An Industry Interview Study of Software Signing for Supply Chain Security

Kelechi G. Kalu
Purdue University
kalu@purdue.edu

Tanmay Singla
Purdue University
singlat@purdue.edu

Chinenye Okafor
Purdue University
okafor1@purdue.edu

Santiago Torres-Arias
Purdue University
santiagotorres@purdue.edu

James C. Davis
Purdue University
davisjam@purdue.edu

Abstract

Many software products are composed of components integrated from other teams or external parties. Each additional link in a software product's supply chain increases the risk of the injection of malicious behavior. To improve supply chain provenance, many cybersecurity frameworks, standards, and regulations recommend the use of software signing. However, recent surveys and measurement studies have found that the adoption rate and quality of software signatures are low. We lack in-depth industry perspectives on the challenges and practices of software signing.

To understand software signing in practice, we interviewed 18 experienced security practitioners across 13 organizations. We study the challenges that affect the effective implementation of software signing in practice. We also provide possible impacts of experienced software supply chain failures, security standards, and regulations on software signing adoption. To summarize our findings: (1) We present a refined model of the software supply chain factory model highlighting practitioner's signing practices; (2) We highlight the different challenges–technical, organizational, and human–that hamper software signing implementation; (3) We report that experts disagree on the importance of signing; and (4) We describe how internal and external events affect the adoption of software signing. Our work describes the considerations for adopting software signing as one aspect of the broader goal of improved software supply chain security.

1 Introduction

Like all engineered products, software can be built more quickly when components are reused from other sources [39, 73]. Although this integration accelerates production, it requires trusting external parties [5]. This trust increases when the provenance of components is known [66], such as via *software signing*. Although historically underutilized [62, 94], due to attacks such as SolarWinds [31], signing has become part of many government, academic, and industry proposals to secure software supply chains [17, 19, 78].

Although software signing has desirable effects, it must be adopted carefully to bring benefits. At present, measurement studies show that missing or erroneous signatures are common across the software supply chain [41, 51, 62, 90, 94]. Existing security frameworks recommend software signing, but the general guidelines in these frameworks omit a detailed implementation model [59, 97]. In order to adopt software signing, organizations would benefit from a detailed understanding of industry practices and challenges. This view is currently lacking in the academic and grey literature.

Our work represents the first interview study that describes software signing practices and rationales in industry. This method gives a deep look at a topic, exposing many directions for further study and implications for practice. In this work, We interviewed 18 experienced security practitioners across 13 organizations. Our interviews focused on four aspects: the software signing practices employed in industry; the challenges that hamper the effective implementation of software signing in practice; the importance accorded to software signing in practice; and the influence of failure events and industry standards.

Our results provide a detailed understanding of industry practices and challenges in using software signing to promote software supply chain security. By integrating our subjects' descriptions, we introduce a refinement of the Software Supply Chain Factory Model that indicates the possible and actual roles of software signing. Our results reveal three kinds of challenges in adopting signing: technical, organizational, and human aspects. Additionally, our findings indicate that practitioners hold varying views on the importance of signing for supply chain security. Lastly, we found that security events — both major failures such as SolarWinds, and new standards and regulations — appear to have a minimal effect on the adoption of software signing.

In summary, we make the following contributions:

1. We refined the software supply chain factory model to reflect diverse industry practices and the forms that signing may take therein (§5.1).

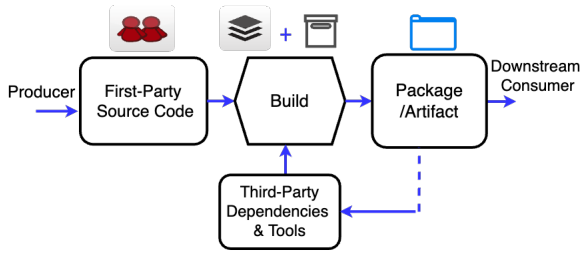

Figure 1: The Software Supply Chain Factory Model [77].

2. We identified common technical, human, and organizational challenges in adopting signing (§5.2).

3. We report on the relative importance of signing in software supply chain security strategies (§5.3).

4. We discuss the effect that failures, standards, and regulations have on software signing practices (§5.4).

2 Background & Related Works

Here, we cover software supply chains (§2.1), cybersecurity considerations (§2.2), and relevant empirical studies (§2.3).

2.1 Software Supply Chains

Software production is often depicted using variations on the early Software Factory Model [14,46] or the more recent Software Supply Chain Factory Model (SSCFM) [1,77], depicted in Figure 1. The SSCFM approximates software product from the perspective of one software engineering organization or team The SSCFM model illustrates that a software product integrates both first-party and third-party components, which are subsequently packaged for downstream use. These integration points also represent potential vulnerabilities where the software product is at risk of malicious code injection from either internal or external sources.

The study of software supply chains has been driven by engineering organizations' greater reliance on external components. Many works have reported on this reliance in various software contexts. For example, a 2024 Synopsys report stated that, in 1,067 projects across 17 industry sectors, ∼96% integrated open-source components, and ∼77% of total code is from open source [74]. Even in government and safety-critical systems, the use of third-party open-source components has moved from anathema to normal [22,32]. Analyses of open-source software package registries have shown large increases year-over-year in the number of available components, the number of actively-used components, and the number of transitive dependencies [38,52,88,96].

The increasing reliance on external components in software production amplifies the potential for the exploitation of vulnerabilities that may be contained in these components. As an example of this risk, Lauinger *et al.* [45] report that ∼37% of the most popular Internet websites rely on at least one library with a publicly acknowledged vulnerability. Given the prevalence of inherited vulnerabilities, Zahan *et al.* [95] and similar studies have summarized attack vectors and potential weak links in software supply chains [9,25,29,42,50,54,69,93].

2.2 Securing Software Supply Chains

2.2.1 Methods and Practices

Numerous methods and tools have emerged to improve software supply chain security. Okafor *et al.* [55] state that these approaches promote three distinct dimensions: *transparency* (knowing all components in the chain), *separation* (isolating components), and *validity* (ensuring integrity of components). For example, transparency is increased by adoption of Software Bills of Materials (SBOMs) [49,92]. Separation is aided by techniques such as sandboxing [4,44,72,80,81] and zero-trust architectures [8,71]. Validity is improved by reproducible builds [43,83], software signatures [62], and attestations about artifact metadata (*e.g.,* authorship) [79].

Security standards, frameworks, and regulations have been established to complement and direct these security efforts. Notable examples include The Linux Foundation's Supply Chain Levels for Software Artifacts (SLSA) [77], the Cloud Native Computing Foundation's (CNCF) Software Supply Chain Best Practices [2], Microsoft's Supply Chain Integrity Model (SCIM) [3], and the US National Institute of Standards and Technology's (NIST) Secure Software Development Framework (NIST-SSDF) [18]. Current national regulations do not force compliance with secure practices [47].

2.2.2 Software Signing in Detail

Software signing is a formally guaranteed method of establishing the authorship of a software component [20,27,28,64]. It ensures the provenance and integrity of components in the supply chain through public-key cryptography. The process involves creating a hash of the software, encrypting it with the author's private key, and attaching the resulting signature to the software. When users download the software, they decrypt the signature with the corresponding public key and compare the hash to ensure the software has not been altered. A typical workflow for a package signing is shown in our linked technical report Appendix A.

Keyless signing [35,51] eliminates the need for authors to manage long-term private keys. Instead, authors authenticate with an identity provider, and an ephemeral key associated with their verified identity is issued for each signing. Users then additionally verify that the public key used for the software verification is associated with the intended author, ensuring both software integrity and authenticity.

Many standards and regulations recommend signing as a base Software Supply Chain security technique [3,17,68,77,

78]. For example, in The Linux Foundation's SLSA framework, software signing is part of Build Security Level 3 [77].

2.3 Empirical SW Supply Chain Security

Although many theoretical techniques for software supply chain security are known, adopting them in practice remains a challenge. Empirical/*in vivo* studies on software supply chain security have examined adoption practices and challenges to identify gaps in theoretical guarantees or usability [29,60,87]. Some studies examine the general activities of component management, *e.g.,* dependency selection [58] and maintenance [11], and examinations of security challenges and incident handling procedures [85]. Others have examined techniques for supply chain security. For example, for the transparency property §2.2.1, researchers have studied the practices of SBOM adoption [40,91] and reported gaps in tooling, interoperability, adoption, and maintenance. For separation, studies have examined the performance of sandboxing [84] and its adoption in open-source ecosystems [7]. For validity, reproducible builds have been analyzed in PyPi [83].

Concerning software signing, researchers have examined challenges in its usability and adoption that create opportunities for attacks [60, 87]. In 2010–2021, roughly 25% of publicly reported attacks exploited the missing or erroneous use of signing [36]. Top signing-related attack vectors include self-signed code, broken signature systems, and stolen certificates [29]. Although software signing is valued, it is considered costly to set up, as reported in Ladisa *et al.'s* survey [42, 86]. Several reports show that signing adoption rates for open-source components are low. For instance, Zahan *et al.* found that $\leq$0.5% of packages in the npm and PyPI registries had signed releases [94]. Schorlemmer *et al.* corroborated this proportion, further noting that signing adoption is correlated with registry policies and that the correctness or quality of signatures depends on usable tooling [62].

3 Knowledge Gaps & Research Questions

In our analysis of the literature discussed in §2.3, we observed that our empirical understanding of software signing challenges and adoption is primarily derived from open-source software components. The available data on industry practices are surveys [42, 86]. We lack in-depth qualitative descriptions of industry experiences, creating a gap in our understanding of software supply chain security. We do not know which signing strategies are effective, the challenges associated with implementing these methods, the roles played by suggested methods and tools, and the factors influencing their adoption.

To fill this gap, we ask four research questions:

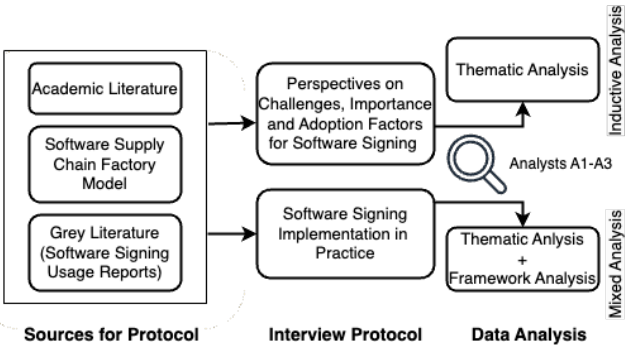

Figure 2: Study methodology. Several sources (*left*) informed our protocol topics (*center*). We applied thematic and framework analyses (*right*).

RQ1: Where and how is software signing implemented by software teams?
RQ2: What are the challenges that affect the implementation and use of software signing?
RQ3: What is the perceived importance of software signing in mitigating risks?
RQ4: How do internal and external signing events influence the adoption of software signing?

4 Methodology

We conducted semi-structured interviews with subjects who were cybersecurity practitioners. We depict our methodology in Figure 2. We describe our research design (§4.1), protocol development (§4.2), participant recruitment (§4.3), data analysis (§4.4), and limitations (§4.5). Our Institutional Review Board (IRB) approved this study.

4.1 Research Design and Rationale

Our research questions require insight into practitioner perspectives. We considered two factors in selecting a methodology. First, because the prior work in this space is limited, there was relatively little basis on which to structure a closed-ended survey. Second, our research questions warrant long-form descriptive-type answers that have to be extracted through open-ended questions, opinions, and detailed observations. We therefore opted for a *semi-structured interview* [61], which would permit us to capture emergent results [21].

The resulting data is expected to be exploratory, and rooted in practitioners' perspectives and experiences [61]. The phenomenon of interest is software signing as a security method, which is a contextual decision based on an organization's distinct policies, security expectations, perceived threats, and customer requirements [67]. We therefore adopt an interpretivist perspective for our research methodology design and analysis. This approach treats each situation as unique [10], focusing on in-depth variables and contextual factors [6]. Our

choice of the interpretivist framework was also motivated by our sampling technique as we discuss later (§4.3).

4.2 Protocol Design and Development

We studied three kinds of resources as we created the initial interview protocol.

1. *Software Supply Chain Security Frameworks:* To ensure a common language and a comprehensive set of topics, we studied software supply chain security frameworks such as SLSA, CNCF, SCIM, and NIST-SSDF (§2.2). These frameworks use variants of the Software Supply Chain Factory Model (SSCFM, cf. Figure 1). We therefore used the SSCFM as our reference framework for structuring the C Sections of our interview protocol. This choice allowed us to investigate signing practices at various stages of the SDLC while using a model familiar to our subjects.
2. *Grey Literature:* To gain insights into practical uses of software signing, we examined grey literature such as industry blogs [30,65], whitepapers [16], and case studies [23,56,90]. These sources documented the adoption of software signing in various industry contexts, and, from these, we derived questions about the contextual and organizational factors that subjects consider. We collected these grey literature sources through Web searches of software signing tools mentioned or recommended by any of the security frameworks we studied. Additionally, we reviewed linked articles related to software signing in these standards.
3. *Background & Literature Review:* Our background and review of related works (§2) provided a foundation for knowing what questions would be effective for answering our research questions while avoiding duplication. Broadly, our literature review process, which yielded all works cited in §2) was based on a forward and backward snowball search. We began with works published in prominent cybersecurity and software engineering venues (USENIX, CCS, IEEE S&P, ICSE, FSE, ASE) and covered a 10-year period.

After protocol development, we refined the protocol through practice interviews (2) and pilot interviews (2). For practice, the lead author interviewed two of the secondary authors. These interviews led to modifications in sections B and C of our interview protocol (see Appendix A). After recruiting subjects, we treated the first two subjects as pilot interviews. During these pilots, we gathered additional insights into how subjects perceived software supply chain attacks and reorganized the protocol, particularly focusing on questions related to the perceived sources of greatest risk (section B). Since few questions were changed or added, we included the data from the pilot interviews in our study where feasible [1]. The final change was made after the seventh interview. The author team reviewed the results up to that point and identified an emergent topic: how practitioners' organizations integrate feedback into their software supply chain (SSC) security processes. Consequently, one additional question was included in the remaining interviews, to address this emerging topic.

Table 1: Summary of interview protocol. Topic D is not analyzed in this paper. See our linked technical report (Appendix A) for full protocol.

Topic (# Questions)	Sample Questions
A. Demographics (4)	What is your role in your team?
B. Perceived software supply chain risks (4)	Can you describe any specific software supply chain attacks (*e.g.,* incidents with 3^{rd}-party dependencies, code contributors, OSS) your team has encountered?
C. Mitigating risks with signing (8)	How does the team use software signing to protect its source code (what parts of the process is signing required)?
D. Signing tool adoption (5)	What factors did the team consider before adopting [TOOL/METHOD] over others?

Defining Internal & External Events to Answer RQ4: Unlike the other research questions, RQ4 focuses on investigating how internal and external factors influence organizational decisions to adopt software signing. Following the definitions of Schorlemmer *et al.* [62], we scope this part of the protocol to the impact of (1) Software Supply Chain failures, and (2) Software Supply Chain standards.

Table 1 summarizes our interview protocol. Beyond this material, the technical report (see Appendix A) provides our full interview protocol.

4.3 Data Collection

Target Population: To effectively address our research questions, the target population was industry professionals with expertise in software signing. An ideal subject would have deep knowledge and experience related to software supply chain practices within their organizations. Such subjects are typically middle- or high-ranking engineering staff.

Recruitment: Given the seniority and expected workload of the target population, we anticipated recruitment to be a challenge. To mitigate this, we employed a non-probability-based purposive and snowball sampling approach to recruit participants [10]. This approach incurs the risk of bias, but has the reward of access. Given the understudied topic at hand, we felt that the reward outweighed the risk.

Our initial sample pool of subjects was made up of members of the Kubecon 2023 conference organizing committee of

[1] We included data from the pilot interviews in our analysis, as qualitative methods guidance supports this practice when the protocol demonstrates its ability to effectively answer research questions [37,76]. This approach is particularly justifiable in contexts where recruitment is challenging, as was the case here and in similar work [48]. Quantitatively, we observed that: (1) the pilot interviews used nearly all (92%) of the final protocol, as only 3 out of 35 questions were changed afterward, and (2) the unique codes derived from the pilot interview transcripts were comparable in quantity and richness to those from other participants (see Figure 4).

Table 2: Subject Demographics. For anonymity, we used generic job roles, not specific titles. "*Senior management*" refers to senior managers, directors, and executives; "*Technical leader*" to senior, lead, partner, and principal engineers; and "*Engineer*" and "*Manager*" to more junior staff. We also highlight type of software subject's immediate team produces. * refers to cases where subject's team built other types of software in addition to the main stated. *O* refers to cases where subject belonged to multiple teams with other products to the one stated.

ID	Role	Experience	Software Type
S1	Research leader	5 years	Internal POC software
S2	Senior mgmt.	15 years	SAAS security tool*^{O}
S3	Senior mgmt.	13 years	SAAS security tool*^{O}
S4	Technical leader	20 years	Open-source tooling
S5	Engineer	2 years	Internal security tooling
S6	Technical leader	27 years	Internal security tooling*^{O}
S7	Manager	6 years	Security tooling*^{O}
S8	Technical leader	8 years	Internal security tooling*
S9	Engineer	2.5 years	SAAS security*
S10	Engineer	13 years	SAAS security*
S11	Technical leader	16 years	Firmware*
S12	Technical leader	4 years	Consultancy
S13	Senior mgmt.	16 years	Internal security tool
S14	Research leader	13 years	POC security Software*
S15	Senior mgmt.	15 years	Internal security tooling
S16	Technical leader	26 years	Security tooling
S17	Senior mgmt.	15 years	SAAS
S18	Manager	11 years	Security tooling

the Cloud Native Computing Foundation. This group matches our target population: all of them either worked as a member of the software security teams of their organizations or worked in organizations with products and services in the domain of software security, and all were medium- to high-ranking members of their organizations. Many of these individuals also contribute to software supply chain security projects hosted by The Linux Foundation, making them well-suited to provide expert insights. We had special access to this group because one of the authors was a member of that committee. However, recognizing the potential biases of this initial sample, we expanded our recruitment through recommendations from initial participants and other industry contacts. This approach allowed us to diversify our subject pool and gather a broader range of insights, while still focusing on individuals with substantial expertise in the domain.

In total, we contacted 30 candidates, of whom 18 agreed to participate (60% response rate). These subjects came from 13 different organizations. The subjects were affiliated with companies with security products or were part of their company's security team. Each participant was offered a $100 gift card as an incentive, in recognition of their high rank.

Subject Demographics: We summarize the demographic and organization information of the subjects in Table 2 and Table 6. We note that six participants were from the initial KubeCon committee invitations; industry contacts of the authors yielded seven participants; and snowball sampling (recommendations from participants) yielded five participants.

In addition to the highlighted, we also note that 13 subjects (from 11 organizations – 4 Small, 2 Medium, and 6 Large organizations) confirmed that they either lead, or belong to, teams involved in software supply chain, infrastructure and tooling, or security control implementation. These subjects indicated that they are responsible for initiating or implementing their organization's security controls and have responsibilities related to compliance. Although not all subjects directly owned compliance, their involvement in security strategy and operational decision making provided them with the appropriate context to assess the utility, influence behind adoption, and challenges affecting the implementation of Software signing.

Subjects from small organizations typically focused on SAAS products related to software supply chain, container, and cloud security; or were consultants. In medium-scale organizations, subjects had more specialized roles, *e.g.,* developing internal security-enabling software for infrastructure. In large organizations, subjects had similar goals, but often belonged to/oversaw multiple teams.

For confidentiality, we refer to organizations by the letters A–L and participants using the notation SX(y), where *X* denotes a unique subject number and *y* represents the subject's product type. This notation is also used to provide context to subject's responses. For example, *S10 (SAAS, dev tools)* refers to Subject S10's direct team's software product, which is a SaaS-type product for an organization whose product area is developer tools.

Addressing Potential Bias: While the connection to the CNCF and specific signing projects could bias responses towards certain approaches, this potential bias is balanced by the deep involvement of participants in the field of software supply chain security. Their contributions are crucial for understanding the current state and future directions of software signing practices. Moreover, the insights provided by these experts, who are at the forefront of the industry, offer a unique and valuable perspective that is essential for the study's objectives.

Generalizability of Results: We acknowledge that the non-random sampling method and the concentration of participants from a specific initiative may limit the generalizability of our findings to the broader population of industry practitioners. However, the purpose of this study was not to achieve broad generalizability but rather to gain in-depth insights from a highly specialized group of experts—hence the use of an interpretivist analysis framework. The findings are particularly relevant to organizations and stakeholders involved in similar software supply chain security initiatives and may inform best practices and future research in this specific context. While broader generalization may be limited, the detailed, expert-driven analysis provided by this study offers significant value in advancing understanding within this niche but critical area.

Interviews: Interviews were conducted by the lead author via Zoom. The median interview duration was 50 minutes.

4.4 Data Analysis

Interview recordings were transcribed by www.rev.com (human transcription). We anonymized identifying information. Our analysis then proceeded via multi-stage coding [13] and a subsequent thematic [12] and framework analysis [70].

Three analysts (A1–A3) participated in the analysis stage of the transcripts. Analyst A1, the lead author, was responsible for all stages of data collection (including protocol design and interview conduct) and contributed to all phases of the analysis. Initial analysis was conducted by A1 and A2. When A2 became unavailable, we recruited A3. To ensure consistency, A3 was trained by memoing and coding transcripts that had already been analyzed. Comparing A3's memos to A1 and A2, we observed strong alignment, ensuring reliability in the analysis process. A1-A3 are graduate students with research experience in cybersecurity, including software signing. The supervisors are researchers in software engineering and cybersecurity.

Multi-stage coding:

Stage 1. We began with data familiarization [75].

Stage 2. Next, we needed to develop and refine our codebook. To provide reliability while conserving resources, we used the technique described by Campbell *et al.* [13] and O'Connor & Joffe [57]. First, we chose a proportion of random transcripts to multiply-code (*i.e.,* multiple analysts coding).[2]

Then, two analysts (A1, A2) conducted multiple rounds of memoing on these transcripts, discussing emerging code categorizations after each round. This process resulted in 506 coded memos, which were reviewed by Analysts A1 and A3 in a series of meetings. Ultimately we obtained a codebook with 36 code categories. This codebook was further refined and extended as discussed in Stages 3-5.

Stage 3. To assess the reliability of our coding, we combine the techniques of Campbell *et al.* [13] and Maxam&Davis [48]. Analysts A1 and A3 randomly selected and coded a transcript independently [13, 53], and we measured their degree of agreement. Since our codebook was collectively defined by analysts A1 and A3, we supposed this coding task would not be difficult, and thus we used the percentage agreement metric as recommended by Feng [26]. In this independent coding step, we observed an 89% agreement rate. The analysts were able to resolve disagreements through discussion and code refinement. According to Campbell *et al.* [13], this high level of agreement suggests that a coding scheme is suitable for a single coding of the remaining uncoded transcripts.

Stage 4. Next, a second set of 6 transcripts was selected and coded by Analyst A1 using the resultant codebook from Stage 3. Following Campbell *et al.'s* recommendation [13], we created new code categories as they emerged; this phase added 6 new code categories, each with less than 6 associated coded memos. Analyst A3 reviewed this updated codebook. *Stage 5.* The same process was applied to the final 6 transcripts, yielding 4 more code categories with fewer than 4 coded memos each. Of the 10 new categories recorded from stage 4 and 5, only 3 provided new insights into our data, as determined through multiple review and discussion rounds by Analysts A1 and A3. Thus A1 and A3 agreed that these new codes be classified as minor code categories.

Thematic and Framework Analyses: From this process, we induced themes following Braune & Clark's method [12]. These themes helped us answer RQs 2-4 directly. To answer RQ1 we also applied a framework analysis [70], mapping the themes to the software supply chain factory model as a reference framework to analyze participants' responses and further categorize the themes. This framework was selected because it serves as the general reference for SSC security.

Through our coding phase's thematic analysis, we identified eight themes Appendix A. Four of these themes were focused on the practical implementation of software signing, while the remaining four addressed various aspects of software signing: the challenges encountered, its perceived importance, the impact of security failures on its adoption, and the influence of security standards and regulations on its adoption.

To determine whether additional aspects of the phenomenon might be expected to emerge if we continued to sample, we measured saturation in the 18 interview transcripts. Following Guest *et al.'s* [33] recommendation, we measured saturation by tracking the cumulative appearance of new codes in each interview. Each interview was rich, with a median of 33 unique codes (Figure 4). As shown in Figure 4, saturation was achieved after interview 11.

4.5 Limitation and Threats to Validity

We discuss construct, internal, and external threats to validity [89]. Following the guidance of Verdecchia *et al.* [82], we focus on substantive threats that might influence our findings. We also describe the role that our own perspectives and experiences may have played in this research (positionality) [24].

Construct Threats are potential limitations of how we operationalized concepts. To mitigate these threats, we grounded our interview protocol in the academic and grey literature on software signing. Following best practices, we also piloted the interview protocol internally and externally to assess its utility in capturing the desired data [15].

Internal Threats are those that affect cause-effect relationships. Qualitative data implies subjective analysis. To improve the reliability of our results, we followed an iterative process to develop the codebook and used multiple raters with a high measured level of agreement. We also acknowledge the possi-

[2] Although there is no common agreement on a suitable proportion of the data set to select for this, O'Connor and Joffe recommend 10–25% as typical [57]. We randomly selected approximately 33% (6 transcripts) to develop our codebook. This decision to slightly exceed O'Connor and Joffe's recommended percentage was to familiarize analysts A2 and A3 with the nature of our collected data, proposed memo, and coding formats.

bility of a response bias, as some participants were recruited from the professional network of one of the authors. To address this, the involved author was not present during the interviews and played no role in the data analysis process.

External Threats may impact generalizability. This work is qualitative, which inherently carries limitations. Given the cost of an interview study, our work has a smaller sample size (N=18) than methods such as surveys can achieve. Our sample size is consistent with that of similar studies [40, 91]. Our subject recruitment approach may also have introduced biases, *e.g.,* towards CNCF/The Linux Foundation-oriented frameworks and tools. As a trade-off, using CNCF members as a starting point improved our access to high-ranking engineering staff. To reduce bias, we also recruited through snowball sampling and other professional networks. We assessed this risk with saturation — by the 11th interview our data saturated, supporting the adequacy of our sample size (Figure 4). We also segmented our interview protocol (Table 1) and intentionally excluded the tool-specific analysis from this study. Instead, we concentrated on parts of the protocol related to the general practice of software signing.

Our study focuses on an expert population to capture nuanced insights into software signing strategies and decisions. While this approach excludes the perspectives of the broader developer community, it aligns with our objective to understand signing practices at a strategic level. Less experienced developers may interpret or prioritize signing differently, reflecting their vantage point within the software development lifecycle. By targeting experts, we ensured that the findings are informed by those directly responsible for implementing and guiding signing strategies.

Positionality Statement: We acknowledge that our backgrounds may have influenced this study [24]. The author team are cybersecurity researchers with expertise in software signing, so our expertise both enabled and influenced the protocol development and analysis. In addition, one of the authors is a contributor to a popular signing tool. To reduce bias resulting from specific aspects of that tool, that author had no involvement in the development of the research protocol or analysis. However, that author informed the framing of the study and reviewed the results.

5 Results

5.1 RQ1: Signing Implementations in Practice

Part of our interview protocol (Table 1) ascertained how subjects use software signing in their security strategies at each stage of the software supply chain factory model (Figure 1). From these responses, we identified when Software signing was required by the subject's team (or organization) and mapped these instances back to the stages of the software supply chain factory model. The summary of our framework analysis is presented in Table 3.

Table 3: Different Signing Use Points, categorized by the stages of the software supply chain factory model highlighted in Figure 1 and subjects who employ software signing at these points. We mark subjects who consider them optional for reasons of demand by customers, nature of products, team requirements, etc. The points highlighted (PI, VI, PE, VE, PS, VS, PB, VB, PA, VA, PD, VD) are described in Figure 3.

Points & description of Software signing use	Subjects
SOURCE CODE	
Internal code contributors sign commits-**PI**	S2-S11, S14-S17 (S4, S8, S14–Optional)
External code contributors sign commits-**PE**	S7, S10 (S11, S14–Optional)
Verify signatures from Internal contributors-**VI**	S7, S14, S17
Verify signatures from external contributors-**VE**	–
Signing after code reviews/audits-**PS**	S2-S11, S14-S17 (S4, S8, S14–Optional)
BUILD	
Verify source code signatures before build-**VS**	S2, S3, S7, S14
Signing of build output-**PB**	S1-S5, S7, S8, S11-S17 (S8-Optional)
DEPLOYMENT/PACKAGE/ARTIFACT	
Verify build signature before deployment-**VB**	S7, S14
Signing of Final Software product-**PA**	All S1–S18
THIRD-PARTY DEPENDENCY	
Signing after verification and certification of dependencies-**PD**	S6, S7, S9, S16, S17, S18
Verify signatures from external dependencies-**VD**	S9, S11, S17 (S9, S11, S17 — Optional)
Verification by final customers-**VA**	Difficult to determine

5.1.1 Source Code

First, we ascertained whether our subjects used any form of external code contributors in their software authorship process. Out of 10 subjects who reported a form of collaboration with open-source or other external contributors, only 5 subjects used external code contributions in the production code of projects they owned (both commercial and OSS). We highlight the points and manner in which subjects reported the use of software signing to create their source code.

External contributors sign commits: Most teams who had input from external code contributors required Software signing. Out of 5 subjects (4 organizations) whose teams used external code contributions in their production code, 4 of these subjects (3 organizations) require that commits be signed. However, two of those do not require signing if the contribution is toward their open-source projects or noncritical projects. Only 2 of our subjects (2 organizations) unconditionally require signing from their external contributors. As S2 *(SAAS, SSC security tool)* put it, *"I got something signed from somebody I don't know...So we do the code review, and then we have a sign-off by somebody on our team that signs off from that code"*.

Internal contributors sign commits: Signing is typically required by teams for their internal code contributors. 14 of 18 subjects report that their team required commit signing from internal contributors. Three subjects (*S4*, *S8*, *S14*) described this as an optional requirement and largely agreed with S2's

reasoning that signing alone does not establish trust in identity. Notably, these subjects' teams often rely on contributions from both internal and external contributors.

Verification of signatures from contributors: Although code contributions are often required to be signed, the signatures are not always verified to ensure the security and provenance of the contributions. According to our subjects, the most commonly used security methods for assessing the security of code contributions include techniques like code reviews, security audits, vulnerability scans, and tests. For example, S10 *(SAAS, dev tools)* states, *"I think everybody on the team does sign their commits, but we don't really verify it...If somebody stopped signing their commits, I don't think much would really happen."*

Code is signed after reviews/audit: The outputs of code reviews and audits are generally required to be cryptographically signed. Subjects whose teams(or organizations) mandate signing for code contributions by internal contributors also report that all activities related to code reviews and audits must be signed as well (14 subjects).

5.1.2 Build

Next, we examine our subjects' responses regarding how signing is used in the build process of software products. For most subjects, the build process and the signing of build outputs (for subjects who do so) are automated. We expect that automation increases the use of software signing.

Verification of source code signatures before build: Signatures from source code are rarely verified before the build process begins. Four subjects mentioned some form of rigorous verification at this stage. For those who do not use or verify signatures, alternative security techniques are regularly employed, such as automating build pipelines with workflow definitions, using build policies, implementing reproducible builds, and utilizing code scanning tools to ensure the security of inputs to the build process.

Signing of build output: 14 subjects (11 organizations) reported signing at this stage of the software delivery process. 10 subjects (6 organizations) reported signing alongside attestations, SBOMS, and other metadata.

5.1.3 Package/Artifact

Next, we analyze the use of signing at the final stage of deploying software products.

Verification of build signature before deployment of final artifact: Similar to the other verification steps, signatures from the build phase are rarely verified before the deployment. Only two subjects (1 organization) mentioned verification at this point. For example, S7 *(security tools, cloud security)* states, *"We sign our code and then we also verify that it was built in our specific build system."*

Signing of Final Software product: All 18 subjects said that signing is done at the deployment stage to establish provenance for the organization on each of their final products.

5.1.4 Third-Party Dependencies

For third-party dependencies in the software supply chain, we assessed how software signing influences dependency selection as a security factor and how signatures are used to verify the authenticity of dependencies in practice.

Presence of a Signature as a Factor for Dependency Selection: When they select third-party dependencies, most subjects do not consider whether it is signed. Only three subjects view signatures as an indicator that the package maintainer prioritizes security. Other security factors influencing the choice of third-party dependencies include the use of OpenSSF's scorecard tool, known vulnerabilities, level of activity, licensing, and the age of the dependency.

Verification of Signatures from external dependencies: Signatures on third-party dependencies are rarely verified in practice. Only three subjects (3 organizations) reported some form of verifying signatures from third-party dependencies. Signatures are mainly verified for commercial off-the-shelf software. As S11 *(firmware & testing software, social technology)* says, *"If it is open source, then probably no...But if it is commercial off the shelf, we want to make sure those are signed."* However, most of our subjects reported several methods for assessing the authenticity and integrity of a third-party dependency. These methods include code and vulnerability scanning (*e.g.,* Coverity), source rebuilding, internal mirroring of dependencies, and reliance on ecosystem internal checks (*e.g.,* Go package manager's checks). Other subjects perform internal reviews of dependencies. Some have a centralized dependency selection and management process, where the organization hosts and maintains dependencies (5 subjects from 4 organizations). A few subjects noted that their organization discourages or forbids third-party dependencies (S13 & S8). In addition to relying on alternative methods to assess dependency security, most of our subjects echoed the sentiment that OSS dependencies are rarely signed and that the identity of individuals signing open-source dependencies does not inherently establish trust.

Signing after verification and certification of dependencies: A few subjects reported that third-party dependencies are thoroughly reviewed using one of the aforementioned techniques before being signed off for use by a team member. Six of our subjects indicated that this process is followed in their software production workflow. For example, S17 *(SAAS, SSC security tool)* described their team's process as this: *"We have a signed report that's signed by us, and we trust us...that says, 'Hey, I scanned this third-party dependency, did all these things. I don't have a signature for that, but I ran all these extra checks and then signed those checks. So now we have a record."'* In contrast to that attestation approach, S18 *(secu-*

rity tooling, internet service) says their team ingests third-party dependencies and signs the internal versions directly.

5.1.5 Refined Software Supply Chain Factory Model

As a result of our framework analysis, we identified various points where software signing is utilized. We compared these to the descriptions in several Software Supply Chain security frameworks, especially the SSCFM (§4.2, Figure 1).

We observed that some of these frameworks and standards are ambiguous, particularly in specifying where within the software supply chain factory model practitioners are required to establish provenance, whether by creating or verifying a signature. We therefore present a refined version of Figure 1 in Figure 3. This updated model clearly identifies the points where provenance should (could) be established, providing guidance for organizations and teams newly adopting software signing in their software creation process. For example, our refined model is compared with the SLSA framework [77]. Despite SLSA's rich description of security recommendations and its focus on build levels, there is surface-level confusion about whether provenance requirements apply solely to the build stage or also to earlier stages such as collaboration.

5.2 RQ2: Challenges Affecting the Implementation of Software Signing in Practice

Our next research objective was to understand the various challenges practitioners face while implementing software signing in practice. We grouped subjects' responses under three main themes for challenges faced by practitioners in practice; Technical, Organizational, and Human factors. Table 4 gives a summary. Due to space limitations, we focus on only a subset of the individual challenges.

5.2.1 Technical Challenges

Key Management: Key management issues were the most reported issues. Issues under this category vary from *key management, key distribution, key generation, and public key infrastructure (PKI)* issues.

Key management issues are the highest categories recorded. *S13* articulates this, especially concerning key distribution: *"a lot of the ways in which I think previous teams' software signing hasn't been useful is...key management...in terms of how the private key is managed but also in how the public key is made available."*

Some subjects discussed recent signing technologies that implement keyless signing (*e.g.,* Sigstore) as a possible solution. S18 *(security tooling, internet service)* said: *"Sigstore comes in...keyless signing...you don't have to worry about long-lasting static SSH keys, or keys being compromised, or rotating keys...you don't have these big servers to store all these signatures"*. S17 *(SAAS, SSC security tool)* also noted, *"[Sigstore's] other strength...I can associate my identity with an OIDC identity as opposed to necessarily needing to generate a key and keep track of that key and yada, yada. So that's super useful because I could say, 'Oh, this is signed with [a] GitHub identity.' So unless my GitHub identity has been compromised, that's much better"*.

Compatibility Issues: Compatibility issues arise during the integration of software signing systems and tools with other tools within the software engineering process, including other security tools and the artifacts to be signed, they also include scenarios where challenges with signing were linked to other security toolings,.

S2, *S7*, *S17*, and *S18* highlight issues in the compatibility of newer signing systems with existing software systems in the software engineering process. As *S18 (security tooling, internet service)* put it, *"There are teams who've been doing this for quite a while...Selling that is a big challenge, right? Because we need to prove that it is much more secure than what has been there in place."*

S8 *(internal security tool & cloud APIs, internet services)*, whose organization's software product is built for different platforms and hardware architectures, noted compatibility issues across various architectures: *"Every platform evolved separately...macOS...Windows...The requirements are changing and evolving independently"*.

Compatibility issues can also occur in situations where current security test tools are not designed to identify issues with Software signing, as noted by S5 *(internal security tool, cloud security)*: *"The vulnerability scan report [is for]...code...CVE published...[but] signatures, I don't think it will pick those things up."*

Ease of Use/Usability: Usability of the signing tool or implementation was a common challenge hampering effective use in practice. This was often expressed as developer experience, *e.g.,* S17 *(SAAS, SSC security tool)* said: *"For a developer, make sure that they can sign the commits easily"*. This point was also raised at a team level by S13 *(internal security tool, telecommunication)*: *"I think that awareness of the value of signing and sophistication around how signing is performed has increased significantly, especially in recent years. But it has also resulted in a lot of smaller teams deciding that this is not a manageable thing."*

Concerning documentation, S11 *(firmware & testing software, social technology)* said: *"I think there are multiple challenges there. When you build some signing infrastructure, you want to make it a self-serve so the teams can onboard themselves, how well you have your self-serve playbooks written down, documentation on using those. I think documentation can be improved."*

Lack of Verification of Signatures: Lack of verification of signatures is also quoted by several subjects as a main challenge. This arises from the factors like signing serves as a "checkmark" security tool (*S4 (open source tooling, cloud dev*

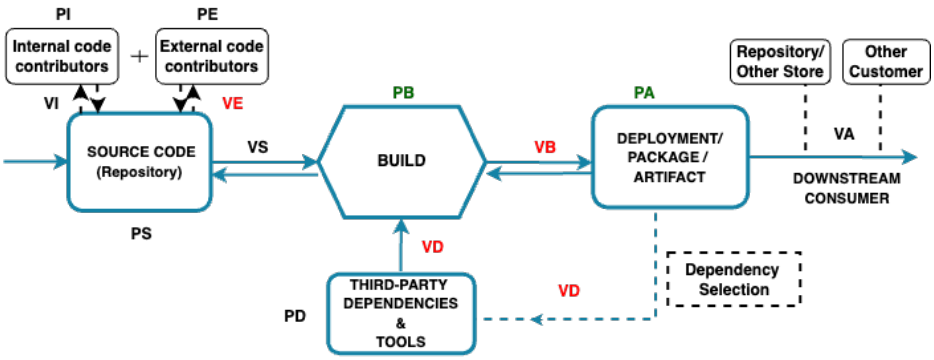

Figure 3: Our refined software supply chain factory model (compare to Figure 1), highlighting different points where provenance and integrity of software could be established (PI, PE, PB, PD, PS, PA) and Verified (VI, VE, VB, VD, VS, VA) using software signatures. These points were identified from participants' responses. Note that while signing at PB and PA stages was commonly practiced (in green), verification at VD, VB and VE stages was minimally performed (in red). Abbreviations are defined in Table 3.

Table 4: Challenges to software signing implementation in practice. We have categorized related challenges highlighted by subjects. Additional technical challenges, 'Managing Signed Artifacts' and 'Identity Management/Authentication,' are detailed in our linked technical report Appendix A, as they are not exclusive to signing tools.

Observed Challenges	#Subjects	#Orgs	Subjects' Proposed Solutions
Technical			
Key Management	10	9	Use of Keyless Signing (*e.g.,* Sigstore)
Compatibility Issues	6	6	—
Lack of Verification of Signatures	6	5	Signed Metadata, Component Data Management
Ease of Use/Usability	4	4	Usable Signing Tools (*e.g.,* Sigstore), Documentation
No Unifying Standard	2	2	—
Organizational			
Operationalization of the Signing Process	4	4	Automating Signing
Resources to Set up Signing	3	3	—
Creating Effective Signing Policy	2	2	Regular Process Feedback Mechanisms
No Management Incentive to Sign	2	2	—
Bureaucracy	1	1	—
Human			
Expertise in setting setup and use of signing	5	4	—
Developer Attitude to Signing	3	3	Automating Signing
Lack of Demand from Customers	1	1	—

tool): *"most people who are strictly signing all commits don't have a meaningful process for verifying those signatures or for enforcing any policy around them. They just do it because"*), and lack of customer interest (*S14 (POC software & security tool, cloud security)*: *"We have tried to ... get potential customers and customers interested...but as far as I know, none of our customers...verify our signatures on images."*)

Verification of Artifact Before Signing: Verification generally was highlighted as a problem, especially verifying inputs to the build process of the software delivery process. The problem noted here is that the content of software artifacts is not readily verified before signing: as S16 *(security tooling, digital technology)* put it, *"if you start with the component, the component gets put into a subassembly. The subassembly gets put into a package. The package can get put into a broader package or whatever. I think the people are signing. But do you know that the sum of all the pieces are accurate? ... what are you really signing, and at what level?"*

5.2.2 Organizational Challenges

Operationalization of the Signing process: The implementations of software signing are typically neither uniform nor centralized across different organizations. Various teams are largely responsible for crafting their own solutions. As S6 *(security tooling, digital technology)* said, *"I can't be more specific, but today, I've talked to three different teams with three different processes"*. S15 *(internal security tooling, aerospace security)* suggested that automation could improve uniformity: *"We try to break down the barriers with really good examples, by*

having common configuration of CI through similar GitHub workflows, and stuff like that. So it's easier for people to use."

Resource to Set up Signing: The issue of organizational resources to set up efficient signing tooling across an organization was also another cause for concern for some of our subjects. Subjects discussed this in terms of finances, time, and engineering effort. S14 *(POC software & security tool, cloud security)* said, *"We need to do a lot of signing...hundreds of container images [daily]...It was a pretty significant engineering effort to build a pipeline that can handle all those signatures."*

Creating Effective Signing Policy: Policies surrounding the implementation of software signing by teams are generally not considered highly effective. These policies include elements such as role-based access, key usage, and key issuance.

> **S3** ***(SAAS, SSC security tool)****: "... the other thing is policies, creating a policy that is, for one, it's hard to do. And then the other part is creating an effective policy and knowing that it's effective."*

A suggested solution to address this issue is the use of regular feedback mechanisms to assess the security policies and processes of different teams within the organization. As S11 *(firmware & testing software, social technology)* said, *"We want to be more proactive...internal audits, and an assessment methodology like the NIST, I think CISF (NIST's CSF)."*

No Management Incentive to Sign: A lack of management incentives was also a challenge. *S11 (firmware & testing software, social technology)* said, *"The investment, the time that the engineer is putting into...signing...before a release, does it get the right attention from the senior management? Do they view it as a time well spent for the product?"*

5.2.3 Human Challenges

Expertise: Some subjects noted that setting up signing tools depends on the engineer's expertise. Subjects felt that most engineers lack the necessary expertise. As S9 *(SAAS, SSC security)* said, *"there's a lift there as far as...expertise"*.

Developer Attitude to Signing: Software developers' attitudes toward software signing pose challenges to the signing process. Disinterest and resistance to adopting new tools are the primary attitude-related obstacles cited. S10 *(SAAS, dev tools)* observed: *"Most developers don't really understand this stuff [or] want to think about it...[they are] happy if they...see a big green tick[mark]."* S6 *(security tooling, digital technology)* added: *"They refuse to get off their old tools."*

Lack of Demand from Customers: Most customer-driven organizations report that software signing is challenging to implement because customers rarely request this feature in their requirements. S14 *(POC software & security tool, cloud security)* put it like this: *"Sign[ing] our images was there from day...we haven't found many customers that are too excited about the signing aspect."*

5.3 RQ3: Perceived Importance of Software Signing in Mitigating Risks

In this section, we aimed to understand how practitioners perceived the importance of software signing in their software delivery security. Opinions on the significance of signing in the software delivery process varied. Subjects identified three distinct perspectives on the importance of software signing, reflecting on their personal views and how signing was treated in their current and previous roles.

5.3.1 Signing as Crucial to Provenance and Integrity

Many of our subjects considered provenance an integral part of security, and felt that software signing is (or will be) a guarantee of that. 10 subjects (representing 9 organizations) had the opinion that Software signing on its own is strong enough to guarantee provenance. *S4* said, *"I think signing is massively important. Signing for open source is probably the single most important thing for software supply chain security"*. *S16*'s response mirrors this, *"I think [Signing's importance] is very high...There's a senior executive VP who has to attest that the software we release is what we say it is ... he could be legally sued if the software we deliver... I mean personally, not just our organization. Him, because of his level in the organization, if we publish software that we have incorrectly attested to, he can be ... So it's a very high priority."*

5.3.2 Signing as a Secondary Security Technique

Some subjects believe that software signing is a supplementary method used to ensure the integrity of primary security techniques. 7 subjects (6 organizations) had the opinion that Software signing does not provide enough guarantee for provenance. They often deemed it important only in conjunction with methods like Attestations and other Manifest forms such as the SBOM. For these organizations, signing is a valuable part of a broader, multi-layered security strategy. Some subjects who view signing as a strong guarantee of provenance also emphasized that adding a manifest or attestation further strengthens this guarantee. As S2 *(SAAS, SSC security tool)* said, *"I think the most important thing is accurate metadata collection and centralization. And even if that data's not signed, we can still get a lot of usefulness out of it. Signatures, in my opinion, they offer an additional layer of security on top of what we should be doing, which is metadata collection."* S17 *(SAAS, SSC security tool)* similarly views signing as a final check on top of other security practices: *"Signing was essentially the way of saying, 'I have checked, we did all the [security practices]'."*

5.3.3 As a Requirement-Driven Practice

Subjects in this category indicated that software signing was primarily used as a "check the box" technique to satisfy regulatory, framework, customer or organizational requirements. They did not view signing as a primary security measure. Instead, it was employed mainly to meet regulatory or contractual obligations, rather than being treated as an essential part of their security strategy. For these subjects, signing was often seen as a procedural necessity—for instance, to submit a pull request, initiate a build, or make a commit—rather than as a measure to enhance security. 4 subjects representing 3 organizations fall into this category.

S17 ***(SAAS, SSC security tool)****: "... at certain organizations it felt like it (software signing) was more of a check-the-box thing of like, "Hey, we should sign it because that's what people tell us to do"*

S8 ***(internal security tool & cloud APIs, internet services)****: "...for signed binaries and code signing ...a lot of those [requirements] are driven...by platform requirements...to prevent us from being blocked from...these platforms, code signing...was largely driven by [these platforms]."*

5.4 RQ4: Impact of Internal & External Events on Software Signing Practices

In this section, we explore the impact of two kinds of security events: security failures such as Log4j, and the advent of security standards and regulations:

1. ***Experienced Software Supply Chain Failures***: We consider these internal if a subject's organization is a direct victim of a Software Supply Chain attack, and external if a subject's organization is not directly impacted in a Software Supply Chain event, *e.g.,* a general Software Supply Chain incident like the Solarwinds attack.
2. ***Software Supply Chain Standards***: These are industry wide external events *e.g.,* the publishing of the EO-14028 executive order on Software Supply Chain security.

We asked subjects about their organizational experiences with software supply chain failures (attacks, vulnerabilities, and other incidents) and how these experiences affected their team's or organization's software supply chain security decisions, particularly regarding software signing. Additionally, we queried subjects on the impact of software supply chain security frameworks, standards, and regulations on their use of software signing and other security techniques.

In their responses, subjects described a security failure they had experienced. We categorized each failure as a Vulnerability (unexploited security flaws), an Incident (SSC failures with no known malicious intent), or an Attack (deliberate and malicious failures). Then, subjects described changes these failures prompted in the organization. We categorized these as Direct Fixes (patched but no change to organizational security approach), General Security Process (large-scale changes involving multiple security techniques), and Signing (adoption or change in the implementation of software signing).

5.4.1 Impact of Experiences with SSC Failures

Table 5 summarizes these events and the resulting changes. Most cases were non-malicious, either Vulnerabilities or Incidents. However, four subjects had experienced attacks, with one major attack leading to extensive collateral damage and a complete overhaul of software and security processes.

Across these events, the typical resolution was via Direct Fix, without an infrastructural change in the general security process or the signing process specifically. When they occurred, changes to the general security process typically involved adoption of a group of tooling such as for SBOMs. As a result of one event, a vulnerability led to changes in the signing process. In this case, the team (which had a safety-critical application) integrated in-toto's signed attestation capability. The subject S2 *(SAAS, SSC security tool)* stated: *"We saw that [in-toto] allowed you to decouple the security metadata collection and validation...that allowed us to solve this problem of how do we know that this opaque process for the software, that opaque certification...was followed without...knowing the details of...that process."*

5.4.2 Impact of Security Regulations, Frameworks, and Standards

Our subjects reported that security frameworks, standards, and regulations had *no direct impact* on their organizations' adoption of software signing as a security method. However, these standards and regulations did influence the general adoption of other security techniques, such as SBOM. 9 participants from 5 organizations shared this view. Only 2 (from 2 organizations) from this group explicitly linked this to signing. 5 participants (from 4 organizations) gave unclear statements (personal opinions; unfamiliarity with how decisions are made).

We asked our subjects, "Are [your SSC security] strategies influenced by regulations and standards?". S1 *(POC software, digital technology)*'s response was typical: *"Yeah, I would say certainly the collection of SBOMs and doing that more broadly has been a response to things like the executive order, at least for our team...The executive order played just a strong role in what we are starting to do now. Whether some teams are now trying to be preventative, that's a good question."*

Table 5: Kinds of software supply chain failures experienced, and associated security changes. Only one participant reported an influence of these failures on their software signing implementation. Software supply chain failures were more likely to prompt a direct fix rather than a change in the security process.

Type	# Subjects (# Orgs)	# Direct fix	# Change in General Security process	# Change in Signing process
Vulnerability	9 (6)	9	2	1
Incident	5 (5)	5	2	0
Attack	4 (4)	4	2	0

6 Discussion

6.1 Impact of Organizational Size and Type of Produced Software on Signing Practices

We did not observe any significant effect of organizational size (and nature of software product) on where signing was implemented (RQ1, Figure 3). However, we did note patterns in the challenges reported (RQ2) and the influence of standards and regulations on the adoption of software signing (RQ4). It is important to note that these observations represent correlations rather than causations. While our findings highlight potential relationships, further study is necessary to establish causative factors and better understand the mechanisms driving these patterns.

Effect on Challenges. We observed a noticeable difference in the organization size of participants who reported certain type of challenges. Only subjects from large organizations (S6, S8, S13, S15) talked about difficulties in operationalizing the signing process. We conjecture this is because of the greater coordination effort required to harmonize the engineering process of several teams.

The nature of the software product also seem to exert some influence on the challenge experienced. Unsurprisingly, only subjects from organizations with non-security-focused product areas (*S11 (firmware & testing software, social technology)* and *S13 (internal security tool, telecommunication)*) reported a lack of management incentive as a barrier to signing. While Only participants from security-focused product teams (*S3 (SAAS, SSC security tool)*, *S10 (SAAS, dev tools)*) reported challenges in creating effective signing policies.

Effect on Influence of Standards and Regulations. As we reported, nine participants from five organizations noted that standards and regulations influenced the adoption of tools like SBOM. Within this group, only two participants (from two of the five organizations) explicitly linked this to signing. Notably, all five organizations were either large or focused on security-related products.

6.2 Relation between Participants' perceived importance of Signing and Signing Implementation

We remark on a recurring dissonance between the perceived importance of Software signing by the participants and their implementation practices. Although all organizations signed their final products, signatures were rarely used to verify integrity or provenance Table 3. Ten subjects (nine organizations) regarded Software signing as a strong guarantee of provenance, but only three participants (from two organizations) reported verifying signatures for internal contributors; none did so for external contributors. This suggests a unidirectional use of Software signing, establishing trust for outputs while placing lower expectations on the provenance of inputs. Many of these organizations publish software with a higher degree of provenance than they demand from their dependencies or external contributions.

6.3 Recommendations from this study

Improving Signature & Artifacts Verification in Signing Workflows. Our refined software supply chain factory model has highlighted a significant gap in the current practice of signature verification. This issue was underscored by several subjects in our study.

In certain automated workflows, the act of signing is often treated as a mere formality, serving as a "checkbox" that, once done, allows engineers to carry out their usual tasks (e.g., creating pull requests or merging code into the main branch). Thus, this checkbox provides little security properties, and requires effort to provide meaningful security gains.

To strengthen provenance guarantees, signing should be complemented with transparency-based methods, ensuring both the signer's identity and artifact integrity. This requires consistent verification of signatures and artifact content to provide meaningful security benefits.

Future works in this area should focus on proposing improved tooling around existing or new software signing tools to include these capabilities in their design. This enhancement would not only improve transparency but also reduce the costs and expertise requirements for setting up such infrastructure. Also, while much of our recommendations focus on "improving tooling", further research is needed to define what kinds of tooling would qualify. This research might

involve experimental and empirical analyses of the current generation of software signing tools, particularly in terms of functionality, integration [63], and usability.

As additional takeaways, we emphasize the need for further research to address critical trust gaps within the open-source ecosystem. Our findings (§5.1.4) highlight an underlying lack of trust in open-source packages. Cross-ecosystem provenance maintenance could serve as a potential solution.

By addressing these issues, we can ensure that signing is not just a procedural element and instead plays a crucial role in maintaining the integrity and authenticity of the software development process.

Addressing Incentive Challenges in Software Signing. One of the most frequently reported challenges in the realm of software signing is the lack of proper incentives for its adoption. These incentives can range from support from management, the ease of using signing tools, as well as comprehensive education on the importance and application of signing. Although some of these issues, such as the need for educating practitioners on the significance and application of signing, can be addressed with relative ease, others necessitate more elaborate planning. For instance, fostering management support might require demonstrating the tangible benefits of software signing in terms of enhanced security and reduced risk of software tampering. Moreover, though the user-friendliness of signing software has been evidenced by various "Why Johnny Can't" papers, more research is needed in lowering the adoption bar for software signing systems. Regardless of the complexity of these challenges, investing in these areas is paramount for enhancing an organization's security posture. By fostering a culture that values and understands the importance of software signing, organizations can ensure the integrity of their software and protect against potential security threats.

Future research could focus on developing strategies to address these incentive-related challenges and exploring their effectiveness in different organizational contexts. This could provide valuable insights for organizations striving to improve their software signing practices.

6.4 Contrast to prior work

Comparisons with Previous Quantitative Studies on Software Signing. We compare our findings with Schorlemmer *et al.'s* [62] quantitative study on software signing in open-source ecosystems. Their results indicate that major software supply chain failures (e.g., SolarWinds, Codecov attacks) and dedicated tooling had no significant impact on the quantity of signatures. In contrast, our findings for the non-open source industry show the opposite. This difference is expected due to the substantial incentives, such as commercial and reputation losses, that drive greater adoption in the non-open source industry. Additionally, our study supports this finding, as some subjects noted that contributions to the open-source ecosystem often involve relaxed security requirements.

Distinctions of Refined Model Compared to Existing Frameworks. While all software supply chain security frameworks agree on the defined threat model, our refinement of this model differs from existing ones in several ways.

1. SLSA: The SLSA framework primarily focuses on securing the build process, offering recommendations as a series of incremental security hardening levels. In contrast, our refinement addresses the entire software supply chain, not just the build step.

2. SCIM and S2C2F: This framework provides threat-reduction recommendations without specific implementation details. Our approach, however, emphasizes the continuous establishment and verification of provenance across the software supply chain.

3. Hammi *et al.* [34]: Their work proposes a blockchain-based cryptographic tool. We do not advocate for a specific approach. Instead, our model allows industry and software teams to create an implementation workflow meeting their needs. Our findings are supported by insights from industry practitioners, enhancing their practical applicability.

7 Conclusion

Software signing is widely recommended for ensuring software provenance, yet many organizations face challenges when integrating it into their processes. In this first qualitative study, we examine the practices and challenges of software signing through interviews with 18 senior practitioners across 13 organizations. We identified that while software products are signed, the signatures often go unvalidated. Key barriers include infrastructure limitations, lack of expertise, and insufficient resources. Although signing offers strong provenance guarantees, it remains secondary in many organizations' cybersecurity strategies. Additionally, organizational behaviors are influenced by major incidents like SolarWinds and industry regulations and standards, with concerns raised about the quality and development of these standards.

Our findings shed light on the real-world practices, challenges, and organizational factors shaping software signing adoption, offering guidance for organizations on their journey toward more secure software supply chains.

8 Acknowledgments

We thank the study participants for contributing their time to this work. We thank the reviewers and T. Schorlemmer, S. Okorafor, and W. Jiang for their feedback. We acknowledge support from Google, Cisco, and NSF #2229703.

9 Ethical Considerations

This work considers the human and organizational factors governing the adoption of software signing techniques in software engineering work. To elicit this data, we developed a research protocol, including subject recruitment, interview content, and data processing and analysis. This protocol was approved by our organization's Institutional Review Board (IRB) prior to the study conduct. All members of the research team who interacted with subjects or their non-anonymized data have completed our organization's training procedures for handling personally identifiable information. Data from individual subjects were anonymized following best practices for qualitative data analysis.

Any human-subjects research effort must weigh the risks and benefits for individual research subjects as well as for society. The primary *risk* for subjects is the hazard to their individual or organizational reputation when describing what might be interpreted as sub-standard cybersecurity practices. We therefore anonymized subjects by rank and by organization. A secondary cost for subjects was their time. Subjects were not coerced to participate, and we offered an incentive of $100 for their participation. The primary potential *benefit* for our subjects, their organizations, and the societies in which they operate, is that the resulting data may motivate improvements in the software signing ecosystem, as we discussed in §6. In our judgment, the potential benefit to software cybersecurity far outweighs the risks posed to our subjects and their organizations.

We therefore attest that:

- We read the ethics considerations discussions in the conference call for papers, the detailed submissions instructions, and the guidelines for ethics document.
- The research team considered the ethics of this research — the authors believe the research was done ethically, and the team's next-step plans (*e.g.,* after publication) are ethical.

10 Compliance with Open Science Policy

We acknowledge that USENIX Security has an open science policy. Authors are expected to openly share their research artifacts by default. The research artifacts associated with this study are:

- Raw transcripts of interviews
- Anonymized transcripts of interviews
- Interview protocol
- Codebook

Things we have shared: The *interview protocol* is a crucial part of any interview study, since it allows for the critical review of a study design as well as its replication. Our full interview protocol is included in our technical report (Appendix A). Since it is semi-structured, we include all of the questions asked of all subjects, as well as examples of the follow-up questions we asked. We also share an excerpt of our *codebook*, with codes, definitions, and example quotes that we coded for each code in our technical report — see Appendix A.

Our full *codebook* is available at https://doi.org/10.5281/zenodo.14660193.

Things we cannot share: For subject privacy reasons, and for IRB compliance, we cannot share the raw transcripts. We are also unwilling to share the anonymized transcripts of the interviews. Given the high organizational ranks of many of our subjects, and the small size of the subject pool resulting from our recruiting strategy, we believe there is a high risk of de-anonymization even of anonymized transcripts. Therefore, we cannot share the anonymized transcripts.

References

[1] tag-security/supply-chain-security/secure-software-factory/secure-software-factory.md at main · cncf/tag-security.

[2] CNCF Security Technical Advisory Group, 2023.

[3] microsoft/scim, 2023.

[4] Marco Abbadini, Dario Facchinetti, Gianluca Oldani, Matthew Rossi, and Stefano Paraboschi. Cage4Deno: A Fine-Grained Sandbox for Deno Subprocesses. In *ACM AsiaCCS*, 2023.

[5] Samuel A. Ajila and Di Wu. Empirical study of the effects of open source adoption on software development economics. *Journal of Systems and Software*, 2007.

[6] Husam Helmi Alharahsheh and Abraham Pius. A review of key paradigms: Positivism vs interpretivism. *Global Academic Journal of Humanities and Social Sciences*, 2020.

[7] Maysara Alhindi and Joseph Hallett. Sandboxing adoption in open source ecosystems. In *Proceedings of the 12th ACM/IEEE International Workshop on Software Engineering for Systems-of-Systems and Software Ecosystems*, SESoS '24, page 13–20, New York, NY, USA, 2024. Association for Computing Machinery.

[8] Paschal C Amusuo, Kyle A Robinson, Santiago Torres-Arias, Laurent Simon, and James C Davis. Ztdjava: Mitigating software supply chain vulnerabilities via zero-trust dependencies. *arXiv:2310.14117*, 2023.

[9] Nafisa Anjum, Nazmus Sakib, Juanjose Rodriguez-Cardenas, Corey Brookins, Ava Norouzinia, Asia Shavers, Miranda Dominguez, Marie Nassif, and Hossain Shahriar. Uncovering Software Supply Chains Vulnerability: A Review of Attack Vectors, Stakeholders, and Regulatory Frameworks. In *IEEE COMPSAC*, 2023.

[10] Sebastian Baltes and Paul Ralph. Sampling in software engineering research: a critical review and guidelines. *Empirical Software Engineering*, 2022.

[11] Chris Bogart, Christian Kästner, James Herbsleb, and Ferdian Thung. When and How to Make Breaking Changes: Policies and Practices in 18 Open Source Software Ecosystems. *ACM Transactions on Software Engineering and Methodology*, 2021.

[12] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 2006.

[13] John L Campbell, Charles Quincy, Jordan Osserman, and Ove K Pedersen. Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement. *Sociological methods & research*, 42(3):294–320, 2013.

[14] G. Cantone. Software factory: modeling the improvement. In *1992 Third International Conference on Factory 2000, 'Competitive Performance Through Advanced Technology'*, pages 124–129, 1992.

[15] Ronald J Chenail. Interviewing the investigator: Strategies for addressing instrumentation and researcher bias concerns in qualitative research. *Qualitative report*, 16(1):255–262, 2011.

[16] CISA. Securing the software supply chain for developers. Technical report, Cybersecurity and Infrastructure Security Agency, 2022. Accessed: 2024-08-28.

[17] Cloud Native Computing Foundation. Software supply chain best practices, May 2021. https://github.com/cncf/tag-security/blob/main/supply-chain-security/supply-chain-security-paper/CNCF_SSCP_v1.pdf.

[18] Information Technology Laboratory Computer Security Division. Secure Software Development Framework | CSRC | CSRC, 2021.

[19] David Cooper, Larry Feldman, and Gregory Witte. Protecting Software Integrity Through Code Signing. Technical Report ITL Bulletin, National Institute of Standards and Technology, May 2018.

[20] David Cooper, Andrew Regenscheid, Murugiah Souppaya, et al. Security Considerations for Code Signing. *NIST Cybersecurity White Paper*, January 2018.

[21] Melissa DeJonckheere and Lisa M Vaughn. Semistructured interviewing in primary care research: a balance of relationship and rigour. *Fam. Med. Community Health*, 2019.

[22] Department of Defense. Open Source Software (OSS) in the Department of Defense (DoD), May 2003.

[23] Docker. Signing docker official images using openpubkey, 2023. https://www.docker.com/blog/signing-docker-official-images-using-openpubkey/.

[24] Joan E Dodgson. Reflexivity in qualitative research. *Journal of human lactation*, 2019.

[25] Ruian Duan, Omar Alrawi, Ranjita Pai Kasturi, Ryan Elder, Brendan Saltaformaggio, and Wenke Lee. Towards Measuring Supply Chain Attacks on Package Managers for Interpreted Languages, 2020. arXiv:2002.01139.

[26] Guangchao Charles Feng. Intercoder reliability indices: disuse, misuse, and abuse. *Quality & Quantity*, 2014.

[27] Hal Finney, Lutz Donnerhacke, Jon Callas, Rodney L. Thayer, and Daphne Shaw. OpenPGP Message Format, 2007.

[28] Simson Garfinkel and Gene Spafford. *Practical UNIX and Internet Security*. O'Reilly Media, Sebastopol, CA, 2003.

[29] Betul Gokkaya, Leonardo Aniello, and Basel Halak. Software supply chain: review of attacks, risk assessment strategies and security controls, 2023. arXiv:2305.14157.

[30] Google Cloud. Best practices for software supply chain security, 2023. Accessed: 2024-08-28.

[31] Wolff Growley, Lerner Gruden, and Welling Canter. Navigating the SolarWinds Supply Chain Attack. 56(2), 2021.

[32] GSA: 18F. 18F: Digital service delivery | Open source policy. https://18f.gsa.gov/open-source-policy/.

[33] Greg Guest, Arwen Bunce, and Laura Johnson. How many interviews are enough? an experiment with data saturation and variability. *Field methods*, 2006.

[34] Badis Hammi and Sherali Zeadally. Software supply-chain security: Issues and countermeasures. *Computer*, 2023.

[35] Ethan Heilman, Lucie Mugnier, Athanasios Filippidis, Sharon Goldberg, Sebastien Lipman, Yuval Marcus, Mike Milano, Sidhartha Premkumar, and Chad Unrein. OpenPubkey: Augmenting OpenID Connect with User held Signing Keys, 2023. Publication info: Preprint.

[36] Trey Herr, William Loomis, Stewart Scott, and June Lee. Breaking trust: Shades of crisis across an insecure software supply chain, 2020. https://www.atlanticcouncil.org/in-depth-research-reports/report/breaking-trust-shades-of-crisis-across-an-insecure-software-supply-chain/.

[37] Immy Holloway. Basic concepts for qualitative research. *(No Title)*, 1997.

[38] Wenxin Jiang, Nicholas Synovic, et al. An Empirical Study of Pre-Trained Model Reuse in the Hugging Face Deep Learning Model Registry. In *International Conference on Software Engineering (ICSE)*, 2023.

[39] Vivek Kant. Is software reuse leading to dependency hell?, 2022. https://www.linkedin.com/pulse/software-reuse-leading-dependency-hell-vivek-kant/.

[40] Berend Kloeg, Aaron Yi Ding, Sjoerd Pellegrom, and Yury Zhauniarovich. Charting the Path to SBOM Adoption:A Business Stakeholder-Centric Approach. 2024.

[41] Trishank Karthik Kuppusamy, Santiago Torres-Arias, Vladimir Diaz, and Justin Cappos. Diplomat: Using delegations to protect community repositories. In *USENIX Symposium on Networked Systems Design and Implementation*, 2016.

[42] Piergiorgio Ladisa, Henrik Plate, Matias Martinez, and Olivier Barais. SoK: Taxonomy of Attacks on Open-Source Software Supply Chains. In *2023 IEEE Symposium on Security and Privacy (SP)*, 2023.

[43] Chris Lamb and Stefano Zacchiroli. Reproducible Builds: Increasing the Integrity of Software Supply Chains. *IEEE Software*, 2022.

[44] Benjamin Lamowski, Carsten Weinhold, et al. Sandcrust: Automatic Sandboxing of Unsafe Components in Rust. In *Workshop on Prog. Langs. and Operating Syst. (PLOS)*, 2017.

[45] Tobias Lauinger, Abdelberi Chaabane, et al. Thou Shalt Not Depend on Me: Analysing the Use of Outdated JavaScript Libraries on the Web. In *Network and Distributed System Security Symposium*, 2017.

[46] Chao Li, Huaizhang Li, and Mingshu Li. A software factory model based on ISO9000 and CMM for Chinese small organizations. In *Proceedings Second Asia-Pacific Conference on Quality Software*, 2001.

[47] Kaspar Rosager Ludvigsen, Shishir Nagaraja, and Angela Daly. Preventing or Mitigating Adversarial Supply Chain Attacks: A Legal Analysis. In *ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*, 2022.

[48] William P Maxam III and James C Davis. An interview study on third-party cyber threat hunting processes in the us department of homeland security. *USENIX Security*, 2024.

[49] National Telecommunications and Information Administration. The minimum elements for a software bill of materials (sbom), July 2021. https://www.ntia.gov/report/2021/minimum-elements-software-bill-materials-sbom.

[50] Shradha Neupane, Grant Holmes, Elizabeth Wyss, Drew Davidson, and Lorenzo De Carli. Beyond typosquatting:

an in-depth look at package confusion. In *Proceedings of the 32nd USENIX Conference on Security Symposium*, SEC '23, pages 3439–3456, USA, August 2023. USENIX Association.

[51] Zachary Newman, John Speed Meyers, and Santiago Torres-Arias. Sigstore: Software Signing for Everybody. In *ACM SIGSAC Conference on Computer and Communications Security*, 2022.

[52] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, et al. You are what you include: large-scale evaluation of remote javascript inclusions. In *ACM conference on Computer and communications security*, 2012.

[53] TJ OConnor, Reham Mohamed, Markus Miettinen, William Enck, Bradley Reaves, and Ahmad-Reza Sadeghi. HomeSnitch: behavior transparency and control for smart home IoT devices. In *Proceedings of the 12th Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '19, pages 128–138, New York, NY, USA, 2019. Association for Computing Machinery. event-place: Miami, Florida.

[54] Marc Ohm, Arnold Sykosch, and Michael Meier. Towards detection of software supply chain attacks by forensic artifacts. In *ACM International Conference on Availability, Reliability and Security*, 2020.

[55] Chinenye Okafor, Taylor R. Schorlemmer, Santiago Torres-Arias, and James C. Davis. Sok: Analysis of software supply chain security by establishing secure design properties. In *ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*. Association for Computing Machinery, 2022.

[56] OpenSSF. Introducing artifact attestations: Now in public beta, 2024. https://openssf.org/blog/2024/05/24/introducing-artifact-attestations-now-in-public-beta/.

[57] Cliodhna O'Connor and Helene Joffe. Intercoder Reliability in Qualitative Research: Debates and Practical Guidelines. *International Journal of Qualitative Methods*, 2020.

[58] Ivan Pashchenko, Duc-Ly Vu, and Fabio Massacci. A Qualitative Study of Dependency Management and Its Security Implications. In *ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[59] Daniel Pigeon. How to implement 3 new software supply chain security frameworks, Jun 2022.

[60] Scott Ruoti, Jeff Andersen, Daniel Zappala, and Kent Seamons. Why johnny still, still can't encrypt: Evaluating the usability of a modern pgp client, 2016. https://cups.cs.cmu.edu/soups/2006/posters/sheng-poster_abstract.pdf.

[61] Johnny Saldana. *Fundamentals of qualitative research*. Oxford university press, 2011.

[62] T. R. Schorlemmer, K. G. Kalu, et al. Signing in four public software package registries: Quantity, quality, and influencing factors. In *2024 IEEE Symposium on Security and Privacy (SP)*, 2024.

[63] Taylor R Schorlemmer, Ethan H Burmane, Kelechi G Kalu, Santiago Torres-Arias, and James C Davis. Establishing provenance before coding: Traditional and next-gen signing. *arXiv preprint arXiv:2407.03949*, 2024.

[64] R. Shirey. Internet security glossary, version 2. Technical report, RFC Editor, August 2007. https://doi.org/10.17487/rfc4949.

[65] Sigstore. Sigstore: A new standard for signing, verifying, and protecting software. https://www.sigstore.dev/.

[66] Kapil Singi, Jagadeesh Chandra Bose R P, Sanjay Podder, and Adam P. Burden. Trusted software supply chain. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2019.

[67] Ian Sommerville. Formal Specification. In *Software Engineering*.

[68] Murugiah Souppaya, Karen Scarfone, and Donna Dodson. Secure Software Development Framework (SSDF) version 1.1 : recommendations for mitigating the risk of software vulnerabilities. Technical report, 2022.

[69] Shreyas Srinivasa, Jens Myrup Pedersen, and Emmanouil Vasilomanolakis. Deceptive directories and "vulnerable" logs: a honeypot study of the LDAP and log4j attack landscape. In *IEEE (EuroS&PW)*, 2022.

[70] Aashish Srivastava and S. Bruce Thomson. Framework Analysis: A Qualitative Methodology for Applied Policy Research, January 2009. https://papers.ssrn.com/abstract=2760705.

[71] V Stafford. Zero trust architecture. *NIST Special Publication*, 800:207, 2020. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207.pdf.

[72] Mengtao Sun and Gang Tan. NativeGuard: protecting android applications from third-party native libraries. In *ACM WiSec*, 2014.

[73] Synopsys. 2020 Open Source Security and Risk Analysis (OSSRA) Report, 2020. https://www.synopsys.com/software-integrity/resources/analyst-reports/2020-open-source-security-risk-analysis.html.

[74] Synopsys. 2024 Open Source Security and Risk Analysis (OSSRA) Report, 2024. https://www.synopsys.com/software-integrity/resources/analyst-reports/open-source-security-risk-analysis.html#introMenu.

[75] Gareth Terry, Nikki Hayfield, Victoria Clarke, Virginia Braun, et al. Thematic analysis. *The SAGE handbook of qualitative research in psychology*, 2(17-37):25, 2017.

[76] Lehana Thabane, Jinhui Ma, Rong Chu, Ji Cheng, Afisi Ismaila, Lorena P. Rios, Reid Robson, Marroon Thabane, Lora Giangregorio, and Charles H. Goldsmith. A tutorial on pilot studies: the what, why and how. *BMC Medical Research Methodology*, 10(1):1, January 2010.

[77] The Linux Foundation. Supply-chain levels for software artifacts. https://slsa.dev/.

[78] The White House. Executive order on improving the nation's cybersecurity, May 2021. https://www.whitehouse.gov/briefing-room/statements-releases/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/.

[79] Santiago Torres-Arias, Hammad Afzali, Trishank Karthik Kuppusamy, Reza Curtmola, and Justin Cappos. in-toto: Providing farm-to-table guarantees for bits and bytes. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1393–1410, Santa Clara, CA, August 2019. USENIX Association.

[80] Nikos Vasilakis, Achilles Benetopoulos, Shivam Handa, Alizee Schoen, Jiasi Shen, and Martin C. Rinard. Supply-Chain Vulnerability Elimination via Active Learning and Regeneration. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, pages 1755–1770, New York, NY, USA, November 2021. Association for Computing Machinery.

[81] Nikos Vasilakis, Ben Karel, Nick Roessler, Nathan Dautenhahn, Andre DeHon, and Jonathan M. Smith. BreakApp: Automated, Flexible Application Compartmentalization. In *NDSS*, 2018.

[82] Roberto Verdecchia, Emelie Engström, Patricia Lago, Per Runeson, and Qunying Song. Threats to validity in software engineering research: A critical reflection. *Information and Software Technology*, 164:107329, 2023.

[83] Duc-Ly Vu, Fabio Massacci, Ivan Pashchenko, Henrik Plate, and Antonino Sabetta. LastPyMile: identifying the discrepancy between sources and packages. In *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Athens Greece, 2021. ACM.

[84] Zhiyuan Wan, David Lo, Xin Xia, and Liang Cai. Practical and effective sandboxing for linux containers. *Empirical Software Engineering*, 24:4034–4070, 2019.

[85] Dominik Wermke, Noah Wohler, Jan H. Klemmer, Marcel Fourne, Yasemin Acar, and Sascha Fahl. Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects. In *IEEE Symposium on Security and Privacy*, 2022.

[86] David A. Wheeler, John Speed Meyers, Mikael Barbero, and Rebecca Rumbul. New slsa++ survey reveals real-world developer approaches to software supply chain security, 2023. https://openssf.org/blog/2023/03/15/new-slsa-survey-reveals-real-world-developer-approaches-to-software-supply-chain-security/.

[87] Alma Whitten and J Doug Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *USENIX security symposium*, volume 348, pages 169–184, 1999.

[88] Erik Wittern, Philippe Suter, and Shriram Rajagopalan. A look at the dynamics of the javascript package ecosystem. In *Proceedings of the 13th international conference on mining software repositories*, pages 351–361, 2016.

[89] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, Anders Wesslén, et al. *Experimentation in software engineering*, volume 236. Springer, 2012.

[90] William Woodruff. PGP signatures on PyPI: worse than useless, May 2023. https://blog.yossarian.net/2023/05/21/PGP-signatures-on-PyPI-worse-than-useless.

[91] Boming Xia, Tingting Bi, Zhenchang Xing, Qinghua Lu, and Liming Zhu. An Empirical Study on Software Bill of Materials: Where We Stand and the Road Ahead. In *IEEE/ACM International Conference on Software Engineering*, 2023.

[92] Boming Xia, Dawen Zhang, Yue Liu, Qinghua Lu, Zhenchang Xing, and Liming Zhu. Trust in Software Supply Chains: Blockchain-Enabled SBOM and the AIBOM Future, 2023. arXiv:2307.02088.

[93] Dapeng Yan, Yuqing Niu, Kui Liu, Zhe Liu, Zhiming Liu, and Tegawende F. Bissyande. Estimating the Attack Surface from Residual Vulnerabilities in Open Source Software Supply Chain. In *International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 2021.

[94] Nusrat Zahan, Parth Kanakiya, Brian Hambleton, Shohanuzzaman Shohan, and Laurie Williams. Openssf

scorecard: On the path toward ecosystem-wide automated security metrics. *IEEE Security & Privacy*, 21(6):76–88, 2023.

[95] Nusrat Zahan, Thomas Zimmermann, Patrice Godefroid, Brendan Murphy, Chandra Maddila, and Laurie Williams. What are Weak Links in the npm Supply Chain? In *IEEE/ACM ICSE-SEIP*, 2022.

[96] Markus Zimmermann, Cristian-Alexandru Staicu, Cam Tenny, and Michael Pradel. Small world with high risks: A study of security threats in the npm ecosystem. In *USENIX Security Symposium*. USENIX Association, 2019.

[97] Miranda Zottmann, Carlos Eduardo, Stuckert do Amaral, Thiago Melo, et al. Comparing Software Supply Chain Protection Approaches. In *Workshop on Communication Networks and Power Systems (WCNPS)*, 2023.

Outline of Appendices

The appendix contains the following material:

- Appendix A: Summary of Technical Report.
- Appendix B: Organizational Demographics.
- Appendix C: Code Saturation Curve.

A Summary of the Technical Report

An extended version of this paper is available as a technical report at arXiv:2406.08198. It includes:

- Explanation of workflow for Software Signing.
- The full interview protocol.
- Excerpts of the codebook, with example codes, code definitions, and example interview quotes.
- Codebook Evolution Diagram illustrating how our codebook changed at each stage of analysis.
- Uncommon (less reported) technical challenges, omitted from the main paper due to space constraints and their broader applicability to tools beyond signing.

B Organizational Demographics

We summarize additional organizational demographic information of our subjects in Table 6. This table provides further context for the subject's organizational size, type of software product, and how many of our subjects belonged to each organization.

Table 6: Organizational Demographics. To enhance anonymity, We only highlight the number of subjects in each organizational category. letters A-L is used to depict each organization.

Type	Breakdown (#Organizations|#Subjects)
Organizational Size (Employee Size)	Small (<100) (4/6), Medium (<1500) (3/4), Large (>1500) (6/8)
Product Area	Digital technology (1/3), SSC Security (2/4), Social technology (1/1), Dev tools (1/1), Telecommunications (1/1), Cloud security (2/3), Aerospace Security (1/1), Internet services (2/2), Cloud + OSS Security (1/1), Cloud + Dev toools (1/1)
Subject Distribution	A (3), B (3), C (1), D (1), E (2), F (1), G (1), H (1), I (1), J (1), J (1), K (1), L (1)

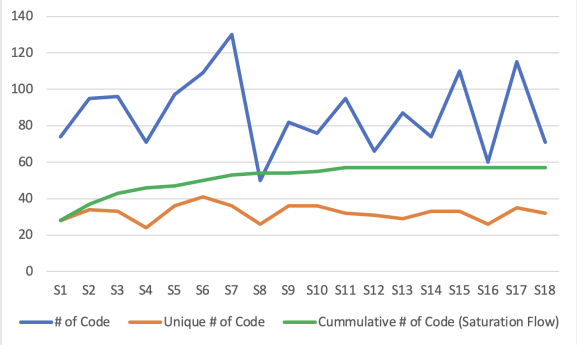

Figure 4: Saturation curve. Interviews are plotted in the order in which they were conducted. Each interview covered many topics in detail (orange line, blue line). However, as the green line shows, our results saturated (*i.e.,* stopped observing new perspectives) around interview 11.

C Code Saturation Curve

We present the trends from the data analysis of our interview subjects in Figure 4. The total number of codes per interview is shown by the blue trend line, with a median of 84.5 codes per interview. Participant S8 had the fewest codes (50), while S7 had the most (130).

The orange trend line represents the unique number of codes in each interview, with a median of 33 unique codes. Subject S4 had the fewest unique codes (24). The green trend line illustrates the saturation curve for the study, indicating that no new unique codes were observed beyond interview 11 (S11). As noted earlier (§4.3), the pilot interviews (S1 and S2) were included due to minimal changes in the interview protocol. These interviews also contained more unique and total codes than the study's median values, further justifying their inclusion.