM SIGSAC Conference on Computer and Communications Security*. 2013, pp. 73–84.

[38] Jun Gao, Pingfan Kong, Li Li, Tegawendé F Bissyandé, and Jacques Klein. “Negative results on mining crypto-api usage rules in android apps”. In: *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE. 2019, pp. 388–398.

[39] Anna-Katharina Wickert, Lars Baumgärtner, Florian Breitfelder, and Mira Mezini. “Python Crypto Misuses in the Wild”. In: *Proc. 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 2021, pp. 1–6.

[40] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. “The most dangerous code in the world: validating SSL certificates in non-browser software”. In: *Proc. 2012 ACM Conference on Computer and Communications Security*. 2012, pp. 38–49.

[41] Harjot Kaur, Sabrina Amft, Daniel Votipka, Yasemin Acar, and Sascha Fahl. “Where to Recruit for Security Development Studies: Comparing Six Software Developer Samples”. In: *31st USENIX Security Symposium, USENIX Security ’22, Boston MA, USA, August 10-12, 2022*. USENIX, Aug. 2022.

[42] Sabrina Amft, Sandra Höltervennhoff, Rebecca Panskus, Karola Marky, and Sascha Fahl. “Everyone for Themselves? A Qualitative Study about Individual Security Setups of Open Source Software Contributors”. In: *45th IEEE Symposium on Security and Privacy, IEEE S&P 2024, May 20-23, 2024*. IEEE Computer Society, May 2024.

[43] Sandra Höltervennhoff, Noah Wöhler, Arne Möhle, Marten Oltrogge, Yasemin Acar, Oliver Wiese, and Sascha Fahl. “A Mixed-Methods Study on User Experiences and Challenges of Recovery Codes for an End-to-End Encrypted Service”. In: *33rd USENIX Security Symposium, USENIX Security ’24, Philadelphia, PA, USA, August 14-16, 2024*. USENIX, Aug. 2024.

[44] Alexander Krause, Jan H. Klemmer, Nicolas Huaman, Dominik Wermke, Yasemin Acar, and Sascha Fahl. “Pushed by Accident: A Mixed-Methods Study on Strategies of Handling Secret Information in Source Code Repositories”. In: *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX, Aug. 2023, pp. 2527–2544.

[45] Dominik Wermke, Noah Wöhler, Jan H. Klemmer, Marcel Fourné, Yasemin Acar, and Sascha Fahl. “Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects”. In: *43rd IEEE Symposium on Security and Privacy, IEEE S&P 2022, May 22-26, 2022*. IEEE Computer Society, May 2022.

[46] Leo A. Goodman. “Snowball Sampling”. In: *The Annals of Mathematical Statistics* 32.1 (1961), pp. 148–170.

[47] Bruce Schneier. *Worldwide Encryption Product Survey Data*. https://www.schneier.com/wp-content/uploads/2016/02/worldwide-encryption-product-survey-data.xls (visited on 05/05/2023.

[48] GitHub. *GitHub Acceptable Use Policies*. https://docs.github.com/en/site-policy/acceptable-use-policies/github-acceptable-use-policies (visited on 05/05/2023.

[49] Emilee Rader, Samantha Hautea, and Anjali Munasinghe. “"I Have a Narrow Thought Process": Constraints on Explanations Connecting Inferences and Self-Perceptions”. In: *16th Symposium on Usable Privacy and Security (SOUPS 2020)*. USENIX, Aug. 2020, pp. 457–488.

[50] RedHat. *What is software supply chain security?* https://www.redhat.com/en/topics/security/what-is-software-supply-chain-security (visited on 14/05/2024.

[51] Zoom Video Communications, Inc. *Zoom*. https://zoom.us/.

[52] BigBlueButton Inc. *Big Blue Button*. https://bigbluebutton.org/.

[53] *Amberscript*. 2024. URL: https://www.amberscript.com.

[54] Melanie Birks and Jane Mills. *Grounded theory: A practical guide*. Sage, 2015.

[55] Cathy Urquhart. *Grounded theory for qualitative research: A practical guide*. Sage, 2012.

[56] Greg Guest, Kathleen M. MacQueen, and Emily E. Namey. *Applied Thematic Analysis*. SAGE Publications, 2012.

[57] Matthew B. Miles and A. Michael Huberman. *Qualitative data analysis: An expanded sourcebook*. 2nd ed. Sage Publications, 1994.

[58] Collins W. Munyendo, Yasemin Acar, and Adam J. Aviv. “"In Eighty Percent of the Cases, I Select the Password for Them": Security and Privacy Challenges, Advice, and Opportunities at Cybercafes in Kenya”. In: *2023 IEEE Symposium on Security and Privacy (SP)*. 2023, pp. 570–587.

[59] Rami Sammak, Anna Lena Rotthaler, Harshini Sri Ramulu, Dominik Wermke, and Yasemin Acar. “Developers’ Approaches to Software Supply Chain Security: An Interview Study”. In: *Proc. 2024 Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*. SCORED ’24. Salt Lake City, UT, USA: ACM, Nov. 2024, pp. 56–66.

[60] Rodrigo Werlinger, Kasia Muldner, Kirstie Hawkey, and Konstantin Beznosov. “Preparation, detection, and analysis: the diagnostic work of IT security incident response”. In: *Information Management & Computer Security* 18.1 (Mar. 2010). Ed. by Steven M. Furnell, pp. 26–42.

[61] Hugh Beyer and Karen Holtzblatt. *Contextual Design: Defining Customer-Centered Systems*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997.

[62] Nora McDonald, Sarita Schoenebeck, and Andrea Forte. “Reliability and Inter-Rater Reliability in Qualitative Research: Norms and Guidelines for CSCW and HCI Practice”. In: *Proc. ACM on Human-Computer Interaction* 3.CSCW (2019), pp. 1–23.

[63] Stefan Albert Horstmann, Samuel Domiks, Marco Gutfleisch, Mindy Tran, Yasemin Acar, Veelasha Moonsamy, and Alena Naiakshina. ““Those things are written by lawyers, and programmers are reading that.” Mapping the Communication Gap Between Software Developers and Privacy Experts”. In: *Proceedings on Privacy Enhancing Technologies*. 2024.

[64] Philip Klostermeyer, Sabrina Amft, Sandra Höltervennhoff, Alexander Krause, Niklas Busch, and Sascha Fahl. “Skipping the Security Side Quests: A Qualitative Study on Security Practices and Challenges in Game Development”. In: *Proc. 2024 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’24. Salt Lake City, UT, USA: ACM, 2024, pp. 2651–2665.

[65] Juliane Schmüser, Philip Klostermeyer, Kay Friedrich, and Sascha Fahl. ““I’m pretty expert and I still screw it up”: Qualitative Insights into Experiences and Challenges of Designing and Implementing Cryptographic Library APIs”. In: *2025 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, May 2025, pp. 26–26.

[66] Benjamin Saunders, Julius Sim, Tom Kingstone, Shula Baker, Jackie Waterfield, Bernadette Bartlam, Heather Burroughs, and Clare Jinks. “Saturation in qualitative research: exploring its conceptualization and operationalization”. In: *Quality & quantity* 52 (2018), pp. 1893–1907.

[67] Warda Usman, Jackie Hu, McKynlee Wilson, and Daniel 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)*. Anaheim, CA: USENIX, Aug. 2023, pp. 473–490.

[68] Shikun Zhang, Yuanyuan Feng, Yaxing Yao, Lorrie Faith Cranor, and Norman Sadeh. “How usable are ios app privacy labels?” In: *Proceedings on Privacy Enhancing Technologies* (2022).

[69] Pardis Emami-Naeini, Henry Dixon, Yuvraj Agarwal, and Lorrie Faith Cranor. “Exploring How Privacy and Security Factor into IoT Device Purchase Behavior”. In: *Proc. 2019 CHI Conference on Human Factors in Computing Systems*. CHI ’19. Glasgow, Scotland, UK: ACM, 2019, pp. 1–12.

[70] Hana Habib, Sarah Pearman, Jiamin Wang, Yixin Zou, Alessandro Acquisti, Lorrie Faith Cranor, Norman Sadeh, and Florian Schaub. “"It’s a scavenger hunt": Usability of Websites’ Opt-Out and Data Deletion Choices”. In: *Proc. 2020 CHI Conference on Human Factors in Computing Systems*. CHI ’20. Honolulu, HI, USA: ACM, 2020, pp. 1–12.

[71] Elaine Barker and Allen Roginsky. *Transitioning the use of cryptographic algorithms and key lengths*. NIST Special Publication 800-131A Revision 2. National Institute of Standards and Technology, Mar. 2019.

[72] Jan H. Klemmer, Marco Gutfleisch, Christian Stransky, Yasemin Acar, M. Angela Sasse, and Sascha Fahl. “"Make Them Change it Every Week!": A Qualitative Exploration of Online Developer Advice on Usable and Secure Authentication”. In: *Proc. 2023 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’23. Copenhagen, Denmark: ACM, 2023, pp. 2740–2754.

[73] Yasemin Acar, Michael Backes, Sascha Fahl, Doowon Kim, Michelle L. Mazurek, and Christian Stransky. “You Get Where You’re Looking for: The Impact of Information Sources on Code Security”. In: *2016 IEEE Symposium on Security and Privacy (SP)*. 2016, pp. 289–305.

[74] Goran Piskachev, Matthias Becker, and Eric Bodden. “Can the configuration of static analyses make resolving security vulnerabilities more effective? - A user study”. In: *Empirical Software Engineering* 28.5 (Sept. 2023).

[75] Stefan Krüger, Sarah Nadi, Michael Reif, Karim Ali, Mira Mezini, Eric Bodden, Florian Göpfert, Felix Günther, Christian Weinert, Daniel Demmler, and Ram Kamath. “CogniCrypt: Supporting developers in using cryptography”. In: *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 2017, pp. 931–936.

[76] Duc Cuong Nguyen, Dominik Wermke, Yasemin Acar, Michael Backes, Charles Weir, and Sascha Fahl. “A Stitch in Time: Supporting Android Developers in WritingSecure Code”. In: *Proc. 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’17. Dallas, Texas, USA: ACM, 2017, pp. 1065–1077.

[77] Vasileios Mavroeidis, Kamer Vishi, Mateusz D Zych, and Audun Jøsang. “The impact of quantum computing on present cryptography”. In: *arXiv preprint arXiv:1804.00200* (2018).

[78] Christian Stransky, Oliver Wiese, Volker Roth, Yasemin Acar, and Sascha Fahl. “27 Years and 81 Million Opportunities Later: Investigating the Use of Email Encryption for an Entire University”. In: *43rd IEEE Symposium on Security and Privacy, IEEE S&P 2022, May 22-26, 2022*. IEEE Computer Society, May 2022.

[79] Elaine Venson, Xiaomeng Guo, Zidi Yan, and Barry Boehm. “Costing Secure Software Development: A Systematic Mapping Study”. In: *Proc. 14th International Conference on Availability, Reliability and Security*. ARES ’19. Canterbury, CA, United Kingdom: ACM, 2019.

[80] Andres Freund. *backdoor in upstream xz/liblzma leading to ssh server compromise*. https://lwn.net/ml/oss-security/20240329155126.kjjfduxw2yrlxgzm@awork3.anarazel.de/, as of January 30, 2025. Mar. 2024.

[81] Redhat Inc. *CVE-2024-3094*. https://access.redhat.com/security/cve/CVE-2024-3094, as of January 30, 2025. Mar. 2024.

[82] Henrik Plate, Serena Elisa Ponta, and Antonino Sabetta. “Impact assessment for vulnerabilities in open-source software libraries”. In: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2015, pp. 411–420.

[83] Lina Boughton, Courtney Miller, Yasemin Acar, Dominik Wermke, and Christian Kästner. “Decomposing and Measuring Trust in Open-Source Software Supply Chains”. In: *Proc. 2024 ACM/IEEE 44th International Conference on Software Engineering: New Ideas and Emerging Results*. ICSE-NIER’24. Lisbon, Portugal: ACM, 2024, pp. 57–61.

[84] Stephen McQuistin, Mladen Karan, Prashant Khare, Colin Perkins, Gareth Tyson, Matthew Purver, Patrick Healey, Waleed Iqbal, Junaid Qadir, and Ignacio Castro. “Characterising the IETF through the lens of RFC deployment”. In: *Proc. 21st ACM Internet Measurement Conference*. IMC ’21. Virtual Event: ACM, 2021, pp. 137–149.

[85] Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. *Report on Post-Quantum Cryptography*. Apr. 2016.

[86] Dustin Moody, Ray Perlner, Regenscheid Andrew, Angela Robinson, and David Cooper. *Transition to Post-Quantum Cryptography Standards*. 2024.

[87] Azadeh Mokhberi and Konstantin Beznosov. “SoK: Human, Organizational, and Technological Dimensions of Developers’ Challenges in Engineering Secure Software”. In: *Proc. 2021 European Symposium on Usable Security*. EuroUSEC ’21. Karlsruhe, Germany: ACM, 2021, pp. 59–75.

[88] Erin Kenneally and David Dittrich. “The menlo report: Ethical principles guiding information and communication technology research”. In: *Available at SSRN 2445102* (2012).

A Survey Questionnaire

1. Do you have any experience with updating cryptographic implementations in any of your projects? [If yes, please briefly explain in one or two sentences (text entry); no] (single choice)
2. Which of the following best describes your current employment status? Please check all that apply.[Employed full-time; Employed part-time; Independent contractor, freelancer, or self-employed; Not employed but looking for work; Not employed but currently not looking for work; Stay-at-home parent; Student; Military; Retired; Unable to work; Prefer not to disclose; Prefer to self-describe (text entry)] (multiple choice)
3. What is your current occupation? (text entry)
4. How many projects have you contributed to in the past? [1–5, 6–10, 11–15, 16–20, >20] (single choice)
5. In which way do you mainly contribute to projects? Please check all that apply. [Solo, As part of a software development team in a company, As a contributor to an open source project, Prefer to self-describe (text entry)] (multiple choice)
6. How many years of experience do you have in software development in total? (please only answer using digits, e.g., 4) (text entry)
7. How many years of experience do you have in computer security in total? ’Experience’ includes years at work or studying in a security-related field (please only answer using digits, e.g., 4). (text entry)
8. What is the highest degree or level of school you have received? (If you’re currently enrolled in school, please indicate the highest degree you have received.) [I never completed any formal education, 10th grade or less (e.g. some American high school credit, German Realschule, British GCSE), Secondary school (e.g. American high school, German Gymnasium, Spanish or French Baccalaureate, British A-Levels), Trade, technical or vocational training, Some college/university study without earning a degree, Associate degree (A.A., A.S., etc.), Bachelor’s degree (B.A., B.S., B.Eng., etc.), Master’s degree (M.A., M.S., M.Eng., MBA, etc.), Professional degree (JD, MD, etc.), Other doctoral degrees (Ph.D., Ed.D., etc.), Prefer not to disclose, Other (please specify) (text entry)] (single choice)
9. If you indicated that you received a degree, which subject did you study or which field did you specialize in? [Optional] (text entry)
10. Did you receive any security education (job training, certificates, university courses)? [Optional] [Yes, No, Prefer not to disclose] (single choice)
11. [If "Yes" in Question 10] Where did you receive your security education? [Optional] [Self-taught, Online class, College/University, On-the-job training, Coding Camp, Prefer not to disclose, Prefer to self-describe (text entry)] (multiple choice)
12. What is your gender? [Optional] [Female, Male, Non-binary, Prefer not to disclose, Prefer to self-describe (text entry)] (single choice)
13. What is your age (in years)? (please only answer using digits, e.g., 19) [Optional] (number entry)
14. Which country do you currently reside in? [Optional] (country-selection drop-down menu)

B Interview Guide

S1: Cryptographic use in software projects

- S1.Q1: [**Crypto Projects**] [*if any crypto project not mentioned for S1.Q2*] Please provide some examples where you typically “get in touch” with **crypto implementations** in your projects. (e.g., encryption, authentication, crypto libraries, verification) With the term **crypto implementations**, we include protocols, primitives, algorithms, and cryptographic libraries.
  - S1.Q1.1: [**Contributions**] What were your contributions to these projects?

- S1.Q1.2.: [**Roles (projects in S2.Q1.)**] What were your roles in these projects?
    - S1.Q1.2.1: [**Responsibilities**] What were your responsibilities?
- S1.Q1.3: Please state the importance of cryptography in your project(s).
- S1.Q1.4: *[if reason is not clear from S2.Q1]* For what reasons are you using crypto implementations in your project(s)?
- S1.Q1.5: Which **crypto implementations** are you using?
- S1.Q2: [**Decisions**] Can you tell us how you decide which crypto implementations you use?
- S1.Q2.1: [**Resources**] What security-related resources do you use/consult for your references (Resources could include everything like online websites, books, specific people, etc.)?
    - S1.Q2.1.1: [**Recommendations**] [*If not mentioned in S2.Q2.1*] What kind of recommendations do you use/follow while deciding on crypto implementations?

S2: Updating Crypto

- S2.Q1: Briefly describe Chrome deprecating SHA1 certificates here
    - S2.Q1.1: Before delving further into the crypto update process-related questions, we would like to briefly state an example of when Chrome deprecated SHA1 certificates.
      In 2005, it was widely known that SHA-1 was weaker than what was originally thought. Google Chrome announced in 2014 to remove the support of SHA-1 certificates. The removal took Google five years, so SHA-1 certificates were supported for a much longer period than Google thought. (Removal by 2019)
- S2.Q2: [**Frequency**] How often have you had to update crypto implementations in your projects in the past?
- S2.Q3: [**Update Situation**] Please elaborate on the most impactful crypto update you have experienced.
- If you have trouble remembering, you can tell us about the latest crypto update.
- S2.Q3.1 (If they say some update was most impactful)
    - [**Impactful**] Why was it most impactful?

S3: Reasons and Awareness for Updating

- S3.Q1: [**Reason**] For which reasons did you update crypto implementations in the past in your project(s)?
- S3.Q2: [**Awareness**]: How did you become aware that you should update your crypto implementation(s)? (e.g., notifications, supply chain security)
    - S3.Q2.1: [**Notification**] [*If not mentioned in S4.Q2*]: How do you get notified about potential issues in your implementation/project / Software Supply Chain? (notification channels may be social media or CV tracker, security researchers)
- S3.Q3: [**Consequences after awareness**]: How did you react after becoming aware that you should update your crypto code?
    - S3.Q3.1: What were the consequences for the project/product (e.g., stop shipping vulnerable/broken versions, reschedule future releases, affect other projects)?
- S3.Q4: Does your project contain any third-party crypto implementations?

Definition [Software supply chain security]: combines best practices from risk management and cybersecurity to help protect the software supply chain from potential vulnerabilities.

- S3.Q4.1: How do you handle supply chain security in your projects?
- S3.Q4.2: [*If yes*]: How do you manage your dependencies?
    - S3.Q4.2.1: In your experience, are there any differences in managing cryptographic dependencies as compared to general/regular dependencies? (e.g., numpy vs. OpenSSL)
        - S3.Q4.2.1.1: [*If yes*]Please elaborate on the differences. (what and why)
    - S3.Q4.2.2: [**Issue handling**] How do you handle issues in your supply chain? (e.g., update version in the supply chain if available, replace the library with another that has no issues, offer a downgrade, etc.)

S4: Update Process

Block 1: Strategies:

- S4.B3.Q1: [**Strategies**] Please describe the overall strategies used to plan and perform the crypto update process.
    - S4.B3.Q1.1 [**Strategies Plan**]: How did you plan the crypto update process?
    - S4.B3.Q1.2 [**Strategies Perform**]: How did you perform the crypto update process?

Block 2: Involved People and Roles:

- S4.B1.Q1: [**Roles & Responsibilities**] What was your part in the crypto update process and what were your responsibilities?
- S4.B1.Q2: [**Other people update process**] Who else apart from yourself was involved in the crypto update process?
    - S4.B1.Q2.1: [**Externals**] Were any externals involved? (e.g., hired professionals, freelancers, hired services from a company), crypto researcher.
        - S4.B1.Q2.1.1[*If yes*] Why were external people involved?
    - S4.B1.Q2.2: [**Experts**] Do you rely on experts for specific problems/implementation? (consultants, freelancers) (internally, externally)
        - S4.B1.Q2.2.1 [*If yes*] For which purposes?

Block 3: Decision Finding Process:

- S4.B2.Q1: [**Decision Finding Process**] Please describe what the decision-finding process for the crypto update was. (e.g., weekly meetings of core developers, mailing lists, GitHub Issues, or polls)
  - S4.B2.Q1.1: Who is in charge of this process?
- S4.B2.Q2: [**Sec & Qual Management**] Which strategies are used for security and quality management? (e.g., QA, audits, internal testing, beta testing)
- S4.B2.Q3: [**Advice**] Did you ask someone, or who would you ask to get help in the crypto update decision-finding process?

Block 4: Success or Failure:

- S4.B4.Q1: Were you finally able to successfully update the crypto implementation?
  - [*If yes*]
    - Did the updated crypto implementations fulfill the requirements?
    - What major changes occurred to the project?
  - [*If no*] Did any of your crypto update processes fail in the past?
    - [*If yes*] Could you elaborate on the reasons for the failure?
    - Did your crypto update process improve after the failure?
    - How did you end up fixing the problem?
    - What consequences occurred due to the crypto update failure? (e.g., delayed release, additional workload, higher costs, someone got fired)

S5: Experiences (Positive and Negative)

- S5.Q1: Please tell us about your experiences when updating crypto implementations in your projects.
  - S5.Q1.1: How long did it take from recognition to fix?
  - S5.Q1.1: Which efforts were required for the crypto updates, and how do they differ from regular software updates?

Block 1: Good Experiences:

- S5.B1.Q1: Can you tell us about things that went well during the crypto update process? (Crypto-specific follow up)
  - S5.B1.Q1.1: [If good exp]: What was most satisfying?

Block 2: Bad Experiences:

- S5.B2.Q1: Did you have any negative experiences during the crypto update process? (Crypto-specific follow up) breaking changes
  - S5.B2.Q1.1: [If yes:] Could you elaborate on those negative experiences?
  - S5.B2.Q1.2: How did the crypto update affect other projects or parts of the same project?

Block 3: Changes and Improvements:

- S5.B3.Q1: What were the [major] changes to the project after you did the crypto update process?
- S5.B3.Q2: [IMPROVEMENTS] Have you improved the overall crypto update process?
  - S5.B3.Q2.1: [If yes]What were the improvements to the update process?

S6: Challenges

- S6.Q1: What challenges or roadblocks did you encounter during the crypto update process?
  - S6.Q1.1: What was the most challenging aspect of updating crypto?
- S6.Q2: What problems did occur in the process of updating crypto? (For example, for forced rollbacks due to unintended changes, are too many code changes needed?)
- S6.Q2.1: [*If they don't say it by themselves*] Which issues did you face regarding back compatibility?
  - S6.Q2.2: Did you encounter any challenges using the documentation?
  - S6.Q3: Have you ever skipped or avoided a crypto update process (crypto implementations)?
  - S6.Q3.1: What were the reasons for not updating?
  - S6.Q3.2: Were there any consequences caused by not updating (e.g., keeping deprecated or outdated implementations/libraries, known/existing vulnerabilities in the codebase)

S7: Wishes

- S7.Q1: Based on your experiences, what do you think can be improved to help yourself better update the crypto code?
  - S7.Q1.1: Are there any unfulfilled wishes that may help you to encounter this process better?
  - S7.Q1.2: Do you think established best practices or a standard would be helpful in the context of updating crypto code?
  - S7.Q1.3: How well do you feel supported in updating crypto implementations in general? (internal documentation, company internal consultancies, onboarding of new developers, experts involved in the project)
- S7.Q2: As a developer [change terms here based on the role of the participant], what do you think is the "perfect solution" for a successful crypto update process?

Conclusion

- [Asking Shadow Interviewer] Does my colleague have any remaining questions?
- Finally, is there anything else you wish we had asked about, or would you like to tell us about it?