Analyzing Cryptography in Context: A Cryptography-Native Approach to Threat Modeling

Ran Canetti*[1], Julie Ha*[1], and Gabriel Kaptchuk*[2]

[1]Boston University, {canetti,hajulie}@bu.edu
[2]University of Maryland, kaptchuk@umd.edu

Abstract

We observe that the existing norms within cryptography do not expect protocol analysts to document the sociotechnical properties that a deployed system should have. To help close this potential gap, we develop a framework that allows bringing sociotechnical dimensions into analyses of cryptographic systems, and in particular facilitates "in-context" analysis of proposed cryptographic deployments on top of widely-accepted cryptographic modeling techniques. To explore the utility of our framework, we use Apple's 2021 CSAM scanning proposal as a case study. We show how our framework naturally surfaces many of the criticisms of Apple's proposal and helps us identify a previously undocumented property of the proposal.

*Authors listed in alphabetical order following the norm in theoretical computer science.

Contents

1 Introduction **3**
- 1.1 Our Contributions . . . 4
- 1.2 Related and Concurrent Work . . . 5

2 Background: Threat Modeling Techniques **5**

3 A Framework for "In-Context" Analysis **8**
- 3.1 Design Goals . . . 8
- 3.2 The need for a cryptography-native approach . . . 9
- 3.3 Our Framework . . . 10
- 3.4 Benefits and Limitations of our Approach . . . 13

4 Case Study: The Apple CSAM Scanning Proposal **14**
- 4.1 Background: Criticism of Apple's Proposal . . . 14
- 4.2 Modeling Apple's Initial CSAM Scanning Proposal . . . 15
- 4.3 Examining Sociotechnical Properties . . . 18
- 4.4 Takeaways from analyzing Apple's proposal . . . 22

5 Updating Apple's Protocol **23**
- 5.1 Step 2: Desirable Sociotechnical Properties . . . 23
- 5.2 Step 0: Reworking the Ideal Functionality . . . 25
- 5.3 Step 1: Extending the Modeling . . . 28
- 5.4 Step 3: Examining Sociotechnical Properties . . . 30

6 Discussion **31**

7 Conclusion and Future Work **33**

8 Acknowledgments. **34**

A Updating Apple's Initial Proposal **44**
- A.1 Apple's Initial Ideal Functionality . . . 44
- A.2 Apple's Initial Protocol . . . 46
- A.3 Updating $\Pi_{\text{Apple-CSAM-Scan}}$ for New Modeling . . . 46

B Protocol Implementing $\mathcal{F}_{\text{CSAM-Scan}}$ **48**
- B.1 Protocol Overview . . . 48
- B.2 Preliminaries . . . 49
- B.3 Protocol Description . . . 53

C Proof that $\Pi_{\text{CSAM-Scan}}$ realizes $\mathcal{F}_{\text{CSAM-Scan}}$ **57**
- C.1 Simulating the Server . . . 57
- C.2 Simulating A Client . . . 61
- C.3 Simulating the Judge . . . 62
- C.4 Simulating NCMEC . . . 63
- C.5 Simulating Public Interfaces . . . 64

1 Introduction

All cryptographic protocols are expected to have an accompanying security proof in a compelling and believable security model. But, designers have a heightened responsibility to document their system's properties when they will be deployed, as people's privacy will be at risk. Responsible deployment, therefore, demands carefully considering the sociotechnical properties of the system, including both cryptographic and non-cryptographic components, and studying the ethical ramifications of deployment.

While security proofs are a convincing way to demonstrate that a protocol realizes a set of idealized properties, we often lack the tools to systematically evaluate sociotechnical properties and ethics. For example, proving that a protocol meets the formal definition of secure multiparty computation (MPC) [Yao86, GMW87, CCD88, BGW88] does not ensure that the MPC's inputs match the context's goals, nor does it prevent computation of an inappropriately disclosive circuit. This work develops a framework aimed at incorporating sociotechnical dimensions within the core analysis of cryptographic systems, rather than leaving such properties out of scope.

A real-world example. The gaps in our existing analysis tools are obvious when we look at recent real-world proposed deployments. Take, for example, Apple's 2021 announcement [App21c] that it would scan end-to-end encrypted photographs stored in iCloud for known instances of child sexual abuse material (CSAM) using a threshold private set intersection protocol. While the soundness of the security proofs accompanying the announcement was largely unquestioned, critics noted that the proposed system put too much trust in Apple's operators and gave clients no ability to detect abuse.

The vast majority of the analysis of this incident has focused on *ethics* (i.e., *should* such a system be deployed?) [AAB+21, Kob21b, FN21, CK21, Gil21, Ops21, GTS+, GS21, KKL+21, San21, MP21, MSST21, PPK21, Res21, Mur21, Esp21]. We, instead, focus on the *process* issue, which has been largely overlooked: Apple's team produced a report [App21a] that met community expectations but failed to highlight many of the aspects of their proposal that were found to be objectionable. That is, because the analysis they conducted was tightly scoped, the research community had to independently extract the sociotechnical properties associated with deployment—without the benefit of being Apple insiders—before even beginning to debate *ethics*. While Apple's analysis followed cryptographic best practice, we conjecture that had Apple's analysis incorporated some of the sociotechnical aspects of the system that were uncovered after release, both the system itself and the reaction to it might have been different. For this to happen, though, the community needs to have a methodology for incorporating societal aspects within the core security analysis.

Cryptography-native threat modeling. We argue that there are three, conceptually distinct steps when it comes to responsibly preparing a cryptosystem for deployment:

(1) proving that it has a set of properties using standard, cryptographic proof techniques,

(2) establishing the sociotechnical properties the cryptographic deployment will have, i.e., the way in which it interacts with its deployment context, and

(3) arguing that deployment is "good," "contextually appropriate," or "ethical."

Each of these steps is a *precondition* that informs the following step, as idealized properties dictate a cryptosystem's interaction with its context and failing to carefully account for the system's sociotechnical properties can radically change the ethical calculus. Moreover, different groups hold responsibility for the steps. Responsibility for (1) and (2) lies solely with cryptographers, who have the technical knowledge to understand the intricacies of the proposed cryptosystems. Effectively doing (3), on the other hand, requires an interdisciplinary team capable of weighing the benefits and risks of deployment from many perspectives.

In this work, we initiate the study of step (2), hopefully paving the way for future work on step (3). We approach step (2) by asking the following question:

What analytical tool should we use to establish the in-context, sociotechnical properties of a cryptographic deployment?

We recognize that building a consensus answer to this question requires full community engagement—and there may be no perfect tool. Our goal, therefore, is not to provide the final answer to this question, but rather to offer a candidate tool that can start this important community-wide conversation.

We observe that *threat modeling* appears to be a promising starting point for developing such a tool, as threat modeling asks analysts to identify the ways in which different system components (both technical and social) interact and the implicit trust assumptions the system makes. Moreover, there is a robust and well-developed threat modeling literature that is often used in practice. Unfortunately, directly importing existing tools poses difficulties, as they are fundamentally non-interoperable with the modeling techniques already commonly used in the cryptographic community and operate at the wrong layer of abstraction (e.g., surfacing the software that implements cryptography, which all cryptographers assume should be correct). Thus, we must develop cryptographic-native analogs of these techniques that will feel natural for cryptographers to use.

1.1 Our Contributions

Our approach demonstrates how existing cryptographic modeling tools can be adapted in a natural way to support the type of sociotechnical analytical work in which we are interested. Specifically, we make the following technical contributions:

– **A Cryptography-Native Threat Modeling Technique.** We develop a threat modeling technique within the Universal Composability (UC) framework [Can01]. Indeed, the UC framework's ability to represent a system at multiple levels of abstraction and detail without losing rigor or precision makes it a convenient platform for capturing sociotechnical properties of security systems. In particular, our framework allows capturing such properties at one level of abstraction and then arguing about how such properties are impacted by implementation details which appear only at significantly lower levels.[1] Our approach asks protocol designers to situate ideal functionalities that will be realized cryptographically within a larger set of functionalities that capture the capabilities of the *full system*, connecting these functionalities with the following, concentrically composed protocols:

  (1) $\Pi_{\text{Intended-use}}$: a protocol describing the envisioned functioning of non-cryptographic software components of the system. Importantly, specifying behavior within $\Pi_{\text{intended-use}}$ (as opposed to an ideal functionality) is an indication that no formal (i.e., cryptographically verifiable) guarantees are going to be made that this software will follow the prescribed steps.

  (2) Π_{Social} : a protocol describing elements of the system that will not (or cannot) be rendered into software. Generally, these steps are ones for which it is difficult to determine if behavior is malicious.

  When seen through the lens of the threat modeling literature, our framework is best understood as a cryptography-native data-flow diagram [GS77] which can be used to examine claimed sociotechnical system properties. These claimed properties can inform a separate, possibly interdisciplinary, discussion determining if the system *should* be deployed.

– **Case Study: Apple's CSAM Scanning Proposal.** To study our threat modeling approach, we apply our framework to a real-world, proposed cryptographic system: Apple's CSAM scanning proposal. In Section 4, we use the framework to retrospectively analyze Apple's initial proposal. We find that this process (a) captures many of the research community's existing critiques, (b) helps identify weaknesses in Apple's initial cryptographic analysis and (c) pinpoints at least one meaningful risk associated with deployment that, to our knowledge, has not been publicly discussed despite multiple years of heightened scrutiny. Specifically, we observe that storing the images scanned by Apple's threshold PSI scheme using convergent encryption can leak the identities of users storing a sub-threshold number of CSAM images.

[1] Still, the conceptual insights motivating our work are not specific to the UC framework. We anticipate that the community will likely need an array of threat modeling frameworks, including some that interoperate with other proof approaches (e.g., game-based security definitions).

We observe, however, that our framework is limited in the ways that it supports reasoning about heuristic properties of systems, like the semantic hash function used in Apple's proposal.

In Section 5, we explore using our framework proactively within the cryptographic design process by first specifying sociotechnical properties and then designing to meet them.

Why choose Apple's CSAM scanning proposal? When selecting a case study for this work, we required a proposal that realistically would be subjected to this type of analysis in the real world, both because it offers sufficient technical complexity and promises to have a rich social context. This ruled out "toy" examples (e.g., the X.509 ecosystem), as they would not help us learn how our framework handled complexity. Moreover, we needed a case study that had been subject to at least some ad-hoc sociotechnical analysis, so we could check if our framework systematically replicated these findings. Without this baseline against which we could compare, it would be difficult to have confidence that our framework performed at some minimum acceptable level. Apple's CSAM scanning proposal satisfied all these criteria.

1.2 Related and Concurrent Work

Our work is inspired by Rogaway's foundational work [Rog15], in addition to recent similar work by Kamara [iac20, use22], Borradaile [Bor21], and Rosenbloom [Ros26], that highlights the need to think about cryptographic systems within the social context in which they will be deployed. Our work differs from theirs by proposing a concrete change to the ways in which cryptographic protocols are documented in service of this shared goal. We are similarly inspired by analyses of real-world deployed cryptosystems that demonstrate gaps between claimed properties and actual guarantees, e.g., [HDWH12, ABD+15, GGK+16, AEP22, AMPS22, BHP23, AHMP23, AP24, TTP26]. These works typically are post-deployment analyses, focusing on cryptography "in the wild," whereas the analyses we consider are intended to be pre-deployment. While these works typically focus on a single deployed system, some of these works, e.g., [TTP26], demonstrate the broader risk of analyzing only sub-components of a larger cryptographic system.

Concurrently with our work, Albrecht et al. [ACDJ26] put forward a compelling alternative vision for understanding the sociotechnical properties of cryptographic systems. Specifically, they outline a process by which researchers deeply embed themselves in a particular deployment context, leveraging observation and grounded theory. This approach can identify security properties that a cryptosystem should provide in order to be effective. We see our works as complementary, solving distinct but related problems. A key difference between our works is that we design a process that every real-world proposal can follow, whereas the type of ethnographic work they consider is not cryptography-native, making it a challenging candidate for broad adoption. Moreover, their process does not directly lend itself to providing documentation. On the other hand, their approach is significantly more thorough and can document the ways in which users interact with cryptographic systems in practice.

2 Background: Threat Modeling Techniques

Before describing the techniques we develop in this work, we summarize the existing threat modeling literature and techniques. The Threat Modeling Manifesto [BSM+] describes the process of threat modeling, at the highest level, with four questions "*(1) What are we working on?, (2) What can go wrong? (3) What are we going to do about it? (4) Did we do a good enough job?.*" While seemingly simplistic, this type of clarity is valuable for bringing a wide variety of stakeholders into the threat modeling process—a value that the authors of the threat modeling manifesto emphasize is important. Shi et al. [SGSM22] give a more detailed set of steps that constitute a "typical" threat modeling process: *"(1) define security requirements; (2) model the system; (3) identify potential threats based on the system model; (4) evaluate the identified threats based on multiple aspects, such as accessibility to attackers, attack complexity, privileges required, and so on; (5) mitigate potential threats according to the security requirements; and (6) validate that the threats are mitigated after applying mitigation techniques."*

This multi-stakeholder (and often interdisciplinary) approach to threat modeling practice (e.g., security professionals, system architects, intelligence experts, etc...) can be used to identify both technical and human vulnerabilities and help identify vulnerabilities that not only threaten digital infrastructure, but also key business interests. Threat modeling most often takes the form of a social process, possibly supported by software packages [OWAa, Mic, Iri, Tar]. A comprehensive summary of threat modeling techniques is beyond the scope of this work (for enumerative lists of existing tools and resources, readers should consult [Inc, SCO+18, SFW18, Sel19, XL19]). Our goal is to familiarize readers—and cryptographers, in particular—with the ways in which threat modeling is practiced.

STRIDE. The most notable threat modeling tool is STRIDE [HL06] (spoofing, tampering, repudiation, information disclosure, denial of service, elevation of privilege), a family of tools initially developed by Microsoft to help identify threats to software systems. The two main variations of STRIDE are STRIDE-per-Element and STRIDE-per-Interaction. Both STRIDE variants comprise a series of steps performed by system architects. First, analysts render the system into a data-flow diagram [GS77], a visualization that maps the way that information passes between external actors (e.g., end users), processing nodes (e.g., web servers), and data stores (e.g., database), each of which is contained within a *trust boundary*. A trust boundary demarcates the scope of control of a single actor or organization and could be imagined as the set of processing nodes that a single corrupted adversary might reasonably control (e.g., client-side code operates in a different trust boundary than the server, even if it is written by the same organization). In STRIDE-per-Element, analysts then iterate through the elements of the data-flow diagram and identify the categories of threats (i.e., spoofing, tampering, etc...) to which each element is vulnerable. In STRIDE-per-Interaction, the same process is applied to data flows that cross trust boundaries instead. The STRIDE toolkit comes with several checklists that then help the analyst elicit concrete threats associated with each element, at which point these threats are documented and can be used to inform required protections.

Alternatives similar to STRIDE. There are numerous alternative threat modeling techniques that share a basic structure with STRIDE: (1) derive a model of the system using data-flow diagrams; (2) consider different ways in which each element or interaction could be manipulated; (3) elicit concrete threats that could result in this manipulation; and (4) document these threats. For example, LINDDUN (linkability, identifiability, non-repudiation, detectability, disclosure of information, unawareness, non-compliance) [WJ15, WVLHJ18] applies a set of threat categories that may be more linguistically familiar to cryptographers to data-flow diagrams before eliciting threats. DREAD (damage potential, reproducibility, exploitability, affected users, discoverability) was an alternative mnemonic used within Microsoft in place of STRIDE, but has since been abandoned. PASTA (process for attack simulation and threat analysis) [UM15] provides a wrapper around STRIDE in the form of an institutional playbook with an emphasis on the organizational dynamics and the leadership buy-in necessary for success. OCTAVE (operationally critical threat, asset, and vulnerability evaluation) [ADSW03, CSYW07] is another organization-focused approach that emphasizes participation by both members of business units and technology units and focuses analysis on key "assets" held by an organization. NIST Special Publication 800-154 outlines a very similar, if less formulaic, version of this template, in which analysts enumerate the data in the system, specify a set of security properties, compile a set of concrete threat vectors, and then produce a subset of these threats against which protections must be installed based on an estimation of the severity of the threats (e.g., cost it would take to execute the attacks).

Attack Trees. Initially described by Bruce Schneier [Sch99, Sch15], attack trees are a method for mapping out routes that an attacker could take to accomplish a particular goal. In particular, the root of the tree represents the goal. Then, each successive layer of children nodes captures options for accomplishing the tasks in parent nodes. For example, Schneier sketches an example in which the root node is "Open Safe," one child of which is labeled "Learn Combo," which in turn has a child with the label "Get Combo from Target." Nodes can be annotated with financial costs, annotated with the possibility/impossibility of the task, and nodes on the same layer can be connected using either OR or AND logic. This cascading structure allows for the specification of attack strategies at many layers of abstraction and eventually computing the total cost of accomplishing the root goal. Attack trees are a common tool used within other threat modeling tools in order to identify and assess the viability of an attacker exploiting a potential threat vector and

identifying security remediation.

Attack and Vulnerability Databases. To support the identification of threats, some threat modeling processes leverage pre-compiled lists of known attacks and vulnerabilities that can be searched systematically or used to support brainstorming. These include CAPEC (common attack pattern enumeration and classification) [MITb], ATT&CK (Adversarial Tactics, Techniques, and Common Knowledge) [MITa], OWASP Top Ten [OWAb], the Threat Trees that are supposed to be used within a full LINDDUN analysis, and many others. There also exist some frameworks that support threat elicitation by pairing known attack vectors with known motivations for attack, like TARA (threat assessment & remediation analysis) [WWU+11]. Alternatively, the Personae non Gratae [CH14] approach provides workups of potential attacker identities (equipped with fictional backstories, motivations, and expertise) that helps analysts embody the adversarial mindset while trying to uncover flaws.

Threat modeling cryptocurrency systems. While it is rare to see a formal threat modeling framework applied to a cryptographic system, the notable exception is cryptocurrency systems. For example, Vandervort et al. [VGJ15] applied STRIDE to a Bitcoin-like cryptocurrency. Almashaqbeh et al. [ABC19] and Boughdiri et al. [BHA24] each develop a custom threat modeling framework that is tailored for cryptocurrencies. Given that their intended target is cryptocurrencies, they do not have the right approach for general cryptographic systems. For example, Almashaqbeh et al. [ABC19] introduce a novel technique, called a collusion matrix which is best suited for the threat domain of cryptocurrencies, as its model helps rule out scenarios that are irrelevant to cryptocurrencies and make sure that it captures specific circumstances (e.g., financially motivated attackers) that are specific to cryptocurrencies.

The other two noteworthy works in this space attempt to systematize the known threats to cryptocurrency systems. Froehlich et al. [FHA22] address the gap of threats that end-users of cryptocurrencies may face. They conduct a study of experts in industry and academia to build a model that will consider six categories of threats that end-users may experience. [FHA22] specify that their model identifies the threats that end-users will experience. Badawi et al. [BJ20] do a systematic literature review on existing threats that target cryptocurrencies. They identify attacks by cyber-criminals on cryptocurrencies, and analyze solutions to these crimes proposed in the literature.

Studies of Threat Modeling Techniques. Although largely beyond the scope of this work, we note that there is a moderate amount of academic work studying the practice of threat modeling itself. These include case studies of performing threat modeling, e.g., [CCK+18, SVR+18, CR21] and descriptive studies of the ways in which threat modeling tools are used by cybersecurity practitioners [TMPV24, VYSJ24, GBBA24]. These later works also surface limitations of existing threat modeling schemes. For example, Thompson et al. [TMPV24] found that threat modeling, as practiced by medical device manufacturers, can be quite ad-hoc and relies heavily on the prior experience of the analyst. Verreydt et al. [VYSJ24] studied the existing threat modeling practices of Dutch organizations, and found that there were significant informal benefits to performing threat modeling (e.g., security awareness), even if the practice seldom identified new threats. Grosse et al. [GBBA24] studied threat modeling as applied to artificial intelligence applications, finding that there is a gap between academic threat models and real-world threat models. Other works have tried to interrogate the value of threat modeling processes in more controlled environments. For example, Fulton et al. [FVA+22] studied students as they attempted to develop secure programming skills within the context of a course that taught threat modeling. Scandariato et al. [SWJ15] also study students, asking them to perform threat modeling using STRIDE. They find that the STRIDE methodology produces threats with a low false positive rate and a moderately high false negative rate, but the process itself is very time consuming. Shull [Shu16] attempted an even more controlled experiment, in which study participants generated a threat model using a randomly assigned threat modeling tool, and found that the outcomes of the threat model process were often inconsistent between analysts.

3 A Framework for "In-Context" Analysis

In this work, we imagine a future in which the *expectation* from the cryptographic community is that real-world proposals for cryptographic systems not only include a rigorous proof of security, but also an analysis that helps illustrate the way in which the cryptographic components will fit into the broader system. This analysis would provide an opportunity to articulate the sociotechnical goals that the composed system is intended to accomplish and analyze the extent to which those goals are met. Unlike standard proofs, this section may not provide iron-clad evidence that the goals have been met, as providing formal proofs of social properties seems prohibitively difficult. Instead, this analysis significantly lowers the barrier for external validators interested in evaluating the merits of the proposed system. Moreover, a clear articulation of the system's sociotechnical goals invites readers to decide if they agree that these goals are appropriate.

3.1 Design Goals

Before considering a viable way to structure this in-context analysis of cryptographic systems, we pause to consider our design goals. At the highest level, we require a framework that is both *convincingly thorough*, in that a reader should be confident that the designers have thought critically both about the properties of their system and the impact of deploying a system with those properties into the world, and *surfaces questions of social importance*, in that the framework should facilitate productive conversation about the ramifications of the system, as deploying cryptography is rarely without trade-offs. We break these high-level properties into concrete requirements below:

(1) **Broaden the analytical frame.** Existing tools for analyzing cryptographic systems focus on modularity and generalizability, in that the properties of the cryptography are kept sufficiently generic such that they can be plugged into future applications. A side effect of this approach is that it naturally suppresses the details of how the system, as deployed, will fit into a larger digital ecosystem. Analysis of these parts of the deployment is—at best—reserved for another document and is often performed without any supporting framework or critical review by security experts, despite their critical importance for analyzing the privacy implications of deployment. We require that our framework provides the affordances necessary to broaden the scope of the analysis performed by system designers.

(2) **Conceptually lightweight, or "cryptography-native."** We require that a new tool does not introduce significant conceptual complexity (beyond the non-trivial complexity associated with standard cryptographic analysis). Requiring too much learning or labor in order to leverage an analytical framework will limit uptake. If we believe that doing this kind of thorough analysis is the "right thing to do," we should make efforts to encourage broad usage. Looking ahead, we do this by piggybacking onto the language of ideal functionalities which are familiar to much of the cryptographic community.

(3) **Systematic and enumerative.** We require that the framework to conduct this analysis provides a clear script for system designers to follow. If a framework is systematic—like existing formal proof techniques—then it acts as compelling evidence, both to the author and the reader, that the system in question has been thoroughly considered. Any new framework that we propose should similarly take a systematic approach to analysis and leave obvious evidence when elements of the analysis have been skipped. We note, however, we stop short of trying to *prove* sociotechnical properties of systems (in a formal sense), but rather make analyzing them more systematic. In this sense, the approach must force system designers to enumerate the relevant aspects of the system completely.

(4) **Highlights implications of trust choices.** Because cryptography enables system designers to re-imagine the trust relationships required to carry out computational tasks, we need our framework to facilitate a clear understanding of the ways in which trust is allocated throughout the entire system. By failing to contextualize protocols, existing norms make it difficult to see these trust relationships—and the implications of discovering that trust was improperly allocated. This includes highlighting the ways in which cryptography can minimize the trust required for the system to operate and aspects of the trust allocation that are beyond the cryptographic component's reach.

3.2 The need for a cryptography-native approach

Threat modeling techniques appear to be a natural starting place for conducting "in-context" analyses. For example, the community could simply require that authors of real-world proposals apply STRIDE/LINDDUN to the proposed system and publish the results of that analysis. Such an approach would likely meet many of the design goals outlined above. STRIDE/LINDDUN would rapidly broaden the analytical frame to new elements of the system, including the software stack, back-end data storage, and human actors. While STRIDE/LINDDUN are not perfect systems, they are widely considered systematic and enumerative enough to become a go-to tool for practitioners protecting digital treasure-troves from motivated attackers. Most importantly, STRIDE/LINDDUN is a mature technique with significant amounts of documentation and educational tools.

There are, of course, significant challenges to directly applying threat modeling tools to cryptographic proposals. First and foremost, there is no interoperability with the concepts cryptographers regularly use to model the properties of cryptographic protocols and existing threat modeling techniques; there is no place to drop an ideal functionality or tuple of algorithms into existing tools. As a result, using existing tools would require re-modeling the cryptographic system from scratch in a compatible form. Not only would this duplicate work that the cryptographers have already done, but it provides an opportunity for details to be lost in translation. That is, even if the standard cryptographic proof holds and the threat modeling analysis is compelling, there is now a *gap* between the abstractions used to conduct each piece. This would, no doubt, require yet another systematic analysis to show that the two system representations are consistent.

As a concrete example, consider trying to do a STRIDE/LINDDUN analysis for a complex cryptographic protocol. Recall that the first step of STRIDE/LINDDUN is to produce a data-flow diagram that shows messages between various computational agents labeled with their content. Cryptographic protocols include many messages that do not directly reveal information (or reveal only partial information about data held by protocol participants). Should these messages be captured in the data-flow diagram, and if so, how should they be labeled? Moreover, cryptographic protocols could be run over time, with non-cryptographic interactions interleaved between cryptographic messages, meaning it is not always possible to just drop multiple rounds of cryptographic interaction into the data-flow diagram, annotated with the eventual outputs to the participating parties. More generally, how should the wide variety of possible malicious behaviors (which could *change* the information logically communicated by a cryptographic protocol) be captured within the diagram? How about the proven properties of these messages? These questions are only scratching the surface. Simply put, the ways in which we formalize cryptographic systems just don't naturally fit in with STRIDE/LINDDUN's data-flow diagrams.

Another notable challenge is that threat modeling tools are far from conceptually lightweight, simply because the language employed by these tools is totally foreign to cryptographers. Moreover, most threat modeling tools—including STRIDE/LINDDUN—are designed with software architects and software engineers in mind. As a result, they operate on the wrong level of abstraction for cryptographers. By their very nature, existing software tools would need to be applied to the software systems *implementing* cryptographic operations, rather than the properties provided by the cryptography directly. For example, STRIDE/LINDDUN's data-flow diagram should capture the various software components in the system (app, web browser, device resources, load balancers, databases, etc...), none of which may independently hold the cryptographic properties of the system. This has two negative effects: (1) cryptographers, as a community, are notoriously bad at software and generally assume that the software implementing their designs is correct as a prerequisite; and (2) treating the system as software, rather than a set of properties, is likely to *distract* from the core, cryptographic parts of the system. This is not to say that a software-first perspective on cryptographic systems would not be valuable—such an analysis should certainly take place within the implementing organization. Rather, it's not clear that this is the domain of cryptographic protocol designers.[2]

If we cannot productively leverage threat modeling tools directly, we should instead attempt to create cryptographic-native analogs of the best parts of threat modeling. For example, all threat modeling tools

[2]As we discuss in the limitations below, this is a subtle argument. For example, there have been numerous arguments that raising complexity of cryptographic protocols will inevitably lead to more complex software with more bugs, thereby reducing the security of the resulting system [AAB+15, AAB+21].

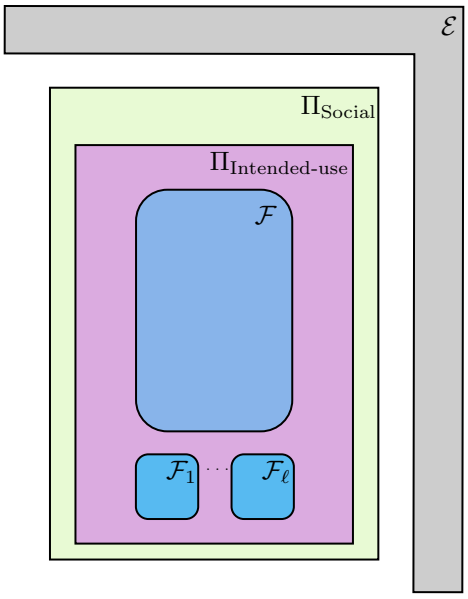

Figure 1: A visualization of our approach. In this rendering, our ideal functionality of interest, $\mathcal{F}$, is composed with several other ideal functionalities, $\mathcal{F}_1, \ldots, \mathcal{F}_\ell$, by $\Pi_{\text{Intended-use}}$, which specifies the expected functioning of the software components of the system not captured by $\mathcal{F}$. The protocol $\Pi_{\text{Intended-use}}$ is further wrapped by Π_{Social}, which describes the aspects of the system that are driven by human decisions or system components for which there is no "correct" behavior.

advocate for a clear and formulaic social process with clear documentation. Additionally, the ubiquity of data-flow diagrams among many threat modeling tools (and the challenges with data-flow diagrams we discussed above) is a clear indication that building the appropriate, cryptographic-native version of a data-flow diagram will be an important step. Notably, these data-flow diagrams trace the *origin* of data back to the human actors, and are not limited to the flow of data between processing nodes. Looking ahead to our framework, **we build a cryptographic-native version of the data-flow diagram** that similarly mandates that the human drivers of the system be in scope. Another key aspect of threat modeling is setting explicit goals (*"what are we working on?"* in the Threat Modeling Manifesto's language [BSM+], the root node label in attack trees [Sch99, Sch15], or the inventory of system data conducted as part of NIST Special Publication 800-154 [SS16]). Looking ahead, we will try to accomplish the same things by asking protocol designers to enumerate sociotechnical goals for their system.

3.3 Our Framework

As our starting point, we take the simulation-based security paradigm and Canetti's UC framework [Can01] in particular. The main motivation behind this choice is that simulation-based security allows cryptographers to define the properties of a cryptosystem using an ideal functionality. A well-written ideal functionality provides a clear rendering of both the security properties of the system and the way in which users can interact with the system. This level of abstraction cleanly matches the data-flow diagrams leveraged in established threat modeling tools, in which analysts are asked to illustrate the way data moves between system components. As our framework is intended to situate a cryptographic protocol within its deployment context, treating the protocol as a black box with some set of properties—an ideal functionality, in other words—is the right fit for the task at hand. While UC makes it possible to formally argue about the security implications of composition, UC does nothing to "force the hand" of the designers to broaden their analytical frame. Indeed, it is possible to specify ideal functionalities that would have terrible ramifications

on society within UC; as we discuss in our case study, Apple's initial proposal was done within the (simple) UC framework and was largely found objectionable. Moreover, UC is generally used as a *proof technique*, and it is not our intention to prove anything about the real world impact of a protocol—rather, we just want to make it clear that designers have considered these implications and support designers as they communicate about them.

Step 0: Defining the ideal functionality $\mathcal{F}$. We assume that the protocol designers have some protocol Π and an ideal functionality $\mathcal{F}$ that they believe Π should realize. Before interacting with our framework, the protocol designers should separately provide a proof that Π realizes $\mathcal{F}$, allowing them to work with $\mathcal{F}$ alone.

Step 1: Extending the modeling. Next, the protocol designer puts $\mathcal{F}$ into context by extending the modeling they have done to cover the full system. Importantly, this modeling is going to use the same paradigms (i.e., language, level of abstraction, etc.) the cryptographer has already used to define $\mathcal{F}$ and prove that Π realizes it. Namely, the protocol designer specifies the following two *protocols*, as visualized in Figure 1:

- $\Pi_{\textbf{Intended-use}}$: This protocol describes the ways that honestly written *software* is expected to interact with $\mathcal{F}$. Most notably, this includes specifying the inputs that are expected to be sent to $\mathcal{F}$ and explicitly describing the ways in which those inputs should be interleaved with other cryptographic components $\mathcal{F}_1, \ldots, \mathcal{F}_\ell$.[3] For example, $\Pi_{\text{Intended-use}}$ might specify that an output of $\mathcal{F}_1$ should be fed into a specific interface of $\mathcal{F}$. Alternatively, this might include the order and timing with which a protocol participant calls the multiple interfaces of $\mathcal{F}$.

  By putting elements of the proposed system in $\Pi_{\text{Intended-use}}$ rather than $\mathcal{F}$, the designer is explicitly stating that cryptographic techniques will not be used to verify their correct operation. Indeed, because $\Pi_{\text{Intended-use}}$ is a *protocol*, rather than an ideal functionality, there is no proving required—and it is possible for software implementing these parts of the protocol to deviate maliciously without detection. Just as attempting (and failing) to formally reduce the security of a cryptographic protocol helps designers identify protocol weaknesses or required properties of cryptographic building blocks, specifying $\Pi_{\text{Intended-use}}$ highlights risks from malicious software.

- $\Pi_{\textbf{Social}}$: Some system components will never be—or can never be—written in software because there is no clear expected behavior. Nonetheless, understanding the implications of these parts of the system is crucial. Π_{Social} provides an opportunity for protocol designers to specify these components. We call this part of the system "social" because we anticipate that much of this protocol will be mediated by human decision-making. These decisions—in which there is no clear *correct* decision—are inevitable in instances where digital systems meet the physical world. For example, Π_{Social} will often be responsible for capturing data and importing it into the system (e.g., manual data entry, recording audio data, etc...).

  Formally specifying Π_{Social} poses a modeling challenge: *if we do not know of a PPT algorithm that captures what* should *happen in Π_{Social}, how can we describe it?* Conceptually, we can solve this by thinking of Π_{Social} as describing the *social meaning* of an arbitrary PPT algorithm that sends messages of the right form. Looking ahead, we put the task of identifying CSAM into Π_{Social}, which captures the *meaning* of an arbitrary algorithm that selects a subset of images. Alternatively, one can see Π_{Social} as describing part of the environment's description and giving social weight to the environment's choices.

These two protocols are composed concentrically (see Figure 1); Π_{Social} makes subroutine calls to $\Pi_{\text{Intended-use}}$, and $\Pi_{\text{Intended-use}}$ makes calls to $\mathcal{F}$ (along with, optionally, $\mathcal{F}_1, \ldots, \mathcal{F}_\ell$). We emphasize that this approach does not change the proofs that designers are required to write; once a protocol Π UC realizes $\mathcal{F}$, it is "safe" (from a provable security perspective) to arbitrarily compose it with other functionalities. Our approach does not require proving anything about $\Pi_{\text{Intended-use}}$ and Π_{Social}. Instead, the process of their specification

[3] $\mathcal{F}_1, \ldots, \mathcal{F}_\ell$ are not ideal functionalities used to *realize* $\mathcal{F}$, but parallel system components. If $\mathcal{F}$ is realized in a hybrid model, those ideal functionalities would be suppressed at this level of abstraction.

helps designers see elements of their system that may require improvement or should be the site for in-depth policy discussions. Moreover, giving external stakeholders access to these protocols helps them understand system limitations and see the values that designers are imbuing into the system.

Guiding Principle for Step 1: all data comes from somewhere. It is clear that the extent to which our framework broadens the analytical frame depends on how far the modeling is extended in Step 1. If the increase in scope is minimal, then threats that are more removed from the system itself are unlikely to be captured. On the other hand, modeling the whole world is clearly infeasible. Thus, a balance must be struck. For the purposes of our framework, we believe a nice balance is achieved if the modeling must capture the *source* of all data in the system. Namely, any data that is relevant to $\mathcal{F}$ must either be imported into the system by an actor in Π_{Social} or be sampled/created within $\Pi_{\text{Intended-use}}$.

Step 2: Articulating high-level, sociotechnical properties. With $\Pi_{\text{Intended-use}}$ and Π_{Social} specified, the framework affords the ability to discuss desirable sociotechnical properties of the system. That is, one can make statements of the form "when [human operator] takes [action], then [result]" or "if all other participants' software components are correctly implemented, then [human operator] cannot [action]." Moreover, the composition of $\mathcal{F}$, $\Pi_{\text{Intended-use}}$ and Π_{Social} allows us to interrogate the plausibility of these statements. For example, we might be able to show how one of these sociotechnical properties can be violated by a deviation of the software captured by $\Pi_{\text{Intended-use}}$ or human decisions within Π_{Social}. This can be done in two ways:

- *Top-down specification of properties.* The first approach to specifying sociotechnical properties is "top-down." This approach requires articulating the properties that the designers *hope* the composed system will achieve from first principles. These goals should be independent of the modeling and could be done *before* Step 1.

- *Bottom-up specification of properties.* Alternatively, the sociotechnical properties of the system can be developed by working with the modeling. For example, one could leverage the systems developed in the threat modeling literature (STRIDE, LINDDUN, etc...) to elicit a set of risks and harms, using our modeling as a modified data-flow diagram. This would include iterating through the various interfaces between ideal functionalities and untrusted code in $\Pi_{\text{Intended-use}}$ (or looking at the decisions made in Π_{Social}) and seeing the extent to which they are vulnerable to spoofing, tampering, etc... Ultimately, this process could still result in sociotechnical goals that could be read as top-level properties the composed system should have (e.g., "even if [human operator] spoofs their identity...").

Clearly, the specificity of these statements, and the quantifiers used therein, require immense care. To illustrate this dynamic, we can look ahead to our case study in Section 4.2: the statements "The system can only reveal CSAM material to Apple's administrators" and "The system can only reveal images that an NCMEC administrator has designated as CSAM to Apple's administrators" are extremely similar, but have wildly different policy implications. Thus, crafting sociotechnical properties requires care tantamount to game-based definitions or ideal functionalities.

Step 3: Interrogating the sociotechnical properties. The final step of the framework is to study the extent to which the system achieves the desired sociotechnical properties. The exact form of the argument that cryptographers are expected to make about these properties is up to them, but the goal should be that the argument is convincing to even a skeptical reader. One potential way to do this is treating the output of our framework as a data-flow diagram and applying STRIDE/LINDDUN to this data-flow diagram. No matter the exact approach, we stop short of proposing that our framework facilitates formal proofs about these sociotechnical properties. While it might be possible to construct proofs for these statements, there are non-trivial modeling choices that must first be overcome, which might come into conflict with our goal to keep the framework conceptually lightweight. We believe that this more formal approach might be valuable future work.

Next steps: ethical analysis. Having established a set of sociotechnical properties that the system will have, there is an opportunity to evaluate the *ethical* implications of those properties. It is clear that in many circumstances there may be significant debate that those properties represent a set of trade-offs that

are appropriate. We note that our framework provides *inputs* to this ethical debate, but is not designed to support it directly. There are, however, emerging efforts to create ethical frameworks for reasoning about computer security that may be valuable for navigating this process [KAL23].

3.4 Benefits and Limitations of our Approach

Hopefully, it is immediately evident that our approach *broadens the designer's analytical frame*, as elements of a deployment beyond $\mathcal{F}$ must be brought into conversation with $\mathcal{F}$. We argue that our approach is naturally *enumerative,* in that system designers can derive $\Pi_{\text{Intended-use}}$ and Π_{Social} by iterating through all of $\mathcal{F}$'s interfaces and ensuring that the origins of the data provided to those interfaces are well specified. Additionally, the division between $\mathcal{F}$, $\Pi_{\text{Intended-use}}$ and Π_{Social} clearly delineates the ways in which *trust is allocated throughout the system.* Procedures within $\mathcal{F}$ are assured cryptographically, whereas the correctness of procedures within $\Pi_{\text{Intended-use}}$ and Π_{Social} are not assured whatsoever. As described above, our approach is simply performing certain protocol design within UC. While there are non-trivial conceptual barriers to the use of UC itself, our framework adds little overhead to cryptographers already familiar with UC. Finally, specifying sociotechnical properties takes a first step at *providing a bridge for cross-disciplinary conversations.* That is not to say that these properties can always be taken directly from policy makers or that they will be interoperable with the language used by non-technical audiences as stated, but rather that the distance between these sociotechnical properties and the natural language that non-cryptographers might use to discuss cryptographic systems is not large.

Limitations. We see four non-trivial limitations that could be improved in future work:

(1) Defining the limits of the system being analyzed can be fraught and contestable; in other words, *how do we know that our analytical frame is sufficiently broad?* Our guiding principle is that the boundaries of the analysis should extend to the actions of the human operators who drive the system and capture the datafication of the physical world—usually by having data be manually imported into the system or designated as relevant within Π_{Social}. This leads to natural boundaries within our chosen case study, but we anticipate that there may be other systems in which boundary navigation will be unclear.

(2) Our framework falls short of guaranteeing that a cryptographer will consider *all* potential attack vectors within a sociotechnical system—a likely insurmountable goal. Our goal of being *systematic and enumerative* should be understood as applying to the *specification* of $\Pi_{\text{Intended-use}}$ and Π_{Social}, which then provide the opportunity for systematic interrogation (e.g., iterating through the interfaces between the three consecutive layers and identifying opportunities for abuse). However, there is no way to guarantee that this systematic process will lead a cryptographer to consider all social dynamics at play or create consensus on the reasonableness of deployment.

(3) Once our framework relegates parts of the system to $\Pi_{\text{Intended-use}}$, it does not provide obvious ways to—or reason to—interrogate the properties of its subcomponents. Rather, all parts of $\Pi_{\text{Intended-use}}$ are treated equally. But, as we will see in Section 4.2, this might suppress the difference between errors that are inherent to the specification of $\Pi_{\text{Intended-use}}$ (i.e., subroutines that do not have perfect robustness or correctness) and errors introduced by a party not following the $\Pi_{\text{Intended-use}}$ honestly. This is an important difference and one which improvements of this work should endeavor to highlight.

(4) Finally, we acknowledge that the technique outlined above implicitly argues for a specific scope of analysis for which cryptographers *should* be responsible. By focusing on the idealized cryptographic properties of the system instead of the software that implements the system, we are making a claim that software is "not the cryptographer's problem." This claim is not inherent. Many members of the cryptographic community focus on elements of software, including constant-time implementations and formal verification. Additionally, members of the cryptographic community have previously argued against backdoors in cryptographic systems by invoking *software complexity* explicitly [AAB$^+$15, AAB$^+$21]—thereby arguing that awareness of the software required to implement cryptographic systems should be the purview of cryptographers. Other approaches to solving this problem could reasonably

take a different position on this issue, resulting in a different cryptography-native threat modeling process.

4 Case Study: The Apple CSAM Scanning Proposal

In August 2021, Apple announced a set of changes to their services with the goal of promoting child safety [App21c]. Most prominent among these changes[4] was the introduction of a mechanism for scanning the contents of end-to-end encrypted image backups for known instances of CSAM [App21b,Bel21a,Bel21b,For21, Pin21]. The announcement was accompanied by overviews of the proposed mechanisms aimed towards a non-cryptographic audience [App21b], a whitepaper by Apple describing the cryptographic protocol underpinning the CSAM scanning system [App21a], and several independent analyses of the proposed system by leading members of the security, privacy, and cryptography communities [Bel21a, Bel21b, For21, Pin21].

The cryptographic core of the CSAM scanning proposal was a *fuzzy, threshold, private set-intersection* protocol that was amenable to incremental computation. At a high level, Apple servers would commit to a set of image hashes (generated using a perceptual hash function) and publicly distribute this (hiding) commitment, called `pdata`, to client devices. Each time a client would back-up an image, they would locally encrypt the image and then produce a *voucher* for the image, both of which would be uploaded to the server. The voucher served to input the perceptual hash of the image into a private set-intersection protocol, where the server's inputs were fixed by `pdata`. When a large enough set of vouchers (thus, *threshold*) indicated a match between the client's images and the server committed set, decryption keys for each matching image would be disclosed to the server.[5] This protocol was further enhanced to hide the number of matches (thus, *fuzzy*); throughout this work, we will focus on the non-fuzzy version of this protocol for simplicity. To demonstrate the security of their protocol, Apple's white paper provides an ideal functionality and proves that their protocol realizes it in simple UC [CCL15].

4.1 Background: Criticism of Apple's Proposal

Apple's system is an instantiation of a *client-side scanning* system, in which content moderation is performed on the client before encryption.[6] This approach to content moderation was discussed for several years before Apple's announcement, e.g., [Wea19, Gre19, Por19, Gra20, Ros20] as part of the ongoing debate about the appropriate role of encryption in society [AAB+15, NAoSM18, BCD+19]. Shortly before Apple's announcement, Kulshrestha and Mayer described a similar system [KM21] and described many of the following criticisms in their limitations discussion. Abelson et al. [AAB+21] remind us, therefore, that many criticisms of Apple's proposal identify inherent properties of client-side scanning as a whole. We note that Apple's proposal sparked a tremendous uproar (e.g., [AAB+21, Kob21b, FN21, CK21, Gil21, Ops21, GTS+, GS21, KKL+21, San21, MP21, MSST21, PPK21, Res21, Mur21, Esp21]), so we limit our summary to only the most salient criticisms.

Criticism 1: The system can be easily re-targeted. No technical measure limited the scanning system to CSAM and therefore other types of content, e.g., politically disruptive content, could be targeted. Abelson et al. [AAB+21] call this *Abuse by Authorized Parties*, while the EFF [MP21] and Green [GTS+] call this *Mission Creep*. While Apple has claimed that it would resist pressure to broaden the scope of its system [App21d], Apple has already made other concessions to governments [Nic21, NZW21, Ism22], indicating that sustained resistance might be difficult.

[4] This announcement also included mechanisms by which sexual content is automatically detected client-side within iMessage and parents are alerted if their children decided to view this content.

[5] The decision to use a threshold is both for policy reasons and technical reasons. First, the system was designed to go after the "worst offenders" who were holding large amounts of illegal content. From a technical perspective, the perceptual hash function has no provable properties and thus could easily produce "false positives." The use of a threshold protects against spurious matches produced by the hash function.

[6] See Scheffler and Mayer [SM23] and Kamara et al. [KKL+21] to contextualize client-side scanning within content moderation.

$\mathcal{F}_{\text{Apple-CSAM-Scan}}$

(The following are parameters of the functionality: identity of the server Apple, identity of client devices $\mathcal{U}_1, \ldots, \mathcal{U}_m$, and a threshold $\mathcal{T}$.) On activation, for all clients $\mathcal{U}_i$, initialize $\mathsf{IsCounting}_{\mathcal{U}_i} = \mathsf{False}$ and $\mathsf{IsInited}_{\mathcal{U}_i} = \mathsf{False}$.

Initializing Clients: Upon input $(\mathsf{InitScan}, \{\mathcal{U}_1, \ldots \mathcal{U}_{m'}\}, \{X_{\mathcal{U}_1}, \ldots, X_{\mathcal{U}_{m'}}\},)$ from Apple:

1. For $\mathcal{U}_i \in \{\mathcal{U}_1, \ldots \mathcal{U}_{m'}\}$, if $\mathsf{IsInited}_{\mathcal{U}_i} = \mathsf{True}$, continue. Otherwise record $X_{\mathcal{U}_i}$ and set $\mathsf{IsInited}_{\mathcal{U}_i} = \mathsf{True}$. Finally, send $(\mathsf{InitScanComplete})$ to Apple.

Process Initialization: Upon input $(\mathsf{ProcessInit})$ from a client $\mathcal{U}$:

1. Rejection sample two hash functions h_0, h_1 such that they could be used to construct a Cuckoo table with $X_{\mathcal{U}}$. If Apple is corrupt, allow $\mathcal{S}$ to supply h_0, h_1.
2. If $\mathsf{IsInited}_{\mathcal{U}} = \mathsf{True}$, set $\mathsf{IsCounting}_{\mathcal{U}} = \mathsf{True}$ and send $(\mathsf{ProcessInitComplete}, |X_{\mathcal{U}}|, h_0, h_1)$ to $\mathcal{U}$. Otherwise, send $(\mathsf{ProcessInitComplete}, \perp)$ to $\mathcal{U}$.

Scan Image: Upon input $(\mathsf{ScanImage}, \mathsf{img_id}, \mathsf{img_hash}, \mathsf{k}, \mathsf{valid} \in \{\mathsf{true}, \mathsf{false}\})$ from a client $\mathcal{U}$:

1. If $\mathsf{IsCounting}_{\mathcal{U}} \neq \mathsf{True}$, return. If this is the first image received from $\mathcal{U}$, initialize variable $\mathsf{counter}_{\mathcal{U}}$, list $\mathsf{img_id_list}_{\mathcal{U}}$, and list $\mathsf{keys}_{\mathcal{U}}$.
2. If $\mathsf{img_hash} \notin X_{\mathcal{U}}$ or $\mathsf{valid} = \mathsf{false}$, send $(\mathsf{ScanImageComplete}, \mathcal{U}, \mathsf{false})$ to Apple. `//valid models malformed messages`
3. If $\mathsf{img_hash} \in X_{\mathcal{U}}$ and $\mathsf{valid} = \mathsf{true}$, increment $\mathsf{counter}_{\mathcal{U}}$, append $\mathsf{img_id}$ to $\mathsf{img_id_list}_{\mathcal{U}}$ and append k to $\mathsf{keys}_{\mathcal{U}}$. Then,
   (a) If $\mathsf{counter}_{\mathcal{U}} \leq \mathcal{T}$ send $(\mathsf{ScanImageComplete}, \mathcal{U}, \mathsf{true})$ to Apple.
   (b) If $\mathsf{counter}_{\mathcal{U}} > \mathcal{T}$, send $(\mathsf{ThresholdMet}, \mathcal{U}, \mathsf{img_id_list}_{\mathcal{U}}, \mathsf{keys}_{\mathcal{U}})$ to Apple.

Figure 2: Apple's CSAM scanning proposal. We have updated the notation and modeling to work within our framework. For a description of their initial ideal functionality, protocol, and our changes, see Appendix A.

Criticism 2: Malicious operators or hackers could abuse the system. Even if Apple were able to resist external pressure to broaden the scope of the system, a malicious operator could include non-CSAM content into the hashset. Abelson et al. [AAB+21] call this *Abuse by Unauthorized Parties* and Green [GTS+] calls this *Unauthorized Surveillance*. Apple's system also provides no mechanism to ensure that each client device receives the same hashset [Kob21a]. Apple suggested that some kind of auditing measure could mitigate the risk of these attacks [App21a, App21d] by making attacks *post-hoc* detectable, but the details of the auditing system were never specified.[7]

Criticism 3: The system is not sufficiently robust. Even if the integrity of the hashset is maintained, critics were concerned about the robustness of the system. Put succinctly by Abelson et al. [AAB+21], *"CSS [Client Side Scanning] Is Less Efficacious in Adversarial Environments."* This criticism spans two concrete concerns: (a) there is evidence that existing perceptual hash functions, including the one designed by Apple, are not robust [Dwy21, SHNK22, PFG+23a, LAP24], which could lead to false positives and false negatives, and (b) malicious clients could encrypt or modify images to evade detection [KM21].

4.2 Modeling Apple's Initial CSAM Scanning Proposal

We now illustrate applying our framework step by step.

Step 0: Defining the ideal functionality $\mathcal{F}_{\text{Apple-CSAM-Scan}}$. The initial formal description of the protocol provided by Apple is very similar to a classic two-party private set intersection protocol, with the added complexity of (1) only releasing the output when the set intersection exceeds some fixed threshold, and (2) switching the semantics of the private set intersection to be more "key-value," such that the intersection is performed on a set of keys, but the data associated with each of those keys constitutes the actual output.

[7] Scheffler, Kulshrestha, and Mayer [SKM23a] study the problem of adding auditability into Apple's proposal.

Working with this initial formalism within our framework poses several challenges. First, the description of the ideal functionality treats the user input as a *batch*, whereas input is actually provided to the clients incrementally. This relaxation suppresses an important leakage: Apple will know when a client is *approaching* the threshold, which itself could be of interest to, for example, law enforcement. Second, Apple's formalism relies on parallel composition to support multiple clients. While this latter choice is technically sound, it makes exposition in this paper more difficult.

We provide an updated model of Apple's ideal functionality, which we call $\mathcal{F}_{\text{Apple-CSAM-Scan}}$, in Figure 2. We modify Apple's initial formalism by addressing the two difficulties outlined above. Specifically, we have clients provide their inputs incrementally rather than as a batch, and all clients interact with the *same* ideal functionality, rather than relying on composition. Additionally, we add some leakage that is missing from Apple's initial modeling (namely, the hash functions used in their protocol that are sampled in an input-dependent way). We provide Apple's initial ideal functionality modeling and protocol in Appendix A. We discuss the small protocol changes (related to equivocation and extraction) required to make their protocol realize $\mathcal{F}_{\text{Apple-CSAM-Scan}}$ in Appendix A.3.

Step 1: Extending the modeling. Following the guiding principle for Step 1 we laid out in Section 3.3, modeling Apple's scanning proposal within our framework requires "explaining" all of the data consumed by $\mathcal{F}_{\text{Apple-CSAM-Scan}}$. In this case, these inputs include the hashset X, the client-supplied images and cryptographic keys that will be input to $\mathcal{F}_{\text{Apple-CSAM-Scan}}$. This requires introducing an additional party and additional ideal functionalities. First, we introduce the National Center for Missing and Exploited Children (NCMEC) as the originator of the hashset X, with a component in both $\Pi_{\text{Intended-use}}$ and Π_{Social}. Next, we surface the *purpose* of running a private set-intersection protocol by introducing two additional simple ideal functionalities: $\mathcal{F}_{\mathsf{PhotoStore}}$ and $\mathcal{F}_{\mathsf{ConvergentEnc}}$. The former captures a system that stores ciphertexts (i.e., an arbitrary blob store) and the latter captures *convergent encryption* [DAB+02, BKR13, KBR13], a deterministic encryption mode Apple uses to reduce server-side storage [App24, App25].[8] In convergent encryption, a large message (i.e., an image) is encrypted under a key derived from the message itself (i.e., $c = \mathsf{Enc}(\mathsf{KDF}(m), m)$) and then the derived key $\mathsf{KDF}(m)$ is encrypted under some user-selected key k. This construction is clearly not IND–CPA and, therefore, the blob store can reuse a single ciphertext for multiple clients. The practice of using convergent encryption to minimize storage is widespread, despite its clear security limitations. Descriptions of these ideal functionalities can be found in Figure 4 and Figure 5. We also implicitly make use of standard secure message transfer through $\mathcal{F}_{\mathsf{SMT}}$ [Can01].

Next, we describe the process of rendering Apple's proposal into our framework. An overview of the result can be seen in Figure 3. $\Pi_{\text{Intended-use}}$ and Π_{Social} can be found in Figure 6 and Figure 7 respectively. There are three primary information flows that we identify, each of which is labeled with a different number in Figure 3: (1) A flow in which the target images are selected, processed, and initialized into the system; (2) A flow describing a client backing up one of their images; and (3) A flow describing a backup that results in the threshold being met, including the resulting decryption.

Process. In order to determine the contents of the protocols $\Pi_{\text{Intended-use}}$ and Π_{Social}, we look through each interface of Figure 14 and note the data that is sent as input for each interface. As described above, it is clear that the hashes for which the functionality is scanning must be made concrete and the meaning of the inputs for each client image submission must be made concrete. Having introduced the additional parties and functionalities described above, we make sure that there is a clear source for each of these pieces of data and it has a path from its source to $\mathcal{F}_{\text{Apple-CSAM-Scan}}$. We note that all modeling in this work assumes *static* corruptions for simplicity.

Flow 1: NCMEC selects the images and Apple initializes scanning. The images in Apple's proposed system originate with NCMEC.[9] Thus, the first step (1a) is to have an administrator at NCMEC select a

[8] The iCloud Security Overview described this by saying that "The raw byte checksum of the photo or video" is stored under "standard encryption" rather than "end-to-end encryption," i.e., the hash of the image is stored after being encrypted under a key controlled by Apple servers [App25]. In Apple's platform security guide, the documentation explicitly names this practice as *convergent encryption* [App24].

[9] Apple later clarified that it would only use images that were held by child protection organizations in at least two organizations.

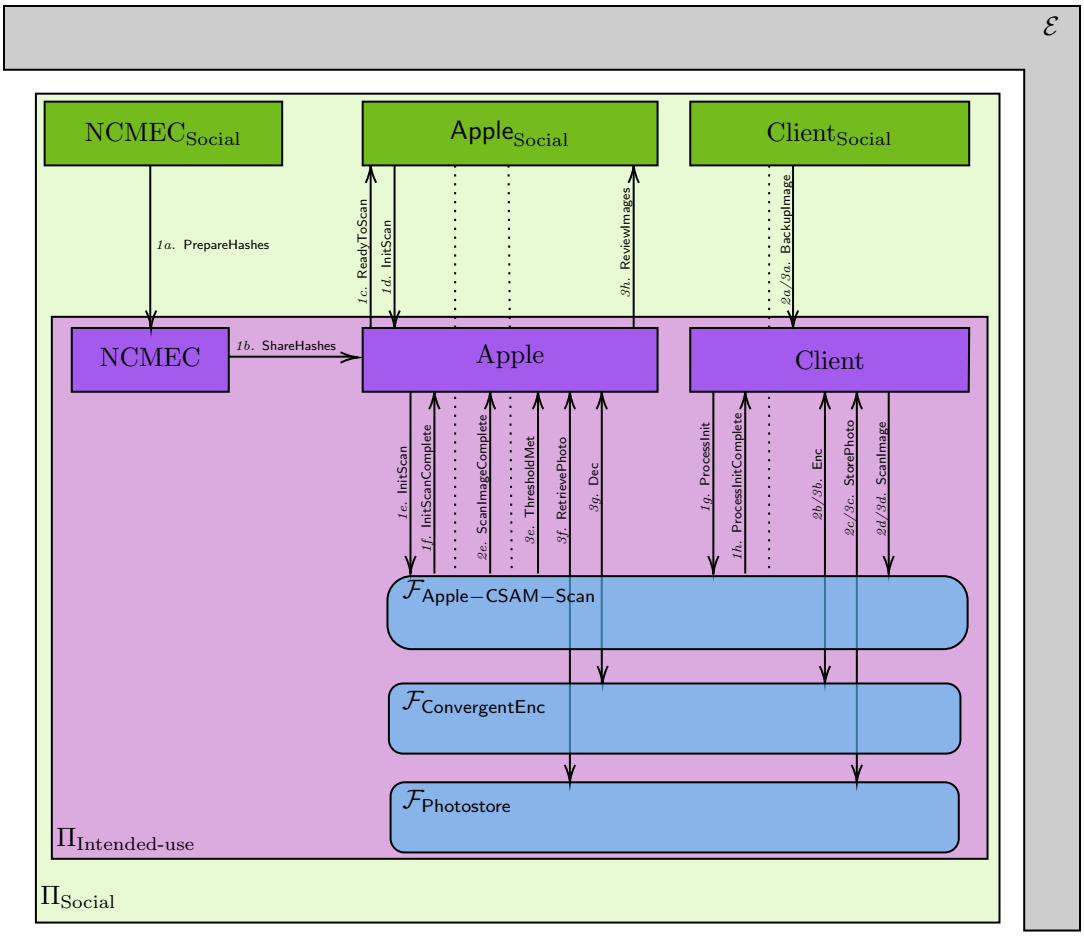

Figure 3: Overview of Apple's proposed protocol in our framework. Numbering indicates flows described in text below.

set of images which should be included in the scanning mechanism. The source of these images—and the mechanisms by which CSAM images are sifted from "other" images—is unspecified; for both of these choices there is no clear normative claim we can make about "honest" behavior or a verification function we could write to ensure that the NCMEC admin has behaved "honestly". As such, this process must happen in Π_{Social}. The NCMEC admin then invokes their software to hash each of the selected images with a perceptual hash function and sends the resulting hashes to Apple's systems (1b). Apple's systems inform an Apple admin that scanning can commence (1c). When the Apple admin is ready, they can initialize the system by calling start (1d) and sending the perceptual hashes to the ideal functionality $\mathcal{F}_{\text{Apple-CSAM-Scan}}$ (1e/f). Finally, client devices check $\mathcal{F}_{\text{Apple-CSAM-Scan}}$ for initialization (1g/h).

Flow 2: Client backing-up image. Clients might receive images from *anywhere* that they then choose to backup. For example, images might be downloaded from the internet, received from another client over an app, or might be a new image captured by the client device itself. As such, the origin of these images is not specified and is just input to the client from the environment in Π_{Social}. The second flow begins with a client choosing to backup some image (2a); notice that the choice to save an image to iCloud is user-driven and there is no assumed correct choice, meaning this choice needs to be situated in Π_{Social}. Next, the client device handles the processing of this image within $\Pi_{\text{Intended-use}}$, by encrypting the image with $\mathcal{F}_{\text{ConvergentEnc}}$ (2b), generating a new cryptographic key, and sending the resulting ciphertext to $\mathcal{F}_{\text{PhotoStore}}$ (2c). The image

$\mathcal{F}_{\mathsf{PhotoStore}}$

(The identity of an authority Auth is a parameter of $\mathcal{F}_{\mathsf{PhotoStore}}$.)

1. Upon receiving $(\mathsf{StorePhoto}, \mathcal{U}, \mathsf{img_id}, \mathsf{ctx})$ from $\mathcal{P} \in \{\mathcal{U}, \mathsf{Auth}\}$:
   (a) If there already exists a record for ctx, associate the tuple $(\mathcal{U}, \mathsf{img_id})$ with that record.
   (b) Otherwise, create a new record for ctx and associate $(\mathcal{U}, \mathsf{img_id})$ with the new record.
   (c) Send $(\mathsf{StorePhotoComplete})$ to $\mathcal{P}$.
2. Upon receiving $(\mathsf{RetrievePhoto}, \mathcal{U}, \mathsf{img_id})$ from $\mathcal{P}$:
   (a) If $\mathcal{P} = \mathcal{U}$ and there exists a record ctx with which $(\mathcal{U}, \mathsf{img_id})$ is associated, then send $(\mathsf{RetrievePhotoComplete}, \mathsf{img_id}, \mathsf{ctx})$ to $\mathcal{P}$.
   (b) If $\mathcal{P} = \mathsf{Auth}$ and there exists a record ctx with which $(\mathcal{U}, \mathsf{img_id})$ is associated, then send $(\mathsf{RetrievePhotoComplete}, \mathsf{img_id}, \mathsf{ctx}, \mathsf{users})$ to $\mathcal{P}$, where users is the set of all tuples $(\mathcal{U}, \mathsf{img_id})$ associated with ctx.
   (c) Otherwise, send $(\mathsf{RetrievePhotoComplete}, \mathsf{img_id}, \perp)$ to $\mathcal{P}$

Figure 4: Ideal Functionality for a Photo Store that holds encrypted images. This is, more generally, an encrypted blob store with an authority charged with managing access to those encrypted blobs.

identifier, the perceptual hash of the image, and the key generated when interacting with $\mathcal{F}_{\mathsf{ConvergentEnc}}$ are then sent to $\mathcal{F}_{\text{Apple-CSAM-Scan}}$ (2d). Finally, $\mathcal{F}_{\text{Apple-CSAM-Scan}}$ notifies Apple's system that the client has submitted a voucher and informs Apple if the submitted voucher is a match.

Flow 3: Client backing-up image that triggers threshold. The process for client uploads that exceed the scanning threshold begins exactly as described in Flow 2 (see 3a-3d). However, when the threshold is met, $\mathcal{F}_{\text{Apple-CSAM-Scan}}$ sends the data associated with all matching submissions to Apple (3e). Using this information, Apple can retrieve the relevant ciphertexts from $\mathcal{F}_{\mathsf{PhotoStore}}$ (3f) and then decrypt each image (3g). These final two steps (3f and 3g) are called for each image in the intersection. The resulting image decryptions are then sent to Apple for manual review (3h). Again, there is no normative process that we can describe at this point. Presumably, an Apple administrator will be responsible for looking at the images and determining if they are indeed CSAM, a process for which there is no algorithm.

4.3 Examining Sociotechnical Properties

Having rendered Apple's proposal into our framework, we now interrogate its sociotechnical properties. We present the results of conducting Step 2 and Step 3 together, although we performed these steps sequentially. We begin with the "top-down" approach, leaning on sociotechnical goals that Apple set in a document release for public consumption. We then turn our attention to the "bottom-up" approach and show how this analytical approach can surface the concerns listed in Section 4.1.

Top-down properties. Within their non-technical overview [App21b], Apple provided a set of informally phrased guarantees that their system was supposed to provide. These represent the most explicit enumeration of the sociotechnical goals of Apple's proposal, so we interrogate them within our framework.[10] We discuss these five properties in the order listed in [App21b]:

(1) **"Apple does not learn anything about images that do not match the known CSAM database."** Within our model, we see that the only way for Apple to learn information about the content of image files that users back-up is when key information is released to Apple via $\mathcal{F}_{\text{Apple-CSAM-Scan}}$.

[10] We acknowledge that the linguistic standards by which we evaluate their claims may be unfair given their initial context.

$\mathcal{F}_{\mathsf{ConvergentEnc}}$

(The deterministic ciphertext generation function G and a key space $\mathcal{K}$ are parameters of $\mathcal{F}_{\mathsf{ConvergentEnc}}$. If desired, G can also be explicitly provided to $\mathcal{F}_{\mathsf{ConvergentEnc}}$ on first activation.)

1. Upon receiving $(\mathsf{Enc}, \mathsf{id}, m)$ from $\mathcal{U}$, generate a ciphertext $\mathsf{ctx} \leftarrow G(m)$. Then,
   (a) if $\mathcal{U}$ is corrupt, send $(\mathsf{RequestKey}, \mathsf{id})$ to $\mathcal{S}$. If $(\mathsf{SetKey}, \mathsf{id}, \mathsf{k})$ is received in response, then record $(\mathsf{id}, \mathsf{ctx}, \mathsf{k}, m)$ and output $(\mathsf{EncComplete}, \mathsf{id}, \mathsf{ctx}, \mathsf{k})$ to $\mathcal{U}$.
   (b) if $\mathcal{U}$ is not corrupt, set $\mathsf{k} \xleftarrow{\$} \mathcal{K}$, record $(\mathsf{id}, \mathsf{ctx}, \mathsf{k}, m)$ and output $(\mathsf{EncComplete}, \mathsf{id}, \mathsf{ctx}, \mathsf{k})$ to $\mathcal{U}$.
2. Upon receiving $(\mathsf{Dec}, \mathsf{id}, \mathsf{ctx}, \mathsf{k})$ from anyone, check if there exists a record with the values $(\mathsf{id}, \mathsf{ctx}, \mathsf{k}, m)$. If such a tuple exists, return $(\mathsf{DecComplete}, \mathsf{id}, m)$. Else, return $(\mathsf{DecComplete}, \mathsf{id}, \perp)$.
3. Upon receiving $(\mathsf{CheckPlaintext}, \mathsf{ctx}, m)$ from anyone, check if $\mathsf{ctx} = G(m)$. If yes, return $(\mathsf{CheckPlaintextComplete}, \mathsf{ctx}, m, \mathsf{true})$. Else, return $(\mathsf{CheckPlaintextComplete}, \mathsf{ctx}, m, \mathsf{false})$.

Figure 5: Ideal functionality for convergent encryption. The CheckPlaintext interface is not strictly necessary, as G could be publicly known. We include this for clarity.

However, the existing phrasing has two ambiguous terms: "match" and "CSAM database." A reasonable reading of the term "match" might include an exact match, but in fact this is trying to capture the semantic matching properties provided by the perceptual hash function. As for the "CSAM Database," this term could reasonably either refer to the set of images selected and passed through the perceptual hash function by NCMEC or could refer to the set of perceptual hashes Apple sends to the ideal functionality. Interrogating this property within the context of our model lends itself to seeing these differences because this property is clearly discussing the input-output behavior of the ideal functionality. As a result, an analyst might rephrase this property to be "Apple does not learn anything about images that do *not have the same perceptual hash as those selected by Apple for scanning.*"

(2) **"Apple can't access metadata or visual derivatives for matched CSAM images until a threshold of matches is exceeded for an iCloud Photos account."** There are two problems with this property: the first is largely an issue of semantics/clarity, while the second undermines the veracity of this claim entirely. First, there is ambiguity in the terms "metadata or visual derivatives." Specifically, when a match is identified, Apple's software is alerted as to the existence of a new match; while this is metadata as technically understood, it might not be metadata as conversationally understood, leading to potential misunderstanding. Moreover, Apple's iCloud data security overview explicitly lists instances of metadata that are accessible to Apple under all circumstances—although this is unlikely to be the specific metadata about which this claim is made.

Much more notably, this claim becomes false *once the composition of the full system becomes apparent*, i.e., once the analysis includes the manner in which the images are stored and the fact that there are many clients in the system. Namely, the interaction between storing the images with convergent encryption, an encryption mode that is explicitly not IND–CPA, and a threshold PSI scheme that is designed to detect semantic matches undermines this claim.

Recall that convergent encryption allows Apple to identify when two users store bit-for-bit identical images. This will directly reveal instances where a user uploads an image that exactly matches one of the instances of known CSAM for which the system is scanning, but does not directly leak when a user uploads a semantically equivalent image (e.g., changing a single pixel value or rotating the image); it is possible for someone to identify the plaintext corresponding to a convergent encryption, but there is likely sufficient entropy in most images to make this challenging in practice. The threshold PSI system hides when different users share inputs, unless they both exceed the threshold. When

$\Pi_{\text{Intended-use}}$ for $\mathcal{F}_{\text{Apple-CSAM-Scan}}$

(The protocol is parameterized by a perceptual hash function PHash*.)*

NCMEC: When NCMEC is invoked with the message $(\mathsf{PrepareHashes}, \mathcal{I} = \{\mathsf{img}_1, \mathsf{img}_2, \ldots\})$ from $\text{NCMEC}_{\text{Social}}$:

1. Set $X = \{\}$, then for $\mathsf{img}_i \in \mathcal{I}$, compute $\mathsf{img_hash}_i = \mathsf{PHash}(\mathsf{img}_i)$ and set $X = X \cup \{\mathsf{img_hash}_i\}$.
2. Send the message $(\mathsf{ShareHashes}, X = \{x_1, \ldots, x_n\})$ to Apple via an instance of $\mathcal{F}_{\mathsf{SMT}}$.

Apple: When Apple receives a message $(\mathsf{ShareHashes}, X = \{\mathsf{img_hash}_1, \ldots, \mathsf{img_hash}_n\})$ from NCMEC via $\mathcal{F}_{\mathsf{SMT}}$:

1. Store X and send $(\mathsf{ReadyToScan})$ to $\text{Apple}_{\text{Social}}$.

Apple: When Apple is invoked with the message $(\mathsf{InitScan})$ from $\text{Apple}_{\text{Social}}$:

1. Retrieve X. Let $\{\mathcal{U}_1, \ldots, \mathcal{U}_m\}$ be the set of all client devices. Send $(\mathsf{InitScan}, \{\mathcal{U}_1, \ldots, \mathcal{U}_m\}, \{X, \ldots, X\})$ to $\mathcal{F}_{\text{Apple-CSAM-Scan}}$. Receive $(\mathsf{InitScanComplete})$ in response and return.

Client: Until a message $(\mathsf{ProcessInitComplete}, |X_{\mathcal{U}}|, h_0, h_1)$ has been received from $\mathcal{F}_{\text{Apple-CSAM-Scan}}$, whenever the client device $\mathcal{U}$ is invoked it sends $(\mathsf{ProcessInit})$ to $\mathcal{F}_{\text{Apple-CSAM-Scan}}$ and processes the result.

Client: When a client device $\mathcal{U}$ is invoked with the message $(\mathsf{BackupImage}, \mathsf{img})$ from $\text{Client}_{\text{Social}}$:

1. Sample a unique random value $\mathsf{img_id}$
2. Send $(\mathsf{Enc}, \mathsf{img_id}, \mathsf{img})$ to $\mathcal{F}_{\mathsf{ConvergentEnc}}$ and receive the output $(\mathsf{EncComplete}, \mathsf{img_id}, \mathsf{ctx}, \mathsf{k})$.
3. Send $(\mathsf{StorePhoto}, \mathcal{U}, \mathsf{img_id}, \mathsf{ctx})$ to $\mathcal{F}_{\mathsf{PhotoStore}}$ and receive the output $(\mathsf{StorePhotoComplete})$ in response.
4. Compute $\mathsf{img_hash} = \mathsf{PHash}(\mathsf{img})$ and send $(\mathsf{ScanImage}, \mathsf{img_id}, \mathsf{img_hash}, \mathsf{k})$ to $\mathcal{F}_{\text{Apple-CSAM-Scan}}$.

Apple: When Apple receives a message $(\mathsf{ScanImageComplete}, \mathcal{U}, \mathsf{img_id}, \mathsf{match} \in \{\mathsf{true}, \mathsf{false}\})$ from $\mathcal{F}_{\text{Apple-CSAM-Scan}}$, return control flow to the environment.

Apple: When Apple receives a message $(\mathsf{ThresholdMet}, \mathcal{U}, \mathsf{img_id}, \mathsf{img_id_list}_{\mathcal{U}}, \mathsf{keys}_{\mathcal{U}})$ from $\mathcal{F}_{\text{Apple-CSAM-Scan}}$:

1. Initialize the empty set $\mathsf{decrypted_photos}_{\mathcal{U}}$.
2. For $i \in [|\mathsf{img_id_list}_{\mathcal{U}}|]$, send $(\mathsf{RetrievePhoto}, \mathcal{U}, \mathsf{img_id}_i)$ to $\mathcal{F}_{\mathsf{PhotoStore}}$. If $(\mathsf{RetrievePhotoComplete}, \mathsf{img_id}_i, \perp)$ is received in response, continue. Otherwise, if $(\mathsf{RetrievePhotoComplete}, \mathsf{img_id}_i, \mathsf{ctx}_i, \mathsf{users}_i)$ is received:
   (a) Send $(\mathsf{Dec}, \mathsf{img_id}_i, \mathsf{ctx}_i, \mathsf{k}_i)$ to $\mathcal{F}_{\mathsf{ConvergentEnc}}$. If $(\mathsf{DecComplete}, \mathsf{img_id}_i, \perp)$ is received in response, continue. Otherwise, if $(\mathsf{DecComplete}, \mathsf{img_id}_i, \mathsf{img}_i)$ is received in response, add img_i to $\mathsf{decrypted_photos}_{\mathcal{U}}$.
3. Send $(\mathsf{ReviewImages}, \mathcal{U}, \mathsf{decrypted_photos})$ to $\text{Apple}_{\text{Social}}$.

Figure 6: $\Pi_{\text{Intended-use}}$ for $\mathcal{F}_{\text{Apple-CSAM-Scan}}$, specifying how non-cryptographic system components *should* operate.

combined, however, Apple could identify that a particular ciphertext is a semantic match for known CSAM if one client passes the threshold and convergent encryption will then allow Apple to identify all other users storing a bit-for-bit match with this image.

To make this concrete, consider the following scenario: Client A sends Client B an image that is a semantic match—but not a bit-for-bit match—with one of the images selected for scanning. When they each upload this image to the system, Client B passes the "threshold of matches," making this image accessible to Apple. The immediate implication is that Apple will also be able to access a "visual derivative" (i.e., the plaintext of the image itself) of one of Client A's photos, even though their iCloud account has not exceeded the match threshold.

(3) **"The risk of the system incorrectly flagging an account is extremely low. In addition, Apple manually reviews all reports made to NCMEC to ensure reporting accuracy."** This is perhaps the most contentious sociotechnical property provided in [App21b]. Without a quantitative way to evaluate "extremely low," this property likely is not analytically helpful. Leblanc-Albarel and Preneel [LAP24] recently started studying the perceptual hash functions used in client-side scanning and found relatively higher rates of hash collisions, which may cast doubt on this sociotechnical property.

(4) **"Users can't access or view the database of known CSAM images."** This is a clearly specified

Π_{Social} for $\mathcal{F}_{\text{Apple-CSAM-Scan}}$

(The protocol is parameterized by a threshold $\mathcal{T}$.)

$\text{NCMEC}_{\text{Social}}$: When $\text{NCMEC}_{\text{Social}}$ is invoked with $\mathcal{I} = \{\mathsf{img}_1, \mathsf{img}_2, \ldots\}$ from Env:

1. Select a subset of images $\mathcal{I}' \subseteq \mathcal{I}$ that are CSAM, then invoke $\Pi_{\text{Intended-use}}$ with the message $(\mathsf{PrepareHashes}, \mathcal{I}')$.

$\text{Apple}_{\text{Social}}$: When $\text{Apple}_{\text{Social}}$ receives $(\mathsf{ReadyToScan})$ from Apple, return control flow to the environment.

$\text{Apple}_{\text{Social}}$: When $\text{Apple}_{\text{Social}}$ is invoked with $(\mathsf{InitScan})$ from Env:

1. If $\mathsf{AppleAdmin}$ is ready to initialize scanning, invoke $\Pi_{\text{intended-use}}$ with the message $(\mathsf{InitScan})$.

$\text{Client}_{\text{Social}}$: When a client Client is invoked with image img by Env :

1. Determine if img is to be backed up. If so, invoke $\Pi_{\text{Intended-use}}$ with the message $(\mathsf{BackupImage}, \mathsf{img})$. Otherwise, return control flow to the environment.

$\text{Apple}_{\text{Social}}$: When $\text{Apple}_{\text{Social}}$ receives $(\mathsf{ReviewImages}, \mathcal{U}, I = \{\mathsf{img}_1, \mathsf{img}_2, \ldots\})$ from Apple:

1. Review the images I and determine if they are CSAM.
2. If any images are not CSAM, report the existence of a design flaw to all other parties.
3. If $|I| < \mathcal{T}$, report that either there is a design flaw or $\mathcal{U}$ is corrupt to all other parties.
4. Send the images to NCMEC or law enforcement as necessary to comply with existing laws.
5. Output I to Env.

Figure 7: Π_{Social} for $\mathcal{F}_{\text{Apple-CSAM-Scan}}$, describing parts of the system that will not be enshrined in software.

sociotechnical property that appears to hold in the system. Specifically, the only leakage about the database of known CSAM images (both the images curated by NCMEC and the perceptual hashes pre-selected by Apple) is the size of the set and the hash functions h_0, h_1. While this latter leakage might allow an attacker to gain partial information about the images, they cannot facilitate "access[ing]" or "view[ing]" the images.

(5) **"Users can't identify which images were flagged as CSAM by the system."** Within the scope of our modeling, there is no way for a user to determine if their image back-up is a match or non-match. We note, however, that the system could be used to flag non-CSAM images, which could motivate slightly rewording this property. Moreover, there are aspects of the described system that are under-specified which make us unable to properly verify this property—imagine the flagged images were later introduced as court evidence.

Bottom-up criticisms. Next, we look to our modeling to see if it successfully allows us to observe the criticisms discussed in Section 4.1 in a bottom-up way. Ideally, these criticisms should follow naturally from asking the questions *What happens if* $\Pi_{\textit{Intended-use}}$ *is not followed?* and *Are there choices within* $\Pi_{\textit{Social}}$ *that could be problematic?* In other words, does the fact that particular parts of the protocol are contained with $\Pi_{\text{Intended-use}}$ and Π_{Social}, rather than the ideal functionality itself, indicate that there is an opportunity for abuse?

– *Flow 1 and Criticism 1:* First, we can see within Flow 1 that the entity responsible for curating the hashset can include arbitrary images, either by explicitly selecting images that others would not see as fitting a normative understanding of the purpose of the system (i.e., within Π_{Social}) or because of software error (i.e., within $\Pi_{\text{Intended-use}}$). Put another way, we can ask the question: *What happens when NCMEC chooses to include images that would not normally be understood to be CSAM in its hashset?* As a result, we see that the selection of images is a *social* process and is, thus, vulnerable to typical pathways of social disruption, i.e., government pressure and shifting norms. This serves to highlight Criticism 1, which could be rephrased as a sociotechnical property of the system as: "the only protection against including non-CSAM images in the hashset is the judgment of the NCMEC administrator."

– *Flow 1 and Criticism 2*: If we examine the second part of Flow 1, we observe that the server's software is expected to submit exactly the hashes supplied by NCMEC, but there is no mechanism to guarantee that this happens. Alternatively, we can see this by observing NCMEC provides no inputs to $\mathcal{F}_{\text{Apple-CSAM-Scan}}$. To frame this as a question, we could ask: *What happens when Apple's software selects images not supplied by NCMEC to the ideal functionality?* This serves to highlight Criticism 2, i.e., "Apple's management software can (untraceably) change the contents of the hashset." It is worth noting that the specifics of the user interface for Apple's management software (i.e., if there is an exposed choice to select photos) could easily shift this from a software concern to a social concern.

– *Flow 2/3 and Criticism 3:* Finally, within Flow 2 and Flow 3, it becomes very clear that the client must voluntarily tie both the perceptual hash and the cryptographic key they submit to the ideal functionality to the images that they backup. Specifically, we could ask the question: *What happens when the client device does not follow* $\Pi_{\textit{Intended-use}}$ *when selecting inputs to* $\mathcal{F}_{\mathsf{PhotoStore}}$, $\mathcal{F}_{\mathsf{ConvergentEnc}}$, *and* $\mathcal{F}_{\textit{Apple-CSAM-Scan}}$? This shows that it is trivial for a malicious client to evade detection, as noted in Criticism 3.

4.4 Takeaways from analyzing Apple's proposal

Applying our framework to Apple's protocol proves to be a valuable process that systematically uncovers the known criticisms of Apple's protocol. More importantly, our process highlighted a weakness of Apple's proposal that has not previously been discussed in the academic or policy discussion of the protocol (to our knowledge) despite multiple years of scrutiny: the leakage that results from running a scanning protocol and convergent encryption within the same system.[11] This leakage is highly non-trivial as it, from a technical perspective, undermines the main aspirational goal that Apple's system will not introduce any privacy leakage to individuals who are not holding multiple instances of CSAM. Moreover, this particular privacy leakage is *actionable*. Specifically, if Apple were to uncover that a particular convergent encryption ciphertext corresponds to a known instance of CSAM via their scanning system, they seem to have a legal responsibility to *report* the event to the relevant authorities (see 18 U.S. Code §2258A [USc]).[12] This report would likely include the existence of the ciphertext, the details of the individual whose input to the PSI protocol resulted in the decryption of the ciphertext, and the information of any other user who interacted with the ciphertext.[13] If Apple had deployed its proposed system, we very likely would have seen legal repercussions from this privacy leakage that the community had never discussed—further highlighting the need for broadening the scope of analysis.

Limitations. Having performed this careful rendering of Apple's proposal into our system, we note that there are several non-trivial limitations. First, there is not a good way to differentiate between the inherent error rate of the perceptual hash function and malicious behavior on the part of the client, even if practical implications of each are quite different. There has been significant work analyzing the security properties of perceptual hash functions (e.g., [PFG+23b,LAP24,HWAM25]) which show that they lack desired properties. Our framework, by treating the hash function as a property-less black-box, fails to highlight these weaknesses. While one direction might be to model this hash function as an ideal functionality, the concrete constructions rely on machine learning techniques that lack provable properties. Thus, at best, this approach would only let us analyze an idealized version of the hash function—creating a gap between the system "as deployed" and "as analyzed."

A second limitation of our analysis is *scope*. For example, a reasonable objection to our modeling might be that we did not address the fact that, in Apple's real proposed deployment, Apple is capable of pushing

[11] There is a non-public podcast with the title "Apple iCloud Encryption, CSAM Scanning and Convergent Encryption", which might potentially touch on this topic, extrapolating from the title alone. Given the non-public nature of the piece, none of the authors have reviewed it. Even if this piece does discuss the interplay between the PSI protocol and convergent encryption, this has not made its way into the academic conversation or policy discussion, to our knowledge.

[12] This is a best-effort reading of the law; the authors are not lawyers.

[13] 18 U.S. Code §2258A states that the report should contain "Information relating to when and how a customer or subscriber of a provider uploaded, transmitted, or received content relating to the report or when and how content relating to the report was reported to, or discovered by the provider, including a date and time stamp and time zone" [USc]

software updates to client devices. We have conducted the modeling as though the client devices are under the client's complete control. This is, of course, not the case in practice. Breaking down this modeling deeply complicates the task of analyzing *this* part of the protocol. After all, a malicious server with the ability to patch software can completely bypass the end-to-end encrypted nature of the image backups and learn about the client's images without needing to use the scanning system altogether. Nevertheless, understanding the social risks that come from using such a power may deserve to be modeled.

5 Updating Apple's Protocol

Having analyzed Apple's initial proposal within our framework, we now continue our case study by exploring how the framework can be used to design and discuss improvements to protocols. The goal of this exercise is to show some alternative, stronger sociotechnical properties and study ways to realize them. Namely, we specify some minimal, additional properties that one would want a cryptographic deployment of this kind to satisfy and then design a simple, updated cryptographic protocol that achieves them. We emphasize (as noted in Section 1.1) that it is not our goal to claim that these additional sociotechnical properties are *sufficient* to mitigate the risks associated with deployment. Instead, we take this as an opportunity to perform a more in-depth analysis of an alternative proposal that Apple easily could have made. Moreover, by updating Apple's proposal, we have an opportunity to explore the ways in which our analysis framework facilitates integrating social processes into the analytical frame—illustrating new ways in which our model can be used. Finally, studying extensions to the Apple protocol provides an opportunity to design protocols with the prior expectation that they will be subjected to analysis within our framework and observing any challenges that this brings.

Rather than follow the steps in the exact order we did above, we instead start by specifying desirable sociotechnical properties (Step 2) that motivate the design of the system itself and then design a system that attempts to provide those properties.

5.1 Step 2: Desirable Sociotechnical Properties

We begin our exploration of extending and updating Apple's proposal with a set of desirable sociotechnical properties that a system of this kind could (and should) provide.

Preventing abuse by the Apple administrator. When considering the criticisms of Apple's initial proposal, it becomes clear that **Criticism 2** (malicious operators could abuse the system) is the most ripe for technical intervention. Engaging with **Criticism 1** (system can be easily retargeted) technically seems to require the specification of a function that could verify if a given image is indeed CSAM, which is a problem that is well beyond our existing technical capabilities—and may be beyond our social and legal capabilities as well. While it may be more plausible to engage with **Criticism 3** (system is not sufficiently robust) technically, the existence of steganographic techniques [Sim83, HLvA02, vAH04] appears to put inherent limits on what is possible. Thus, we focus on integrating techniques that limit Apple operators to searching for *exactly* the perceptual hashes generated by NCMEC. Specifically, we want to improve the system such that it provides the following sociotechnical property:

(1) Apple cannot scan client's devices for any perceptual hashes not designated as CSAM by NCMEC.

Integrating with legal processes. Next, we can consider integrating this system with the legal system: the legal system should be *aware* of instances in which Apple's administrators are able to decrypt the photos of a client. Making sure that another entity is aware of this event (especially one that may have a legal responsibility to care about such an event) can mitigate some of the risks associated with false positives within the perceptual hash function. More generally, we might also want the legal system to be "in the loop" for the decryption process (as opposed to just "on the loop" and getting notifications), similar to the search warrant process in the United States. To describe these properties, we introduce the role of the *judge* who will be responsible for the actions of the legal system:

$\mathcal{F}_{\text{CSAM-Scan}}$

(The following are parameters of the functionality: identity of NCMEC, identity of the server Apple, identity of the judge $\mathcal{J}$, identity of client devices $\mathcal{U}_1, \ldots, \mathcal{U}_m$, a threshold $\mathcal{T}$, and a de-commitment string distribution $\mathcal{D}$.) On activation, obtain string generation algorithms $\mathcal{S}_{PSI}$, $\mathcal{S}_J$ from $\mathcal{S}$. Initialize $\mathsf{HashesSpecified} = \mathsf{False}$ and for all clients $\mathcal{U}_i$, initialize $\mathsf{IsCounting}_{\mathcal{U}_i} = \mathsf{False}$ and $\mathsf{IsInited}_{\mathcal{U}_i} = \mathsf{False}$.

Specify Hashes: On receiving $(\mathsf{SpecifyHashes}, X = \{\mathsf{img_hash}_1, \ldots, \mathsf{img_hash}_n\})$ from NCMEC:

1. If $\mathsf{HashesSpecified} = \mathsf{True}$, ignore the request. Otherwise set $\mathsf{HashesSpecified} = \mathsf{True}$ and continue.
2. For each $\mathsf{img_hash}_i \in X$, send $(\mathtt{HashQuery}, \mathsf{img_hash}_i, \lambda)$ to $\mathcal{G}_{\mathsf{pRO}}$ and collect the response values in a set ROHashes.
3. Sample authorization token $\mathsf{d} \leftarrow \mathcal{D}$.
4. Send $(\mathsf{HashesSpecified})$ to $\mathcal{S}$. When (OK) received in response, continue.
5. Store X and $(\mathsf{ROHashes}, \mathsf{d})$, and then output $(\mathsf{SpecifyHashesComplete}, \mathsf{ROHashes}, \mathsf{d})$ to NCMEC.

Initializing Clients: On receiving $(\mathsf{InitScan}, \{\mathcal{U}_1, \ldots, \mathcal{U}_\ell\}, \{\mathsf{ROHashes}_{\mathcal{U}_1}, \ldots, \mathsf{ROHashes}_{\mathcal{U}_\ell}\}, \mathsf{d}')$ from Apple:

1. If $\mathsf{HashesSpecified} \neq \mathsf{True}$, ignore the request.
2. For each $\mathcal{U}_i \in \{\mathcal{U}_1, \ldots, \mathcal{U}_\ell\}$: if $(\mathsf{ROHashes}_{\mathcal{U}_i}, \mathsf{d}')$ does not match the stored value $(\mathsf{ROHashes}, \mathsf{d})$ or $\mathsf{IsInited}_{\mathcal{U}_i} = \mathsf{True}$, continue. Otherwise, set $\mathsf{IsInited}_{\mathcal{U}_i} = \mathsf{True}$.
3. Send $(\mathsf{InitScanComplete})$ to Apple.

Process Initialization: Upon input $(\mathsf{ProcessInit})$ from a client $\mathcal{U}$:

1. Rejection sample two hash functions h_0, h_1 such that they could be used to construct a Cuckoo table with X. If Apple is corrupt, allow $\mathcal{S}$ to supply h_0, h_1.
2. If $\mathsf{IsInited}_{\mathcal{U}} = \mathsf{True}$, set $\mathsf{IsCounting}_{\mathcal{U}} = \mathsf{True}$ and send $(\mathsf{ProcessInitComplete}, |X|, h_0, h_1)$ to $\mathcal{U}$. Otherwise, send $(\mathsf{ProcessInitComplete}, \perp)$ to $\mathcal{U}$.

Scan Image: On receiving $(\mathsf{ScanImage}, \mathsf{img_id}, \mathsf{img_hash}, \mathsf{k}, \mathsf{valid} \in \{\mathsf{true}, \mathsf{false}\})$ from a client $\mathcal{U}$:

1. If $\mathsf{IsCounting}_{\mathcal{U}} \neq \mathsf{True}$, ignore the request. If this is the first image received from $\mathcal{U}$, initialize variable $\mathsf{counter}_{\mathcal{U}} = 0$, list $\mathsf{img_id_list}_{\mathcal{U}} = \emptyset$, and list $\mathsf{keys}_{\mathcal{U}} = \emptyset$.
2. If $\mathsf{img_hash} \notin X$ or $\mathsf{valid} = \mathsf{false}$, send $(\mathsf{ScanImageComplete}, \mathcal{U}, \mathsf{img_id}, \mathsf{false})$ to Apple.
3. If $\mathsf{img_hash} \in X$ and $\mathsf{valid} = \mathsf{true}$, increment $\mathsf{counter}_{\mathcal{U}}$, append $\mathsf{img_id}$ to $\mathsf{img_id_list}_{\mathcal{U}}$ and append k to $\mathsf{keys}_{\mathcal{U}}$. Then,
   (a) If $\mathsf{counter}_{\mathcal{U}} \leq \mathcal{T}$ send $(\mathsf{ScanImageComplete}, \mathcal{U}, \mathsf{img_id}, \mathsf{true})$ to Apple.
   (b) If $\mathsf{counter}_{\mathcal{U}} > \mathcal{T}$:
      – Generate a scan verification string and judge approval token as $(\pi_{\mathcal{U}}, \tau_{\mathcal{U}}) \leftarrow \mathcal{S}_{PSI}(\mathcal{U}, \mathsf{counter}_{\mathcal{U}})$.
      – Record $(\mathcal{U}, \pi_{\mathcal{U}}, \tau_{\mathcal{U}})$ and send $(\mathsf{ThresholdMet}, \mathcal{U}, \mathsf{img_id}, \pi_{\mathcal{U}}, \tau_{\mathcal{U}})$ to Apple. If both $\mathcal{J}$ and Apple are corrupt, send $(\mathsf{ThresholdMet}, \mathcal{U}, \mathsf{img_id}, \pi_{\mathcal{U}}, \tau_{\mathcal{U}}, \mathsf{img_id_list}, \mathsf{keys}, \pi^{\mathcal{J}}_{\mathcal{U}})$ to $\mathcal{S}$ instead.

Judge Approval: On receiving $(\mathsf{Approve}, \mathcal{U}, \tau_{\mathcal{U}})$ from the judge $\mathcal{J}$:

1. If there does not exist a record $\tau_{\mathcal{U}}$ associated with client $\mathcal{U}$, return. Otherwise,
   (a) Generate a judge verification token as $\pi^{\mathcal{J}}_{\mathcal{U}} \leftarrow \mathcal{S}_J(\tau_{\mathcal{U}})$
   (b) Record $(\mathcal{U}, \tau_{\mathcal{U}}, \mathsf{img_id_list}, \mathsf{keys}, \pi^{\mathcal{J}}_{\mathcal{U}})$ and send $(\mathsf{ApproveComplete}, \mathcal{U}, \mathsf{img_id_list}, \mathsf{keys}, \pi^{\mathcal{J}}_{\mathcal{U}})$ to $\mathcal{J}$. If both $\mathcal{J}$ and Apple are corrupt, send the same message to $\mathcal{S}$ instead.

Check If Hashes Specified: On receiving $(\mathsf{CheckSpecified})$ from anyone:

1. If $\mathsf{HashesSpecified} = \mathsf{True}$, respond $(\mathsf{CheckedSpecified}, \mathsf{Yes})$. Otherwise, return $(\mathsf{CheckedSpecified}, \mathsf{No})$.

Check Hashes: On receiving $(\mathsf{CheckHashes}, \mathsf{ROHashes}, \mathsf{d})$ from anyone:

1. If there exists a record $(\mathsf{ROHashes}, \mathsf{d})$, return $(\mathsf{CheckedHashes}, \mathsf{Yes})$. Otherwise, return $(\mathsf{CheckedHashes}, \mathsf{No})$.

Verify PSI Output: Upon input $(\mathsf{PublicVerifyPSI}, \mathcal{U}, \pi, \tau)$ from anyone:

1. If there exists a record $(\mathcal{U}, \pi, \tau)$ return $(\mathsf{PublicVerifyPSIComplete}, \mathsf{Yes})$. Otherwise, return $(\mathsf{PublicVerifyPSIComplete}, \mathsf{No})$

Verify Judge Output: On receiving $(\mathsf{PublicVerifyJudge}, \mathcal{U}, \tau, \mathsf{img_id_list}, \mathsf{keys}, \pi^{\mathcal{J}})$ from anyone:

1. If there exists a record $(\mathcal{U}, \tau, \mathsf{img_id_list}, \mathsf{keys}, \pi^{\mathcal{J}})$ return $(\mathsf{PublicVerifyJudgeComplete}, \mathsf{Yes})$. Otherwise, return $(\mathsf{PublicVerifyJudgeComplete}, \mathsf{No})$

Figure 8: $\mathcal{F}_{\text{CSAM-Scan}}$, an updated version of Apple's CSAM scanning protocol.

(2) A predetermined judge must provide approval before Apple administrators can decrypt the images with perceptual hashes matching images designated as CSAM by an NCMEC administrator.

(3) If a client does not back up sufficiently many images with the same perceptual hash as those designated as CSAM by the NCMEC administrator, then no Apple administrator can see any of the client's photos. This holds even if the Apple administrator colludes with the judge or the judge's secret state is compromised.

Adding public verifiability. In order for the public to have confidence that the system is operating in a socially responsible way, it is important that the public can interrogate some of its properties. As such, we want to augment the system to provide *public verifiability*:

(4) It is possible for any member of the public to verify

  (i) that the system is scanning for the same perceptual hashes for all clients;

  (ii) that the perceptual hashes for which Apple is scanning are those selected by NCMEC.

(5) Claims that a client backed up CSAM to the photostore should be publicly verifiable—implying that clients cannot be framed. In other words, it should be intractable for anyone to produce evidence that a client backed up an image that they did not.

The latter of these properties is particularly important if evidence collected from this system could be used to convict and punish individuals who are holding CSAM images. The public verifiability of this property means that, for example, the evidence introduced as part of a prosecution could be verified by anyone—even those suspicious of the judicial system.

5.2 Step 0: Reworking the Ideal Functionality

We begin by reworking the ideal functionality, with these sociotechnical goals in mind. In practice, our team iteratively sketched and refined this ideal functionality, taking into consideration both what we thought was feasible to realize and the ways in which it would interact with $\Pi_{\text{Intended-use}}$. Thus, we did not cleanly execute Step 0 followed by Step 1; rather Steps 0 and 1 formed a feedback loop. What we present below is the final result of that process.

The main goal of our new ideal functionality is to include more of the overall system within the ideal functionality's boundaries. We give a brief overview of the new ideal functionality below; the full updated ideal functionality $\mathcal{F}_{\text{CSAM-Scan}}$ can be found in Figure 8.

- **NCMEC provides input directly to $\mathcal{F}_{\text{CSAM-Scan}}$:** We modify Flow 1 from the initial proposal such that NCMEC provides the perceptual hashes directly into the ideal functionality $\mathcal{F}_{\text{CSAM-Scan}}$ through a new interface. By having NCMEC interact with $\mathcal{F}_{\text{CSAM-Scan}}$, they join the cryptographic part of the protocol and $\mathcal{F}_{\text{CSAM-Scan}}$ can reason about NCMEC's decisions directly. We also change the information passed between NCMEC and Apple to be random oracle hashes of the input set, which destroys any structure about the image contained in the perceptual hash function's output.

- **InitScan will only process matching inputs:** Next, we modify $\mathcal{F}_{\text{CSAM-Scan}}$ such that Apple can only initiate the scanning process using exactly the set of perceptual hashes supplied by NCMEC above. If Apple provides a set of perceptual hashes that do not match those provided by NCMEC, $\mathcal{F}_{\text{CSAM-Scan}}$ will simply ignore the input—meaning that clients will not be properly initialized. Looking ahead, this will be implemented by having Apple prove using standard commit-and-prove zero-knowledge that public parameters are consistent with NCMEC's committed perceptual hashes.

- **Judge interface to $\mathcal{F}_{\text{CSAM-Scan}}$:** To integrate the judge into the system, we open a new interface in $\mathcal{F}_{\text{CSAM-Scan}}$. This interface gives the judge the access to the output of the private set intersection protocol (rather than giving it directly to Apple). The judge can then send this information to Apple

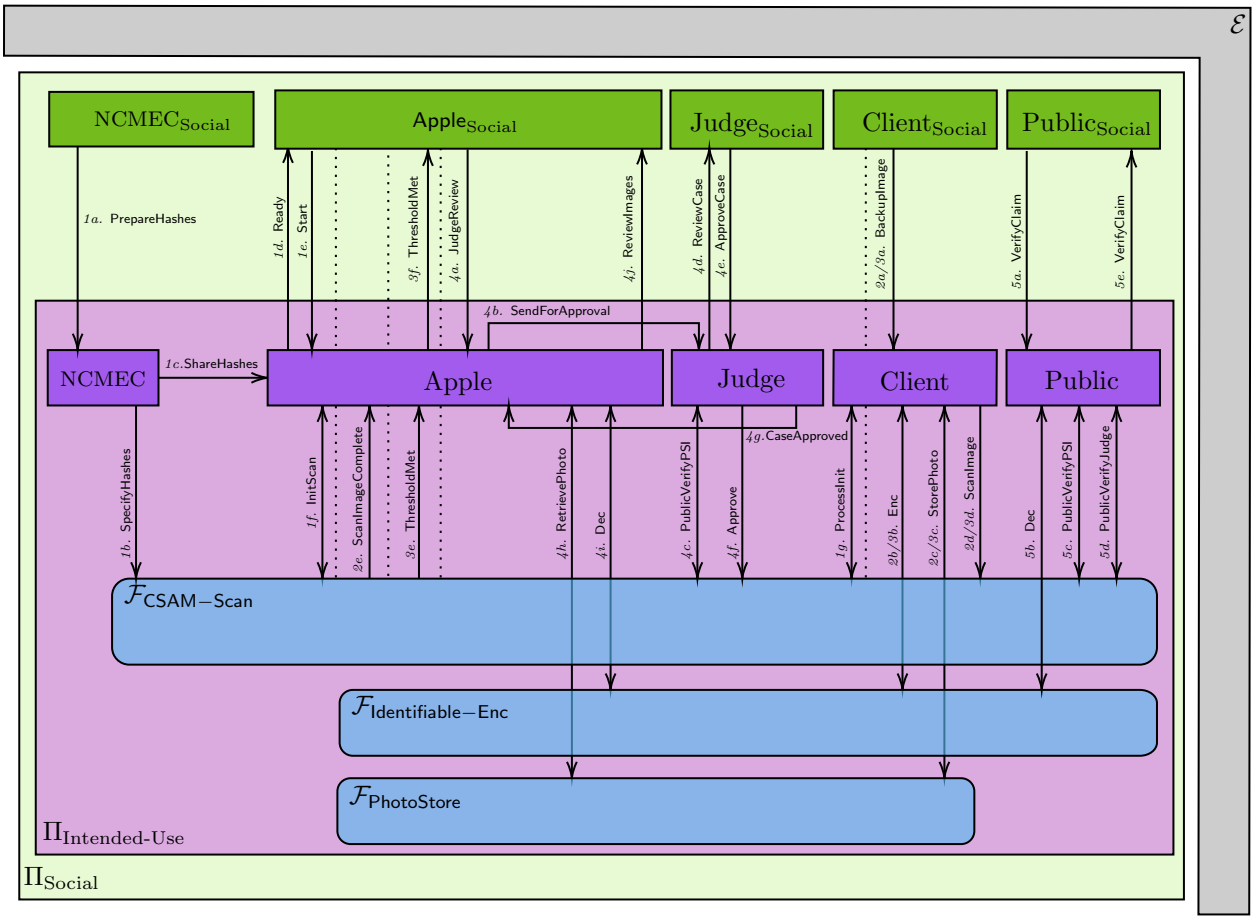

Figure 9: An overview of the rendering of Apple's proposed protocol into our framework. We suggest reading the data flows in the labeled order, which generally move left to right. For a more complete walk-through of this figure, see Section 5.3.

within $\Pi_{\text{Intended-use}}$ and Apple can subsequently decrypt the images. To manage this indirection, $\mathcal{F}_{\text{CSAM-Scan}}$ outputs a "judge approval token" $\tau_{\mathcal{U}}$ rather than keys directly, and the Judge can recover the output of the set intersection by sending this approval token into the ideal functionality.

– **Public verification interfaces:** Finally, we add two verification interfaces to $\mathcal{F}_{\text{CSAM-Scan}}$ that can be called by anyone. These collectively empower the public to verify that a claimed set of keys were really generated by an interaction between the client and Apple's server. Implementing these interfaces requires changes to the underlying private set-intersection protocol. Specifically, the public must be able to verify that the transcripts of interactions are real, which is accomplished using signatures.

Realizing $\mathcal{F}_{\text{CSAM-Scan}}$. We now give an overview to the ways in which Apple could have changed its initial proposal such that it could realize the updated ideal functionality. Formal descriptions of these protocol changes are deferred to Appendix B, as the details of this protocol do not help us study or evaluate our framework.

– **Publicly posting commitment to X on $\mathcal{F}_{BB}$.** An obvious approach to including NCMEC in the protocol is to simply have NCMEC generate the public parameters pdata of the private set intersection

protocol independently. While tempting, this clearly breaks the way that the protocol "should" work socially—NCMEC is responsible for curating sets of hashes and therefore should not be involved in the scanning protocol itself. Moreover, this protocol could, in principle, be run by many different image hosting providers, so it makes sense for NCMEC's role to be something that need only be run once, independently of the number of service providers. Thus, we take a different approach: in order to ensure that there is a global set of hashes that will be used for all clients, NCMEC begins the protocol by posting a signed cryptographic commitment to X to the bulletin board functionality $\mathcal{F}_{BB}$. The key properties of the bulletin board are that all posts there are immutable and available to any member of the public. Concretely, NCMEC actually does this by passing each input hash $\mathsf{img_hash} \in X$ through the random oracle, and committing to the output. We do this both for social reasons and technical reasons. On the social side, we do not know how much information about the images $\mathsf{img_hash}$ holds, and thus it might be valuable to destroy any remaining structure before passing the results to non-NCMEC parties. Technically, we also need to be careful where the random oracle is evaluated such that the zero-knowledge proofs discussed next are instantiable (i.e., do not reason over the random oracle evaluations).

– **Zero-knowledge proofs of correctness for pdata.** We propose to prevent abuse by the Apple administrator in the *simplest* way possible: using generic zero-knowledge.[14] In more detail, when generating pdata, Apple also generates a zero-knowledge proof of consistency between pdata and the commitment posted to $\mathcal{F}_{BB}$. This is slightly non-trivial, as the algorithm for generating $\mathsf{pdata} = (L, p_1, p_2, \ldots p_{n'})$ requires sampling random elements to fill the empty entries in the cuckoo table (see the Apple Initializing Scanning section of Figure 14 in Appendix A). This can undermine security, by allowing Apple to insert arbitrarily chosen perceptual hashes by controlling the randomness. As such, the generation of pdata must be de-randomized, which can be done by ensuring that the exponents among $p_1, p_2, \ldots p_{n'}$ lie along a degree n polynomial. When clients process pdata, they verify this proof, ensuring that the embedded hash set is exactly the one chosen by NCMEC. Notice that the zero-knowledge proof cannot reason over evaluations of the random oracle, so we must instead change the modeling of the hash functions used within the protocol to be cryptographic hash functions that have a circuit description.

– **Key encapsulation under judge's key.** Intuitively, we integrate the judge into the process of recovering the output of the PSI protocol by encrypting the contents of each voucher under the judge's public key. The one subtlety here is that the server should be able to check to see if the threshold for decryption has been met *without* the judge's help in order to reduce the workload burden on the judicial system—checking to see if the threshold is met is a very common task whereas very few clients will pass the threshold. As such, the client first encrypts their inputs under the judge's key and then processes that ciphertext through the voucher generation process.

– **Public verifiability.** Adding public verifiability for the protocol is straightforward. Namely, messages within the protocol must be appropriately signed and proofs of proper decryption must be created. Specifically, vouchers are signed by the clients, the server creates a zero-knowledge proof of the voucher processing steps, and the judge proves that they decrypted the key encapsulation properly in zero-knowledge. We note that we also require that the image encryption be properly key-committing, which we did in the initial modeling of $\mathcal{F}_{\mathsf{Enc}}$.

[14] Since the initial development of the ideas in this work (see [CK21]), the research community has made significant strides on ensuring consistency between PSI inputs and other protocol sub-components [SKM23b, SLY+25, GMPS25, BGJP25]. Our decision to use the "simple" approach of generic zero-knowledge, rather than fine-tune a protocol, is inspired by the observation that Apple likely had the computation resources available to them to simply use generic zero-knowledge without needing any other techniques. In other words, Apple was in the position to spend time and money (in compute time), rather than pay cryptographers to design a protocol. Clearly, now that protocols exist that have similar properties, the value of using generic zero-knowledge may no longer out-weigh the computational costs. Nevertheless, optimizing the details of this approach is not the focus on our work, so we proceed with this more simple approach.

$\mathcal{F}^{\mathcal{D}}_{\mathsf{Identifiable-Enc}}$

Upon initialization, obtain a ciphertext generation algorithm G from $\mathcal{S}$.

1. Upon receiving $(\mathsf{Enc}, \mathsf{id}, m)$ from $\mathcal{U}$, generate a ciphertext $\mathsf{ctx} \leftarrow G(\mathsf{id})$. Then,
   (a) if $\mathcal{U}$ is corrupt, send $(\mathsf{RequestKey}, \mathsf{id})$ to $\mathcal{S}$. If $(\mathsf{SetKey}, \mathsf{id}, \mathsf{k})$ is received in response, then record $(\mathsf{id}, \mathsf{ctx}, \mathsf{k}, m, \mathcal{U})$ and output $(\mathsf{EncComplete}, \mathsf{id}, \mathsf{ctx}, \mathsf{k})$ to $\mathcal{U}$.
   (b) if $\mathcal{U}$ is not corrupt, set $\mathsf{k} \xleftarrow{\$} \mathcal{D}$, record $(\mathsf{id}, \mathsf{ctx}, \mathsf{k}, m, \mathcal{U})$ and output $(\mathsf{EncComplete}, \mathsf{id}, \mathsf{ctx}, \mathsf{k})$ to $\mathcal{U}$.
2. Upon receiving $(\mathsf{Dec}, \mathsf{id}, \mathsf{ctx}, \mathsf{k})$ from anyone, check if there exists a record with the values $(\mathsf{id}, \mathsf{ctx}, \mathsf{k}, m, \cdot)$. If such a tuple exists, return $(\mathsf{DecComplete}, \mathsf{id}, m)$. Else, return $(\mathsf{DecComplete}, \mathsf{id}, \perp)$.
3. Upon receiving $(\mathsf{Dec}, \mathsf{id}, \mathsf{ctx}, \mathsf{k}, \mathcal{U})$ from anyone, check if there exists a record with the values $(\mathsf{id}, \mathsf{ctx}, \mathsf{k}, m, \mathcal{U})$. If such a tuple exists, return $(\mathsf{DecComplete}, \mathsf{id}, m, \mathcal{U})$. Else, return $(\mathsf{DecComplete}, \mathsf{id}, \perp)$.

Figure 10: Identifiable, key-committing encryption ideal functionality

5.3 Step 1: Extending the Modeling

We can now continue to Step 1, in which we render the ideal functionality $\mathcal{F}_{\text{CSAM-Scan}}$ into its broader context by specifying $\Pi_{\text{Intended-use}}$ (see Figure 11) and Π_{Social} (see Figure 12). As we did in Section 5.2, we present the results of the iterative refinement process. We visualize the composition of $\Pi_{\text{Intended-use}}$ and Π_{Social} in Figure 9.

To start, we must modify the encryption functionality used to encrypt the images that will be stored in $\mathcal{F}_{\mathsf{PhotoStore}}$. The basic, key-committing encryption provided by $\mathcal{F}_{\mathsf{Enc}}$ does nothing to prove that a ciphertext was created by a client $\mathcal{U}$. Specifically, the Auth associated with $\mathcal{F}_{\mathsf{PhotoStore}}$ can upload ciphertexts on behalf of $\mathcal{U}$! Thus, we need some way to ensure that each ciphertext is tightly bound to the identity of the party who encrypts. This is trivially realizable by having the client sign each ciphertext using a digital signature. We capture this change in functionality by swapping $\mathcal{F}_{\mathsf{Enc}}$ with $\mathcal{F}_{\mathsf{Identifiable-Enc}}$, which can be found in Figure 10.

Flow 1: NCMEC selects the images and Apple initializes scanning. Now that we have exposed an interface in $\mathcal{F}_{\text{CSAM-Scan}}$ for NCMEC, $\Pi_{\text{Intended-use}}$ begins with NCMEC specifying the hashes directly. As before, the first step (1a) is to have an administrator at NCMEC select a set of images which should be included in the scanning mechanism. In a departure from Apple's initial proposal, NCMEC can then directly send these hashes to $\mathcal{F}_{\text{CSAM-Scan}}$ (1b) before sharing the hashes with Apple (1c). This, in turn, informs the Apple administrator that the system is ready for scanning (1d) and the Apple administrator can start the scanning process (1e) and (1f). Finally, client devices process the initialization (1g) and are ready to begin sending messages.

Flow 2: Client backing-up image. This flow works the same as Apple's initial proposal. Clients decide to back up an image (2a), encrypt that image with $\mathcal{F}_{\mathsf{Identifiable-Enc}}$ (2b), and send the resulting ciphertext to $\mathcal{F}_{\mathsf{PhotoStore}}$ (2c). The client then sends the key material and perceptual hash to $\mathcal{F}_{\text{CSAM-Scan}}$ (2d), which notifies Apple's system (2e).

Flow 3: Client backing-up image that triggers threshold. This flow starts exactly as Flow 2 (3a-3d), but once the threshold has been met, $\mathcal{F}_{\text{CSAM-Scan}}$ notifies Apple to that effect (3e), which in turn notifies the Apple administrator that a client has met the threshold (3f).

Flow 4: Requesting approval from the Judge. In order to access a client's images, the Apple administrator asks the Judge for approval (4a and 4b). The Judge's software then verifies that the information provided by Apple is correct (4c) before handing the case off to the Judge themselves for manual review (4d). If the Judge approves, they activate their software (4e), which relays approval to $\mathcal{F}_{\text{CSAM-Scan}}$ (4f)

$\Pi_{\text{Intended-use}}$ for $\mathcal{F}_{\text{CSAM-Scan}}$

(The protocol is parameterized by a perceptual hash function PHash*.)*

NCMEC: When NCMEC invoked on the set $\mathcal{I}' = \{\mathsf{img}_1, \mathsf{img}_2, \ldots\}$:

1. Set $X = \{\}$. For $\mathsf{img}_i \in \mathcal{I}'$, compute $\mathsf{img_hash}_i = \mathsf{PHash}(\mathsf{img}_i)$ and set $X = X \cup \{\mathsf{img_hash}_i\}$.
2. Send the message $(\mathsf{SpecifyHashes}, X = \{\mathsf{img_hash}_1, \ldots, \mathsf{img_hash}_n\})$ to $\mathcal{F}_{\text{CSAM-Scan}}$, and receive $(\mathsf{SpecifyHashesComplete}, \mathsf{ROHashes}, \mathsf{d})$.
3. Send $(\mathsf{ShareHashes}, \mathsf{ROHashes}, \mathsf{d})$ to Apple via an instance of $\mathcal{F}_{\mathsf{SMT}}$.

Apple: When Apple receives $(\mathsf{ShareHashes}, \mathsf{ROHashes}, \mathsf{d})$ from NCMEC via $\mathcal{F}_{\mathsf{SMT}}$:

1. Send $(\mathsf{CheckHashes}, \mathsf{ROHashes}, \mathsf{d})$ to $\mathcal{F}_{\text{CSAM-Scan}}$. If $(\mathsf{CheckedHashes}, \mathsf{Yes})$ is not received in response, return.
2. Record $(\mathsf{ROHashes}, \mathsf{d})$, and send $(\mathsf{ReadyToScan})$ to $\mathsf{Apple}_{\mathsf{social}}$

Apple: When Apple receives a message $(\mathsf{InitScan})$ from $\mathsf{Apple}_{\mathsf{social}}$:

1. Retrieve $(\mathsf{ROHashes}, \mathsf{d})$. Let $\{\mathcal{U}_1, \ldots, \mathcal{U}_m\}$ be the set of all user devices. Send $(\mathsf{InitScan}, \{\mathcal{U}_1, \ldots, \mathcal{U}_m\}, \{\mathsf{ROHashes}, \ldots, \mathsf{ROHashes}\}, \mathsf{d})$ to $\mathcal{F}_{\text{CSAM-Scan}}$. Receive $(\mathsf{InitScanComplete})$ in response and return.

Client: Until a message $(\mathsf{ProcessInitComplete}, |X|, h_0, h_1)$ has been received from $\mathcal{F}_{\text{CSAM-Scan}}$, regularly send $(\mathsf{ProcessInit})$ to $\mathcal{F}_{\text{CSAM-Scan}}$.

Client: When a user device $\mathcal{U}$ receives the message $(\mathsf{BackupImage}, \mathsf{img})$ from $\text{Client}_{\text{Social}}$:

1. Sample a unique random value $\mathsf{img_id}$. Send $(\mathsf{Enc}, \mathsf{img_id}, \mathsf{img})$ to $\mathcal{F}_{\mathsf{Identifiable-Enc}}$ and receive the output $(\mathsf{EncComplete}, \mathsf{img_id}, \mathsf{ctx}, \mathsf{k})$.
2. Send $(\mathsf{StorePhoto}, \mathcal{U}, \mathsf{img_id}, \mathsf{ctx})$ to $\mathcal{F}_{\mathsf{PhotoStore}}$ and receive $(\mathsf{StorePhotoComplete})$ in response.
3. Compute $\mathsf{img_hash} = \mathsf{PHash}(\mathsf{img})$ and send $(\mathsf{ScanImage}, \mathsf{img_id}, \mathsf{img_hash}, \mathsf{k})$ to $\mathcal{F}_{\text{CSAM-Scan}}$.

Apple: When Apple receives $(\mathsf{ScanImageComplete}, \mathcal{U}, \mathsf{img_id}, b)$ from $\mathcal{F}_{\text{CSAM-Scan}}$, return control flow to Env.
Apple: When Apple receives $(\mathsf{ThresholdMet}, \mathcal{U}, \mathsf{img_id}, \pi_{\mathcal{U}}, \tau_{\mathcal{U}})$ from $\mathcal{F}_{\text{CSAM-Scan}}$:

1. Record $(\mathcal{U}, \pi_{\mathcal{U}}, \tau_{\mathcal{U}})$ and send $(\mathsf{Match}, \mathcal{U})$ to Apple

Apple: When Apple receives $(\mathsf{JudgeReview}, \mathcal{U})$ from $\mathsf{Apple}_{\text{social}}$:

1. Retrieve $(\mathcal{U}, \pi_{\mathcal{U}}, \tau_{\mathcal{U}})$ and send $(\mathsf{SendForApproval}, \mathcal{U}, \pi_{\mathcal{U}}, \tau_{\mathcal{U}})$ to $\mathcal{J}$ via an instance of $\mathcal{F}_{\mathsf{SMT}}$.

Judge: When $\mathcal{J}$ receives $(\mathsf{SendForApproval}, \mathcal{U}, \pi_{\mathcal{U}}, \tau_{\mathcal{U}})$ from Apple via $\mathcal{F}_{\mathsf{SMT}}$:

1. Send $(\mathsf{PublicVerifyPSI}, \mathcal{U}, \pi_{\mathcal{U}}, \tau_{\mathcal{U}})$ To $\mathcal{F}_{\text{CSAM-Scan}}$. If $(\mathsf{PublicVerifyPSIComplete}, \mathsf{No})$ is received in response, stop and return. Otherwise, record $(\mathcal{U}, \pi_{\mathcal{U}}, \tau_{\mathcal{U}})$ and send $(\mathsf{ReviewCase}, \mathcal{U})$ to $\mathcal{J}_{\mathsf{social}}$.

Judge: When $\mathcal{J}$ receives $(\mathsf{ApproveCase}, \mathcal{U})$ from $\mathcal{J}_{\mathsf{social}}$:

1. Retrieve a record $(\mathcal{U}, \pi_{\mathcal{U}}, \tau_{\mathcal{U}})$. Send $(\mathsf{Approve}, \mathcal{U}, \tau_{\mathcal{U}})$ to $\mathcal{F}_{\text{CSAM-Scan}}$. If $(\mathsf{ApproveComplete}, \mathcal{U}, \mathsf{img_id_list}, \mathsf{keys}, \pi_{\mathcal{U}}^{\mathcal{J}})$ is received in response, send $(\mathsf{CaseApproved}, \mathcal{U}, \tau_{\mathcal{U}}, \mathsf{img_id_list}_{\mathcal{U}}, \mathsf{keys}_{\mathcal{U}}, \pi_{\mathcal{U}}^{\mathcal{J}})$ to Apple via $\mathcal{F}_{\mathsf{SMT}}$.

Apple: When Apple receives $(\mathsf{CaseApproved}, \mathcal{U}, \tau_{\mathcal{U}}, \mathsf{img_id_list}_{\mathcal{U}}, \mathsf{keys}_{\mathcal{U}}, \pi_{\mathcal{U}}^{\mathcal{J}})$ from $\mathcal{J}$ via $\mathcal{F}_{\mathsf{SMT}}$:

1. Send $(\mathsf{PublicVerifyJudge}, \mathcal{U}, \tau_{\mathcal{U}}, \mathsf{img_id_list}_{\mathcal{U}}, \mathsf{keys}_{\mathcal{U}}, \pi_{\mathcal{U}}^{\mathcal{J}})$ to $\mathcal{F}_{\text{CSAM-Scan}}$. If $(\mathsf{PublicVerifyJudgeComplete}, \mathsf{No})$ is received in response, stop and return.
2. Initialize the empty set $\mathsf{decrypted_photos}_{\mathcal{U}}$.
3. For $i \in [|\mathsf{img_id_list}_{\mathcal{U}}|]$, send $(\mathsf{RetrievePhoto}, \mathcal{U}, \mathsf{img_id}_i)$ to $\mathcal{F}_{\mathsf{PhotoStore}}$. If $(\mathsf{RetrievePhotoComplete}, \mathsf{img_id}_i, \perp)$ is received in response, continue. Otherwise, if $(\mathsf{RetrievePhotoComplete}, \mathsf{img_id}_i, \mathsf{ctx}_i, \mathsf{users}_i)$ is received:
   (a) Send $(\mathsf{Dec}, \mathsf{img_id}_i, \mathsf{ctx}_i, \mathsf{k}_i)$ to $\mathcal{F}_{\mathsf{Identifiable-Enc}}$. If $(\mathsf{DecComplete}, \mathsf{img_id}_i, \perp)$ is received in response, continue. Otherwise, if $(\mathsf{DecComplete}, \mathsf{img_id}_i, \mathsf{img}_i)$ is received in response, add img_i to $\mathsf{decrypted_photos}_{\mathcal{U}}$.
4. Send $(\mathsf{ReviewImages}, \mathcal{U}, \mathsf{decrypted_photos})$ to $\mathsf{Apple}_{\mathsf{social}}$.

Public: When any party receives $(\mathsf{VerifyClaim}, \mathcal{U}, \{(\mathsf{img_id}_1, \mathsf{img}_1), \ldots, (\mathsf{img_id}_{\mathcal{T}}, \mathsf{img}_{\mathcal{T}})\}, \mathsf{evidence})$ from Env:

1. Parse $\mathsf{evidence} = (\{(\mathsf{k}_1, c_1), \ldots, (\mathsf{k}_{\mathcal{T}}, c_{\mathcal{T}})\}, \pi_{\mathcal{U}}, \tau_{\mathcal{U}}, \pi_{\mathcal{U}}^{\mathcal{J}})$. Initialize $\mathsf{Verified} = \mathsf{Yes}$.
2. For $i \in [\mathcal{T}]$, send $(\mathsf{Dec}, \mathsf{img_id}_i, c_i, \mathsf{k}_i, \mathcal{U})$ to $\mathcal{F}_{\mathsf{Identifiable-Enc}}$.
   If any of the responses are $(\mathsf{DecComplete}, \mathsf{img_id}_i, \perp)$, set $\mathsf{Verified} = \mathsf{No}$.
3. Send $(\mathsf{PublicVerifyPSI}, \mathcal{U}, \pi_{\mathcal{U}}, \tau_{\mathcal{U}})$ to $\mathcal{F}_{\text{CSAM-Scan}}$. If $(\mathsf{PublicVerifyPSIComplete}, \mathsf{No})$ received in response, set $\mathsf{Verified} = \mathsf{No}$.
4. Send $(\mathsf{PublicVerifyJudge}, \mathcal{U}, \tau_{\mathcal{U}}, \{\mathsf{img_id}_1, \ldots, \mathsf{img_id}_{\mathcal{T}}\}, \{\mathsf{k}_1, \ldots, \mathsf{k}_{\mathcal{T}}\}, \pi_{\mathcal{U}}^{\mathcal{J}})$ to $\mathcal{F}_{\text{CSAM-Scan}}$.
   If $(\mathsf{PublicVerifyJudgeComplete}, \mathsf{No})$ received in response, set $\mathsf{Verified} = \mathsf{No}$.
5. Send $(\mathsf{VerifyClaim}, \mathcal{U}, \mathsf{Verified})$ to caller.

Figure 11: $\Pi_{\text{intended-use}}$, capturing software components of the system with properties that will not be captured by the cryptographic proof.

and to Apple (4g). Once approval for the case has been received, Apple can run the same image recovery procedure: retrieving the ciphertexts (4h), decrypting the images (4i), and returning the plaintext images for manual review to Apple (4j).

Flows 5: Verifying image provenance. Finally, Flow 5 allows anyone with the proper evidence to verify that Apple recovered a set of images through the system. Specifically, this flow allows checking that a set of images were initially encrypted by a particular client and the key material to decrypt those images was accessed through the scanning protocol. We model this process as any party in Π_{Social} sending some images and evidence into $\Pi_{\text{Intended-use}}$ (5a). To verify this evidence, queries are made to $\mathcal{F}_{\mathsf{Identifiable-Enc}}$ (5b) and $\mathcal{F}_{\text{CSAM-Scan}}$ (5c and 5d). The result of this confirmation process is then sent back (5e).

5.4 Step 3: Examining Sociotechnical Properties

In Section 5.1, we stated security properties that we wanted the system to provide. Having now specified the system, we can evaluate the extent to which we were successful.

(1) **Apple cannot scan client's devices for any perceptual hashes not designated as CSAM by NCMEC.** Notice that this property is directly enforced by $\mathcal{F}_{\text{CSAM-Scan}}$. Namely, the hashes that serve as Apple's input to the private set intersection protocol (1f) must match exactly those specified by NCMEC (1b).

(2) **A predetermined judge must provide approval before Apple administrators can decrypt the images with perceptual hashes matching images designated as CSAM by an NCMEC administrator.** There are several cases to consider: (1) if both the Judge and Apple are corrupted by the same adversary, then approval is always given implicitly, making this property trivially satisfied; (2) if the Judge is honest, then $\mathcal{F}_{\text{CSAM-Scan}}$ will only release the images *after* approval is provided by the Judge.

(3) **If a client does not back up sufficiently many images with the same perceptual hash as those designated as CSAM by the NCMEC administrator, then no Apple administrator can see any of the client's photos. This holds even if the Apple administrator colludes with the judge or the judge's secret state is compromised.** Notice that the Judge can only provide approval *after* the threshold has been met. This is enforced by the requirement that the judge must provide the appropriate $\tau_{\mathcal{U}}$ in order to approve, and this value is not sampled before the threshold is met. Importantly this holds even when both the Judge and Apple are corrupted.

(4) **It is possible for any member of the public to verify**

   (i) **that the system is scanning for the same perceptual hashes for all clients;** This is implicitly guaranteed by the fact that there is no self composition within this system—the entire functionality is provided by $\mathcal{F}_{\text{CSAM-Scan}}$ itself. Moreover, $\mathcal{F}_{\text{CSAM-Scan}}$ enforces that all devices are scanned for exactly the hashes provided by NCMEC (which cannot be updated later).

   (ii) **that the perceptual hashes for which Apple is scanning are those selected by NCMEC.** This property is provided for the same reasons as 4i.

(5) **Claims that a client backed up CSAM to the photostore should be publicly verifiable—implying that clients cannot be framed. In other words, it should be intractable for anyone to produce evidence that a client backed up an image that they did not.** $\mathcal{F}_{\text{CSAM-Scan}}$ provides interfaces for public verification that, together, provide this property. Namely, a member of the public can verify that a plaintext image img_1 corresponds to a ciphertext c by querying the Dec interface of $\mathcal{F}_{\mathsf{Identifiable-Enc}}$ with the appropriate key k. Moreover, $\pi_{\mathcal{U}}, \tau_{\mathcal{U}}$, and $\pi^{\mathcal{J}}$ can certify that k was truly released by $\mathcal{F}_{\text{CSAM-Scan}}$, as any member of the public can call the $\mathsf{PublicVerifyPSI}$ and $\mathsf{PublicVerifyJudge}$ interfaces.

Thus, we claim that the updated system provides the properties specified in Section 5.1.

Π_{Social} for $\mathcal{F}_{\text{CSAM-Scan}}$

$\text{NCMEC}_{\text{Social}}$: When $\text{NCMEC}_{\text{Social}}$ receives $\mathcal{I} = \{\mathsf{img}_1, \mathsf{img}_2, \ldots\}$ from Env:

(a) Select a subset of images $\mathcal{I}' \subseteq \mathcal{I}$ that are CSAM, then invoke $\Pi_{\text{intended-use}}$ with the message $(\mathsf{PrepareHashes}, \mathcal{I}')$.

$\text{Apple}_{\text{Social}}$: When $\text{Apple}_{\text{Social}}$ receives $(\mathsf{ReadyToScan})$ from Apple:

(a) Return control flow to the environment.

$\text{Apple}_{\text{Social}}$: When $\text{Apple}_{\text{Social}}$ receives $(\mathsf{InitScan})$ from Env:

(a) Invoke $\Pi_{\text{intended-use}}$ with the message $(\mathsf{InitScan})$.

$\text{Client}_{\text{Social}}$: When a user $\text{Client}_{\text{Social}}$ receives a message $(\mathsf{ReceiveImage}, \mathsf{img})$ from Env
//Captures taking a photo or receiving an image via another communication channel

(a) If the client wants to back up the image, send $(\mathsf{BackupImage}, \mathsf{img})$ to $\mathcal{U}$. Otherwise, return control flow to the environment.

$\text{Apple}_{\text{Social}}$: When $\text{Apple}_{\text{Social}}$ receives $(\mathsf{Match}, \mathcal{U})$ from Apple:

(a) When prepared, send $(\mathsf{JudgeReview}, \mathcal{U})$ to Apple.

$\text{Judge}_{\text{Social}}$: When $\text{Judge}_{\text{Social}}$ receives $(\mathsf{ReviewCase}, \mathcal{U})$ from $\mathcal{J}$:

(a) When prepared, send $(\mathsf{ApproveCase}, \mathcal{U})$ to $\mathcal{J}$

$\text{Apple}_{\text{Social}}$: When $\text{Apple}_{\text{Social}}$ receives $(\mathsf{ReviewImages}, \mathcal{U}, I = \{\mathsf{img}_1, \mathsf{img}_2, \ldots\})$ from Apple:

(a) Review the images I and determine if they are CSAM.

(b) If any images are not CSAM, report the existence of a design flaw to all other parties.

(c) If $|I| < \mathcal{T}$, report that either there is a design flaw or $\mathcal{U}$ is corrupt to all other parties.

(d) Send the images to NCMEC or law enforcement as necessary to comply with existing laws.

(e) Output I to Env.

$\text{Public}_{\text{Social}}$: When any party receives $(\mathsf{VerifyClaim}, \mathcal{U}, \{(\mathsf{img_id}_1, \mathsf{img}_1), \ldots, (\mathsf{img_id}_{\mathcal{T}}, \mathsf{img}_{\mathcal{T}})\}, \mathsf{evidence})$ from Env:

(a) Send $(\mathsf{VerifyClaim}, \mathcal{U}, \{(\mathsf{img_id}_1, \mathsf{img}_1), \ldots, (\mathsf{img_id}_{\mathcal{T}}, \mathsf{img}_{\mathcal{T}})\}, \mathsf{evidence})$ to their equivalent party in $\Pi_{\text{Intended-use}}$.

(b) When $(\mathsf{VerifyClaim}, \mathcal{U}, \mathsf{Verified})$ is received in response, return it to the caller.

Figure 12: Π_{social}, capturing human decisions that drive the protocol.

6 Discussion

The fallout following Apple's CSAM scanning proposal exposed severe *process* limitations in the ways that cryptographers analyze their protocol proposals and document that analysis. Namely, the existing norm in cryptography is that all protocol proposals must be accompanied by a *proof of security* that shows that the protocol provides some defined set of properties. However, Apple's proposal is a clear indication that this is not enough; if the documentation stops at the proof of security, as is often the case, the analysis falls short of documenting the sociotechnical properties the system has—let alone making a compelling argument that the set of proven properties are the *right* ones. Without expanding the analytical frame included in the proposal, it risks giving the reader the impression that the authors are unaware of system limitations (or intentionally trying to hide these flaws).[15] Moreover, the cryptographic protocol might only be one sub-component of the larger system and the documented analysis may make it difficult to understand the role the proven properties play in the fully composed system.

We take a first step towards building a cryptography-native approach to illustrating sociotechnical properties of cryptographic systems. We see four significant benefits to making the use of our framework a communal *expectation*:

(1) Being very transparent helps ensure that there is never the appearance that protocol designers are trying to "hide" anything. Especially when there are real trade-offs associated with deployment, being clear about the costs and benefits is important.

[15] We emphasize that we do not believe that this was the case with Apple's proposal. But, the opportunity to miss or suppress important details is a real hazard as increasing numbers of complex cryptographic systems are proposed for deployment.

(2) Agreeing upon a specific process helps ensure protocol designers do not take short cuts when thinking through their proposals. This is similar to the way *writing* a formal security proof can help highlight details missed during the protocol design process—in addition to being a compelling artifact for external reviewers of the protocol;

(3) Taking responsibility for establishing sociotechnical properties helps ensure that the marketing of cryptographic deployments stays accurate. If cryptographers do not do this work, someone less qualified *will* do it—and will get it wrong.

(4) When proposals prove to be controversial, the artifact created by conducting this analysis can serve as a starting point for debating ethics. Importantly, this helps differentiate between disagreement about the sociotechnical properties a system has and whether those properties are "good enough" for deployment.

Post-hoc vs. proactive use of our framework. In the main body of our paper, we study applying our framework to an existing proposal in a post-hoc way, when a set of sociotechnical properties (i.e., critiques) were already well known. While this gives us confidence that our framework does not miss important properties, our prior exposure to these known sociotechnical properties might have made them easier to identify. On the other hand, the fact that our framework helped us identify a new property of Apple's proposal, despite years of community attention, indicates our framework represents progress. In the weeks following Apple's announcement, the research community quickly iterated on how best to understand and contextualize the proposal. This was a process that included careful reading of the protocol description and proof of security, and an extended, decentralized debate over the validity of the implicit threat model captured by the analysis. There was, however, no shared language to which the community had access during this discussion. Namely, weaknesses in the protocol, the formalisms, and discussions of geo-politics were interleaved into a single discussion that was difficult to follow. Our framework would have provided an opportunity to bind these conversational threads together cleanly as the community worked together to conduct a version of the analysis we did in Section 4.

Our case study also shows that applying our framework *post-hoc* is a difficult process: we had to rework the formal modeling of the system, make low-level protocol modifications to accommodate these modeling changes, and apply a pedantic meticulousness to sociotechnical statements that were never intended to be subjected to that level of attention. Much of this overhead was, no doubt, valuable. For instance, the process of modifying the formalisms helped highlight limitations of the prior formalisms (e.g., that h_0, h_1 were not treated as leakage), other modifications made the system's functionality more readable to those with less expertise in cryptography, and composing the scanning system with the data storage system revealed new, previously unobserved leakage patterns. Nonetheless, the overhead of applying our framework is non-trivial, leading us to believe that *post-hoc* application of our framework shows the most promise when proposals are *controversial*. Controversial proposals often spark lively and multifaceted debate, which is rarely systematic; using our framework would help quickly on-board interested members of the community as they joined the debate and minimize the need for repeating work.

In Section 5, we explore an alternative approach: using our framework proactively within the protocol design process.This pushes for making sociotechnical goals explicit at the beginning of the process and then iteratively revisiting those goals throughout protocol refinement.Specifically, we flip the order of the steps in the following way: (1) enumerate the sociotechnical properties that the system should have; (2) design an ideal functionality and associated, realizing protocol; (3) extend the modeling to include $\Pi_{\text{Intended-use}}$ and Π_{Social}; and (4) interrogate the extent to which the design meets the desired sociotechnical goals. We study sociotechnical properties like *"Apple cannot scan client's devices for any perceptual hashes not designated as CSAM by NCMEC," "It is possible for any member of the public to verify that the system is scanning for the same perceptual hashes for all clients,"* and *"Claims that a client backed up CSAM to the photostore should be publicly verifiable—implying that clients cannot be framed."* We imagine this proactive approach might be particularly valuable when working with interdisciplinary teams in order to design cryptographic systems in response to community needs.

Towards ethics and cross-disciplinary conversations. As discussed in Section 1, we see establishing sociotechnical properties and arguing ethics as two conceptually distinct tasks, the first of which is a

precondition for the second. Our work is aimed at supporting the former process, at least in part because we believe that arguing the ethics of deployment cannot be the responsibility of cryptographers alone. Input from experts with other disciplinary expertise (e.g., lawyers, policymakers, ethicists, etc.) is critical. Indeed, transitioning advanced cryptographic primitives to practice implicates the privacy of real people. Responsible deployment, therefore, demands (1) re-examining the legitimacy of the threat models used to initially motivate protocol designs, (2) carefully considering the social properties the cryptographic system will have when composed with non-cryptographic components, and (3) studying the ethical ramifications of deployment. Our framework is targeted specifically at step (2), but steps (1) and (3) are no less important. When it comes to ethics in particular, there have been recent steps to import long-standing ethical dilemmas into the computer security context [KAL23].

Eventually, we hope the community comes to consensus on a process by which sociotechnical properties of cryptosystems are established. It is important that this process produces artifacts that are valuable for this interdisciplinary conversation and keep it technically accurate. This type of artifact—one that can cross disciplinary boundaries without losing fidelity—is called a "boundary object" in sociology [SG89]. We observe that the properties produced by our framework have the potential to be high-quality boundary objects, as they are expressed in natural language and are exactly the types of properties that would be interesting to lawyers or policymakers. In principle, this could also provide an avenue for these communities to provide input, feedback, or set goals for cryptographic systems.

Value as a tool for cryptographic innovation. We also see a more subtle benefit of our framework when used as an ideation tool by cryptographers. When using our framework, there becomes a clear division between the components of a system captured by ideal functionalities $\mathcal{F}_1, \ldots$ and those relegated to the composing protocol $\Pi_{\text{Intended-use}}$. This division highlights opportunities for new cryptographic work by attempting to shift *more* of the system logic into the ideal functionalities. Clearly it will not be feasible to push all system functionality into the ideal functionality, but our framework provides a sandbox in which cryptographers can experimentally move system components back and forth. The increasing literature on PSI systems that are *authenticated* or have *committed inputs* [SKM23b, SLY+25, GMPS25, BGJP25] were directly inspired by the public criticisms of Apple's protocol and can be seen as attempting to move more of $\Pi_{\text{Intended-use}}$ into $\mathcal{F}_{\text{Apple-CSAM-Scan}}$. We expect that more widespread use of tools like our framework would, therefore, help to identify other interesting and challenging cryptographic research problems.

On the use of universal composability. Throughout this work, we have leaned heavily on universal composability as the runtime within which we use our framework. As a consequence, advocating for widespread adoption of this type of analysis implicitly appears to be advocating for the use of UC. While there are certainly significant benefits to proving security of a protocol within UC, there are also non-trivial barriers (e.g., the impossibility of UC commitments in the plain model [CF01]). Additionally, mandating that protocols are UC secure may come with reduced efficiency—although this is not always the case. These difficulties might seem to undermine some of the applicability of our framework or hamstring the ways in which cryptographers design protocols.

We see our work as highlighting the need for frameworks of this kind and a first step towards exploring their structure. As future work, we anticipate that there will be a need for new frameworks that allow for similar analyses to be conducted for cryptosystems that leverage other formal proof techniques, be they stand-alone simulation or game based definitions. We chose to work with UC because of the affordances it offers: the use of ideal functionalities as convenient abstractions for cryptographic protocols, composability with other cryptographic protocols for "free," clear ways to interleave the inputs and outputs of ideal functionalities, etc... But, we will eventually need "cryptographic-native" threat modeling techniques that can interoperate with all of the formal techniques for which cryptographers are eager to reach.

7 Conclusion and Future Work

In this work, we identified a gap in the existing expectations for analyzing cryptographic systems which allowed documentation to silently omit important properties of an eventual deployment—not out of malice,

but simply because these properties were out of scope. This observation motivates the need for a new generation of analysis tools that expand the scope of analysis that cryptographers regularly conduct while also feeling native to the cryptographers using them. We re-examined UC security through the lens of the threat modeling literature and showed that it could be repurposed to meet this need. Our case study revealed that our framework is promising, as it both systematically identified known sociotechnical properties and helped us identify a property of the system that had been overlooked. Beyond this *post-hoc* analysis, we also show (in Section 5) how our framework can be integrated directly into the protocol development process.

We hope that our work illustrates the need for more community attention on this issue. The framework we propose is intended to be a *first step* and significantly more community effort will be required before we reach consensus as to the right analysis tool.

We see several next research problems that are particularly important: (1) our work focused on UC security, but it is clear we will need analogous approaches that naturally interoperate with game-based definitions; (2) we stopped short of trying to *prove* that systems have sociotechnical properties. While designing a framework that would enable these proofs is challenging, it does not appear to be impossible. Future work could also consider using formal analysis tools from the programming literature to automate some of this work; and (3) rigorously analyzing the usability of our system (or similar future proposals) would be valuable.

8 Acknowledgments.

The second author was supported by the National Science Foundation under Grant #2209194, #1718135, #1915763 and #1931714. A significant amount of this work was done while the third author was at Boston University. This work was supported by DARPA under Agreement No. HR00112020021. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Government or DARPA. Additionally, the third author was supported by the National Science Foundation under Grant #2030859 to the Computing Research Association for the CIFellows Project.

The authors would like to thank the many people who read early drafts of the work and provided invaluable feedback, including Zoe Ruha Bell, Shaanan Cohney, Rachel Cummings, Shlomi Hod, Palak Jain, Ryan Little, Priyanka Nanayakkara, Daniel Roche, Jayshree Sarathy, and Kris Shrishak. We would like to especially thank Mayank Varia for supporting us through multiple resubmissions of this work and Daniel Votipka for encouraging us to properly contextualize this work within the threat modeling literature. Finally, we would like to thank the organizers of the Cryptographic Applications Workshop 2024, Miro Haller and Matilda Backendal, for providing a venue in which we could present an early version of this work and gather important feedback.

References

[AAB+15] Harold Abelson, Ross Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Whitfield "Whit" Diffie, John Gilmore, Matthew Green, Susan Landau, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, Michael A. Specter, and Daniel J. Weitzner. Keys under doormats. *Commun. ACM*, 58(10):24–26, sep 2015.

[AAB+21] Hal Abelson, Ross Anderson, Steven M. Bellovin, Josh Benaloh, Matt Blaze, Jon Callas, Whitfield Diffie, Susan Landau, Peter G. Neumann, Ronald L. Rivest, Jeffrey I. Schiller, Bruce Schneier, Vanessa Teague, and Carmela Troncoso. Bugs in our pockets: The risks of client-side scanning, 2021.

[ABC19] Ghada Almashaqbeh, Allison Bishop, and Justin Cappos. Abc: A cryptocurrency-focused threat modeling framework. In *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 859–864, 2019.

[ABD+15] David Adrian, Karthikeyan Bhargavan, Zakir Durumeric, Pierrick Gaudry, Matthew Green, J. Alex Halderman, Nadia Heninger, Drew Springall, Emmanuel Thomé, Luke Valenta, Benjamin VanderSloot, Eric Wustrow, Santiago Zanella-Béguelin, and Paul Zimmermann. Imperfect forward secrecy: How Diffie-Hellman fails in practice. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015*, pages 5–17. ACM Press, October 2015.

[ACDJ26] Martin R. Albrecht, Simone Colombo, Benjamin Dowling, and Rikke Bjerg Jensen. Authenticated encryption in the face of key compromise: The case for alert blindness. In *Advances in Cryptology – EUROCRYPT 2026: 45th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Rome, Italy, May 10–14, 2026, Proceedings, Part II*, page 191–221, Berlin, Heidelberg, 2026. Springer-Verlag.

[ADSW03] Christopher Alberts, Audrey Dorofee, James Stevens, and Carol Woody. Introduction to the octave approach. *Pittsburgh, PA, Carnegie Mellon University*, pages 72–74, 2003.

[AEP22] Martin R. Albrecht, Raphael Eikenberg, and Kenneth G. Paterson. Breaking bridgefy, again: Adopting libsignal is not enough. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022*, pages 269–286. USENIX Association, August 2022.

[AHMP23] Martin R. Albrecht, Miro Haller, Lenka Mareková, and Kenneth G. Paterson. Caveat implementor! Key recovery attacks on MEGA. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part V*, volume 14008 of *LNCS*, pages 190–218. Springer, Cham, April 2023.

[AMPS22] Martin R. Albrecht, Lenka Mareková, Kenneth G. Paterson, and Igors Stepanovs. Four attacks and a proof for Telegram. In *2022 IEEE Symposium on Security and Privacy*, pages 87–106. IEEE Computer Society Press, May 2022.

[AP24] Martin R. Albrecht and Kenneth G. Paterson. Analyzing cryptography in the wild: A retrospective. *IEEE Security & Privacy*, 22(6):12–18, 2024.

[App21a] Apple. The apple psi system. `https://www.apple.com/child-safety/pdf/Apple_PSI_System_Security_Protocol_and_Analysis.pdf`, August 2021. Since initial publication, Apple has modified the contents of the webpage. The initial announcement can be view at `https://web.archive.org/web/20210805191006/https://www.apple.com/child-safety/pdf/Apple_PSI_System_Security_Protocol_and_Analysis.pdf`. Accessed on 18 Decemeber, 2023.

[App21b] Apple. Csam detection: Technical summary. https://www.apple.com/child-safety/pdf/CSAM_Detection_Technical_Summary.pdf, August 2021. Since initial publication, Apple has modified the contents of the webpage. The initial announcement can be view at https://web.archive.org/web/20210805191352/https://www.apple.com/child-safety/pdf/CSAM_Detection_Technical_Summary.pdf. Accessed on 18 Decemeber, 2023.

[App21c] Apple. Expanded protections for children. https://www.apple.com/child-safety/, August 2021. Since initial publication, Apple has modified the contents of the webpage. The initial announcement can be view at https://web.archive.org/web/20210805191220/https://www.apple.com/child-safety/. Accessed on 18 Decemeber, 2023.

[App21d] Apple. Expanded protections for children: Frequently asked questions. https://www.apple.com/child-safety/pdf/Expanded_Protections_for_Children_Frequently_Asked_Questions.pdf, Aug 2021.

[App24] Apple. Apple platform security. https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf, December 2024. Apple regularly updates their documentation. As such, a permenant copy of the version we consulted for this work can be found at https://web.archive.org/web/20250705153759/https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf.

[App25] Apple. icloud data security overview. https://support.apple.com/en-us/102651, Feb 2025. Apple regularly updates their documentation. As such, a permenant copy of the version we consulted for this work can be found at https://web.archive.org/web/20250714063502/https://support.apple.com/en-us/102651.

[BCD+19] Jim Baker, Katherine Charlet, Tom Donahue, Ed Felten, Avril Haines, Susan Hennessey, Chris Inglis, Sean Joyce, Susan Landau, Christy Lopez, Alex Macgillivray, Jason Matheny, Tim Maurer, Denis McDonough, Lisa Monaco, Laura Moy, Michelle Richardson, Ronald L. Rivest, Ari Schwartz, Harlan Yu, and Denise Zheng. Moving the encryption policy conversation forward. https://carnegieendowment.org/2019/09/10/moving-encryption-policy-conversation-forward-pub-79573, 2019.

[Bel21a] Mihir Bellare. The apple psi protocol. https://www.apple.com/child-safety/pdf/Technical_Assessment_of_CSAM_Detection_Mihir_Bellare.pdf, July 2021. Since initial publication, Apple has modified the contents of the webpage. The initial announcement can be view at https://web.archive.org/web/20210805192048/https://www.apple.com/child-safety/pdf/Technical_Assessment_of_CSAM_Detection_Mihir_Bellare.pdf. Accessed on 18 Decemeber, 2023.

[Bel21b] Mihir Bellare. A concrete-security analysis of the apple psi protocol. https://www.apple.com/child-safety/pdf/Alternative_Security_Proof_of_Apple_PSI_System_Mihir_Bellare.pdf, July 2021. Since initial publication, Apple has modified the contents of the webpage. The initial announcement can be view at https://web.archive.org/web/20210826041030/https://www.apple.com/child-safety/pdf/Alternative_Security_Proof_of_Apple_PSI_System_Mihir_Bellare.pdf. Accessed on 18 Decemeber, 2023.

[BGJP25] James Bartusek, Sanjam Garg, Abhishek Jain, and Guru-Vamsi Policharla. Laconic PSI on authenticated inputs and applications. Cryptology ePrint Archive, Paper 2025/1108, 2025.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[BHA24] Maher Boughdiri, Mohamed Hkima, and Takoua Abdelattif. A threat modeling approach for blockchain security assessment. In *2024 IEEE/ACS 21st International Conference on Computer Systems and Applications (AICCSA)*, pages 1–6. IEEE, 2024.

[BHP23] Matilda Backendal, Miro Haller, and Kenneth G. Paterson. MEGA: Malleable encryption goes awry. In *2023 IEEE Symposium on Security and Privacy*, pages 146–163. IEEE Computer Society Press, May 2023.

[BJ20] Emad Badawi and Guy-Vincent Jourdan. Cryptocurrencies emerging threats and defensive mechanisms: A systematic literature review. *IEEE Access*, 8:200021–200037, 2020.

[BKR13] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 296–312. Springer, Berlin, Heidelberg, May 2013.

[Bor21] Glencora Borradaile. *Defend Dissent: Digital Suppression and Cryptographic Defense of Social Movements*. 2021.

[BSM+] Zoe Braiterman, Adam Shostack, Jonathan Marcil, Stephen de Vries, Irene Michlin, Kim Wuyts, Robert Hurlbut, Brook S.E. Schoenfield, Fraser Scott, Matthew Coles, Chris Romeo, Alyssa Miller, Izar Tarandach, Avi Douglen, and Marc French. Threat modeling manifesto. `https://www.threatmodelingmanifesto.org/`. Accessed on 22 November 2024.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[Can03] Ran Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239, 2003.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (abstract) (informal contribution). In Carl Pomerance, editor, *CRYPTO'87*, volume 293 of *LNCS*, page 462. Springer, Berlin, Heidelberg, August 1988.

[CCK+18] Geumhwan Cho, Jusop Choi, Hyoungshick Kim, Sangwon Hyun, and Jungwoo Ryoo. Threat modeling and analysis of voice assistant applications. In Brent ByungHoon Kang and Jin Soo Jang, editors, *WISA 18*, volume 11402 of *LNCS*, pages 197–209. Springer, Cham, August 2018.

[CCL15] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 3–22. Springer, Berlin, Heidelberg, August 2015.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Berlin, Heidelberg, August 2001.

[CH14] Jane Cleland-Huang. How well do you know your personae non gratae? *IEEE software*, 31(4):28–31, 2014.

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 597–608. ACM Press, November 2014.

[CK21] Ran Canetti and Gabriel Kaptchuk. The broken promise of apple's announced forbidden-photo reporting system – and how to fix it. `https://www.bu.edu/riscs/2021/08/10/apple-csam/`, 2021.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

[CR21] Hsin Yi Chen and Siddharth Prakash Rao. On adoptability and use case exploration of threat modeling for mobile communication systems. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2417–2419. ACM Press, November 2021.

[CSV16] Ran Canetti, Daniel Shahaf, and Margarita Vald. Universally composable authentication and key-exchange with global PKI. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 265–296. Springer, Berlin, Heidelberg, March 2016.

[CSYW07] Richard Caralli, James Stevens, Lisa Young, and William Wilson. Introducing octave allegro: Improving the information security risk assessment process. Technical Report CMU/SEI-2007-TR-012, Carnegi Mellon University Software Engineering Insitute, May 2007. Accessed: 2024-Nov-21.

[DAB+02] J.R. Douceur, A. Adya, W.J. Bolosky, P. Simon, and M. Theimer. Reclaiming space from duplicate files in a serverless distributed file system. In *Proceedings 22nd International Conference on Distributed Computing Systems*, pages 617–624, 2002.

[Dwy21] Brad Dwyer. Imagenet contains naturally occurring neuralhash collisions. `https://blog.roboflow.com/neuralhash-collision/`, Aug 2021.

[Esp21] Filipe Espósito. Apple employees express concerns about new csam scanning. `https://9to5mac.com/2021/08/12/apple-employees-express-concerns-about-new-csam-scanning/`, Aug 2021.

[FHA22] Michael Froehlich, Philipp Hulm, and Florian Alt. Under pressure. a user-centered threat model for cryptocurrency owners. In *Proceedings of the 2021 4th International Conference on Blockchain Technology and Applications*, ICBTA '21, page 39–50, New York, NY, USA, 2022. Association for Computing Machinery.

[FN21] Sharon Bradford Franklin and Greg Nojeim. International coalition calls on apple to abandon plan to build surveillance capabilities into iphones, ipads, and other products. `https://cdt.org/insights/international-coalition-calls-on-apple-to-abandon-plan-to-build-surveillance-capabilities-into-iphones-ipads-and-other-products/`, Aug 2021.

[For21] David Forsyth. Apple's csam detection technology. `https://www.apple.com/child-safety/pdf/Technical_Assessment_of_CSAM_Detection_David_Forsyth.pdf`, July 2021. Since initial publication, Apple has modified the contents of the webpage. The initial announcement can be view at `https://web.archive.org/web/20210805192137/https://www.apple.com/child-safety/pdf/Technical_Assessment_of_CSAM_Detection_David_Forsyth.pdf`. Accessed on 18 Decemeber, 2023.

[FVA+22] Kelsey R. Fulton, Daniel Votipka, Desiree Abrokwa, Michelle L. Mazurek, Michael Hicks, and James Parker. Understanding the how and the why: Exploring secure development practices through a course competition. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1141–1155. ACM Press, November 2022.

[GBBA24] Kathrin Grosse, Lukas Bieringer, Tarek R. Besold, and Alexandre Alahi. Towards more practical threat models in artificial intelligence security. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.

[GGK+16] Christina Garman, Matthew Green, Gabriel Kaptchuk, Ian Miers, and Michael Rushanan. Dancing on the lip of the volcano: Chosen ciphertext attacks on apple iMessage. In Thorsten Holz and Stefan Savage, editors, *USENIX Security 2016*, pages 655–672. USENIX Association, August 2016.

[Gil21] Daniel Kahn Gillmor. Apple's new 'child safety' plan for iphones isn't so safe. `https://www.aclu.org/news/privacy-technology/apples-new-child-safety-plan-for-iphones-isnt-so-safe`, Aug 2021.

[GMPS25] Aarushi Goel, Peihan Miao, Phuoc Van Long Pham, and Satvinder Singh. PICS: Private intersection over committed (and reusable) sets. Cryptology ePrint Archive, Paper 2025/1071, 2025.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[Gra20] Julie Inman Grant. End-to-end encryption: a challenging quest for balance. `https://www.esafety.gov.au/newsroom/blogs/end-end-encryption-challenging-quest-for-balance`, Feb 2020.

[Gre19] Matthew Green. Can end-to-end encrypted systems detect child sexual abuse imagery? `https://blog.cryptographyengineering.com/2019/12/08/on-client-side-media-scanning/`, Dec 2019.

[GS77] Chris Gane and Trish Sarson. *Structured systems analysis: tools and techniques*. McDonnell Douglas Systems Integration Company, 1977.

[GS21] Matthew D. Green and Alex Stamos. Apple wants to protect children. but it's creating serious privacy risks. `https://www.nytimes.com/2021/08/11/opinion/apple-iphones-privacy.html`, Aug 2021.

[GTS+] Matthew Green, Vanessa Teague, Bruce Schneier, Alex Stamos, and Carmela Troncoso. An evaluation of the risks of client-side scanning. Real World Crypto 2022 (RWC 23). Recording available at `https://www.youtube.com/live/AGlwtmfgMjk?si=l6nnmGFRckmdOwxj&t=1520`.

[HDWH12] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In Tadayoshi Kohno, editor, *USENIX Security 2012*, pages 205–220. USENIX Association, August 2012.

[HL06] Michael Howard and Steve Lipner. *The Security Development Lifecycle*, volume 34. 06 2006.

[HLvA02] Nicholas J. Hopper, John Langford, and Luis von Ahn. Provably secure steganography. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 77–92. Springer, Berlin, Heidelberg, August 2002.

[HWAM25] Sophie Hawkes, Christian Weinert, Teresa Almeida, and Maryam Mehrnezhad. Perceptual hash inversion attacks on image-based sexual abuse removal tools. *IEEE Security & Privacy*, 23(3):64–73, 2025.

[iac20] Crypto for the People, Invited talk at Crypto 2020 by Seny Kamara, August 2020.

[Inc] Hysn Technologies Inc. awesome-threat-modelling. Available at `https://github.com/hysnsec/awesome-threat-modelling`. Accessed on 19 November, 2024.

[Iri] IrisRisk. Irisrisk. Available at `https://www.iriusrisk.com/`. Accessed on 21 November 2024.

[Ism22] Benjamin Ismail. Apple's censorship and compromises in hong kong. https://wp.applecensorship.com/wp-content/uploads/2022/12/Apps-at-Risk_-Apples-Censorship-and-Compromises-in-Hong-Kong.pdf, Dec 2022.

[JL10] Stanislaw Jarecki and Xiaomin Liu. Fast secure computation of set intersection. In Juan A. Garay and Roberto De Prisco, editors, *SCN 10*, volume 6280 of *LNCS*, pages 418–435. Springer, Berlin, Heidelberg, September 2010.

[KAL23] Tadayoshi Kohno, Yasemin Acar, and Wulf Loh. Ethical frameworks and computer security trolley problems: Foundations for conversations. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 5145–5162. USENIX Association, August 2023.

[KBR13] Sriram Keelveedhi, Mihir Bellare, and Thomas Ristenpart. DupLESS: Server-aided encryption for deduplicated storage. In Samuel T. King, editor, *USENIX Security 2013*, pages 179–194. USENIX Association, August 2013.

[KKL+21] Seny Kamara, Mallory Knodel, Emma Llansó, Greg Nojeim, Lucy Qin, Dhanaraj Thakur, and Caitlin Vogus. Outside looking in: Approaches to content moderation in end-to-end encrypted systems. , Aug 2021.

[KM21] Anunay Kulshrestha and Jonathan R. Mayer. Identifying harmful media in end-to-end encrypted communication: Efficient private membership computation. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 893–910. USENIX Association, August 2021.

[Kob21a] Nadim Kobeissi. Nadim kobeissi on twitter. https://twitter.com/kaepora/status/1423387147172724741, Aug 2021.

[Kob21b] Nadim Kobeissi. An open letter against apple's privacy-invasive content scanning technology. https://appleprivacyletter.com/, August 2021. Accessed on 18 December, 2023.

[LAP24] Diane Leblanc-Albarel and Bart Preneel. Black-box collision attacks on widely deployed perceptual hash functions. Cryptology ePrint Archive, Report 2024/1869, 2024.

[Mic] Microsoft. Threat modeling tool. https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool. Accessed on 21 November 2024.

[MITa] MITRE. ATT&CK. https://attack.mitre.org/. Accessed on 22 November 2024.

[MITb] MITRE. Capec - common attack pattern enumerations and classifications. https://capec.mitre.org/. Accessed on 22 November 2024.

[MP21] India McKinney and Erica Portnoy. Apple's plan to "think different" about encryption opens a backdoor to your private life. https://www.eff.org/deeplinks/2021/08/apples-plan-think-different-about-encryption-opens-backdoor-your-private-life, Aug 2021.

[MSST21] Nat Meysenburg, Lauren Sarkesian, Ross Schulman, and Andi Wilson Thompson. A technical explainer on apple's concerning privacy changes. https://www.newamerica.org/oti/briefs/a-technical-explainer-on-apples-concerning-privacy-changes/, Aug 2021.

[Mur21] Steven J. Murdoch. Apple letting the content-scanning genie out of the bottle. https://www.benthamsgaze.org/2021/08/17/apple-letting-the-content-scanning-genie-out-of-the-bottle/, Aug 2021.

[NAoSM18] Engineering National Academies of Sciences and Medicine. *Decrypting the Encryption Debate: A Framework for Decision Makers*. The National Academies Press, Washington, DC, 2018.

[Nic21] Jack Nicas. Apple's compromises in china: 5 takeaways. https://www.nytimes.com/2021/05/17/technology/apple-china-privacy-censorship.html, May 2021.

[NZW21] Jack Nicas, Raymond Zhong, and Daisuke Wakabayashi. Censorship, surveillance and profits: A hard bargain for apple in china. https://www.nytimes.com/2021/05/17/technology/apple-china-censorship-data.html, May 2021.

[Ops21] Kurt Opsahl. If you build it, they will come: Apple has opened the backdoor to increased surveillance and censorship around the world. https://www.eff.org/deeplinks/2021/08/if-you-build-it-they-will-come-apple-has-opened-backdoor-increased-surveillance, Aug 2021.

[OWAa] OWASP. Threat dragon. Available at https://owasp.org/www-project-threat-dragon/. Accessed on 21 November 2024.

[OWAb] OWASP. Top ten. https://owasp.org/www-project-top-ten/. Accessed on 22 November 2024.

[PFG+23a] Jonathan Prokos, Neil Fendley, Matthew Green, Roei Schuster, Eran Tromer, Tushar Jois, and Yinzhi Cao. Squint hard enough: Attacking perceptual hashing with adversarial machine learning. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 211–228, Anaheim, CA, August 2023. USENIX Association.

[PFG+23b] Jonathan Prokos, Neil Fendley, Matthew Green, Roei Schuster, Eran Tromer, Tushar M. Jois, and Yinzhi Cao. Squint hard enough: Attacking perceptual hashing with adversarial machine learning. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023*, pages 211–228. USENIX Association, August 2023.

[Pin21] Benny Pinkas. A review of the cryptography behind the apple psi system. http://www.apple.com/child-safety/pdf/Technical_Assessment_of_CSAM_Detection_Benny_Pinkas.pdf, July 2021. Since initial publication, Apple has modified the contents of the webpage. The initial announcement can be view at https://web.archive.org/web/20210805190856/http://www.apple.com/child-safety/pdf/Technical_Assessment_of_CSAM_Detection_Benny_Pinkas.pdf. Accessed on 18 Decemeber, 2023.

[Por19] Erica Portnoy. Why adding client-side scanning breaks end-to-end encryption. EFF Blog. https://www.eff.org/deeplinks/2019/11/why-adding-client-side-scanning-breaks-end-end-encryption, Nov 2019.

[PPK21] Nilay Patel, Riana Pfefferkorn, and Jen King. Here's why apple's new child safety features are so controversial. https://www.theverge.com/22617554/apple-csam-child-safety-features-jen-king-riana-pfefferkorn-interview-decoder, Aug 2021.

[Res21] Eric Rescorla. Overview of apple's client-side csam scanning. https://educatedguesswork.org/posts/apple-csam-intro/, Aug 2021.

[Rog15] Phillip Rogaway. The moral character of cryptographic work. Cryptology ePrint Archive, Report 2015/1162, 2015.

[Ros20] Paul Rosenzweig. The law and policy of client-side scanning. https://www.lawfaremedia.org/article/law-and-policy-client-side-scanning, Aug 2020.

[Ros26] Leah Namisa Rosenbloom. Cryptography and Collective Power. In Daniel Escudero and Ivan Damgård, editors, *Progress in Cryptology – LATINCRYPT 2025*, pages 3–39, Cham, 2026. Springer Nature Switzerland.

[San21] Julian Sanchez. Apple's iphone: Now with built-in surveillance. https://www.cato.org/blog/apples-iphone-now-built-surveillance, Aug 2021.

[Sch99] Bruce Schneier. Attack trees. *Dr. Dobb's journal*, 24(12):21–29, 1999.

[Sch15] Bruce Schneier. *Secrets and lies: digital security in a networked world*. John Wiley & Sons, 2015.

[SCO+18] Nataliya Shevchenko, Timothy A Chick, Paige O'Riordan, Thomas Patrick Scanlon, and Carol Woody. Threat modeling: a summary of available methods. *Software Engineering Institute—Carnegie Mellon University*, pages 1–24, 2018.

[Sel19] J Selin. Evaluation of threat modeling methodologies a case study. *School of Technology Information and Communication Technology*, (May), 2019.

[SFW18] Nataliya Shevchenko, Brent R Frye, and Carol Woody. Threat modeling for cyber-physical system-of-systems: Methods evaluation. *Software Engineering Institute: Pittsburgh, PA, USA*, 2018.

[SG89] Susan Leigh Star and James R Griesemer. Institutional ecology,translations' and boundary objects: Amateurs and professionals in berkeley's museum of vertebrate zoology, 1907-39. *Social studies of science*, 19(3):387–420, 1989.

[SGSM22] Zhenpeng Shi, Kalman Graffi, David Starobinski, and Nikolay Matyunin. Threat modeling tools: A taxonomy. *IEEE Security & Privacy*, 20(4):29–39, 2022.

[SHNK22] Lukas Struppek, Dominik Hintersdorf, Daniel Neider, and Kristian Kersting. Learning to break deep perceptual hashing: The use case neuralhash. In *Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '22, page 58–69, New York, NY, USA, 2022. Association for Computing Machinery.

[Shu16] Forrest Shull. Evaluation of competing threat modeling methodologies. *Evaluation*, 2016.

[Sim83] Gustavus J. Simmons. The prisoners' problem and the subliminal channel. In David Chaum, editor, *CRYPTO'83*, pages 51–67. Plenum Press, New York, USA, 1983.

[SKM23a] Sarah Scheffler, Anunay Kulshrestha, and Jonathan Mayer. Public verification for private hash matching. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 253–273, 2023.

[SKM23b] Sarah Scheffler, Anunay Kulshrestha, and Jonathan R. Mayer. Public verification for private hash matching. In *2023 IEEE Symposium on Security and Privacy*, pages 253–273. IEEE Computer Society Press, May 2023.

[SLY+25] Yunqing Sun, Hanlin Liu, Kang Yang, Yu Yu, Xiao Wang, and Chenkai Weng. Committed vector oblivious linear evaluation and its applications. Cryptology ePrint Archive, Paper 2025/1037, 2025.

[SM23] Sarah Scheffler and Jonathan R. Mayer. SoK: Content moderation for end-to-end encryption. *PoPETs*, 2023(2):403–429, April 2023.

[SS16] Murugiah Souppaya and Karen Scarfone. Draft nist special publication 800-154: Guide to data-centric system threat modeling. Available at https://csrc.nist.gov/pubs/sp/800/154/ipd. Accessed on 30 December 2024., March 2016.

[SVR+18] Rock Stevens, Daniel Votipka, Elissa M. Redmiles, Colin Ahern, Patrick Sweeney, and Michelle L. Mazurek. The battle for new york: A case study of applied digital threat modeling at the enterprise level. In William Enck and Adrienne Porter Felt, editors, *USENIX Security 2018*, pages 621–637. USENIX Association, August 2018.

[SWJ15] Riccardo Scandariato, Kim Wuyts, and Wouter Joosen. A descriptive study of microsoft's threat modeling technique. *Requirements Engineering*, 20:163–180, 2015.

[Tar] Izar Tarandach. pytm. https://github.com/izar/pytm. Accessed on 21 November 2024.

[TMPV24] Ronald Thompson, Madeline McLaughlin, Carson Powers, and Daniel Votipka. "There are rabbit holes I want to go down that I'm not allowed to go down": An investigation of security expert threat modeling practices for medical devices. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024*. USENIX Association, August 2024.

[TTP26] Kien Tuong Truong, Noemi Terzo, and Kenneth G. Paterson. Signal lost (integrity): The signal app is more than the sum of its protocols. Cryptology ePrint Archive, Paper 2026/484, 2026.

[UM15] Tony UcedaVelez and Marco M Morana. *Risk Centric Threat Modeling: process for attack simulation and threat analysis*. John Wiley & Sons, 2015.

[USc] U.S. Code Title 18 - Crimes and Criminal Procedures. Accessed on 1 Aug 2025 at `https://www.law.cornell.edu/uscode/text/18/2258A`.

[use22] USENIX Enigma 2022 - Crypto for the People (part 2), April 2022.

[vAH04] Luis von Ahn and Nicholas J. Hopper. Public-key steganography. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 323–341. Springer, Berlin, Heidelberg, May 2004.

[VGJ15] David Vandervort, Dale E. Gaucas, and Robert St. Jacques. Issues in designing a bitcoin-like community currency. In Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff, editors, *FC 2015 Workshops*, volume 8976 of *LNCS*, pages 78–91. Springer, Berlin, Heidelberg, January 2015.

[VYSJ24] Stef Verreydt, Koen Yskout, Laurens Sion, and Wouter Joosen. Threat modeling state of practice in dutch organizations. In *Twentieth Symposium on Usable Privacy and Security (SOUPS 2024)*, pages 473–486, Philadelphia, PA, August 2024. USENIX Association.

[Wea19] Nicholas Weaver. Encryption and combating child exploitation imagery. `https://www.lawfaremedia.org/article/encryption-and-combating-child-exploitation-imagery`, Oct 2019.

[WJ15] Kim Wuyts and Wouter Joosen. Linddun privacy threat modeling: a tutorial. *CW Reports*, 2015.

[WVLHJ18] Kim Wuyts, Dimitri Van Landuyt, Aram Hovsepyan, and Wouter Joosen. Effective and efficient privacy threat modeling through domain refinements. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, pages 1175–1178, 2018.

[WWU+11] Jackson Wynn, Joseph Whitmore, Geoff Upton, Lindsay Spriggs, Dan McKinnon, Richard McInnes, Richard Graubart, and Lauren Clausen. Threat assessment & remediation analysis (tara). `https://www.mitre.org/sites/default/files/2021-10/pr-11-4982-tara-methodology-and-description.pdf`, October 2011.

[XL19] Wenjun Xiong and Robert Lagerström. Threat modeling–a systematic literature review. *Computers & security*, 84:53–69, 2019.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

A Updating Apple's Initial Proposal

In this section we cover the ways in which we have (slightly) updated the modeling provided by Apple in Figure 2 such that it can more faithfully capture the way it would have been used in practice—and as a result, is more amenable to analysis in our framework. We also discuss the (minor) changes required to make their protocol function with this new modeling. We divide this discussion into three subsections: First, in Appendix A.1, we present a verbatim replication of the ideal functionality present in [App21a] and discuss the ways in which we update their modeling to that presented in Figure 2. Next, in Appendix A.2, we present the initial protocol proposed by Apple. Finally, in Appendix A.3, we discuss the needed updates to their protocol necessary to make it compatible with the updated ideal functionality.

A.1 Apple's Initial Ideal Functionality

We include a verbatim description for the ideal functionality Apple included in their initial proposal in Figure 13. We use initial notation, which requires some clarifications in the context of our work. Specifically:

– $\bar{Y}$ is a set of m tuples of the form $(\mathsf{img_hash}, \mathsf{img_id}, ad)$, *i.e.,* if

$$\bar{Y} = \{(\mathsf{img_hash}_1, \mathsf{img_id}_1, ad_1), \ldots, (\mathsf{img_hash}_m, \mathsf{img_id}_m, ad_m)\}$$

then $\bar{Y}_{id} = \{\mathsf{img_id}_1, \ldots, \mathsf{img_id}_m\}$, $id(\bar{Y})$ is $\bar{Y}_{id}$ with no duplicates, and

$$\bar{Y}_{\{id,ad\}} = \{(\mathsf{img_id}_1, ad_1), \ldots, (\mathsf{img_id}_m, ad_m)\}.$$

– If $X = \{\mathsf{img_hash}'_1, \ldots, \mathsf{img_hash}'_n\}$, then

$$id(\bar{Y} \cap X) = \{\mathsf{img_id}_i \mid \exists\ (\mathsf{img_hash}_i, \mathsf{img_id}_i, ad_i) \in \bar{Y} \text{ and } \mathsf{img_hash}_i \in X\}$$

– If $T = \{\mathsf{img_id}'_1, \ldots, \mathsf{img_id}'_n\}$, then

$$\bar{Y}[T] = \{(\mathsf{img_hash}_i, \mathsf{img_id}_i, ad_i) \mid (\mathsf{img_hash}_i, \mathsf{img_id}_i, ad_i) \in \bar{Y} \text{ and } \mathsf{img_id}_i \in T\}.$$

$\mathcal{F}_{\text{ftPSI-AD}}$

Parameters known to all parties:

- two parties: server and client,
- B is the maximum set size for the server and client,
- t is the threshold,
- $s_{\max}$ is the maximum size of the set S of synthetics,
- all associated data values ad in $\mathcal{D}$ have the same public length.

The functionality $\mathcal{F}_{\text{ftPSI-AD}}$:

- Wait for input $X = \{x_1, x_2, \ldots\}$ from the server; abort if the server is corrupt and $|X| > B$.
- Send $|X|$ to the client; abort if the client is corrupt and aborts.
- Wait for input $\bar{Y}$ and $S \subseteq id(\bar{Y})$ from the client; abort if the client is corrupt and ($m > B$ or $|S| > s_{\max}$).
- Send $\bar{Y}_{id}$ to the server.
- If $|id(\bar{Y} \cap X) \setminus S| > t$:
  send $\bar{Y}[id(\bar{Y} \cap X) \setminus S]_{\{id,ad\}}$ and S to the server,
  otherwise
  send $id(\bar{Y} \cap X) \cup S$ to the server.

Figure 13: Apple's formalization of their ideal functionality.

$\Pi_{\text{Apple-CSAM-Scan}}$

Parameters: threshold $\mathcal{T}$, generator $g \in \mathbb{G}$, set size n, and slack factor $\epsilon \in [0, 1]$.

Apple Initializing Scanning: When Apple is initialized on $X = \{\mathsf{img_hash}_1, \ldots, \mathsf{img_hash}_n\}$ and a client $\mathcal{U}$:

1. Sample a scalar α and set $L = g^\alpha$.
2. Sample h_0, h_1 and compute $T \leftarrow \mathsf{CuckooTable}_{h_0,h_1}(X)$. Repeat this process until $T \neq \perp$.
3. For $i \in [n']$, if $T[i] \neq \perp$, $p_i \leftarrow \mathsf{HashToCurve}(T[i])^\alpha$, otherwise set $p_i \xleftarrow{\$} \mathbb{G} \setminus \{0\}$
4. Send $\mathsf{pdata} = (L, p_1, \ldots, p_{n'}, h_0, h_1)$ to the client $\mathcal{U}$.

Client Device Setup: When the client $\mathcal{U}$ receives a message $\mathsf{pdata} = (L, p_1, \ldots, p_{n'}, h_0, h_1)$ from Apple:

1. If $L \notin \mathbb{G} \setminus \{0\}$, or $\exists\ i$ s.t. $p_i \notin \mathbb{G} \setminus \{0\}$, or $\exists\ i, j$ such that $p_i = p_j$, return $\perp$.
2. Sample and store an encryption key $adkey$ and PRF key $hkey$.

Client Device Voucher Upload: When the client inputs a tuple $(\mathsf{img_id}, \mathsf{img_hash}, \mathsf{k})$:

1. Set $adct \leftarrow \mathsf{Enc}(adkey, \mathsf{k})$ and $sh \leftarrow \mathsf{SecretShareAtPoint}(adkey, \mathcal{T}, \mathsf{PRF}(hkey, \mathsf{img_id}))$
2. Sample encryption key $rkey$, and compute $rct \leftarrow \mathsf{Enc}(rkey, (adct, sh))$
3. For $j \in \{0, 1\}$:
   (a) Sample scalars β_j, γ_j and compute $q_j \leftarrow \mathsf{HashToCurve}(\mathsf{img_hash})^{\beta_j} \cdot g^{\gamma_j}$ and $s_j \leftarrow (p_{h_j(\mathsf{img_hash})})^{\beta_j} \cdot L^{\gamma_j}$
   (b) Set $ct_j \leftarrow \mathsf{Enc}(\mathsf{KDF}(s_j), rkey)$
4. Sample $b \xleftarrow{\$} \{0, 1\}$ and send $(\mathsf{img_id}, q_b, ct_b, q_{1-b}, ct_{1-b}, rct)$ to the server (i.e., shuffle the contents).

Apple Voucher Processing: When Apple receives $(\mathsf{img_id}, q_0, ct_0, q_1, ct_1, rct)$ from a client $\mathcal{U}$:

1. For $j \in \{0, 1\}$:
   (a) Set $rkey_j \leftarrow \mathsf{Dec}(\mathsf{KDF}(q_j^\alpha), ct_j)$
   (b) If $rkey_j \neq \perp$, set $(adct_j, sh_j) \leftarrow \mathsf{Dec}(rkey_j, rct)$
2. If there is exactly one index $j \in \{0, 1\}$ for which $rkey_j \neq \perp$, add $(\mathsf{img_id}, adct_j, sh_j)$ to $\mathsf{SHARES}_\mathcal{U}$.
3. If $|\mathsf{SHARES}_\mathcal{U}| > \mathcal{T}$, reconstruct $adkey$ using $\mathcal{T} + 1$ shares in $\mathsf{SHARES}_\mathcal{U}$. Then, for each triple $(\mathsf{img_id}_i, adct_i, sh_i) \in \mathsf{SHARES}_\mathcal{U}$, compute $k_i \leftarrow \mathsf{Dec}(adkey, adct_i)$ and add $(\mathsf{img_id}_i, k_i)$ to $\mathsf{OUTSET}_\mathcal{U}$. Finally, output $\mathsf{OUTSET}_\mathcal{U}$.

Figure 14: Apple's Initial CSAM Scanning Protocol.

Modeling limitations. There are three main weaknesses in the modeling and formalism within Apple's initial proposal. The re-rendering of Apple's ideal functionality that we use in the main body of the work (Figure 2) directly addresses these limitations.

(1) Hash functions are not treated as leakage: In Apple's initial ideal functionality specification, $\mathcal{F}_{\text{ftPSI-AD}}$, the hash functions h_0, h_1 used to generate the Cuckoo table are not treated as leakage. This is clearly a problem, as the hash functions are rejection sampled as a function of the server's input. Simply put, there is no way around having this be a leakage associated with the system (even if that leakage, in practice, has very little impact). Therefore, we simply add this as an explicit leakage to the ideal functionality.

(2) PSI protocol is modeled as "one shot:" The second noteworthy weakness of $\mathcal{F}_{\text{ftPSI-AD}}$ is the choice to have the client supply all their inputs to the ideal functionality in one shot. It is clear, however, that this is not how the proposed system was going to work in practice. In fact, one of the beautiful aspects of Apple's proposal (from a cryptographic perspective) is that clients provide inputs *incrementally*: each time a client device wants to back-up an image, it creates a voucher for that image independently—a process which has computational and communication complexity independent of the number of images previously backed-up by the client device.

(3) Self composition for multiple parties: Apple's initial formation $\mathcal{F}_{\text{ftPSI-AD}}$ is between a single server and a single client, and implicitly relies on *self composition* to argue that such a system can be safely used between Apple and all of the client devices in its network. Using self-composition in this way, however, means that the server is free to choose its inputs to each instance of $\mathcal{F}_{\text{ftPSI-AD}}$ independently. Concretely, this allows a

malicious Apple to covertly select different hash sets for each client. This weakness becomes immediately apparent when analyzing their protocol within our framework as the server's choice of inputs to each instance of $\mathcal{F}_{\text{ftPSI-AD}}$ happens only within $\Pi_{\text{Intended-use}}$.

A.2 Apple's Initial Protocol

We provide a description of Apple's proposed protocol in Figure 14. Let $(\mathsf{Enc}, \mathsf{Dec})$ be a standard $\mathsf{IND\$-CCA}$ encryption scheme that provides *random key robustness*. In Apple's implementation, this is done with AES128-GCM with a random 96-bit nonce and PRF is a standard pseudorandom function. For brevity, we make use of several sub-protocols:

- $\mathsf{CuckooTable}_{h_0,h_1}(X)$ maps a set X into a table T such that for all $x \in X$, either $T[h_0(x)] = x$ or $T[h_1(x)] = x$, and empty elements are $\perp$. If a cycle exists, this function returns $\perp$.
- $\mathsf{HashToCurve}$ is modeled as a (programmable) random oracle that maps elements of the image hash space to a uniform element of the group $\mathbb{G}$.
- $\mathsf{SecretShareAtPoint}(x, \mathcal{T}, r)$ generates the Shamir Secret share of x with threshold $\mathcal{T}$ at point r. Each time a client calls this subroutine, we assume that it is *de-randomized*, i.e., the polynomial that is to be evaluated at r is fixed.
- KDF is a key derivation function that is modeled as a (programmable) random oracle.

Protocol limitations. One limitation of Apple's protocol is that Apple only proves *privacy* with respect to a malicious client. They do not, however, make any claims about the "correctness guarantee for the outputs" [App21a, Page 20]. Namely, there is no (formally proven) guarantee that the associated data that the server receives as output from the ideal functionality will match the inputs provided by the client. This follows from the well-known difficulty of designing fully malicious secure PSI protocols from DDH-like assumptions (see, for example, the discussion in [JL10]). While this is not a problem if the PSI protocol is considered in isolation, this can be problematic when we require composability.

Apple's simulator for a malicious client [App21a, p. 24-25] stops after producing a simulated pdata and there is no description of the simulator's procedure for *extracting* the client's inputs for the ideal functionality. Creating such a valid extraction strategy appears to require non-trivial changes to their protocol. While some information can be extracted by modeling the hashing operations as random oracles, the properties of the Naor-Reingold randomized self-reduction mean that a simulator cannot determine if vouchers produced by a malicious client are well-formed.

To see this problem, consider that the pdata message constructed by the simulator can embed at most n' candidate perceptual hashes. At the same time, the simulator will need to extract *all* of the perceptual hashes input by the client—which might be significantly more than n'. Thus, the simulator may eventually receive a voucher that it cannot "decrypt." For this voucher, there are two cases: the client honestly encoded a perceptual hash (which may or may not be in the ideal server's hash set) or the voucher is malformed and would never be "decryptable." The simulator cannot distinguish between these two settings by the properties of the randomized self-reduction, breaking the simulation.

A.3 Updating $\Pi_{\text{Apple-CSAM-Scan}}$ for New Modeling

We make two protocol modifications to Apple's proposal to make it work with our new modeling. We do not provide the full details of these changes independently, as they are integrated directly into the updated protocol we describe in Section 5.

Enabling equivocation. The consequence of Apple's choice to model the private set intersection as "one-shot" is a subtle, cryptographic one, rather than a catastrophic compromise. Namely, when simulating the server, the simulator in Apple's initial formulation knows the final result of the private set-intersection protocol *before* it needs to simulate vouchers. As such, the simulation strategy is nearly trivial: simply

generate vouchers honestly for each "matching" image and generate uniform random vouchers for non-matching images. When switching to an incremental formation of the ideal functionality (as we have done in Figure 2), this strategy is no longer possible; the simulator knows if the voucher represents a match or a non-match, but does not know the "contents" of the voucher until *after* the threshold number of matches has been met. Thus, we require a modification of the protocol that would allow the simulator to equivocate to the contents of vouchers. We update the protocol to allow for this equivocation using standard techniques in the programmable random oracle model (an assumption already present in Apple's initial construction).

Specifically, we add an additional layer of indirection through the programmable random oracle. For each perceptual hash for which the client wants to generate a voucher, they sample a random value r, query the random oracle on $(adkey \| r \| \mathcal{U} \| \mathsf{img_id})$ to key a per-input key $\mathsf{k}_{\mathsf{img_id}}$, where $\mathcal{U}$ is a unique identifier for the client. $\mathsf{k}_{\mathsf{img_id}}$ is then used to encrypt the image encryption key k (i.e., the data payload that the server should get from the private set intersection protocol) using an encryption scheme that is easy to equivocate (e.g., the one-time pad). The value r can then be released alongside $adct$. Once a server reconstructs $adkey$, as in the base protocol, they can query the random oracle consistently and retrieve the same key.

Our equivocation strategy follows directly: when the simulator creates a voucher, it simply samples the ciphertext of k as a random string and otherwise proceeds as usual. Then, once the threshold has been matched, the simulator can program the random oracle such that all the decryption yields the correct values. Notice that the server does not know the high entropy value $adkey$ until after the threshold is met, so the probability of anyone querying the random oracle *before* the simulator is able to program the output is negligible.

Enabling extraction. There are two "easy" options for enabling extraction without making significant changes to Apple's protocol. The first alternative is to model the client device as strictly honest, making extraction trivial. This is the implicit assumption present in Apple's initial proposal and might be reasonable when the client software is written and managed by the server itself.

The other option is much more conceptually simple—and, perhaps, more compelling: simply have the client prove that the voucher is well-constructed in zero-knowledge. The simulator can then extract directly from the zero-knowledge proof, circumventing the complexity of making changes to the internals of the protocol structure altogether. In practice, this approach is slightly non-trivial, given that the protocol makes use of multiple hash functions modeled as random oracles (specifically, $\mathsf{HashToCurve}$ and KDF) which cannot be used within the zero-knowledge proof. As such, we chose to change the modeling of KDF to just be a simple mapping function from group elements to the keyspace of the random-key robust encryption scheme. This change makes it difficult to extract s_j, but we can "fix" this problem by simply extracting from the zero-knowledge proof. Specifically, we generate the following proof:

$$\pi_{\text{voucher}} = \mathsf{PoK}\left\{(\{\beta_j, \gamma_j, g_j, p_{i_j}, s_j, i_j\}_{j\in\{0,1\}}) : q_j = g_j^{\beta_j} g^{\gamma_j}, s_j = p_{i_j}^{\beta_j} L^{\gamma_j}, p_{i_j} \in \mathsf{pdata}, \mathsf{Dec}(\mathsf{KDF}(s_j), ct_j) \neq \perp\right\}$$

The simulator can extract the witness $\{\beta_j, \gamma_j, g_j, p_{i_j}, s_j, i_j\}_{j\in\{0,1\}}$ from π_{voucher} and begin to search through the random oracle queries to see if g_j was generated as the output of a random oracle query. If yes, then the input to that random oracle query was the $\mathsf{img_hash}$ that should be passed to the ideal functionality. If not, then the voucher will never be decryptable.

We note that this approach critically relies upon the random-key robustness of the encryption scheme. Without this property, it is conceivable that a malicious client could "work backwards" to make a decryptable voucher without picking any corresponding $\mathsf{img_hash}$. Specifically, they could find a key $\mathsf{KDF}(\hat{s_j})$ that makes the decryption pass, and then solve for values of g_j, β_j, γ_j that produce such a value. But, if the value $\hat{s_j}$ is not selected at random, then this process would require solving a discrete log problem (i.e., by finding β_j, γ_j such that $\hat{s_j} = p_{i_j}^{\beta_j} L^{\gamma_j}$). Thus, it is enough to rely on the random-key robustness property.

Another alternative would be to change the modeling on the $\mathsf{HashToCurve}$ function. While this would be a feasible choice, it would immediately break the current extraction strategy for pdata (i.e., when simulating the server). Thus, going down this route would require additional modifications.

B Protocol Implementing $\mathcal{F}_{\text{CSAM-Scan}}$

B.1 Protocol Overview

In an effort to keep this section self-contained, we briefly give an overview of the changes that we make to Apple's initial proposal in order to have it realize $\mathcal{F}_{\text{CSAM-Scan}}$.

Proving pdata's consistence with NCMEC's chosen hashes. There are three protocol changes we make in order to allow clients to know that every client's backups are being scanned for exactly the hashes specified by NCMEC:

- *Derandomizing* pdata*.* The first important change is to the generation process of pdata, such that Apple has no opportunity to inject their chosen images into the selected set. We do this by generating the first n elements of pdata as before, but then computing the remaining $n' - n$ elements as linear combinations of the first (i.e., by Lagrange interpolation in the exponent).
- *Prove consistency using the commit-and-prove paradigm.* NCMEC begins by producing a cryptographic commitment c to its chosen hashes, which it then signs σ_{NCMEC}. Apple is then provided with both the chosen hashes and the decommitment randomness for the commitment scheme. Using this information, Apple can provide a zero-knowledge proof of consistency between the parameters pdata sent to clients and the commitment c. We do this by making use of $\mathcal{F}_{\mathsf{CommitAndProve}}$. We note that a consequence of this strategy is that we can no longer model the HashToCurve as a random oracle, so we switch the modeling to be a cryptographic hash function with pre-image resistance (such that the zero-knowledge proof can make non-black box use of the hash function).
- *NCMEC posts their commitment to the bulletin board.* Rather than having Apple forward c, σ_{NCMEC} to each client directly, we instead have NCMEC post these values to the bulletin board $\mathcal{F}_{BB}$. This prevents NCMEC from producing multiple different commitments and sending each to a different client. For both social and technical reasons (as outlined in Section 5.2) we have NCMEC push the hashes through the random oracle before committing. This allows Apple's initial security argument for privacy against malicious clients from [App21a] to hold.

Modifying the voucher creation process. We make three main changes to the way we build vouchers, two of which are described in Appendix A.

- *Enabling equivocation.* We begin by making the changes outlined in Appendix A.3. Namely, rather than using a single key $\mathsf{k}_{\text{threshold}}$ to encrypt all of the image encrypted keys, we derive a unique, per-input key $\mathsf{k}_{\mathsf{img_id}}$ by sampling a random value r and passing $\mathsf{k}_{\text{threshold}}$ and r through the random oracle.[16]. This allows for the simulator to equivocate using standard techniques (i.e., use of a one-time pad with the random oracle's output for encryption).
- *Integrating the judge's key.* In order to ensure that the judge must be involved in the process of decrypting images, we have the client encrypt the ciphertext *adct* (along with the randomness r used to derive the key $\mathsf{k}_{\mathsf{img_id}}$) under the judge's public key to produce the ciphertext $ct_{\mathcal{J}}$.
- *Adding extractability via zero-knowledge.* As discussed in Appendix A.3, there are minor changes required if we want to make the vouchers *extractable*, allowing us to write a UC proof. The specific zero-knowledge proof in Appendix A.3 is no longer sufficient in order to make the proof go through, as HashToCurve is no longer modeled as a random oracle. Thus, the zero-knowledge proof is slightly expanded. Specifically, we have the client commit to their long-term key material and then use $\mathcal{F}_{\mathsf{CommitAndProve}}$ to prove consistency.

Public Verifiability. In order to enable public verifiability, we simply need to add signatures and zero-knowledge proofs in the appropriate places to ensure that a full transcript of the decryption process is verifiable (i.e., the voucher submission, the decryption process by Apple, the decryption process by the judge, having the image ciphertexts themselves be identifiable, etc. . .).

[16] We switch the notation from the base protocol slightly because there are too many keys throughout the construction. Namely, in Apple's initial proposal (and in Appendix A), they call this key *adkey*, which we rename to $\mathsf{k}_{\text{threshold}}$

B.2 Preliminaries

We make direct use of several concrete cryptographic protocols.

1. We make use of the same symmetric key encryption scheme with random key robustness, as used in Apple's initial scheme [App21a]. We denote the algorithms in this scheme as Enc and Dec. We assume that the key space for this encryption algorithm is simply $\{0,1\}^\lambda$.
2. In order to facilitate key encapsulation under the judge's key, we make use of an explicit public key encryption scheme, with associated algorithms $\mathsf{PKKeyGen}$, PKEnc, and PKDec. We also require that there is a deterministic function $\mathsf{DerivePublicKey}$ that takes the secret key and re-derives the associated public key.
3. We make use of a standard pseudorandom function PRF. We assume that the keyspace for the PRF is simply $\{0,1\}^\lambda$.

Our protocol also makes use of multiple ideal functionalities, and is thus defined in a hybrid model. For the most part, these ideal functionalities are borrowed directly from prior work. On some occasions, we have to stylistically change the way in which these ideal functionalities are used. This is specifically because we pass some information between parties within $\Pi_{\text{Intended-use}}$, meaning that the parties who call the ideal functionalities may change from the originally envisioned usage. Additionally, we make use of uploading adversarial code and string generation algorithms to reduce the logistical overhead of requesting strings from $\mathcal{S}$.

Our signature functionality $\mathcal{F}^P_{\mathsf{Cert}}$. We provide indirect authentication of messages using the $\mathcal{F}^P_{\mathsf{Cert}}$ functionality from [Can03] (shown in Figure 15). Notice that this functionality is parameterized by the identity of a party P, such that it can be used to validate that a message m is associated with P.

$\mathcal{F}^P_{\mathsf{Cert}}$

Upon initialization, obtain a signature generation algorithm G from $\mathcal{S}$.

Signature Generation: Upon receiving a request (Sign, m) from P,

1. Let $\sigma = G(m)$. If there exists a record $(m, \sigma, 0)$, output an error message to P and halt.
2. Record the entry $(m, \sigma, 1)$ and send $(\mathsf{Signature}, m, \sigma)$ to P.

Signature Verification: Upon receiving a value $(\mathsf{Verify}, m, \sigma)$ from some party V:

1. Send $(\mathsf{Verify}, m, \sigma)$ to the adversary.
2. Upon receiving $(\mathsf{Verify}, m, \phi)$ from the adversary, do:
   (a) If $(m, \sigma, 1)$ is recorded, set $f = 1$.
   (b) Else if the signer is not corrupted, and no entry $(m, \sigma', 1)$ for any σ' is recorded, then set $f = 0$ and record the entry $(m, \sigma, 0)$.
   (c) Else if there is an entry (m, σ, f') recorded, then set $f = f'$.
   (d) Else, set $f = \phi$ and record the entry (m, σ', ϕ).
3. Output $(\mathsf{Verified}, m, f)$ to V.

Figure 15: Verification/Signature Functionality [Can03]

The Commit and Prove Functionality $\mathcal{F}_{\mathsf{CommitAndProve}}$. In order to prove that pdata is consistent with the hashes specified by NCMEC, we use a commit-and-prove paradigm. We provide this functionality with $\mathcal{F}_{\mathsf{CommitAndProve}}$ from [CLOS02]. We note that we have made non-trivial changes to this ideal functionality such that it fits better into our paradigm. The majority of these are stylistic—the identities of the prover,

committer, and verifier are not fixed ahead of time and we expose more of the commitment functionality (i.e., Open). Additionally, we expose strings that can be passed between parties, which were not present in the initial rendering. The most important consequence of this change is that the proof functionality becomes non-interactive (i.e., there is a string that it produces that can be verified without directly interacting with the prover). We make this change mostly for ergonomic reasons (i.e., such that the contents of each voucher can be written out explicitly, rather than having some values passed by $\mathcal{F}_{\mathsf{SMT}}$ and then interacting with a zero-knowledge ideal). In other words, this could be re-written to remove the non-interactive requirement, at the expense of making the protocol more difficult to read. We make use of two relation circuits:

– $\mathcal{F}^{\mathsf{pdata}}_{\mathsf{CommitAndProve}}$: The first relation checks that some particular pdata is constructed correctly with respect to the committed set of elements X. That is, we create a commitment to X and then produce the following proof:

$$\pi_{\mathsf{pdata}} = \mathsf{PoK}\left\{(X, \alpha) : L = g^{\alpha},\ \text{for } x \in X, (p_{h_0(x)} = \mathsf{HashToCurve}(x)^{\alpha} \text{ or } p_{h_1(x)} = \mathsf{HashToCurve}(x)^{\alpha})\right\}$$

– $\mathcal{F}^{\mathsf{voucher}}_{\mathsf{CommitAndProve}}$: The relation for this instance is the one described in Appendix A.3. It produces a proof of knowledge that the voucher is correctly constructed with respect to pdata for some input value $\mathsf{img_hash}$. Importantly, the construction of the vouchers includes use of the random oracle, but the proof does not cover this aspect of the generation algorithm. In more detail, we have the client commit to the values $(\mathsf{k}_{\text{threshold}}, hkey)$, then they produce the following proof of knowledge with respect to that commitment:

$$\begin{aligned}\pi_{\text{voucher}} = \mathsf{PoK}\Big\{(\mathsf{k}_{\text{threshold}}, hkey, \mathsf{img_hash}, \{\beta_j, \gamma_j, g_j, s_j, i_j\}_{j\in\{0,1\}}) : q_j = g_j^{\beta_j} g^{\gamma_j}, s_j = p_{i_j}^{\beta_j} L^{\gamma_j},\\ g_j = \mathsf{HashToCurve}(\mathsf{img_hash}), ct_{\mathcal{J}} = \mathsf{Enc}(pk_{\mathcal{J}}, (r, adct)),\\ sh \leftarrow \mathsf{SecretShareAtPoint}(\mathsf{k}_{\text{threshold}}, \mathcal{T}, \mathsf{PRF}(hkey, \mathsf{img_id})),\\ (\mathsf{Dec}(\mathsf{KDF}(s_0), ct_0) = \bot, (\mathsf{Dec}(\mathsf{KDF}(s_1), ct_1) = rkey), \mathsf{Dec}(rkey, rct) = (ct_{\mathcal{J}}, sh)) \text{ or}\\ (\mathsf{Dec}(\mathsf{KDF}(s_0), ct_0) = rkey, (\mathsf{Dec}(\mathsf{KDF}(s_1), ct_1) = \bot), \mathsf{Dec}(rkey, rct) = (ct_{\mathcal{J}}, sh))\Big\}\end{aligned}$$

$\mathcal{F}_{\mathsf{CommitAndProve}}$

$\mathcal{F}_{\mathsf{CommitAndProve}}$ is parameterized by a binary relation R and a distribution $\mathcal{D}$. Upon initialization $\mathcal{S}$ gives $\mathcal{F}_{\mathsf{CommitAndProve}}$ a string generation algorithms $\mathcal{S}_{\text{Com}}$ and $\mathcal{S}_{\text{Proof}}$.

Commit Phase: Upon receiving a message $(\mathsf{Commit}, \mathsf{sid}, w)$ from a party P, if such a message for sid has been received previously, ignore it. Otherwise, sample $d \xleftarrow{\$} \mathcal{D}$ and $c \leftarrow \mathcal{S}_{\text{Com}}(\mathsf{sid})$, then record (sid, c, d, w) and send $(\mathsf{CommitReceipt}, \mathsf{sid}, c, d)$ back to P.

Open Phase: Upon receiving a message $(\mathsf{Open}, \mathsf{sid}, c, d, w)$ from a party P, if there is a record (sid, c, d, w) then send $(\mathsf{OpenReceipt}, \mathsf{sid}, c, d, w, 1)$ to P. Otherwise, send $(\mathsf{OpenReceipt}, \mathsf{sid}, c, d, w, 0)$ to P.

Prove Phase: Upon receiving a message $(\mathsf{Prove}, \mathsf{sid}, x, c, d, w, w')$ from a party P, if there is no record (sid, c, d, w) then ignore the message. Otherwise, sample $\pi \leftarrow \mathcal{S}_{\text{Proof}}(\mathsf{sid})$ and create a record $(\mathsf{sid}, x, c, d, w, \pi, R(x, w\|w'))$. Then, send $(\mathsf{ProveReceipt}, \mathsf{sid}, x, c, d, w, \pi, R(x, w\|w'))$ back to P.

Verification Phase: Upon receiving a message $(\mathsf{Verify}, \mathsf{sid}, x, c, \pi)$ from a party P, if there is a record $(\mathsf{sid}, x, c, \cdot, \cdot, \pi, b)$, then send $(\mathsf{VerifyReceipt}, \mathsf{sid}, x, c, \pi, b)$ back to P. Otherwise send $(\mathsf{VerifyReceipt}, \mathsf{sid}, x, c, \pi, 0)$ back to P.

Figure 16: Commit and Prove Functionality adapted from [CLOS02]

The Programmable Random Oracle Functionality $\mathcal{G}_{\mathsf{pRO}}$. Just like Apple's initial proposal, we make use of a programmable random oracle. We make this assumption explicit through $\mathcal{G}_{\mathsf{pRO}}$. Notably, because of the modeling, we require that this random oracle is modeled as a global functionality [CJS14]. The description for $\mathcal{G}_{\mathsf{pRO}}$ can be found in Figure 17.

$\mathcal{G}_{\mathsf{pRO}}$

On input $(\texttt{HashQuery}, m, \ell)$:

1. If there is a record (m, h)
   - If $|h| \geq \ell$: let h' be the first ℓ bits of h. //$\mathcal{G}_{\mathsf{pRO}}$ returns prefixes of already-computed entries.
   - If $|h| < \ell$: choose $h_{\text{end}} \xleftarrow{\$} \{0,1\}^{\ell-|h|}$, let $h' = h||h_{\text{end}}$, and replace the record (m, h) with (m, h').

   Else choose $h' \xleftarrow{\$} \{0,1\}^{\ell}$ and record (m, h').
2. Output $(\texttt{HashQuery}, h')$ to the caller.

On message $(\texttt{Program}, m, h)$ from the adversary:

1. If there is no record (m, h'), then record (m, h). Send $(\texttt{Program})$ to the adversary. //If m has already been queried then programming fails silently.

Figure 17: Programmable Random Oracle Functionality [CJS14]

The Non-interactive Zero-Knowledge Proof Functionality $\mathcal{F}_{NIZK}$. In addition to $\mathcal{F}_{\mathsf{CommitAndProve}}$, we also make use of non-interactive zero-knowledge proofs in order to make the entire chain of decryption verifiable (i.e., that the server got a particular key because it was submitted as part of a voucher). The description of this functionality can be found in Figure 18. We make use of $\mathcal{F}_{NIZK}$ for two relations:

– $\mathcal{F}_{NIZK}^{\mathsf{decryption}}$: The relation for this instance checks that a set of ciphertext and plaintext are related by decryption under a secret key corresponding to a public key. Specifically:

$$\pi_{\text{decrypt}} = \mathsf{PoK}\left\{(sk_{\mathcal{J}}) : pk_{\mathcal{J}} = \mathsf{DerivePublicKey}(sk_{\mathcal{J}}) \text{ and } i \in [\ell], (r_i, adct_i) = \mathsf{PKDec}(sk_{\mathcal{J}}, ct_{\mathcal{J},i})\right\}$$

– $\mathcal{F}_{NIZK}^{\mathsf{KeyDev}}$: The relation checks that a key was derived from two Diffie-Hellman key shares:

$$\pi_{\text{KeyDev}} = \mathsf{PoK}\left\{(\alpha) : L = g^{\alpha}, \mathsf{k} = \mathsf{KDF}(q^{\alpha})\right\}$$

$\mathcal{F}_{NIZK}$

$\mathcal{F}_{NIZK}$ is parameterized by an NP relation $\mathcal{R}$.

- **Proof:** On input $(\mathsf{Prove}, \mathsf{sid}, x, w)$ from party P: If $\mathcal{R}(x, w) = 1$ then send $(\mathsf{Prove}, P, \mathsf{sid}, x)$ to $\mathcal{S}$. Upon receiving $(\mathsf{Proof}, \mathsf{sid}, \pi)$ from $\mathcal{S}$, store $(\mathsf{sid}, x, w, \pi)$ and send $(\mathsf{Proof}, \mathsf{sid}, \pi)$ to P.
- **Verification:** On input $(\mathsf{Verify}, \mathsf{sid}, x, \pi)$ from party V: If $(\mathsf{sid}, x, w, \pi)$ is stored, then return $(\mathsf{Verification}, \mathsf{sid}, x, \pi, \mathcal{R}(x, w))$ to V. Else, send $(\mathsf{Verify}, V, \mathsf{sid}, x, w)$ to $\mathcal{S}$. Upon receiving $(\mathsf{witness}, \mathsf{sid}, w)$ from $\mathcal{S}$, store $(\mathsf{sid}, x, w, \pi)$, and return $(\mathsf{Verification}, \mathsf{sid}, x, \pi, \mathcal{R}(x, w))$ to V.
- **Corruption:** When receiving $(\mathsf{corrupt}, \mathsf{sid})$ from $\mathcal{S}$, mark sid as corrupted. If there is a stored tuple $(\mathsf{sid}, x, w, \pi)$, then send it to $\mathcal{S}$.

Figure 18: NIZK Functionality

The Bulletin Board Functionality $\mathcal{F}_{BB}$. We make use of bulletin board functionality $\mathcal{F}_{BB}$, taken from [CSV16]. In our protocol, we make use of $\mathcal{F}_{BB}$ to make NCMEC's commitment to the hashes public and globally accessible. The description of $\mathcal{F}_{BB}$ can be found in Figure 19.

$\mathcal{F}_{BB}$

Upon initialization of the bulletin board, initialize a counter c.

Post: Upon receiving the message $(\mathsf{Post}, m, \sigma)$ from a party P, send $(\mathsf{Posted}, P, m, \sigma)$ to the adversary. Upon receiving OK from the adversary, record (c, P, m, σ). Increment c and send (Ok) to P.

RetrieveRequest: Upon receiving a message $(\mathsf{Retrieve}, m)$ from anyone, send public output $(\mathsf{Retrieve}, c, P, m, \sigma)$, if m was previously posted. Otherwise, return $\perp$.

RetrieveAll: Upon request receiving a message $(\mathsf{RetrieveAll})$ from anyone, send all recorded messages.

Figure 19: Ideal functionality for bulletin board [CSV16]

B.3 Protocol Description

We now provide a description of the modified protocol $\Pi_{\text{CSAM-Scan}}$ that realizes $\mathcal{F}_{\text{CSAM-Scan}}$. Given the size of $\Pi_{\text{CSAM-Scan}}$, it is not possible to present it in a single figure or linearly in text. Instead, we divide the protocol into the components run by each protocol participant. Specifically, NCMEC's part of the protocol can be found in Figure 20, clients' part of the protocol can be found in Figure 22, the server's part of the protocol can be found in Figure 21, the judge's part of the protocol can be found in Figure 23, and the public verifiability parts of the protocol, which could be run by any party, can be found in Figure 24.

The full protocol has several public parameters: a threshold $\mathcal{T}$, generator $g \in \mathbb{G}$, set size n, a slack factor $\epsilon \in [0, 1]$, and the security parameter λ.

NCMEC's part of the protocol. NCMEC's part of the protocol $\Pi_{\text{CSAM-Scan}}$ is straightforward. First, it invokes the commitment interface of $\mathcal{F}_{\mathsf{CommitAndProve}}$ on its set of input hashes $X = \{\mathsf{img_hash}_1, \ldots, \mathsf{img_hash}_n\}$. This generates a commitment string c and a decommitment string d. NCMEC then generates a signature σ_{NCMEC} on c by invoking the signature interface of $\mathcal{F}_{\mathsf{Cert}}$ and posts $(c, \sigma_{\text{NCMEC}})$ onto the bulletin board $\mathcal{F}_{BB}$. The values $(c, d, X, \sigma_{\text{NCMEC}})$ are then returned to the component of NCMEC in $\Pi_{\text{Intended-use}}$, which can share them with the $\Pi_{\text{Intended-use}}$ component of Apple. The formal description of this protocol can be found in Figure 20.

NCMEC's part of $\Pi_{\text{CSAM-Scan}}$

Specify Hashes On input $X = \{\mathsf{img_hash}_1, \ldots, \mathsf{img_hash}_n\}$:

1. For each $\mathsf{img_hash}_i \in X$, send $(\texttt{HashQuery}, \mathsf{img_hash}_i, \lambda)$ to $\mathcal{G}_{\mathsf{pRO}}$ and collect the response values in a set ROHashes.
2. Send $(\mathsf{Commit}, \mathsf{sid}, \mathsf{ROHashes})$ to $\mathcal{F}^{\mathsf{pdata}}_{\mathsf{CommitAndProve}}$ and receive $(\mathsf{CommitReceipt}, \mathsf{sid}, c_{\text{NCMEC}}, d)$ in response.
3. Send $(\mathsf{Sign}, c_{\text{NCMEC}})$ to $\mathcal{F}^{\text{NCMEC}}_{\mathsf{Cert}}$ and receive $(\mathsf{Signature}, c_{\text{NCMEC}}, \sigma_{\text{NCMEC}})$ in response.
4. Send $(\mathsf{Post}, (c_{\text{NCMEC}}, \sigma_{\text{NCMEC}}))$ to $\mathcal{F}_{BB}$ and receive (Ok) in response.
5. Return $(c_{\text{NCMEC}}, d, \mathsf{ROHashes}, \sigma_{\text{NCMEC}})$.

Figure 20: Description of NCMEC's part of $\Pi_{\text{CSAM-Scan}}$

Apple's part of the protocol. Apple has three parts of the protocol $\Pi_{\text{CSAM-Scan}}$: initialization of clients, processing vouchers that are uploaded by clients, and recovery of the image encryption keys once the threshold has been met and the judge's approval has been given. During initialization, Apple checks that NCMEC's inputs are well formed and then builds pdata. The main difference from the initial protocol can be found in steps (4d and 4e) in Figure 21: first, the non-$\perp$ elements of the cuckoo table are generated normally and then the remaining elements (denoted as the indices $T_\perp$) are computed by Lagrange interpolation. The correctness of this generation process is then proved using $\mathcal{F}_{\mathsf{CommitAndProve}}$.

To process vouchers, the server trial decrypts the provided ciphertexts. If it succeeds, it extracts a public key ciphertext $ct_{\mathcal{J}}$ (under the judge's key) and a secret share sh, which it stores until the threshold is met. Notice that we have here replaced the KDF used in Appendix A with an explicit call to the random oracle. Once the threshold is met, the shares can be used to reconstruct the client $\mathcal{U}$'s key $\mathsf{k}_{\text{threshold},\mathcal{U}}$ and all the ciphertexts can be returned to the $\Pi_{\text{Intended-use}}$ component of Apple and sent to the $\mathcal{J}$.

Finally, Apple takes the information passed from the judge (a random value $r_{\mathsf{img_id}}$ and a ciphertext $adct_{\mathsf{img_id}}$ for each image identifier img_id) and recovers the image encryption keys. Specifically, Apple derives the per-image encryption key by passing $\mathsf{k}_{\text{threshold},\mathcal{U}}$, $r_{\mathsf{img_id}}$, and other client specific-information into the random oracle to recover a one-time pad $\mathsf{k}_{\mathsf{img_id}}$. The image encryption key can then be recovered as $\mathsf{k}_{\mathsf{img_id}} \oplus adct_{\mathsf{img_id}}$.

The full description of this protocol can be found in Figure 21.

Server's part of $\Pi_{\text{CSAM-Scan}}$

Initialize Clients: On input $\{\mathcal{U}_1, \ldots, \mathcal{U}_{m'}\}, \{\mathsf{ROHashes}_{\mathcal{U}_1}, \ldots, \mathsf{ROHashes}_{\mathcal{U}_{m'}}\}, c_{\text{NCMEC}}, \mathsf{d}, \sigma_{\text{NCMEC}}$:

1. Send $(\mathsf{RetrieveAll})$ to $\mathcal{F}_{BB}$ and find the *first* post of the form $(c_{\text{NCMEC}}, \sigma_{\text{NCMEC}})$. Then, send $(\mathsf{Verify}, c_{\text{NCMEC}}, \sigma_{\text{NCMEC}})$ to $\mathcal{F}_{\mathsf{Cert}}^{\text{NCMEC}}$. If $(\mathsf{Verified}, c_{\text{NCMEC}}, 1)$ is not received in response, return $\perp$.
2. If $\{\mathsf{ROHashes}_{\mathcal{U}_1}, \ldots, \mathsf{ROHashes}_{\mathcal{U}_{m'}}\}$ are not identical, return.
3. Send $(\mathsf{Open}, \mathsf{sid}, c_{\text{NCMEC}}, d, \mathsf{ROHashes}_{\mathcal{U}_1})$ to $\mathcal{F}_{\mathsf{CommitAndProve}}^{\mathsf{pdata}}$. If $(\mathsf{OpenReceipt}, \mathsf{sid}, c_{\text{NCMEC}}, d, \mathsf{ROHashes}_{\mathcal{U}_1}, 1)$ is not received, return.
4. For each $\mathcal{U}_j$ (for $j \in [m']$) remove any duplicates in $\mathsf{ROHashes}_{\mathcal{U}_j}$. Then construct pdata:
   (a) Sample $\alpha \xleftarrow{\$} \mathbb{F}_q$ and set $L \leftarrow g^\alpha$. Sample h_0, h_1 and compute $T \leftarrow \mathsf{CuckooTable}_{h_0,h_1}(\mathsf{ROHashes})$. Repeat this process until $T \neq \perp$.
   (b) Denote the set $i \in [n']$ where $T[i] = \perp$ as $T_\perp$ and the set $i \in [n']$ where $T[i] \neq \perp$ as $\overline{T_\perp}$.
   (c) For $i \in \overline{T_\perp}$, set $p_i \leftarrow \mathsf{HashToCurve}(T[i])^\alpha$.
   (d) For $i \in T_\perp$, set $p_{i'}$ by Lagrange interpolation in the exponent. Namely, set
   $$p_{i'} \leftarrow \prod_{i \in \overline{T_\perp}} p_i^{\mathsf{Lag}_{\overline{T_\perp}}(i', i)}$$
   where $\mathsf{Lag}_{\overline{T_\perp}}(i', i)$ outputs the ith coefficient for reconstructing the point i' from the points $\overline{T_\perp}$.
   (e) Set sid to be $\mathcal{U}_j$ and then send $(\mathsf{Prove}, \mathsf{sid}, (L, p_1, \ldots, p_{n'}, h_0, h_1), c_{\text{NCMEC}}, d, \mathsf{ROHashes}, \alpha)$ to $\mathcal{F}_{\mathsf{CommitAndProve}}^{\mathsf{pdata}}$ and get $(\mathsf{ProveReceipt}, \mathsf{sid}, (L, p_1, \ldots, p_{n'}, h_0, h_1), c_{\text{NCMEC}}, d, \mathsf{ROHashes}, \alpha, \pi_{\mathsf{pdata}}, 1)$ in return.
   (f) Send $\mathsf{pdata}_{\mathcal{U}_j} = (L, p_1, \ldots, p_{n'}, h_0, h_1, \pi_{\mathsf{pdata}})$ to the client $\mathcal{U}_j$ via $\mathcal{F}_{\mathsf{SMT}}$.

Processing Vouchers: Upon receiving $\mathsf{voucher} = (\mathsf{img_id}, q_0, ct_0, q_1, ct_1, rct, c_{\text{voucher}}, \pi_{\text{voucher}}, \sigma_{\mathcal{U}})$ from $\mathcal{U}$:

1. Send $(\mathsf{Verify}, (\mathsf{pdata}_{\mathcal{U}}, \mathsf{img_id}, q_0, ct_0, q_1, ct_1, rct, c_{\text{voucher}}, \pi_{\text{voucher}}), \sigma_{\mathcal{U}})$ to $\mathcal{F}_{\mathsf{Cert}}^{\mathcal{U}}$. If $(\mathsf{Verified}, (\mathsf{pdata}_{\mathcal{U}}, \mathsf{img_id}, q_0, ct_0, q_1, ct_1, rct, c_{\text{voucher}}, \pi_{\text{voucher}}), \sigma_{\mathcal{U}}, 1)$ is not received, return $\perp$.
2. Set $\mathsf{sid} = \mathsf{img_id}$, set $x = (\mathsf{pdata}_{\mathcal{U}}, pk_{\mathcal{J}}, (\mathsf{img_id}, q_0, ct_0, q_1, ct_1, rct))$ and send $(\mathsf{Verify}, \mathsf{sid}, x, c_{\text{voucher}}, \pi_{\text{voucher}})$ to $\mathcal{F}_{\mathsf{CommitAndProve}}^{\mathsf{voucher}}$. If $(\mathsf{VerifyReceipt}, \mathsf{sid}, x, c_{\text{voucher}}, \pi_{\text{voucher}}, 1)$ is not received, return $\perp$.
3. If this is the first voucher received from $\mathcal{U}$, then record the value c_{voucher}. Otherwise, check that the value c_{voucher} is consistent with the recorded value and return $\perp$ if it is not.
4. For $j = \{0, 1\}$:
   (a) Set $rkey \leftarrow \mathsf{Dec}(\mathsf{KDF}(q_j^\alpha), ct_j)$. If decryption fails and $j = 0$ continue and if $j = 1$ return $\perp$.
   (b) Set $(ct_{\mathcal{J}}, sh) \leftarrow \mathsf{Dec}(rkey, rct)$. Add the tuple $(\mathsf{img_id}, \mathsf{voucher}, s_j, ct_{\mathcal{J}}, sh)$ to $\mathsf{SHARES}_{\mathcal{U}}$.
5. For the index j that passed decryption, send $(\mathsf{Prove}, \mathsf{sid}, (L, q_j, \mathsf{KDF}(q_j^\alpha)), \alpha)$ to $\mathcal{F}_{NIZK}^{\mathsf{KeyDev}}$ and receive $(\mathsf{Proof}, \mathsf{sid}, \pi_{\text{KeyDev}})$ in response.
6. If $|\mathsf{SHARES}_{\mathcal{U}}| > \mathcal{T}$, Reconstruct $\mathsf{k}_{\text{threshold},\mathcal{U}}$ using $(\mathcal{T}+1)$ distinct Shamir secret shares in $\mathsf{SHARES}_{\mathcal{U}}$. Store this key.
7. Return $\mathsf{SHARES}_{\mathcal{U}}, \pi_{\text{KeyDev}}$

Recovering Keys: Upon receiving $(\mathcal{U}, \{(\mathsf{img_id}_1, (r_{\mathsf{img_id}_1}, adct_{\mathsf{img_id}_1}), \ldots, (\mathsf{img_id}_{\mathcal{T}}, (r_{\mathsf{img_id}_{\mathcal{T}}}, adct_{\mathsf{img_id}_{\mathcal{T}}})\})$ from $\mathcal{J}$:

1. Retrieve $\mathsf{k}_{\text{threshold},\mathcal{U}}$ and initialize and empty list OUTSET. Then, for each tuple $(\mathsf{img_id}_i, (r_{\mathsf{img_id}_i}, adct_{\mathsf{img_id}_i}))$:
   – Send $(\texttt{HashQuery}, \text{"imgkeygen"} \| \mathsf{k}_{\text{threshold},\mathcal{U}} \| r_{\mathsf{img_id}_i} \| \mathcal{U} \| \mathsf{img_id}_i, \lambda)$ for $\mathcal{G}_{\mathsf{pRO}}$. Receive $(\texttt{HashQuery}, \mathsf{k}_{\mathsf{img_id}_i})$ in response.
   – Set $\mathsf{OUTSET} \leftarrow \mathsf{OUTSET} \cup (\mathsf{k}_{\mathsf{img_id}_i} \oplus adct_{\mathsf{img_id}_i})$
2. Return OUTSET.

Figure 21: Description of the server's part of $\Pi_{\text{CSAM-Scan}}$

The client's part of the protocol. The client has two main components of the protocol: processing the initialization from the server and generating vouchers to upload into the system. When processing the server's initialization, $\mathcal{U}$ pulls down the commitment from $\mathcal{F}_{BB}$ and verifies the signature and proof. Additionally, it generates long-term key material $k_{\text{threshold}}$ and $hkey$ that will be used during voucher generation.

Voucher generation has the following steps: (1) generation of the per-$\mathsf{img_id}$ key $\mathsf{k}_{\mathsf{img_id}}$ using freshly sampled randomness $r_{\mathsf{img_id}}$. The key $\mathsf{k}_{\mathsf{img_id}}$ encrypts the input key k as a one-time pad to produce $adct$. The client then generates a secret share of their long-term key $k_{\text{threshold}}$ using $\mathsf{SecretShareAtPoint}$, and $(r, adct)$ is then encrypted under the judge's public key to produce $ct_{\mathcal{J}}$. This ciphertext is then used to produce a voucher in the exact same manner as in Apple's base protocol. The final change to the protocol is that the client uses the randomness β_j, γ_j to produce zero-knowledge proofs that the voucher was produced consistently (which will primarily be used in conjunction with random oracle queries for extraction).

A formal description of this protocol can be found in Figure 22.

Client's part of $\Pi_{\text{CSAM-Scan}}$

Client Initialization: On input $\mathsf{pdata} = (L, p_1, \ldots, p_{n'}, h_0, h_1, \pi_{\mathsf{pdata}})$:

1. If $L \notin \mathbb{G} \setminus \{0\}$, or $\exists\ i$ s.t. $p_i \notin \mathbb{G} \setminus \{0\}$, or $\exists\ i, j$ such that $p_i = p_j$, return $\perp$.
2. Send $(\mathsf{RetrieveAll})$ to $\mathcal{F}_{BB}$ and find the *first* post of the form $(c_{\text{NCMEC}}, \sigma_{\text{NCMEC}})$. Then, send $(\mathsf{Verify}, c_{\text{NCMEC}}, \sigma_{\text{NCMEC}})$ to $\mathcal{F}_{\mathsf{Cert}}^{\text{NCMEC}}$. If $(\mathsf{Verified}, m, 1)$ is not received in response, return $\perp$.
3. Send $(\mathsf{Verify}, \mathsf{sid}, (L, p_1, \ldots, p_{n'}, h_0, h_1), c_{\text{NCMEC}}, \pi_{\mathsf{pdata}})$ to $\mathcal{F}_{\mathsf{CommitAndProve}}^{\mathsf{pdata}}$. If $(\mathsf{VerifyReceipt}, \mathsf{sid}, (L, p_1, \ldots, p_{n'}, h_0, h_1), c_{\text{NCMEC}}, \pi_{\mathsf{pdata}}, 1)$ is not received in response, return $\perp$
4. Verify that there are only n degrees of freedom in $p_1, \ldots, p_{n'}$. Specifically, check that for $i' \in [n+1, n']$
$$p_{i'} = \prod_{i \in [n]} p_i^{\mathsf{Lag}_{[n]}(i', i)}$$
where $\mathsf{Lag}_{\{1,\ldots,n\}}(i', i)$ outputs the ith coefficient for reconstructing the point i' from the points $[n]$.
5. Choose and store an encryption key $\mathsf{k}_{\text{threshold}} \xleftarrow{\$} \{0,1\}^\lambda$, and a PRF key $hkey \xleftarrow{\$} \{0,1\}^\lambda$.
6. Set sid to be $\mathcal{U}_j$ and then send $(\mathsf{Commit}, \mathsf{sid}, (\mathsf{k}_{\text{threshold}}, hkey))$ to $\mathcal{F}_{\mathsf{CommitAndProve}}^{\mathsf{voucher}}$ and get $(\mathsf{CommitReceipt}, \mathsf{sid}, (\mathsf{k}_{\text{threshold}}, hkey), c_{\text{voucher}}, d)$ in return.

Voucher Generation: Upon input $(\mathsf{img_hash}, \mathsf{img_id}, \mathsf{k})$:

1. Send $(\mathtt{HashQuery}, \mathsf{img_hash}, \lambda)$ to $\mathcal{G}_{\mathsf{pRO}}$ and receive $(\mathtt{HashQuery}, \mathsf{ro_img_hash})$ in response.
2. Sample $r \xleftarrow{\$} \{0,1\}^\lambda$ and send $(\mathtt{HashQuery}, \text{"imgkeygen"} \| \mathsf{k}_{\text{threshold}} \| r \| \mathcal{U} \| \mathsf{img_id}, \lambda)$ for $\mathcal{G}_{\mathsf{pRO}}$. Receive $(\mathtt{HashQuery}, \mathsf{k}_{\mathsf{img_id}})$ in response.
3. Set $sh \leftarrow \mathsf{SecretShareAtPoint}(\mathsf{k}_{\text{threshold}}, \mathcal{T}, \mathsf{PRF}(hkey, \mathsf{img_id}))$.
4. Set $adct \leftarrow \mathsf{k}_{\mathsf{img_id}} \oplus \mathsf{k}$ and set $ct_{\mathcal{J}} \leftarrow \mathsf{Enc}(pk_{\mathcal{J}}, (r, adct))$.
5. Sample $rkey \xleftarrow{\$} \mathcal{K}_{\mathsf{Enc}}$ and set $rct \leftarrow \mathsf{Enc}(rkey, (ct_{\mathcal{J}}, sh))$
6. For $j \in \{0, 1\}$:
   (a) Sample $\beta_j, \gamma_j \xleftarrow{\$} \mathbb{F}_q$ and compute
   $$q_j \leftarrow \mathsf{HashToCurve}(\mathsf{ro_img_hash})^{\beta_j} \cdot g^{\gamma_j} \quad \text{and} \quad s_j \leftarrow (p_{h_j(\mathsf{ro_img_hash})})^{\beta_j} \cdot L^{\gamma_j}$$
   (b) Set $ct_j \leftarrow \mathsf{Enc}(\mathsf{KDF}(s_j), rkey)$
7. Sample $b \xleftarrow{\$} \{0,1\}$. Set $\mathsf{sid} = \mathsf{img_id}$ and set $x = (\mathsf{pdata}, pk_{\mathcal{J}}, (\mathsf{img_id}, q_b, ct_b, q_{1-b}, ct_{1-b}, rct))$ and $w' = (\mathsf{ro_img_hash}, \{\beta_j, \gamma_j, \mathsf{HashToCurve}(\mathsf{ro_img_hash}), s_j, h_j(\mathsf{ro_img_hash})\}_{j \in \{0,1\}}, r)$ and send $(\mathsf{Prove}, \mathsf{sid}, x, c_{\text{voucher}}, d, (\mathsf{k}_{\text{threshold}}, hkey), w')$ to $\mathcal{F}_{\mathsf{CommitAndProve}}^{\mathsf{voucher}}$ and get $(\mathsf{ProveReceipt}, \mathsf{sid}, x, c_{\text{voucher}}, d, (\mathsf{k}_{\text{threshold}}, hkey), w', \pi_{\text{voucher}}, 1)$ in response.
8. Send $(\mathsf{Sign}, (\mathsf{pdata}, \mathsf{img_id}, q_b, ct_b, q_{1-b}, ct_{1-b}, rct, c_{\text{voucher}}, \pi_{\text{voucher}}))$ to $\mathcal{F}_{\mathsf{Cert}}^{\mathcal{U}}$ and receive $(\mathsf{Signature}, (\mathsf{pdata}, \mathsf{img_id}, q_b, ct_b, q_{1-b}, ct_{1-b}, rct, c_{\text{voucher}}, \pi_{\text{voucher}}), \sigma_{\mathcal{U}})$ in response.
9. Send $(\mathsf{img_id}, q_b, ct_b, q_{1-b}, ct_{1-b}, rct, c_{\text{voucher}}, \pi_{\text{voucher}}, \sigma_{\mathcal{U}})$ to Apple via $\mathcal{F}_{\mathsf{SMT}}$.

Figure 22: Description of the client's part of $\Pi_{\text{CSAM-Scan}}$

The judge's part of the protocol. The judge's responsibility in $\Pi_{\text{CSAM-Scan}}$ is simply to decrypt the provided ciphertexts and prove that the decryption was done honestly. Note that within $\Pi_{\text{Intended-use}}$, the judge will also verify the proofs associated with this data, demonstrating that the ciphertexts were actually sent by a client (the protocol for doing so is described below in Figure 24). However, there is no cryptographic way to verify that this has happened, so it is relegated to $\Pi_{\text{Intended-use}}$. The formal description of this protocol can be found in Figure 23.

The judge's part of $\Pi_{\text{CSAM-Scan}}$

Decrypt Ciphertexts: Upon input $\mathsf{SHARES}_{\mathcal{U}} = \{ct_{\mathcal{J},1}, \ldots ct_{\mathcal{J},\ell}\}$,

1. For $j \in [\ell]$, set $(r_j, adct_j) \leftarrow Dec(sk_{\mathcal{J}}, ct_{\mathcal{J},j})$
2. Send $(\mathsf{Prove}, \mathsf{sid}, (\{(r_1, adct_1), \ldots, (r_\ell, adct_\ell)\}, \{ct_{\mathcal{J},1}, \ldots ct_{\mathcal{J},\ell}\}, pk_{\mathcal{J}}), (sk_{\mathcal{J}}))$ to an instance of $\mathcal{F}_{NIZK}^{\mathsf{decryption}}$ and get $(\mathsf{Proof}, \mathsf{sid}, \pi_{\text{decryption}})$ in return.
3. Return $(\{(r_1, adct_1), \ldots, (r_\ell, adct_\ell)\}, \pi_{\text{decryption}})$.

Figure 23: Description of the judge's part of $\Pi_{\text{CSAM-Scan}}$

The public part of the protocol. Finally, we specify the verification parts of the protocol that can be run by any party with access to the right information. Specifically, to certify that a set of ciphertexts were actually the product of an interaction between a client and the server, a party checks that the vouchers were properly signed by the client, the zero-knowledge proofs that the key material used to decrypt rct was generated honestly verify, and the ciphertexts were the product of correct decryption. To verify that the judge correctly decrypted these ciphertexts, anyone can verify the corresponding zero-knowledge proof. The formal description of this protocol can be found in Figure 24.

Part of $\Pi_{\text{CSAM-Scan}}$ that can be run by anyone

Verify PSI Output Ciphertexts: Upon input $(\mathcal{U}, \mathsf{pdata}, (\mathsf{voucher}_1 = (\mathsf{img_id}_1, q_{0,1}, ct_{0,1}, q_{1,1}, ct_{1,1}, rct_1, \pi_{\text{voucher},1}, \sigma_{\mathcal{U},1}), \ldots, \mathsf{voucher}_\ell), (\mathsf{k}_1, \ldots, \mathsf{k}_\ell), (ct_{\mathcal{J},1}, sh_1) \ldots (ct_{\mathcal{J},\ell}, sh_\ell))$:

1. For $i \in [\ell]$:
   - Send $(\mathsf{Verify}, (\mathsf{pdata}, \mathsf{img_id}_i, q_{0,i}, ct_{0,i}, q_{1,i}, ct_{1,i}, rct_i, \pi_{\text{voucher},i}), \sigma_{\mathcal{U},i})$ to $\mathcal{F}_{\mathsf{Cert}}^{\mathcal{U}}$. If $(\mathsf{Verified}, (\mathsf{pdata}, \mathsf{img_id}_i, q_{0,i}, ct_{0,i}, q_{1,i}, ct_{1,i}, rct_i, \pi_{\text{voucher},i}), 1)$ is not received, return 0.
   - For $j \in \{0,1\}$, send $(\mathsf{Verify}, \mathsf{sid}, (L, q_{j,i}, \mathsf{k}_i), \pi_{\text{KeyDev}})$ to $\mathcal{F}_{NIZK}^{\mathsf{KeyDev}}$. If $(\mathsf{Verification}, \mathsf{sid}, (L, q_{j,i}, \mathsf{k}_i), \pi_{\text{KeyDev}}, b \in \{0,1\})$ is not received *exactly once* in response, return 0.
   - For $j \in \{0,1\}$, verify that $(ct_{\mathcal{J},i}, sh_i) = \mathsf{Dec}(\mathsf{k}_i, rct_j)$ for exactly one value of j. If not, return 0.
2. Return 1.

Verify Judge Proof: On input $(\{(r_1, adct_1), \ldots, (r_\ell, adct_\ell)\}, \{ct_{\mathcal{J},1}, \ldots ct_{\mathcal{J},\ell}\}, pk_{\mathcal{J}}, \pi_{\text{decryption}})$:

1. Send $(\mathsf{Verify}, \mathsf{sid}, (\{(r_1, adct_1), \ldots, (r_\ell, adct_\ell)\}, \{ct_{\mathcal{J},1}, \ldots ct_{\mathcal{J},\ell}\}, pk_{\mathcal{J}}), \pi_{\text{decryption}})$ to $\mathcal{F}_{NIZK}^{\mathsf{decryption}}$ and receive $(\mathsf{Verification}, \mathsf{sid}, (\{(r_1, adct_1), \ldots, (r_\ell, adct_\ell)\}, \{ct_{\mathcal{J},1}, \ldots ct_{\mathcal{J},\ell}\}, pk_{\mathcal{J}}), \pi_{\text{decryption}}, b \in \{0,1\})$ in response.
2. Return b.

Figure 24: Description of a part of $\Pi_{\text{CSAM-Scan}}$ that could be run by anyone.

C Proof that $\Pi_{\text{CSAM-Scan}}$ realizes $\mathcal{F}_{\text{CSAM-Scan}}$

We now prove that $\Pi_{\text{CSAM-Scan}}$ UC-realizes $\mathcal{F}_{\text{CSAM-Scan}}$. Specifically we prove the following theorem:

Theorem 1. *$\Pi_{CSAM\text{-}Scan}$ UC-realizes $\mathcal{F}_{CSAM\text{-}Scan}$ in the $(\mathcal{F}_{BB}, \mathcal{F}_{\mathsf{Cert}}, \mathcal{F}_{\mathsf{CommitAndProve}}, \mathcal{G}_{\mathsf{pRO}}, \mathcal{F}_{NIZK}, \mathcal{F}_{\mathsf{SMT}})$-hybrid model with static corruptions assuming a secure symmetric key encryption scheme with random key robustness, a secure public key encryption scheme, and a pseudorandom function.*

We do this by giving a simulator for each party in the protocol. When the adversary corrupts multiple of these parties, the simulator for the join of them is trivial, so we do not include them here. This is easily seen by observing that clients have no joint information and no interaction (except via the bulletin board, which only requires a single message to be posted for all parties). When both the server and a subset of the clients are corrupt, then making their views consistent is trivial.

The simulator for the server is in Appendix C.1, the simulator for the client is in Appendix C.2, the simulator for the judge is in Appendix C.3, the simulator for NCMEC is in Appendix C.4, and the protocol for the public parts of the protocol is in Appendix C.5.

C.1 Simulating the Server

We now prove that the simulation of the server is computationally close to the server's real view of the protocol. The protocol run by the server can be found in Figure 21. The simulator for the server can be found in Figure 25.

Hybrid_0: Let Hybrid_0 denote the real world experiment.

Hybrid_1: For each non-matching voucher, i.e., vouchers for which $\mathcal{F}_{\text{CSAM-Scan}}$ sends the message

$$(\mathsf{ScanImageComplete}, \mathcal{U}, \mathsf{img_id}, \mathsf{False}),$$

generate the voucher at random. That is, sample a random image hash $\mathsf{img_hash}$ and then generate the voucher honestly. Notice that the probability that $\mathsf{img_hash}$ is a match for pdata is inversely proportional to the size of the image hash space (which is exponential in the security parameter). Therefore, by the security argument in Apple's initial paper [App21a], $|\mathsf{Hybrid}_1 - \mathsf{Hybrid}_0|$ is negligible in the security parameter.

Hybrid_2: For each matching voucher, instead let the proof π_{voucher} be simulated. By the properties of the zero-knowledge proof, $|\mathsf{Hybrid}_2 - \mathsf{Hybrid}_1|$ is negligible in the security parameter.

Hybrid_3: Let Hybrid_3 be the same as Hybrid_2 except generate the voucher with respect to a uniformly selected element $\mathsf{img_hash} \in X$ instead of using the true input hash value. Notice that by the Naor-Reingold transformation (see [App21a]), the resulting voucher is distributed independently of the input hash. That is, if $\mathsf{HashToCurve}(\mathsf{ro_img_hash}), L, p_{h_j(\mathsf{ro_img_hash})}$ is a DDH tuple, then the result will be a uniform DDH tuple, and if $\mathsf{HashToCurve}(\mathsf{ro_img_hash}), L, p_{h_j(\mathsf{ro_img_hash})}$ is not a DDH tuple then the result is three uniform elements in the group. Thus, $|\mathsf{Hybrid}_3 - \mathsf{Hybrid}_2|$ is negligible in the security parameter.

Hybrid_4: Let Hybrid_4 be the same as Hybrid_3 except that for each matching voucher v_i sample $adct_i \xleftarrow{\$} \{0,1\}^\lambda$. Then, before the final voucher is sent to the server, program the Random Oracle such that $adct_i$ will decrypt to k_i. We argue that $|\mathsf{Hybrid}_4 - \mathsf{Hybrid}_3|$ is negligible in two parts: (1) before there are $\mathcal{T}$ matches, the view of the server is the same as in Hybrid_3 because it has no information about $\mathsf{k}_{\text{threshold}}$ by the privacy of Shamir Secret sharing. (2) After there are $\mathcal{T}$ matches, the view of the server is the same as in Hybrid_3 unless it queried the random oracle before programming. Note that r and $\mathsf{k}_{\text{threshold}}$ are sampled at random from $\{0,1\}^\lambda$, so the probability that the server queried the random oracle on one of the $\mathcal{T}$ points of interest is $< \frac{q\mathcal{T}}{2^\lambda} = \mathsf{poly}(\lambda)$, where q is the number of queries the server has made to the random oracle. Thus, $|\mathsf{Hybrid}_4 - \mathsf{Hybrid}_3|$ is negligible in the security parameter.

Hybrid_5: For each matching voucher, instead let the proof π_{voucher} be computed honestly. Notice that this is a valid proof, simply with a different witness. By the properties of the zero-knowledge proof, $|\mathsf{Hybrid}_5 - \mathsf{Hybrid}_4|$ is negligible in the security parameter.

$\mathsf{Sim}_{\text{server}}$: Simulating the Server

Setup of $\mathcal{S}_{PSI}$: During setup, $\mathsf{Sim}_{\text{server}}$ sets up the string generation algorithm $\mathcal{S}_{PSI}$ such that it uses the same randomness as in **Simulating Voucher Creation** below. For a client $\mathcal{U}$, $\tau_{\mathcal{U}}$ is the associated public key encryption generated below and $\pi_{\mathcal{U}}$ is generated in the same manner as the proof string is generated by the simulator for $\mathcal{F}_{NIZK}^{\mathsf{KeyDev}}$.

Notification of Hashes Specification: Upon receiving (HashesSpecified) from $\mathcal{F}_{\text{CSAM-Scan}}$, simulate (1) generation of a commitment string c_{NCMEC} (as $\mathcal{F}_{\mathsf{CommitAndProve}}$), (2) a signature string σ_{NCMEC} (as $\mathcal{F}_{\mathsf{Cert}}$), and (3) posting $(c, \sigma_{\text{NCMEC}})$ (as $\mathcal{F}_{BB}$). Then record c_{NCMEC} and send (OK) to $\mathcal{F}_{\text{CSAM-Scan}}$.

Receiving Hashes from NCMEC: Upon receiving $(\mathsf{ShareHashes}, \mathsf{ROHashes}_{\text{NCMEC}}, d_{\text{NCMEC}})$ from NCMEC via an instance of $\mathcal{F}_{\mathsf{SMT}}$, record $(\mathsf{ROHashes}_{\text{NCMEC}}, d_{\text{NCMEC}})$ and forward the message.

Extracting Hashes: Upon initialization, create an empty set of potential witnesses W.

1. Upon receiving a message $(\mathsf{Prove}, \mathsf{sid}, (L, p_1, \ldots, p_{n'}, h_0, h_1), c, d, \mathsf{ROHashes}, \alpha)$ from Apple for $\mathcal{F}_{\mathsf{CommitAndProve}}$, if $c \neq c_{\text{NCMEC}}$ or $d \neq d_{\text{NCMEC}}$, then ignore the message. Otherwise, $\mathsf{Sim}_{\text{server}}$ simulates $\mathcal{F}_{\mathsf{CommitAndProve}}$ to create the proof string π_{pdata} add the tuple $(c, d, \mathsf{ROHashes}, \alpha, \mathsf{pdata} = (L, p_1, \ldots, p_{n'}, \pi_{\mathsf{pdata}}))$ to W.
2. Upon receiving a message $\mathsf{pdata} = (L, p_1, \ldots, p_{n'}, h_0, h_1, \pi_{\mathsf{pdata}})$ from Apple for a client $\mathcal{U}$, then:
   (a) If there exists no matching entry $(c, d, \mathsf{ROHashes}, \alpha, \mathsf{pdata} = (L, p_1, \ldots, p_{n'}, \pi_{\mathsf{pdata}}))$ in W, ignore the message.
   (b) Otherwise, send $(\mathsf{InitScan}, \{\mathcal{U}\}, \{\mathsf{ROHashes}\}, \mathsf{d}_{\text{NCMEC}})$ to $\mathcal{F}_{\text{CSAM-Scan}}$.
3. When prompted by $\mathcal{F}_{\text{CSAM-Scan}}$, $\mathsf{Sim}_{\text{server}}$ passes the appropriate values of h_0, h_1.

Simulating Voucher Creation: Upon initialization, sample and record keys $\mathsf{k}_{\text{threshold},\mathcal{U}}$ and $hkey_{\mathcal{U}}$ for each client $\mathcal{U}$ and a keypair $pk_{\mathcal{J}}, sk_{\mathcal{J}}$ for the judge (If the judge is also correct, use the Judge's public key directly). Also initialize an empty set $\mathsf{BannedROQueries} = \emptyset$. $\mathsf{Sim}_{\text{server}}$ considers three cases:

1. Upon receiving $(\mathsf{ScanImageComplete}, \mathcal{U}, \mathsf{img_id}, \mathsf{true})$:
   (a) Sample $\mathsf{img_hash}$ from X and generate a voucher honestly as in Figure 22. Let r be the value sampled in step 1 of **Voucher Generation**.
   (b) Add "imgkeygen"$\|\mathsf{k}_{\text{threshold},\mathcal{U}}\|r\|\mathcal{U}\|\mathsf{img_id}$ to $\mathsf{BannedROQueries}$ and record $(adct_{\mathsf{img_id}}, r_{\mathsf{img_id}})$.
   (c) Send the voucher to Apple.
2. Upon receiving $(\mathsf{ScanImageComplete}, \mathcal{U}, \mathsf{img_id}, \mathsf{false})$:
   (a) Sample a uniform $\mathsf{img_hash}$ from the hash space, generate the voucher honestly with random input key, and send it to Apple.
3. Upon receiving $(\mathsf{ThresholdMet}, \mathcal{U}, \mathsf{img_id}, \pi_{\mathcal{U}}, \tau_{\mathcal{U}})$:
   (a) If a message ThresholdMet has been previously received for $\mathcal{U}$, $\mathsf{Sim}_{\text{server}}$ samples an additional voucher honestly. Otherwise, $\mathsf{Sim}_{\text{server}}$ follows the voucher creation procedure listed above in response to the message $(\mathsf{ScanImageComplete}, \mathcal{U}, \mathsf{img_id}, \mathsf{true})$.

Equivocating Vouchers: If both Apple and $\mathcal{J}$ are corrupt, then we must equivocate as soon as $\mathcal{J}$ gets access to the keys. Otherwise, we equivocate when an honest $\mathcal{J}$ sends the key information to Apple. Specifically:

1. If $\mathcal{J}$ is corrupt, then run the following when $(\mathsf{ApproveComplete}, \mathcal{U}, \mathsf{img_id_list} = \{\mathsf{img_id}_1, \ldots, \mathsf{img_id}_\ell\}, \mathsf{keys} = \{\mathsf{k}_1, \ldots, \mathsf{k}_\ell\}, \pi_{\mathcal{U}}^{\mathcal{J}})$ is received from $\mathcal{F}_{\text{CSAM-Scan}}$. Otherwise, run the following when $(\mathsf{CaseApproved}, \mathcal{U}, \tau_{\mathcal{U}}, \mathsf{img_id_list} = \{\mathsf{img_id}_1, \ldots, \mathsf{img_id}_\ell\}, \mathsf{keys} = \{\mathsf{k}_1, \ldots, \mathsf{k}_\ell\}, \pi_{\mathcal{U}}^{\mathcal{J}})$ is received from $\mathcal{J}$, for $j \in [\ell]$:
   (a) Retrieve appropriate $\mathsf{k}_{\text{threshold},\mathcal{U}}, adct_{\mathsf{img_id}_j}, r_{\mathsf{img_id}_j}$ and run $\mathsf{k}_{\mathsf{img_id}_j} \leftarrow (adct_{\mathsf{img_id}_j} \oplus \mathsf{k}_j)$
   (b) Send $(\texttt{Program}, \text{"imgkeygen"}\|\mathsf{k}_{\text{threshold},\mathcal{U}}\|r_{\mathsf{img_id}}\|\mathcal{U}\|\mathsf{img_id}, \mathsf{k}_{\mathsf{img_id}})$ to $\mathcal{G}_{\mathsf{pRO}}$ and receive $(\texttt{Program})$ in response. $\mathsf{Sim}_{\text{server}}$ then ensures that $\mathcal{G}_{\mathsf{pRO}}$ was programmed, and aborts the simulation with an error if it is not. Finally, remove "imgkeygen"$\|\mathsf{k}_{\text{threshold},\mathcal{U}}\|r_{\mathsf{img_id}}\|\mathcal{U}\|\mathsf{img_id}$ from $\mathsf{BannedROQueries}$.
2. Then send $(\mathcal{U}, \{(\mathsf{img_id}_1, (r_{\mathsf{img_id}_1}, adct_{\mathsf{img_id}_1}), \ldots, (\mathsf{img_id}_\ell, (r_{\mathsf{img_id}_\ell}, adct_{\mathsf{img_id}_\ell})\})$ to Apple.

Random Oracle Queries: Upon receiving $(\texttt{HashQuery}, m, \cdot)$ from Apple for $\mathcal{G}_{\mathsf{pRO}}$, if $m \in \mathsf{BannedROQueries}$, abort the simulation with an error. Otherwise, forward queries as usual.

Figure 25: Simulator $\mathsf{Sim}_{\text{server}}$ for simulating the view of NCMEC.

Finally, notice that Hybrid_5 is distributed the same as the ideal experiment with two exceptions: (Case 1) if Env prompts an honest client on an input that is a collision on the composition of random oracle and the function HashToCurve, or (Case 2) if Env prompts an honest client on an input that matches one of the interpolated elements of pdata. We can show that each case happens with only negligible probability. To do so, we show that an environment Env that meets one of these cases could be used, when interacting with an alternative ideal functionality $\mathcal{F}'_{\text{CSAM-Scan}}$, to do something that should only be possible with negligible probability. For each case, $\mathcal{F}'_{\text{CSAM-Scan}}$ is a copy of $\mathcal{F}_{\text{CSAM-Scan}}$ with several lines added, which can be found in Figure 26. The added lines are in red, with the cases for which those lines are active denoted at the beginning of the line.

$\mathcal{F}'_{\text{CSAM-Scan}}$

...

Initializing Clients: On receiving $(\mathsf{InitScan}, \{\mathcal{U}_1, \ldots, \mathcal{U}_\ell\}, \{\mathsf{ROHashes}_{\mathcal{U}_1}, \ldots, \mathsf{ROHashes}_{\mathcal{U}_\ell}\}, \mathsf{d}')$ from Apple:

1. If $\mathsf{HashesSpecified} \neq \mathsf{True}$, ignore the request.
2. For each $\mathcal{U}_i \in \{\mathcal{U}_1, \ldots, \mathcal{U}_\ell\}$:
   (a) If $(\mathsf{ROHashes}_{\mathcal{U}_i}, \mathsf{d}')$ does not match the stored value $(\mathsf{ROHashes}, \mathsf{d})$ or $\mathsf{IsInited}_{\mathcal{U}_i} = \mathsf{True}$, continue. Otherwise, set $\mathsf{IsInited}_{\mathcal{U}_i} = \mathsf{True}$.
   (b) (Case 2) If Apple is corrupt, request ϵn elements in $\mathbb{G}$ from $\mathcal{S}$ and add those elements to $\mathsf{Watchlist}_{\mathcal{U}_i}$.
3. Send $(\mathsf{InitScanComplete})$ to Apple.

...

Scan Image: On receiving $(\mathsf{ScanImage}, \mathsf{img_id}, \mathsf{img_hash}, \mathsf{k}, \mathsf{valid} \in \{\mathsf{true}, \mathsf{false}\})$ from a client $\mathcal{U}$:

1. If $\mathsf{IsCounting}_{\mathcal{U}} \neq \mathsf{True}$, ignore the request. If this is the first image received from $\mathcal{U}$, initialize variable $\mathsf{counter}_{\mathcal{U}} = 0$, list $\mathsf{img_id_list}_{\mathcal{U}} = \emptyset$, and list $\mathsf{keys}_{\mathcal{U}} = \emptyset$.
2. (Case 1 & 2) Send $(\mathtt{HashQuery}, \mathsf{img_hash}, \lambda)$ to $\mathcal{G}_{\mathsf{pRO}}$ and receive $(\mathtt{HashQuery}, \mathsf{ro_img_hash})$ in response.
3. (Case 1) If $\mathsf{img_hash} \notin X$ but $\exists x \in X$ such that $\mathsf{HashToCurve}(\mathsf{ro_img_hash}) = \mathsf{HashToCurve}(x)$ send $(\mathsf{CollisionError}, \mathsf{ro_img_hash}, x)$ to $\mathcal{S}$.
4. (Case 2) If $\mathsf{HashToCurve}(\mathsf{ro_img_hash}) \in \mathsf{Watchlist}_{\mathcal{U}}$, send $(\mathsf{WatchlistError}, \mathsf{img_hash}, \mathsf{HashToCurve}(\mathsf{ro_img_hash}))$ to $\mathcal{S}$.
5. If $\mathsf{img_hash} \notin X$ or $\mathsf{valid} = \mathsf{false}$, send $(\mathsf{ScanImageComplete}, \mathcal{U}, \mathsf{img_id}, \mathsf{false})$ to Apple.
6. If $\mathsf{img_hash} \in X$, increment $\mathsf{counter}_{\mathcal{U}}$, append $\mathsf{img_id}$ to $\mathsf{img_id_list}_{\mathcal{U}}$ and append k to $\mathsf{keys}_{\mathcal{U}}$. Then,
   (a) If $\mathsf{counter}_{\mathcal{U}} < \mathcal{T}$ send $(\mathsf{ScanImageComplete}, \mathcal{U}, \mathsf{img_id}, \mathsf{true})$ to Apple.
   (b) If $\mathsf{counter}_{\mathcal{U}} \geq \mathcal{T}$:
      – Generate a scan verification string and judge approval token as $(\pi_{\mathcal{U}}, \tau_{\mathcal{U}}) \leftarrow \mathcal{S}_{PSI}(\mathcal{U})$.
      – Record $(\mathcal{U}, \pi_{\mathcal{U}}, \tau_{\mathcal{U}})$ and send $(\mathsf{ThresholdMet}, \mathcal{U}, \mathsf{img_id}, \pi_{\mathcal{U}}, \tau_{\mathcal{U}})$ to Apple. If both $\mathcal{J}$ and Apple are corrupt, send $(\mathsf{ThresholdMet}, \mathcal{U}, \mathsf{img_id}, \pi_{\mathcal{U}}, \tau_{\mathcal{U}}, \mathsf{img_id_list}, \mathsf{keys}, \pi^{\mathcal{J}}_{\mathcal{U}})$ to $\mathcal{S}$ instead.

...

Figure 26: $\mathcal{F}'_{\text{CSAM-Scan}}$, a modified version of $\mathcal{F}_{\text{CSAM-Scan}}$ used to prove that $\Pi_{\text{CSAM-Scan}}$ realizes $\mathcal{F}_{\text{CSAM-Scan}}$. The differences between the two functionalities is highlighted in red.

– Case 1: Notice that if Env does not prompt an honest client on an input that is a collision on the composition of random oracle and the function HashToCurve, then it cannot distinguish between $\mathcal{F}'_{\text{CSAM-Scan}}$ and $\mathcal{F}_{\text{CSAM-Scan}}$, as they have the same functionality, and thus Env cannot change its strategy in its interactions. In the event that Env does prompt an honest client on an input that is a collision on the composition of random oracle and the function HashToCurve, notice that the simulator receives a message $(\mathsf{CollisionError}, \mathsf{ro_img_hash}, x)$, and that $\mathsf{ro_img_hash}, x$ will be a collision with respect to HashToCurve. Recall that HashToCurve is collision resistant, thereby causing a contradiction with all but negligible probability. Therefore, the probability that Env prompts an honest client on an input that is a collision on the composition of random oracle and the function HashToCurve is negligible.

– Case 2: Similarly, for case 2, notice that the Env can only distinguish between $\mathcal{F}'_{\text{CSAM-Scan}}$ and $\mathcal{F}_{\text{CSAM-Scan}}$ if it prompts an honest client such that the Watchlist is triggered. To set the Watchlist, when $\mathcal{S}$ receives a message

$$\mathsf{pdata} = (L, p_1, \ldots, p_{n'}, h_0, h_1, \pi_{\mathsf{pdata}})$$

from Apple, it uses the previous calls to $\mathcal{F}_{\mathsf{CommitAndProve}}$ to extract the set ROHashes. Using these values, note that it can determine the set of $n \subset [n']$ that correspond to the values $\mathsf{HashToCurve}(\mathsf{ro_img_hash})^\alpha$ for $\mathsf{ro_img_hash} \in \mathsf{ROHashes}$. As we did in Figure 21, denote this set of indices as $\overline{T_\perp}$, and its complement $[n'] \setminus \overline{T_\perp}$ as $T_\perp$. $\mathcal{S}$ sets the Watchlist as $\{p_i^{\alpha^{-1}} | i \in T_\perp\}$.

If during an interaction with $\mathcal{F}'_{\text{CSAM-Scan}}$ Env causes $\mathcal{F}'_{\text{CSAM-Scan}}$ to output the message

$$(\mathsf{WatchlistError}, \mathsf{img_hash}, \mathsf{HashToCurve}(\mathsf{ro_img_hash})),$$

note that img_hash is a preimage for the random oracle for ro_img_hash, which in turn is a preimage for HashToCurve for some value $p_i^{\alpha^{-1}}$ for $i \in T_\perp$. But, by the soundness of the zero-knowledge proof p_i is computed deterministically from the elements $\{p_i | i \in \overline{T_\perp}\}$, up to re-orders of the elements introduced by Apple's choice of h_0, h_1. For each possible choice of h_0, h_1, this fixes the values of $p_i^{\alpha^{-1}}$ for $i \in T_\perp$. Given that Env is polynomial time, this means it can produce at most a polynomial set of values $p_i^{\alpha^{-1}}$ by sampling different values of h_0, h_1. Thus, img_hash must constitute a value such that when evaluated on the random oracle and HashToCurve has some fixed value within some polynomial set of values. Clearly, given the random oracle's distributional properties, this can only happen with negligible probability.

Thus, by the hybrid lemma, the distribution of the view of the server in the real world and the view of Apple in the ideal world are computationally close.

C.2 Simulating A Client

We now prove that the simulation of a client is computationally close to the client's real view of the protocol. The protocol run by the client can be found in Figure 22. The simulator for the client can be found in Figure 27.

Hybrid_0: Let Hybrid_0 denote the real world experiment.

Hybrid_1: Let Hybrid_1 be the same as Hybrid_0, but the zero-knowledge proof π_{pdata} is simulated. By the zero-knowledge property of the zero-knowledge proof, $|\mathsf{Hybrid}_1 - \mathsf{Hybrid}_0|$ is negligibly small in the security parameter.

Hybrid_2: Let Hybrid_2 be the same as Hybrid_1, except the commitment c_{NCMEC} is performed with respect to a uniformly selected set. By the hiding property of the commit and prove functionality, $|\mathsf{Hybrid}_2 - \mathsf{Hybrid}_1|$ is negligibly small in the security parameter.

Hybrid_3: Let Hybrid_3 be the same as Hybrid_2, except pdata is generated with respect to a uniform set. We argue that Hybrid_3 and Hybrid_2 are close in a series of n sub-hybrids, $\mathsf{Hybrid}_{2.1}, \ldots, \mathsf{Hybrid}_{2.n}$, within which each one element of the set is swapped out. The argument for each of these holds by the same argument outlined in [App21a] (note that the random oracle in our construction is evaluated by NCMEC when committing to their input set). Thus, $|\mathsf{Hybrid}_3 - \mathsf{Hybrid}_2|$ is negligibly small in the security parameter.

Note that Hybrid_3 is distributed in the same way as the simulator $\mathsf{Sim}_{\text{client}}$ described in Figure 27. Thus, by the hybrid lemma, the distribution of the view of the client in the real world and the view of the client in the ideal world are computationally close.

$\mathsf{Sim}_{\text{client}}$: Simulating the Client

Simulating NCMEC's Commitment: Upon receiving ($\mathsf{HashesSpecified}$), simulate the creation of a commitment string c_{NCMEC} (i.e., the Commit Phase of $\mathcal{F}^{\mathsf{pdata}}_{\mathsf{CommitAndProve}}$), simulate an interaction with $\mathcal{F}^{\text{NCMEC}}_{\mathsf{Cert}}$ to generate a signature σ_{NCMEC}, and finally simulate posting $c_{\text{NCMEC}}, \sigma_{\text{NCMEC}}$ to the $\mathcal{F}_{BB}$.

Client Initialization: When the client makes a call to $\mathcal{F}_{\mathsf{SMT}}$ to receive pdata from Apple, $\mathsf{Sim}_{\text{client}}$ sends the message ($\mathsf{ProcessInit}$) to $\mathcal{F}_{\text{CSAM-Scan}}$. If ($\mathsf{ProcessInitComplete}, |X|, h_0, h_1$) is received in response, then:

1. Rejection sample a set X' of size n such that it would fit into a Cuckoo Table with has functions h_0, h_1 and generate $\mathsf{pdata} = (L, p_1, \ldots, p_{n'}, h_0, h_1, \pi_{\mathsf{pdata}})$ honestly and send it to the client. Record the secret key value α.

Extracting from a Voucher: Upon receiving $\mathsf{voucher} = (\mathsf{img_id}, q_0, ct_0, q_1, ct_1, rct, c_{\text{voucher}}, \pi_{\text{voucher}}, \sigma_{\mathcal{U}})$ from $\mathcal{U}$, $\mathsf{Sim}_{\text{client}}$ does the following:

1. If this is the first voucher received from $\mathcal{U}$, then record the value c_{voucher}. Otherwise, check that the value c_{voucher} is consistent with the recorded value. If not, sample uniform $\mathsf{img_hash}$ and k and send ($\mathsf{ScanImage}, \mathsf{img_id}, \mathsf{img_hash}, \mathsf{k}, \mathsf{false}$) to $\mathcal{F}_{\text{CSAM-Scan}}$.
2. Search for a recorded query to $\mathcal{F}^{\mathsf{voucher}}_{\mathsf{CommitAndProve}}$ with response π_{voucher}. If no such record exists sample uniform $\mathsf{img_hash}$ and k and send ($\mathsf{ScanImage}, \mathsf{img_id}, \mathsf{img_hash}, \mathsf{k}, \mathsf{false}$) to $\mathcal{F}_{\text{CSAM-Scan}}$. Otherwise, extract the values $\mathsf{img_hash}, \mathsf{k}_{\text{threshold}}, r, adct$ from the recorded queries.
3. Query $\mathcal{G}_{\mathsf{pRO}}$ on ($\mathtt{HashQuery}$, "imgkeygen" $\|\mathsf{k}_{\text{threshold}}\|r\|\mathcal{U}\|\mathsf{img_id}, \lambda$) and get $\mathsf{k}_{\mathsf{img_id}}$ in response.
4. Send ($\mathsf{ScanImage}, \mathsf{img_id}, \mathsf{img_hash}, adct \oplus \mathsf{k}_{\mathsf{img_id}}, \mathsf{true}$) to $\mathcal{F}_{\text{CSAM-Scan}}$.

$\mathcal{F}^{\mathsf{voucher}}_{\mathsf{CommitAndProve}}$ **Queries:** On queries to $\mathcal{F}^{\mathsf{voucher}}_{\mathsf{CommitAndProve}}$, $\mathsf{Sim}_{\text{client}}$ records the queries and responses.

Random Oracle Queries: On queries ($\mathtt{HashQuery}$, "imgkeygen" $\|\mathsf{k}_{\text{threshold}}\|r\|\mathcal{U}\|\mathsf{img_id}, \lambda$) to $\mathcal{G}_{\mathsf{pRO}}$, $\mathsf{Sim}_{\text{client}}$ records the query and the response $\mathsf{k}_{\mathsf{img_id}}$.

Figure 27: Simulator $\mathsf{Sim}_{\text{client}}$ for simulating the view of a client.

C.3 Simulating the Judge

We now prove that the simulation of the judge is computationally close to the client's real view of the protocol. The protocol run by the judge can be found in Figure 23. The simulator for the judge can be found in Figure 28.

Notice that the simulation is distributed exactly according to the protocol, providing that the judge does not make any banned queries to the random oracle. But, the banned queries are sampled with high entropy, so the probability that the judge queries one of these before receiving a $\tau_{\mathcal{U}}$ from the server is negligible. After receiving the value, the simulator removes the associated queries from the banned list and can program the random oracle appropriately. Therefore, the simulation and the real protocol are computationally close.

$\mathsf{Sim}_{\mathcal{J}}$: Simulating the Judge

Setup of $\mathcal{S}_{PSI}$: If the server is corrupt, then $\mathcal{S}_{PSI}$ is setup as in $\mathsf{Sim}_{\text{server}}$ above. Otherwise, for a client $\mathcal{U}$, $\tau_{\mathcal{U}}$ is a set of $\mathsf{counter}_{\mathcal{U}}$ public key encryptions under the judge's public keys to known, client-dependent, random values $r, adct$. $\pi_{\mathcal{U}}$ is computed using the same string generation algorithm as $\mathcal{F}_{NIZK}^{\mathsf{KeyDev}}$. Not that this will require sampling a valid looking statement for $\mathcal{F}_{NIZK}^{\mathsf{KeyDev}}$, which can be done by having $\mathsf{Sim}_{\mathcal{J}}$ generate values $\mathsf{k}_{\text{threshold},\mathcal{U}}$ and $hkey_{\mathcal{U}}$ for each honest client as well as key material for the honest server.

Random Oracle Queries: For all honest clients (for which the simulator has sampled key material $\mathsf{k}_{\text{threshold},\mathcal{U}}$), add all "imgkeygen"$\|\mathsf{k}_{\text{threshold},\mathcal{U}}\|r\|\mathcal{U}\|\cdot$ to $\mathsf{BannedROQueries}$, where the values of r are determined by the same randomness used to $\mathcal{S}_{PSI}$. Then, upon receiving $(\texttt{HashQuery}, m, \cdot)$ from Apple for $\mathcal{G}_{\mathsf{pRO}}$, if $m \in \mathsf{BannedROQueries}$, abort the simulation with an error. Otherwise, forward queries as usual.

Setup of $\mathcal{S}_{\mathcal{J}}$: During setup, $\mathsf{Sim}_{\mathcal{J}}$ sets $\mathcal{S}_{\mathcal{J}}$ according to the string generation algorithm of $\mathcal{F}_{NIZK}^{\mathsf{decryption}}$.

Equivocating Vouchers: If both Apple and $\mathcal{J}$ are corrupt, then we must equivocate as soon as the threshold is met. This case is handled by Figure 25. Thus, we only need to handle the case when only the judge is corrupted (ie., when the server is honest). There are two ways in which the simulator can detect that it is time to equivocate: either (1) $\mathcal{J}$ makes a valid call to $\mathcal{F}_{NIZK}^{\mathsf{decryption}}$, or (2) $\mathcal{J}$ makes a call to the random oracle that would allow it to re-derive the output key. More specifically:

(1) When $\mathcal{J}$ sends a message $(\mathsf{Prove}, \mathsf{sid}, (\{(r_1, adct_1).\ldots,(r_\ell, adct_\ell)\}, \{ct_{\mathcal{J},1}, \ldots ct_{\mathcal{J},\ell}\}, pk_{\mathcal{J}}), (sk_{\mathcal{J}}))$ to $\mathcal{F}_{NIZK}^{\mathsf{decryption}}$, $\mathsf{Sim}_{\mathcal{J}}$ forwards the message to $\mathcal{F}_{NIZK}^{\mathsf{decryption}}$. If the proof is valid and the values $ct_{\mathcal{J},1}, \ldots ct_{\mathcal{J},\ell}$ match those generated by $\mathcal{S}_{PSI}$, then $\mathsf{Sim}_{\mathcal{J}}$ send $(\mathsf{Approve}, \mathcal{U}, \tau_{\mathcal{U}})$ to $\mathcal{F}_{\text{CSAM-Scan}}$ and receives $(\mathsf{ApproveComplete}, \mathcal{U}, \mathsf{img_id_list}, \mathsf{keys}, \pi_{\mathcal{U}}^{\mathcal{J}})$ in response. It then proceeds to the **Equivocation Procedure** described below.

(2) When $\mathcal{J}$ receives a set of ciphertext $\{ct_{\mathcal{J},1}, \ldots ct_{\mathcal{J},\ell}\}$ that match those generated by $\mathcal{S}_{PSI}$, remove the associated entries from $\mathsf{BannedROQueries}$. Then, if $\mathcal{J}$ sends a $\texttt{HashQuery}$ to $\mathcal{G}_{\mathsf{pRO}}$ on one of these values, $\mathsf{Sim}_{\mathcal{J}}$ send $(\mathsf{Approve}, \mathcal{U}, \tau_{\mathcal{U}})$ to $\mathcal{F}_{\text{CSAM-Scan}}$ and receives $(\mathsf{ApproveComplete}, \mathcal{U}, \mathsf{img_id_list}, \mathsf{keys}, \pi_{\mathcal{U}}^{\mathcal{J}})$ in response. It then proceeds to the **Equivocation Procedure** described below before forwarding the request.

(3) **Equivocation Procedure**: Let $\mathsf{img_id_list} = \{\mathsf{img_id}_1, \ldots, \mathsf{img_id}_\ell\}, \mathsf{keys} = \{\mathsf{k}_1, \ldots, \mathsf{k}_\ell\}$. For each $j \in [\ell]$:

(a) Retrieve appropriate $\mathsf{k}_{\text{threshold},\mathcal{U}}, adct_{\mathsf{img_id}_j}, r_{\mathsf{img_id}_j}$ and run $\mathsf{k}_{\mathsf{img_id}_j} \leftarrow (adct_{\mathsf{img_id}_j} \oplus \mathsf{k}_j)$

(b) Send $(\texttt{Program}, \text{"imgkeygen"}\|\mathsf{k}_{\text{threshold},\mathcal{U}}\|r_{\mathsf{img_id}}\|\mathcal{U}\|\mathsf{img_id}, \mathsf{k}_{\mathsf{img_id}})$ to $\mathcal{G}_{\mathsf{pRO}}$ and receive $(\texttt{Program})$ in response. $\mathsf{Sim}_{\text{server}}$ then ensures that $\mathcal{G}_{\mathsf{pRO}}$ was programmed, and aborts the simulation with an error if it is not.

Figure 28: Simulator $\mathsf{Sim}_{\mathcal{J}}$ for simulating the view of the judge.

C.4 Simulating NCMEC

We provide the simulator $\mathsf{Sim}_{\text{NCMEC}}$ for simulating the view of NCMEC. Note that NCMEC has no output and receives no messages from other parties. Thus, the only issue here is extracting their input. This is trivial given the use of the hybrid model. The description of $\mathsf{Sim}_{\text{NCMEC}}$ can be found in Figure 29.

Note that the simulator is distributed exactly as the real protocol, providing that there are no collisions in the random oracle queries. This only happens with negligible probability, given the distribution of the random oracle.

$\mathsf{Sim}_{\text{NCMEC}}$: Simulating NCMEC

Random Oracle Queries: Upon receiving $(\texttt{HashQuery}, m, \cdot)$ from NCMEC for $\mathcal{G}_{\mathsf{pRO}}$, forward the message normally. $\mathsf{Sim}_{\text{NCMEC}}$ then records a tuple (m, h) where h is the response.

Extracting Committed Values: $\mathsf{Sim}_{\text{NCMEC}}$ extracts the values X as follows:

1. When NCMEC sends a message $(\mathsf{Commit}, \mathsf{sid}, \mathsf{ROHashes})$ to the commitment interface of $\mathcal{F}^{\mathsf{pdata}}_{\mathsf{CommitAndProve}}$, $\mathsf{Sim}_{\text{NCMEC}}$ forwards the messages as normal and records a tuple $(\mathsf{ROHashes}, c)$, where c is the resulting commitment string.
2. $\mathsf{Sim}_{\text{NCMEC}}$ forwards messages to $\mathcal{F}^{\text{NCMEC}}_{\mathsf{Cert}}$ as normal.
3. When NCMEC sends a message $(\mathsf{Post}, (c_{\text{NCMEC}}, \sigma_{\text{NCMEC}}))$ to $\mathcal{F}_{BB}$, $\mathsf{Sim}_{\text{NCMEC}}$ validates σ_{NCMEC}. If it does not validate, then $\mathsf{Sim}_{\text{NCMEC}}$ does not need to send a message to $\mathcal{F}_{\text{CSAM-Scan}}$.
4. $\mathsf{Sim}_{\text{NCMEC}}$ then looks up the tuple $(\mathsf{ROHashes}, c_{\text{NCMEC}})$. If no such query exists, then send $(\mathsf{SpecifyHashes}, X = \{\mathsf{img_hash}_1, \ldots, \mathsf{img_hash}_n\})$ for a randomly sampled set $\{\mathsf{img_hash}_1, \ldots, \mathsf{img_hash}_n\}$.
5. Then, for each element $\mathsf{ro_img_hash} \in \mathsf{ROHashes}$, $\mathsf{Sim}_{\text{NCMEC}}$ looks up a tuple $(\mathsf{img_hash}, \mathsf{ro_img_hash})$ and
   - If there is no record $(\mathsf{img_hash}, \mathsf{ro_img_hash})$, then sample a uniform $\mathsf{img_hash}$ and program the random oracle by sending $(\texttt{Program}, \mathsf{img_hash}, \mathsf{ro_img_hash})$ to $\mathcal{G}_{\mathsf{pRO}}$. Then add $\mathsf{img_hash}$ to X.
   - If there is a record $(\mathsf{img_hash}, \mathsf{ro_img_hash})$, then add $\mathsf{img_hash}$ to X.
6. Finally, send $(\mathsf{SpecifyHashes}, X)$ to $\mathcal{F}_{\text{CSAM-Scan}}$.

Figure 29: Simulator $\mathsf{Sim}_{\text{NCMEC}}$ for simulating the view of NCMEC.

C.5 Simulating Public Interfaces

Finally, we provide the simulator $\mathsf{Sim}_{\text{Public}}$ for simulating the public verification components of all parties. Note that this simulator is active no matter which parties are corrupted. The protocol for this public verification can be found in Figure 24, and the description of $\mathsf{Sim}_{\text{Public}}$ can be found in Figure 30.

Notice that the distributions match exactly for this simulation, as the string generation algorithms used in the real protocol are exactly those used in the ideal functionality.

$\mathsf{Sim}_{\text{Public}}$: Simulating the Public (i.e., anyone)

Setup of $\mathcal{S}_{PSI}$, $\mathcal{S}_{\mathcal{J}}$, and $\mathcal{D}$: The simulator sets the string generation algorithms to match the string generation algorithms of $\mathcal{F}_{NIZK}^{\mathsf{KeyDev}}$ and $\mathcal{F}_{NIZK}^{\mathsf{decryption}}$ respectively. The distribution $\mathcal{D}$ matches the distribution for $\mathcal{F}_{\mathsf{CommitAndProve}}$.

Figure 30: Simulator $\mathsf{Sim}_{\text{Public}}$ for simulating public verification for all parties.