Establishing trust in confidential computation and communication systems

Ceren Kocaoğullar

King's College

This dissertation is submitted in May, 2025 for the degree of
Doctor of Philosophy

Declaration

This thesis is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the preface and specified in the text. It is not substantially the same as any work that has already been submitted, or is being concurrently submitted, for any degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the preface and specified in the text. It does not exceed the prescribed word limit for the relevant Degree Committee.

Abstract

Establishing trust in confidential computation and communication systems

Ceren Kocaoğullar

Modern confidential computation and communication systems aim to safeguard data in-use and protect metadata, extending privacy beyond the limits of conventional approaches. Trust is foundational to their security and adoption, yet difficult to establish due to strict privacy requirements, technical complexity, and conflicting stakeholder incentives.

Anonymity networks provide metadata-private communication, protecting information such as who is talking to whom. Current anonymity systems require users to manually exchange key material and network information, a cumbersome operation which hinders adoption. This dissertation presents Pudding, a user discovery protocol that automates trust establishment through email addresses, hides usernames from unauthorised parties, and provides fault-tolerance.

Confidential Computing uses Trusted Execution Environments (TEEs) built for secure and isolated computation to protect data privacy and integrity during processing. As TEEs combine specialised hardware and software from multiple vendors, users must trust a complex and often opaque ecosystem. This dissertation introduces the Confidential Computing Transparency framework as a structured, progressive model to help users make informed decisions by increasing transparency and accountability. A user study, involving over 800 participants, is conducted, demonstrating higher transparency improves trust, and that detailed explanations further increase willingness to share sensitive data.

TEEs can also be used to support dynamic peer-to-peer networks, such as vehicle-to-vehicle communication systems for semi- or fully-autonomous driving, where machine-to-machine trust is essential for collaboration. This dissertation presents Careful Whisper, a gossip-based protocol for establishing trust in such environments. The protocol can reduce attestation cost from quadratic to linear, allows cross-protocol interoperability, and performs reliably in unreliable networks.

These contributions demonstrate practical, scalable ways to build more trustworthy confidential computing and communication. In doing so, they provide some of the foundations required for future secure and private computer systems.

Acknowledgements

Cambridge once felt impossibly far away. I never imagined that reading by the Cam, or walking to Grantchester to sit under the apple trees at the Orchard would become my favourite ways to spend a day. I have never stopped feeling lucky to be here, and I am forever grateful to the many people who helped me get here.

First, thank you to my supervisor, Alastair Beresford. Alastair has supported me unfailingly from even before we met, by helping me secure funding to come to Cambridge for my MPhil. That single act changed the course of my life. His guidance and generosity have shaped the researcher I have become, and I am profoundly grateful.

I am grateful to my family—Mehtap, Yalçın, and Yekta—for their unwavering support, from the time I was a preschooler proudly declaring I would become a *bilim kadını*—a woman of science, even when the default term was the male form—until I became one. Thank you for your unwavering love, your patience, every sacrifice, and for nurturing my curiosity. And to Furkan, you have been my anchor and my encouragement, and your presence has made all the difference. I am endlessly grateful to walk through life with you.

To my lab mates and collaborators—Alex, Alice, Daniel, Diana, Jenny, Laurie, Luis, Martin, Michael, and Tina—to those I worked with during my internships at Amazon, Google, and Arm, and to everyone else who belongs on this list: thank you for the ideas, conversations, and solidarity.

I am sincerely thankful to King's College, the Cambridge Trust, Nokia Bell Labs, and the Department of Computer Science and Technology for generously funding my MPhil and PhD.

Finally, thank you, reader, for choosing to spend time with this work.

Contents

Chapter 1

Introduction

Real-world privacy-enhancing systems are complex mechanisms made of many interconnected components, including hardware, software, people, and institutions, that must work together to function effectively. End-to-end encrypted messaging apps are widely deployed examples of such systems. They can be viewed as complex ecosystems involving cloud service providers, server infrastructure, mobile devices and applications, end users, and regulatory bodies, all of which contribute to determining their design, implementation, use, and governance.

While these components and stakeholders need to interoperate for the system to function, they often cannot—and, in many instances, should not—trust one another naively, because their interests may be in conflict. For example, regulatory authorities might seek access to message contents to tackle criminal activity, while users turn to these apps specifically to keep their conversations private. This tension between working together and against each other at the same time, makes trust establishment in privacy-enhancing systems both critical and complicated.

The trust challenge becomes even more pronounced in confidential computation and communication systems, which aim for even stronger privacy guarantees, namely protecting the privacy of data in-use and metadata-private communication. This dissertation aims to provide usable, efficient, and reliable ways to build trust in such systems across three key areas: *human-to-human* trust in metadata-private communication systems (Chapter 3), *human-to-machine* trust in Confidential Computing with empirical validation (Chapters 4 and 5), and *machine-to-machine*

trust in peer-to-peer Confidential Computing systems (Chapter 6). Trust has a different meaning in each of these contexts, as I define below.

Anonymity networks like Loopix [172] and Nym [67] not only protect message contents, but also provide privacy for metadata, such as who is talking to whom and when. However, the current mechanism for establishing trust between users stands as a potential barrier to the wider adoption of anonymity networks. In this context, trust refers to users obtaining each other's public encryption keys in an authenticated way, so that they know they are talking to the right person and no one can read the message contents unless they have access to the corresponding private key. Currently, communication over anonymity networks requires users to manually exchange public keys and network addresses, which are cumbersome and not human-readable.

In this dissertation, I present Pudding, a practical user discovery protocol for establishing human-to-human trust in anonymity networks. Pudding automates user discovery and uses familiar, human-readable usernames, such as email addresses, while preserving privacy. It protects usernames from unwanted exposure, supports fault tolerance in the face of server failures, and includes safeguards against impersonation. In doing so, Pudding improves usability without compromising the privacy goals of anonymity networks.

Much like confidential communication systems, Confidential Computing—a class of technologies designed to protect the privacy and integrity of data in use—introduces distinct trust challenges [54]. Confidential Computing relies on Trusted Execution Environments (TEEs) [53], which provide isolated areas within processors to isolate and protect data and code during processing, even from higher-privileged software, such as the operating system or hypervisor. TEEs consist of hardware and software components, often sourced from multiple third-party vendors. For Confidential Computing to uphold its guarantees, these components must be correctly designed and implemented. Therefore, users implicitly place significant trust in the underlying technologies. Here, trust refers to the user's confidence in the correct and secure operation of the hardware and software components that make up the Confidential Computing system.

One important challenge is to ensure that trust is not assumed naively. Instead, there must be mechanisms to promote transparency, allowing users to make informed

decisions and hold those who build these systems accountable in the event of a security breach. Nevertheless, complexity of these systems and multiple stakeholders involved make achieving transparency particularly difficult. While the industry has made some progress in improving transparency in Confidential Computing, there is no standard playbook. Each company implements transparency differently, making it hard to compare approaches or establish trust consistently. In this dissertation, I present the Confidential Computing Transparency Framework, which provides a structured roadmap to increasing user trust through transparency. This framework defines multiple levels that build on top of each other, allowing for progressive implementation and flexibility in the face of IP restrictions, resource constraints, and proprietary components.

To evaluate how different transparency levels translate into real-world trust and data-sharing behaviour, this dissertation presents a user study. We conducted two versions of the study with over 800 participants in total: one where we provided a concise explanation of Confidential Computing and transparency, and one with a more detailed version. Results show that user trust increases with transparency, and that clear, detailed explanations help minimise misconceptions and increase trust. Additionally, users were more willing to share sensitive data when higher levels of transparency were provided.

Establishing human trust in machines is one side of the trust challenge in Confidential Computing; the other side is establishing machine-to-machine trust. TEEs can support peer-to-peer networks—such as vehicular ad hoc networks of semi- or fully-autonomous vehicles—by enabling secure data processing and sharing without compromising confidentiality. In such networks, nodes must establish mutual trust to collaborate effectively. In this context, trust refers to the confidence that one machine has in another machine's ability to operate securely and according to expected confidentiality and integrity guarantees through using TEEs.

TEEs can achieve this through remote attestation, a process where a prover TEE presents evidence of its trustworthiness to a verifier, who can then decide whether or not to trust the prover. However, a naive peer-to-peer attestation approach where every TEE directly attests every other TEE, results in quadratic communication overhead, which is inefficient in dynamic environments where nodes frequently join and leave the network.

To address this, I present Careful Whisper, a protocol that uses gossip-based trust dissemination to reduce the overhead of peer-to-peer attestation, achieving linear complexity under ideal conditions. It supports interoperability by enabling transitive trust across heterogeneous networks, and allows trust establishment with offline nodes via relayed attestations. Its effectiveness is demonstrated using a custom discrete-event simulator, evaluating propagation speed, scalability, and resilience across various topologies. In short, Careful Whisper provides a scalable, flexible foundation for machine-to-machine trust in decentralised TEE networks.

Confidential computation and communication systems need efficient trust establishment mechanisms to ensure they provide privacy protection for the people who rely on them and broader adoption of the technology. The need for a high level of privacy, the involvement of various stakeholders with their own interests, and the technical complexity of these systems make this a particularly challenging task. The goal of this dissertation is to design secure, practical and efficient systems for establishing human-to-human, human-to-machine and machine-to-machine trust in confidential computing and communication systems.

1.1 Publications

This dissertation includes collaborative work, both published and in prepublication.

- *Ceren Kocaoğullar, Daniel Hugenroth, Martin Kleppmann, Alastair R. Beresford.* ***Pudding: Private User Discovery in Anonymity Networks****. In Proceedings of the IEEE Symposium on Security and Privacy (S&P), 2024.*
  Chapter 3 is based on the Pudding paper. I led the protocol design, implementation, and security analysis, and was the primary author. Daniel contributed to implementation and experiments; Martin to security analysis. All co-authors helped shape the core ideas and their presentation.
- *Ceren Kocaoğullar, Tina Marjanov, Ivan Petrov, Ben Laurie, Al Cutter, Christoph Kern, Alice Hutchings, Alastair R. Beresford.* ***A Confidential Computing Transparency Framework for a Comprehensive Trust Chain****. Prepublication.*

Chapters 4 and 5 are based on the Confidential Transparency Framework and user study presented in this work. I led the framework development and was the lead author, with industry insights from Ivan, Ben, Al, and Christoph, and conceptual input from Alastair. Tina and I co-designed the user study; she led the analysis, while I contributed additional analysis for this chapter. Ivan, Alice, and Alastair also supported the user study design and presentation.

- *Ceren Kocaoğullar, Gustavo Petri, Dominic P. Mulligan, Derek Miller, Hugo J. M. Vincent, Shale Xiong, Alastair R. Beresford.* ***Building Trust in Ad-Hoc Trusted Execution Environment (TEE) Networks****. Prepublication.*
  The initial idea for gossip-based attestation and the related patent application came from Gustavo, Dominic, Derek, and Hugo. I developed it into a full protocol, extended it for stronger adversaries, authored the text, and built a simulator for evaluation. Shale and Alastair contributed to the design and presentation.
- *Daniel Hugenroth, Ceren Kocaoğullar, and Alastair R. Beresford.* ***Choosing Your Friends: Shaping Ethical Use of Anonymity Networks****. In Proceedings of Security Protocols XXVIII: 28th International Workshop, 2023.*
  §2.3.5 is based on this paper, which was shaped by all three co-authors and informed by workshop discussions.

1.2 Contributions

In this dissertation, I make the following contributions:

- In §2.3.5, I discuss how anonymity networks can be designed with ethical considerations in mind, arguing that technical and policy choices can help reduce potential for misuse without compromising privacy.
- In Chapter 3, I present *Pudding*, a user discovery protocol that enables practical contact discovery in anonymity networks without compromising metadata privacy. Pudding automates user discovery using email addresses, hides contact relationships and protects username queries from unauthorised users, even when some servers are compromised or unreliable.

- In Chapter 4, I present the *Confidential Computing Transparency framework*, which provides a structured approach for achieving transparency in Confidential Computing. It enables users to make informed trust decisions by introducing verifiable mechanisms and clearly defined levels of transparency, rather than relying on naive trust. The framework defines a tiered model of transparency levels that can be tailored to different needs and constraints.
- In Chapter 5, I present a large-scale user study evaluating the impact of transparency on trust in Confidential Computing systems. The study reveals that transparency has a measurable impact on trust, especially when users are given clear, targeted explanations. It also demonstrates how addressing common misconceptions can improve the effectiveness of transparency measures.
- In Chapter 6, I present *Careful Whisper*, a scalable attestation protocol for establishing trust in peer-to-peer TEE networks without relying on a central authority. It uses gossiping to disseminate trust and reduce attestation complexity. This protocol also supports heterogeneous TEEs and unreliable connectivity. I also propose an extension to the protocol that limits the influence of adversaries capable of compromising TEE protections.

Chapter 2

Background

Data exists in three states: at rest (stored), in transit (transferred), and in use (processed). This dissertation focuses on systems that preserve data privacy while in use, and those that protect metadata privacy during transfer. This chapter begins by outlining the evolution of data protection in mobile communications, resulting in the widespread adoption of end-to-end encryption. Anonymity networks are explored next, which offer even stronger privacy guarantees for data in transit by protecting metadata privacy. A discussion on design strategies that may mitigate potential misuse of such networks is also included. Following this, Confidential Computing is examined, which is a family of techniques aimed at safeguarding data while in use. The chapter concludes with a discussion of transparency, which is crucial in establishing trust in systems that include Confidential Computing.

2.1 End-to-end encryption for data in transit

In the early days of electronic mobile communication, prior to the 1990s, encryption was almost entirely absent, leaving voice and data transmissions susceptible to interception. Analogue mobile networks such as the Advanced Mobile Phone Service (AMPS) [138] were particularly vulnerable due to their lack of built-in cryptographic protections. Eavesdropping on these systems required nothing more sophisticated than a radio scanner, as signals were transmitted in plaintext. The introduction of digital mobile technologies in the 1990s brought advances to security,

most notably with the Global System for Mobile Communications (GSM) standard [101]. GSM introduced encryption mechanisms, including the A5 family of ciphers, which aimed to secure the communication channel between mobile devices and base stations. The A5/1 algorithm was the main encryption scheme, developed for use in European markets [76]. A deliberately weakened variant called A5/2 was used for export to countries restricted by Cold War era regulation [77, 76, 101]. Despite bringing encryption to mass communications for the first time, GSM was far from invulnerable. Both A5/1 and A5/2 were shown to be susceptible to cryptanalysis due to design flaws and short key lengths [34, 71, 82, 171].

As public awareness of digital security grew, the late 1990s witnessed the emergence of end-to-end encryption tools, such as Pretty Good Privacy (PGP), designed to secure email and file transmissions. PGP, developed by Phil Zimmermann, used strong asymmetric encryption schemes, including RSA [176] and later Diffie-Hellman [68], paired with the IDEA symmetric cipher to protect user data [221]. Despite its strong cryptographic foundation, PGP was plagued by significant usability issues, as users had to manage key generation, distribution, and trust manually. Whitten and Tygar's foundational usability study, "Why Johnny Can't Encrypt" [216] found that even technically literate users struggled to perform basic encryption tasks without mistakes. This usability gap limited the adoption of PGP beyond the technical and activist communities [91].

In early to mid-2010s, modern encrypted messaging platforms like Signal and WhatsApp addressed these usability challenges by hiding technical tasks from the user and providing intuitive interfaces. These platforms employ end-to-end encryption, which ensures that message contents can only be seen in plaintext by the receiver and sender. In end-to-end encrypted messaging, each pair of communication partners establish a shared secret using a secure key exchange protocol, such as Diffie-Hellman [68]. They then encrypt the contents of each message they exchange with this shared secret, so that only the recipient of the message decrypt and read them. Most of these platforms are built on the Signal Protocol [140], a cryptographic framework that incorporates the Double Ratchet algorithm, X3DH (Extended Triple Diffie-Hellman), and prekeys to ensure forward secrecy and post-compromise security [51].

A major usability innovation of these end-to-end encrypted messaging platforms

is the use of phone numbers for user discovery, which eliminates the need for manual key exchange. Instead, public keys are generated and distributed by centralised servers, while private keys remain stored on user devices. By lowering the barriers to secure communication, these messaging platforms have enabled billions of users to adopt end-to-end encryption.

2.2 Metadata privacy

Metadata privacy concerns protecting information about a message beyond its actual content. This includes details such as who is communicating with whom, when and how often they exchange messages, the location of the communicating parties, and other characteristics like message size and content type. Metadata can be highly sensitive and, in some cases even more revealing than the message content itself.

Various adversaries including service providers, state actors, and network attackers can exploit metadata to track individuals, infer behaviours, and uncover sensitive information. In some cases, metadata is even accessible to the public, making unintended disclosures more likely. In 2018, publicly available FitBit location data inadvertently revealed the locations of secret military bases, as patrol officers shared their movements [3]. That same year, Strohmeier et al. showed that publicly accessible aircraft metadata could be used to infer corporate mergers before they were officially announced [187]. Investigative journalists at Bellingcat have repeatedly leveraged metadata in many high-profile cases, including a Russian FSB plot targeting opposition leader Alexei Navalny [196] and potential ties between the Columbia's General Prosecutor's Office and the killing of protestor Lucas Villa [191].

Although current generation of end-to-end encrypted messaging apps offer confidentiality of message contents against malicious servers and network adversaries (provided users verify each other's keys) these apps offer little metadata privacy. Even Signal's Sealed Sender mechanism [136], which is designed to hide the sender's identity from the service provider, does not offer protection at the network level. Unlike regular Signal messages, Sealed Sender messages include a sender certificate in the payload. Instead of authenticating in the usual way, the sender uses a

delivery token *for the recipient* to submit the message. This way, Signal servers do not learn the sender of a Sealed Sender message. However, this mechanism does not protect against a network-level adversary capable of monitoring traffic patterns, who may still infer who is communicating with whom based on connection metadata. Without metadata protection at the network level, end-to-end encrypted messaging apps alone are insufficient to fully protect user privacy.

2.3 Anonymity networks for metadata in transit

Anonymity networks provide metadata privacy, hiding information such as which users communicate with whom and when. Different network designs can provide protection against adversaries with different capabilities, and be different in terms of latency, scalability and usability. Depending on these, anonymity networks can be useful for private browsing, censorship resistance, whistleblowing, and secure communication.

2.3.1 Tor anonymity network

Tor [69] is perhaps the most well-known anonymity network. It is a circuit-based overlay network, where all communication between a client and a destination follow the same predefined path made of three relay nodes: (1) a guard node that knows the client's IP address, but not the destination; (2) an exit node, which knows the vice versa; and (3) a middle relay node between these two nodes, which forwards the message from the guard to the exit node. This way, each relay node can only know the node coming before it and the one after it; therefore, the full circuit and the communicating parties remain hidden from the relay nodes as long as they are not all controlled by the same party. To reduce this risk, Tor nodes are volunteer-operated and the network has no central authority [69]. Circuits are short-lived and are reestablished every $\sim$10 minutes to mitigate traffic correlation attacks [194].

In Tor, data packets are encapsulated in multiple layers of encryption, also known as onion encryption. The client establishes a separate session key with each of the nodes in the circuit using Diffie-Hellman key exchange [68]. To ensure

forward secrecy, new keys are generated for each circuit. Each data packet is then encrypted in reverse order of the circuit; starting with the key shared with the exit node, then the middle relay, and finally the guard node. As the message traverses the circuit, each node in the path removes the corresponding layer of encryption, revealing the next hop. As the message leaves the exit node and forwarded to the final destination, all Tor-specific encryption has been removed [69].

Tor achieves low latency and high throughput, allowing real-time communications. It is widely used for private browsing, censorship resistance, and anonymous communication. However, despite its strengths, Tor has several well-documented vulnerabilities. Most notably, it does not defend against traffic correlation attacks by a global passive adversary, who is capable of observing multiple points in the network simultaneously to infer user behaviour or deanonymise users [160, 212]. Additional threats include distributed denial-of-service (DDoS) attacks that can degrade or disrupt the network's performance [113, 112], as well as credential sniffing and person-in-the-middle attacks carried out by malicious exit nodes [218].

2.3.2 Mix networks

Mix networks, or mixnets, are another anonymity network architecture aimed at hiding communication patterns. The first mix network design introduced by David Chaum in 1981 [46] was the first to route data through multiple nodes and use layers of encryption that are removed at each step, ideas that were later used in other anonymity network architectures, including Tor. In Chaumian mixnets, each message follows a different route, unlike the circuit-based design of Tor. Also different from Tor, relay nodes, known as mixes, change the ordering of messages they receive before forwarding them [46].

Chaumian mixnets prevented timing correlation attacks and provided protection against global passive adversaries. However, they did not protect active attacks, such as tagging or replay attacks and also had very high message latency. To address some of these issues, Mixmaster [155] emerged in the 1990s. It provided several improvements, such as padding all packets to a fixed size to prevent traffic analysis attacks, adding randomised delays before sending messages, and the first ever suggestion of using dummy messages to hide traffic volume. In 2003,

Mixminion [62] added the ability to send anonymous replies with Single-Use Reply Blocks (SURBs), which will be discussed in §2.3.4, introduced the first structured dummy traffic, as well as directory servers for public key discovery and topology management.

While reordering and delaying messages strengthens metadata privacy, the downside is a higher latency than non-anonymity networks or even designs like Tor. Despite this limitation, mix networks can support many other types of applications, including asynchronous messaging. To better balance the latency-anonymity trade-off, "medium-latency" mixnets have been developed; most notably Loopix [172], which I describe next.

2.3.3 Loopix architecture

Loopix is a prominent medium-latency mix network design that protects sender-receiver relationships from third parties, including global passive and active adversaries, enabling metadata-private email or instant messaging applications [172].

The network consists of three types of nodes, as illustrated in Figure 2.1: (1) *user devices* are associated with a given user and provide send/receive functionality for end-user applications; (2) multiple layers of *mix nodes* transmit real and cover messages; (3) *provider nodes* manage access to the network. Each user device is registered at a provider node and can only send and receive packets through its provider. A provider node can receive messages on the user's behalf, even when the user device is offline.

Loopix hides communication patterns using cover traffic and Poisson mixing [172]. Cover traffic is made of dummy messages that can either be dropped before reaching a destination or loop back to the sender. Poisson mixing refers to the mix nodes determining the delay time before sending each message at random based on a Poisson distribution. Loopix network packets use the Sphinx packet format described in §2.3.4. Additionally, Loopix design is adopted by the Nym[1] anonymity network, currently deployed on hundreds of nodes.

[1] `https://nymtech.net/`

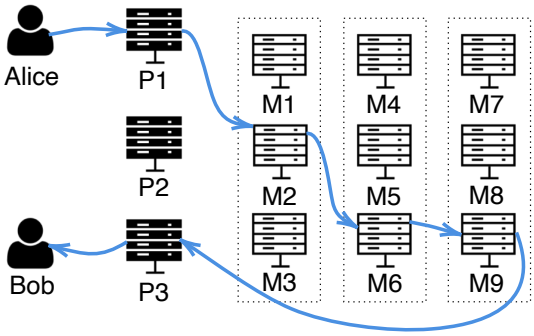


Figure 2.1: Diagram showing Alice sending Bob a message through a Loopix network with three layers of mix nodes *(M1–M9)* and three provider nodes *(P1, P2, P3)*. The arrows show a possible route for Alice's message. Each message passes through one mix node from each layer. Alice and Bob access the network via provider nodes, which are connected to all the mix nodes in the outermost layers.

2.3.4 Sphinx packets and Single-Use Reply Blocks (SURBs)

Sphinx is a cryptographic packet format designed to send messages through mix networks [59]. A Sphinx packet consists of two main components: a header and an encrypted payload. The header contains the metadata necessary to route and process the message correctly, i.e. to verify the integrity of the packet at each hop, to send the packet to the next node in the path, and to decrypt the outer-most payload layer. The payload is encrypted in layers, ensuring that each node in the message path learns only the routing information to the next hop and a network observer cannot correlate a mix node's incoming and outgoing packets.

A defining characteristic of Sphinx packets is their compactness. For an onion-encrypted message to be routed through a mix network, each node on its route must be able to decrypt its corresponding layer and learn the routing information. To do so, each node must share or establish a key with the sender. Previous systems, such as Mixminion [62], use RSA encryption [176] for this purpose, resulting in an overhead of at least one RSA ciphertext. Sphinx instead uses Diffie-Hellman [68], which allows each relay node to derive keys from a single public group element that is blinded at each hop. As a result, Sphinx achieves 128-bit security with just 448 bytes of overhead for a 5-hop path, compared to Mixmaster's [155] 2136 bytes and Mixminion's 1040 bytes [59].

In addition to its compactness, Sphinx provides bitwise unlinkability, making it difficult to correlate messages entering and exiting a mix node. It also offers protection against replay attacks and active tagging attacks. Sphinx packets are source-routed, meaning that the creator of the header decides and encodes the routing information into the packet [59].

The Sphinx packet format includes a special mechanism named Single-Use Reply Blocks (SURBs), which allows a user to receive a reply to their message without revealing their identity to the recipient [62, 59]. A SURB functions as a one-time return path through a mix network, precontructed by the original sender of a message and attached to the message to allow the receiver to respond. In the typical onion encryption model, if Alice sends a message to Bob, Bob needs to know Alice's encryption key (and potentially her network address) to be able to respond to her. SURBs eliminate this requirement. They allow Alice to receive a reply from Bob without having to share any such information that may be tied to her identity. Structurally, a SURB is a Sphinx header containing onion-encrypted routing information that leads through a sequence of mix nodes and ultimately back to Alice. This header also includes the necessary keys for payload encryption.

While SURBs are most commonly used for enabling anonymous replies to the original sender, they can be created by anyone who knows the destination's encryption key (and network address, if needed by the network's design).

2.3.5 Ethical considerations about anonymity networks

While privacy is essential for protecting individuals, it can also provide cover for criminal activities. Some systems designed for secure and private communication, such as Tor and the end-to-end encrypted messaging application Telegram, are widely associated with illegal activities. In our paper "Choosing Your Friends: Shaping Ethical Use of Anonymity Networks" presented at the Security Protocols Workshop (SPW) in 2023 with Daniel Hugenroth and Alastair Beresford, we propose some design strategies that can encourage ethical use of anonymity networks [99].

One approach we propose is restricting anonymity, which can be done in several ways. For example, anonymity can be limited to only one side of a communication, like the CoverDrop [12] system designed for whistleblowers to contact journalists.

Other strategies include limiting the user base or requiring non-anonymous access during onboarding while maintaining anonymity afterward.

Another category of tactics is using technical limitations as a tool for restricting system's use cases. For example, reducing bandwith or increasing latency can make it difficult to disribute exploitative images or video. The design of mechanisms for discovering other users and content can also impact how an anonymity network is used. For instance, having a shared index searchable by users, a subscriber-based system, or one-to-one contact establishment can influence the user base and interactions within the network. Finally, policy decisions play a crucial role in safeguarding users. For example, distributing operational responsibilities among actors with competing incentives, as Tor does with its nodes, helps prevent abuse. Limiting or moderating access to external resources can also discourage the unethical use of an anonymity network. These strategies offer a balance between privacy and responsible usage, helping mitigate the risks associated with anonymity networks.

2.4 Confidential Computing for data in use

Anonymity networks and end-to-end encryption are ways to protect data in transit. They are built on top of our long-standing ability of securing data at rest using encryption. What remains as a challenge is protecting the data in use, that is, while they are being processed. Confidential Computing addresses this challenge by using hardware-based Trusted Execution Environments (TEEs) built for secure and isolated computation. Several TEE implementations have been developed by major hardware vendors, such as Intel SGX [108] and TDX [109], AMD SEV-SNP [20], and Arm CCA [26]. Commercial cloud platforms, including Google Cloud [50], Microsoft Azure [147], and Amazon Web Services [15] have made Confidential Computing widely available.

By using specialised hardware, Confidential Computing provides distinctive advantages over other privacy-preserving computation techniques like homomorphic encryption and secure multiparty computation. Specifically, TEEs provide confidentiality for both data and code in execution through techniques like hardware isolation and memory encryption. Moreover, TEEs protect the integrity of the data and code throughout execution [53]. As a result, Confidential Computing is a

versatile tool that can be used independently or complement other private computation methods, such as homomorphic encryption. It is an essential component in building a foundation for comprehensive private computation.

A significant caveat of Confidential Computing is that it places an implicit trust in the manufacturers to design and implement TEEs securely and correctly. If the TEE hardware or software has any vulnerabilities or hidden backdoors, privacy and integrity of data and code may be compromised.

2.4.1 Remote attestation

Remote attesattion is a key feature of Confidential Computing [54]. It allows a *prover* device to provide information about the authenticity, integrity, and certain runtime properties of a TEE to a *verifier* by providing hardware-signed evidence. The evidence contains relevant measurements of the Trusted Computing Base (TCB) [54], which is the collection of firmware and software that the security of a system depends on [125]. The verifier then compares the attestation evidence against a set of trusted reference measurements and establishes trust in the prover if the evidence is found valid. If the attestation is successful, the verifier may provision secrets or establish encrypted sessions tied to the TEE's integrity. These protocols can leverage different hardware or software environments to ensure that the prover generates legitimate attestation evidence [33].

2.5 Transparency

Transparency enables community scrutiny, which can accelerate detection and resolution of issues and promote greater accountability. Although transparency does not guarantee security, as evidenced by notable vulnerabilities found in open-source software [81, 188, 25, 157, 63], it offers a significant improvement over the alternative of naive trust.

The most established example of transparency is Certificate Transparency, which has become the standard method for monitoring TLS certificates [128, 127]. This approach inspired similar initiatives such as Key Transparency [144, 100, 130, 214] and Binary Transparency [13, 134], which aim to bring similar auditability to public

key infrastructures and software distribution systems, respectively. In the context of Confidential Computing, transparency plays an equally important role in mitigating the risk of implicit trust in TEE providers. Given that remote attestation and TEE guarantees ultimately depend on the correctness and integrity of hardware and firmware implementations, introducing mechanisms for transparency can help detect vulnerabilities and backdoors early, and build trust across the ecosystem.

This section starts with an overview of the current transparency practices in Confidential Computing (§2.5.1), followed by discussions of Certificate Transparency (§2.5.2), Key Transparency (§2.5.3), and Binary Transparency (§2.5.4).

2.5.1 Transparency practices in Confidential Computing

Transparency and open-source adoption are becoming more common in security-critical software of Confidential Computing systems. In August 2023, AMD released the source code for the SEV firmware used in 4th Generation AMD EPYC processors [19]. According to the company's press release that same month, these processors are available through major cloud providers, including Amazon Web Services (AWS), Google Cloud, Microsoft Azure, and Oracle Cloud Infrastructure (OCI) [21]. However, as of this writing, the repository lacks a descriptive README file or supporting documentation. Additionally, no new commits have been made since its initial publication.

In June of the same year, Intel also open-sourced the TDX module [110]. The repository has been consistently updated to align with new version releases. It includes documentation on TDX, along with guidelines for debugging and contributing via opening issues. Additionally, the code is reproducibly buildable—by following Intel's provided instructions, users can generate the exact binary released from the same source code.

Arm leads the Trusted Firmware project [27], providing open-source reference implementations of firmware that form the foundation of its TEE specifications. This suite of security-focused components establish the core of Arm-based TEEs, covering boot security (TF-A [203], TF-M [204], MCUBoot [200]), secure execution environments (OP-TEE [201], Hafnium [199]), cryptographic services (PSA Crypto [202]), and security management and services (TS [205]). At the time of writing,

these repositories are actively maintained, with updates ranging from monthly to almost daily. Each project includes detailed documentation, and (with the exception of MCUBoot) features a full review history, documenting internal evaluations of every commit. The project also provides clear contribution and review guidelines. Unlike Intel and AMD, the Trusted Firmware project allows contributors to make direct commits rather than just submitting suggestions. Most of the firmware projects, except OP-TEE, maintain open forums, where anyone can join conference calls to engage with maintainers. Additionally, some projects offer mailing lists for discussions and idea-sharing within the community.

In May 2023, AWS announced that the architecture of its Nitro system underwent a review by the cybersecurity firm NCC group [175]. The findings have been made public as a report [16]. The NCC Group's assessment found no design flaws that compromise AWS's security claims, that the Nitro System enforces least privilege access, strong encryption, and secure software deployment, ensuring AWS employees cannot access customer data. All API access is authenticated, logged, and monitored, with no backdoors for manual intervention. However, the review relied on AWS's documentation and developer interviews, with no hands-on validation or testing of Nitro firmware, hypervisor, or physical security [175]. Therefore, there is no guarantee that the audited component designs match those deployed in production environments.

In October 2024, Apple introduced a combination of transparency measures for Private Cloud Compute (PCC) to facilitate independent security research [39]. It has published a security guide outlining PCC's architecture, security mechanisms, and privacy protections. It also released source code for certain PCC components, including CloudAttestation [105] for attestation validation and Thimble [106] for enforcing transparency. It also released a Virtual Research Environment (VRE) [107], which allows researchers to inspect software releases, verify transparency logs, and conduct security testing in a controlled environment. Additionally, PCC has been added to Apple's Security Bounty Program, which offers rewards for findings that challenge its security and privacy claims [39].

2.5.2 Certificate Transparency

The security of our HTTPS connections relies on domains sending their X.509 certificates to browsers, with the browser verifying if the certificate is issued by an authorised Certificate Authority (CA). This mechanism by itself places blind trust on the CAs, which creates a vulnerability–exploited or malicious CAs can issue false certificates, leading to potential man-in-the-middle attacks [11, 149, 10]. To detect and mitigate rogue certificates, Google launched the Certificate Transparency project in 2013 [128, 127], which is now considered to be the most widely-adpoted application of transparency. Since 30 April 2018, Google Chrome Policies requires all CAs to support Certificate Transparency [83].

Certificate Transparency makes certificates publicly verifiable through a three step process: (1) the CA sends the data it will put inside the certificate to a transparency log, (2) the transparency log signs this data and sends this signature back to the CA, (3) the CA appends the transparency log's signature to the certificate data, signing them together to form the final certificate [127]. The transparency log's signature serves as an inclusion promise (see §4.3.2). Monitors (§4.3.1) observe the transparency log, and download certificates to check their inclusion proofs. If a monitor finds an anomaly in a certificate or in a CA's logging behaviour, it can ring the alarm bells and trigger a certificate revocation mechanism.

2.5.3 Key Transparency

In end-to-end encrypted communication, there is a parallel challenge to CAs having too much power over SSL certificates: users have to trust key directory servers for providing them with the correct key corresponding to a username. Addressing this concern and drawing inspiration from the principles of Certificate Transparency, CONIKS [144] has pioneered the concept of Key Transparency. Differently from Certificate Transparency, CONIKS places a strong emphasis to privacy, limiting the information that each transparency log query reveals.

Key Transparency has gained traction in practice as well. In 2017, Google launched its open-source Key Transparency project [100]. In 2023, both WhatsApp and Apple introduced Key Transparency to automatically verify the authenticity of users' encryption keys on their messaging platforms [130, 214, 73].

2.5.4 Binary Transparency

Binary Transparency [197], addresses a parallel concern to Certificate and Key Transparency in the context of software supply chains. The primary objective is to establish accountability for software originating from central repositories and to counter targeted attacks. Binary Transparency is actively deployed in Google Pixel binaries [87] and Go modules [56]. Sigstore [79] is a collective effort for automating logging, signing, and monitoring in Binary Transparency.

In research, several systems have been proposed to strengthen transparency in software distribution. Contour [13] uses a blockchain to achieve consistency and availability of transparency logs, even in the presence of powerful adversaries. CHAINIAC [162] also takes a decentralised stance, combining skipchains with verified builds to deliver tamper-evident and trustworthy software updates. Building on the need for verifiable software supply chains in mobile ecosystems, Mobile App Distribution Transparency (MADT) [134] applies Binary Transparency to distribution of Android app binaries. Together, these efforts indicate a growing trend towards using transparency to secure software supply chains, both in practice and in theory.

2.6 Summary

This dissertation explores new approaches to establishing trust in two key areas of privacy-preserving technologies: systems that protect data in use—namely, Confidential Computing—and systems that protect metadata in transit—namely, anonymity networks. This chapter provides background information on the evolution of anonymity networks and how they differ from end-to-end encrypted systems. It then focuses on a specific subset of anonymity networks, mix networks, and provide details of a medium-latency mix network design named Loopix. It also includes a discussion of design choices aimed at encouraging ethical use of anonymity networks.

The chapter then shifts to Confidential Computing, introducing its core concepts and describing why transparency is critical for fostering informed trust to its users. Lastly, it describes the current industry practices of transparency in Confidential

Computing, as well as how transparency has been established and used in more established contexts of Certificate, Key and Binary Transparency.

The following chapter will introduce the first trust establishment mechanism proposed in this thesis: Pudding, a user discovery protocol designed to automate contact discovery in anonymity networks while protecting privacy.

Chapter 3

Private and practical user discovery in anonymity networks

End-to-end encryption is widely used in apps like Signal, iMessage, and WhatsApp to protect the confidentiality and integrity of our correspondence (§2.1). Unlike earlier systems such as PGP, which required users to explicitly manage cryptographic key material, today's secure messaging apps allow users to contact their friends using only a low-entropy username, such as a phone number. It is likely that this convenience contributed to widespread adoption of end-to-end encrypted messaging apps, while PGP has remained a niche tool.

Despite offering practical ways to discover contacts while also hiding the contents of conversations, these apps do not provide metadata privacy (§2.2). Although anonymity networks support messaging with metadata privacy (§2.3), they overall do not consider user discovery as a part of their designs and instead defer the problem to other work. The user discovery model of encrypted messaging apps is unsuitable for anonymity networks as it undermines metadata privacy, and existing research fails to achieve key properties such as hiding usernames from unauthorised users, Byzantine fault tolerance, or usability (§3.1.1).

This chapter introduces Pudding[1], a practical and metadata-private user discovery protocol. Pudding aims to establish human-to-human trust in anonymity networks. Trust here means users get each other's public keys in an authenticated

[1] A wordplay on the abbreviation of 'private user discovery' (PUD).

way, confirming who they are communicating with and that only someone with the right private key can read the messages. Pudding hides contact relationships between users, so that neither a global network observer nor a compromised discovery server can determine who is talking to whom. It also ensures that an adversarial user cannot learn whether a given username is registered on Pudding, except when authorised by the owner of that username. Pudding allows user discovery even when up to one third of the directory servers are unavailable or return incorrect responses. Pudding is an application-layer protocol built on top of the Loopix [172] architecture. We have implemented a prototype of it on the Nym network[2].

In my MPhil in Advanced Computer Science dissertation at the University of Cambridge[3], I developed a protocol also named Pudding to address the problem of private and practical user discovery in anonymity networks. However, the protocol presented in my MPhil dissertation is fundamentally different from the one described here, both in terms of its security and privacy properties as well as the underlying mechanisms. In §3.6.3, I outline the most notable differences between the two protocols.

This chapter is based on our paper "Pudding: Private User Discovery in Anonymity Networks" [118]. I adapted the text and figures from the paper to fit this dissertation. I led the protocol design, prototype implementation, and security analysis, and was the primary author of the text. Daniel contributed to implementation and managed the experimental setup and data analysis; Martin assisted with security proof sketches (§A). Daniel, Martin, and Alastair have all contributed to the overall idea development and presentation.

3.1 User discovery in anonymity networks

To establish contact with a user on an anonymity network, one needs to know some network-specific information about that user. We use the term *contact information* to refer to the information that is needed to make initial contact with a user through the anonymity network. Although the content of contact information differs among

[2] `https://nymtech.net/`

[3] My MPhil dissertation can be accessed at `https://www.cl.cam.ac.uk/~ck596/ceren_kocaogullar_mphil_dissertation.pdf`

anonymity network schemes, it generally includes a user's public key. In some systems, such as Talek [47] and Vuvuzela [208], the public key is used as a user ID as well. In Loopix [172], the contact information is comprised of the public key, the IP address of the user's provider, and the account ID at that provider (see §3.2.1). Nym is similar to Loopix, only using a Nym-specific provider ID instead of an IP address.

We define *user discovery* as initiating contact with a user based on her username, either by obtaining her contact information, or by directly sending a message to the user without revealing her contact information.

3.1.1 Challenges in metadata-private user discovery

The user discovery model used by modern end-to-end encrypted messaging applications is fundamentally incompatible with strong metadata privacy, as it undermines the anonymity guarantees that anonymity networks are designed to provide. While these messaging apps ensure confidentiality by encrypting message content, their key discovery mechanisms expose metadata, which can be leveraged by adversaries to infer communication patterns, track users, or manipulate trust relationships.

Specifically, encrypted messaging services such as Signal, iMessage, and WhatsApp rely on trusted key directory servers to store and distribute public key material. To establish a secure communication channel, users query these directories by submitting usernames, such as phone numbers (Signal, WhatsApp and Telegram), or email addresses (allowed in iMessage). The key directories then provide the users with the corresponding public keys of their contacts. However, this approach introduces a significant privacy risk. This discovery model enables the servers to learn which contacts in a user's address book are registered to the messaging app (e.g. Signal [141]), and in some cases even those that are not registered (e.g. WhatsApp [215]).

Communicating with a key directory via an anonymity network may provide some metadata privacy against a network observer, but it does not provide protection against malicious or compromised directory servers. If a key server is compromised, an attacker can alter or add public keys linked to user identities. In messaging apps that support multiple devices per account, an adversary could

stealthily register an additional device under a user account. As a result, when others send encrypted messages to the user, the messages would also be delivered to the attacker, who could decrypt them. This method, known as a ghost-user attack [209], was proposed by GCHQ as a way to intercept encrypted communications without breaking encryption itself [133].

Moreover, these directory-based discovery models are vulnerable to enumeration attacks, where an adversarial user systematically queries the key directory with all possible phone numbers to determine which ones are associated with registered accounts. Previous research has demonstrated the feasibility of such attacks at scale, revealing that an attacker can extract large portions of a service's user base simply by iterating through likely phone numbers [92]. This compromises user anonymity and exposes sensitive social graphs, potentially allowing adversaries to map entire communication networks.

As a result, the user discovery model employed by current end-to-end encrypted messaging applications do not provide strong metadata privacy, and is fundamentally incompatible with anonymity networks.

3.1.2 Privacy requirements

A private user discovery mechanism must not undermine the level of privacy provided by the anonymity network. Therefore, a user discovery system should ensure that:

1. The information "Alice is searching for Bob" is kept secret from everyone other than Alice and Bob. The providers of the discovery service and anonymity network, and any active or passive network adversaries, should not learn which users are communicating.

2. A user should not be able to learn whether a given username is registered, except by contacting the owner of that username and the recipient choosing to respond.

The first requirement is important, because contact relationships may reveal sensitive information that can lead to multiple forms of victimisation, such as targeted social engineering attacks, blackmailing, defamation, and discrimination.

The second requirement is critical in cases where exposing the membership information can harm the user; for instance, in a scenario where the anonymous communication network is known to be used by the LGBTQ+ community in an oppressive state, where members of this community are heavily punished. In addition, a large database of contact information allows an adversary to perform spamming and phishing on a large scale. Moreover, an adversary with exploits that target users of the particular anonymity network can use such a database to disseminate the exploits. As mentioned in §3.3, user discovery schemes used by encrypted messaging apps do not provide this requirement; nor do content discovery mechanisms such as DNS and DHT, as we describe in §3.6.

3.1.3 Usability requirements

We believe that user discovery in anonymity networks should be simple, so that it is accessible to all users. Inspired by trust establishment usability definitions from Unger et al.'s work [206], we identify two key usability requirements for user discovery. A *usable* discovery mechanism in an anonymity network should:

1. Require only a short, memorable, and hence low-entropy username to establish contact. The username can either be a 'pseudonym' that is unique only within the system, or an identifier that has meaning outside of the anonymity network and therefore may already be in the local address books of the user's friends (e.g. a phone number or an email address).

2. Allow the user discovery to happen without requiring the user to leave the anonymity network, because any communication outside the anonymity network risks revealing to a network adversary the very metadata that the anonymity network is trying to hide.

The first requirement is analogous to DNS and address books: Users easily recall URLs and people's names, not IP addresses or phone numbers, because memorising random-looking strings requires repetition over an extended period [36]. Users can learn the username of other users in many ways. For instance, demonstrators meeting at a protest may exchange usernames to stay in touch. Although using a non-anonymous identifier such as a phone number or an email address may appear

counter-intuitive in an anonymity network, it is useful for users who are willing to reveal that they are a member of the network, but do not wish to reveal who they are communicating with. For example, a journalist may publish their username as part of their social media presence, so that a potential whistleblower can contact them securely.

The requirement to allow user discovery without leaving the network draws from research showing that authentication ceremonies of encrypted messaging apps have poor usability [210, 94]. In such ceremonies, users share public keys or verification codes manually with each other via QR code, near-field communication (NFC), or through speech [206].

3.2 The Pudding protocol

As we detail in §3.1, there is no existing solution for user discovery that fits the usability and privacy needs of anonymity networks, is a lightweight design that is compatible with applications running on mobile devices, and provides fault tolerance. In this section, we describe the Pudding protocol that aims to address these needs and objectives.

3.2.1 Setup

Pudding is an application-layer protocol—it does not require any modification of the anonymity network for which it provides a user discovery service. Pudding can be built upon any anonymity network that allows messaging without explicitly knowing the recipient's contact information (see single-use reply blocks in §2.3.4), such as Nym. The one exception to this rule is the validation of email addresses, as discussed in §3.4. We build Pudding on Nym/Loopix, because this anonymity network design provides a high level of metadata privacy, is suitable for asynchronous messaging, and is designed to be feasible to use on mobile devices.

Pudding nodes. For our protocol, we introduce a new type of server, the *discovery node*, which provides the user discovery service. Discovery nodes can be independently managed and maintained, and no single entity should control more than one-third of the discovery nodes (see §3.2.3). There is a fixed set of n

discovery nodes, where $n = 4$ or $n = 7$ would be reasonable choices (see *Security parameters* in §3.3). Each discovery node has a database that maps usernames to contact information. From a Loopix protocol point of view, discovery nodes act as user devices (although in practice, they may be co-located with mix nodes or provider nodes). This allows discovery nodes to be added to the network without changes to the Loopix protocol, without disturbing its security properties.

System assumptions. We assume that all user devices know the public contact information for all discovery nodes, so that they can send anonymous messages to the discovery nodes. Provider nodes can supply this information to user devices.

3.2.2 Security and privacy goals

Our security and privacy goal definitions are inspired by AnoA [30], which is also the approach taken by Loopix [172]. In terms of notation, each user (U) possesses a username (ID) and contact information (Δ). We denote the ID and Δ associated with a certain user U as $(U : ID, \Delta)$. We write $\{U_i \to U_j\}$ to indicate that user U_i searches for user U_j. For now we state our goals informally, and we formalise them in §3.5.1.

G1 Unlinkability: The adversary cannot learn *who is searching for whom*. This goal ensures that the *sender-receiver third-party unlinkability* goal of Loopix is preserved. In other words, assuming a sufficiently large Loopix anonymity set, the adversary cannot tell the difference between $\{U_1 \to U_*\}$ and $\{U_2 \to U_*\}$, for all honest users U_1 and U_2 chosen by the adversary. Essentially, when a user searches for U_*, the adversary learns U_* but not who sent the query.

G2 Security against impersonation: Assume a user U_1 has registered contact information Δ_1 for a particular username ID_1. When another user subsequently looks up ID_1, the contact request should be sent to the registered contact information Δ_1 and no other user. In simpler terms, an adversary cannot cause any honest user to send a message intended for U_1 to a $\Delta_2 \neq \Delta_1$.

G3 External identity verification: A user's identity within the system should be verifiably associated with an external identity by the discovery nodes. For

instance, this can be achieved by discovery nodes sending a code to an email address or phone number, assuming that the communication channel used to send the code is secure and the user's device is trusted. This goal provides assurance to the user that she is talking to the correct person, defined as the owner of an email address. Let us assume that the user U is a user controlled by the adversary, and ID is an external identity, such as an email address at a trusted email service, that is *not* controlled by the adversary. We define external identity verification as the inability of the adversary to convince any honest discovery node into registering the user $(U : ID, \Delta)$.

G4 Membership unobservability: No user should be able to determine whether a username has been registered, unless the owner of that username decides to reveal this information. Membership unobservability is important in situations where exposing the fact that someone is a member of a network may harm the user. For a username ID chosen by the adversary, we define *membership unobservability* as the inability of the adversary to distinguish with better than negligible probability whether there exists some U, Δ such that $(U : ID, \Delta)$ is a registered honest user.

3.2.3 Threat model

For the *unlinkability* (**G1**), *security against impersonation* (**G2**), and *external identity verification* (**G3**) goals, we assume the following threat model. We assume that the adversary can join Pudding with any number of user devices, and may observe the network traffic globally, or interfere with it in arbitrary ways. We assume that at most f out of $n = 3f + 1$ discovery nodes are faulty. A faulty node is either non-Byzantine (crashed, disconnected), or Byzantine (may maliciously deviate from the protocol). We assume that Byzantine nodes are controlled by the adversary, and that they may collude. Non-Byzantine nodes follow an honest-but-curious model [170]: they correctly follow the protocol, but any messages they receive may be leaked to the adversary. The set of non-Byzantine-faulty nodes may change over time, whereas a Byzantine node is assumed to remain Byzantine forever. The users and the non-faulty nodes do not need to know which nodes are faulty. This is a common approach for Byzantine fault-tolerant protocols [124].

We assume that the underlying Loopix construction protects against active and passive network adversaries who wish to learn which nodes are communicating with each other. We base these assumptions upon the Loopix threat model [172]. Lastly, we assume that (1) the email services employed for username verification ensure that an email originating from a specific email address contains a DKIM signature (see §3.4.1) corresponding to that email address, and (2) that email traffic is encrypted with TLS so that a network adversary cannot read or modify messages.

For the *membership unobservability* (**G4**) goal, we assume a weaker threat model, where the adversary controls any number of user devices but no mix, provider, or discovery nodes (an adversary with access to a single discovery node's database can trivially determine which usernames are registered). Despite this weaker threat model, providing membership unobservability against user devices significantly limits the number of parties that have access to this information, and also prevents crawling the database of usernames, which has been an issue in existing encrypted messaging apps [92].

Security parameters. We assume $n = 3f + 1$ discovery nodes for the following reason. Assume f discovery nodes are temporarily unavailable during registration, so $n-f$ discovery nodes have a copy of the contact information for a given username. When searching for that username, a different set of f discovery nodes may be unavailable, but at least $n - 2f \geq f + 1$ nodes will be both available and have a copy of the contact information of that username, which will be sufficient to allow discovery of this username.

3.3 User discovery with Pudding

Pudding user discovery allows a user device to make contact with the registered owner of a particular username. Registering to Pudding with a username allows a user to become discoverable through Pudding by that username, but a user who wants to search for another user does not need to register with Pudding. We first discuss the discovery protocol, and defer the description of the registration protocol to §3.4.

3.3.1 Deterministic SURB generation

Pudding's membership unobservability goal (**G4**) requires that a searcher cannot tell from the discovery nodes' responses whether a given username has been registered. We achieve this by always returning a SURB for a discovery request. If the searched username is registered at the discovery nodes, the SURB will allow the searcher to send a message to the username's owner. Otherwise, the result of the discovery query is a fake SURB that does not route to any real user, but simply causes the message to be dropped in the mix network.

This provides membership unobservability because (i) a user cannot distinguish whether a SURB was generated with fake or real contact information, and (ii) a user cannot distinguish whether a message was dropped by the network, or the recipient user chose not to respond to the message. Although the final mix node in a packet's route can tell whether a SURB is fake, as per §3.2.3, the threat model for membership unobservability only includes malicious users, not adversarial mix or discovery nodes.

SURB generation cannot simply be performed by a single discovery node, since otherwise that discovery node could impersonate any user by routing the searcher's message to an adversary-controlled destination. Instead, we define a deterministic scheme for generating SURBs, so that all honest discovery nodes generate the same SURB for the same query. We then require that a user receives an identical SURB from at least $f + 1$ distinct discovery nodes before sending a message using that SURB. Since we assume that a maximum of f discovery nodes may be Byzantine, there is no way that the adversary can cause the user to receive $f + 1$ incorrect SURBs, and therefore any SURB received from at least $f + 1$ discovery nodes must be correct.

Traditional SURB generation. The original SURB generation procedure for Sphinx [59] takes two inputs: (1) a destination contact information Δ and (2) a sequence of mix nodes $\{n_0, n_1, \ldots, n_{v-1}\}$ through which the message will be routed, where $v < r$ and r is the maximum number of mix nodes in a message route. The SURB generation procedure in Loopix needs randomness for two tasks: (1) picking a delay value d_i for the message to be delayed at each hop n_i, chosen from an exponential distribution so that each mix node behaves as a Poisson process; and

(2) picking an element of a prime-order group, which is used by each mix node in the path to derive a shared secret key, known by the mix node and the SURB's creator. Each mix node uses this shared secret key to derive the keys for validating the integrity of the message, removing a layer of encryption, and learning the next hop in the message route [59, 172].

Deterministic SURB generation. Our SURB generation process is identical to the original procedure, with the exception that all sources of randomness are replaced with a pseudorandom generator seeded with $\mathsf{KDF}(\mathit{nonce} \parallel \mathit{username} \parallel k)$, where *username* is the username being queried, *nonce* is a nonce in the query message, k is a long-lived secret that is shared among all the discovery nodes (set up when the discovery nodes are created), and KDF is a key derivation function. Instead of taking the sequence of mix nodes $\{n_0, n_1, \ldots, n_{v-1}\}$ (the message route) as an input, it also generates this sequence pseudorandomly from the KDF output. The inputs to our modified procedure are therefore the *username*, the *nonce*, and the destination contact information Δ. Determinism ensures that all honest discovery nodes generate the same SURB for the same destination contact information and nonce. Since k is not known to users, a malicious user cannot run the same SURB generation algorithm to break membership unobservability.

While not explicitly outlined in our protocol, potential leaks of the k value can be mitigated through an epoch-based mechanism, where the discovery nodes periodically agree on fresh k values. Lookup failures during key rotation can be reduced by having overlapping epochs. The epoch number can also be incorporated into nonces, which avoids having to store nonces forever.

Creating fake SURBs. When a user queries a username that is not registered on the system, Pudding nevertheless returns a valid SURB in order to provide membership unobservability. This SURB is generated in the same way as a SURB for a genuine user, except that the discovery nodes use a fixed false contact information Δ_{fake} instead of Δ. This Δ_{fake} is a non-existent "black hole" destination, so that any messages sent to it are dropped by the mix network. The querying user cannot distinguish a SURB for the black hole from a SURB for real contact information, because the outermost layer of the SURB header is encrypted with the public key of the first mix node on the path [59], and thus cannot be decrypted

by the user.

If a user makes queries for two different usernames that are both unregistered, the SURB responses will be constructed based on the same Δ_{fake} destination. However, since the seed of the pseudorandom generator incorporates the queried username, the SURBs are encrypted with different keys, and therefore the querying user cannot tell whether the SURBs are based on the same or different contact information. Having a single, fixed Δ_{fake} eliminates the need for coordination between discovery nodes.

The public key within Δ_{fake} must be chosen in a way that the corresponding private key is not known by any party in the system. For elliptic curves, this can be achieved by running a hash-to-curve algorithm [78] with a simple constant input, such as the string "Pudding fake identity", to obtain a group element whose discrete logarithm is unknown.

3.3.2 Authenticating contact information

The SURB described in the previous section allows Alice to send a single message to Bob without Alice learning Bob's contact information, and without Bob learning Alice's identity. However, if Alice wishes to identify herself to Bob by including her username in her message, the SURB does not allow Bob to verify that the message did indeed come from Alice, nor does it allow Alice to verify that replies to her message did indeed come from Bob. Security against impersonation (**G2**) requires users to authenticate each other.

If membership unobservability (**G4**) were not a concern, the discovery nodes could return a user's public key (which is part of their contact information Δ) when queried for a username. Alice and Bob could query the discovery servers for each other's usernames to learn each other's public keys, and then perform an authenticated key exchange protocol to establish a mutually authenticated channel.

To enable mutual authentication while preserving membership unobservability, we use key-blinded signatures [66, 72], a technique also used by Tor to maintain anonymity of Tor onion services [1, 97]. (Not to be confused with blind signatures [44]: key-blinded signatures hide the unblinded public key from the verifier, whereas blind signatures hide the message from the signer.)

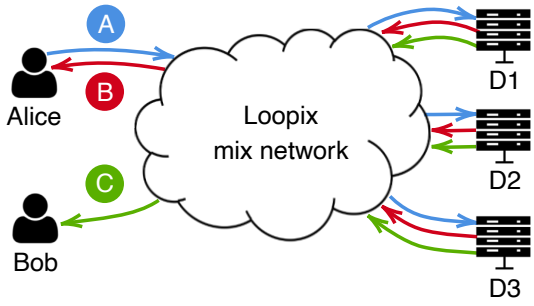
The scheme we use [66] works as follows. Let g be a generator of an elliptic curve group with prime order p. Let $x \in \mathbb{Z}_p$ be a private key and let g^x be the corresponding public key. Let $y \in \mathbb{Z}_p$ be a blinding factor chosen uniformly at random. One can then sign and verify digital signatures similarly to regular ECDSA or EdDSA signatures, using $xy \pmod p$ as private key and g^{xy} as public key. Given a user's public key g^x, a discovery node can therefore generate blinded public keys $(g^x)^{y_1}$, $(g^x)^{y_2}, \ldots$ that cannot be linked with each other, or with the unblinded public key g^x, without knowledge of the blinding factors $y_1, y_2, \ldots$ [72]. We write the blinded-key signing operation as $\mathsf{SignBK}(sk, bk, msg)$ where sk is the unblinded private key, bk is the blinding key, and msg is the data to be signed.

3.3.3 Discovery protocol

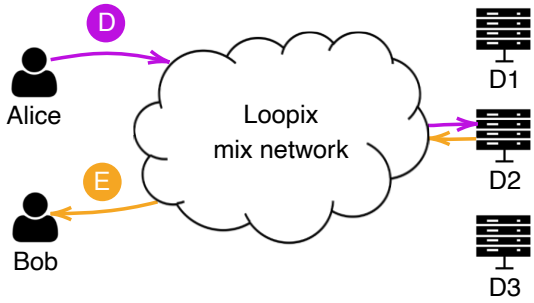
User discovery in Pudding consists of three protocol phases: Lookup queries the database of users, ContactInit allows the searcher to send an initial message to the queried user, and AddFriend establishes an authenticated channel between the two users.

Lookup **protocol.** Let Alice be a user who is searching for Bob, and let ID_A and ID_B be their usernames (having a username is optional in Alice's case). Note that we use "Alice/Bob" as shorthand for "Alice's/Bob's user device". Alice and Bob are both Loopix users, with contact information Δ_A and Δ_B respectively. Each user's contact information includes a long-term public key for which the user's device holds the corresponding private key. Alice's private key is $x_\mathsf{A} \in \mathbb{Z}_p$ and her public key is g^{x_A}, whereas Bob's private and public keys are x_B and g^{x_B} respectively. We also assume that each discovery node has a private signing key, and that Alice and Bob know the corresponding public key for each discovery node (this information can be part of the network topology that a user downloads when first connecting to the Loopix network). The protocol is illustrated in Figure 3.1:

1. Alice generates a random *nonce*. For each discovery node i, Alice also generates a SURB S_i that allows that discovery node to reply to her. She then sends a discovery message $(ID_\mathsf{B}, nonce, S_i)$ to each discovery node via the anonymity network.



(a) During the LOOKUP protocol: (A) Alice sends a query message containing Bob's username to each discovery node; (B) each discovery node replies with a SURB and a blinded public key for Bob; (C) each discovery node also sends Bob the blinding key.



(b) Then, Alice runs the CONTACTINIT protocol where: (D) Alice creates a Sphinx packet M_{init} using the SURB she received in LOOKUP. Alice then sends a message to an arbitrary discovery node, where M_{init} is the payload. (E) The discovery node places M_{init} in the network, which routes the message to Bob.

Figure 3.1: Alice discovering Bob (LOOKUP) and and initiating contact with him (CONTACTINIT) through a Pudding setup with three discovery nodes *(D1, D2, D3)*.

2. Upon receiving a discovery message, each discovery node checks if *nonce* has been used before; if so, it drops the message. This prevents a user from detecting when the contact data for another user changes by making repeated queries using the same nonce.

3. Each discovery node seeds a pseudorandom generator with $\mathsf{KDF}(\mathit{nonce} \parallel \mathit{ID}_{\mathsf{B}} \parallel k)$, where k is a secret shared with the other discovery nodes as described in §3.3.1. If the discovery node has a record for ID_{B} with contact information Δ_{B} in its database, it creates a deterministic SURB *surb* using Δ_{B} and the pseudorandom generator; if not, it generates the SURB using

Δ_{fake} instead. Each discovery node also obtains a blinding key for Bob $y_{\mathsf{B}} \in \mathbb{Z}_p$ from the pseudorandom generator, obtains Bob's public key $g^{x_{\mathsf{B}}}$ from Δ_{B}, and computes the blinded public key $bpk_{\mathsf{B}} = (g^{x_{\mathsf{B}}})^{y_{\mathsf{B}}}$. If ID_{B} is not registered, it computes a blinded key $bpk_{\mathsf{B}} = (pk_{\mathsf{fake}})^{y_{\mathsf{B}}}$ based on the public key pk_{fake} in Δ_{fake}.

4. Each discovery node computes a signature $sig_{\mathsf{A}} = \mathsf{Sign}_{sk}(nonce \parallel surb \parallel bpk_{\mathsf{B}})$ using its signing key sk, and sends $(nonce, surb, bpk_{\mathsf{B}}, sig_{\mathsf{A}})$ to Alice using the SURB S_i in Alice's discovery message.

5. If ID_{B} is registered with contact information Δ_{B}, each discovery node also computes a signature $sig_{\mathsf{B}} = \mathsf{Sign}_{sk}(nonce \parallel y_{\mathsf{B}})$ using its signing key sk, and sends $(nonce, y_{\mathsf{B}}, sig_{\mathsf{B}})$ to Bob using Δ_{B}. This will allow Bob to generate signatures for the blinded public key.

6. For each message received, Alice checks that sig_{A} is a valid signature by one of the discovery nodes. She waits until she has received the same $surb$ and bpk_{B}, along with the $nonce$ she generated, from at least $f + 1$ distinct discovery nodes. If she does not receive them within some timeout, she discards all values and restarts the protocol.

7. For each message received, Bob checks that sig_{B} is a valid signature by one of the discovery nodes. If Bob receives $f + 1$ copies of the same blinding key y_{B} and $nonce$ from distinct discovery nodes, he stores the two values locally in anticipation of a future contact request (see ContactInit Step 3).

ContactInit **protocol.** The SURB received from $f + 1$ discovery nodes during the Lookup protocol allows Alice to send a message to Bob anonymously. However, if Alice sends her initial message directly to Bob using this SURB, an adversary who controls a discovery node in addition to Alice's provider or the first mix node on the message path can recognise the SURB and link it to Alice. To prevent such linking attacks, the message is reflected via a different node. For simplicity, we choose a discovery node as the reflector. Alternatively, the system can incorporate separate, untrusted reflector nodes, or even allow other users to serve as message reflectors. Our protocol allows Alice to incorporate a pseudonym or a codeword in

her initial message, enabling Bob to make an informed decision about whether to respond to her.

1. Alice generates an ephemeral Diffie-Hellman private key $a \in \mathbb{Z}_p$, public key g^a, and initial message key $K_e = \mathsf{KDF}((bpk_{\mathsf{B}})^a \parallel \text{“init key”})$ from the blinded public key received in LOOKUP Step 6. She then uses the SURB from LOOKUP Step 6 to create a Sphinx packet M_{init} [59]. The payload of M_{init} contains g^a, the nonce value she used when searching for Bob, as well as the following fields encrypted under K_e: (1) a SURB that allows Bob to respond to her, (2) an optional pseudonym or codeword for Bob, and (3.a) either her username ID_{A}, or (3.b) if she is not registered or does not want to share her username, Alice samples a blinding key $y_{\mathsf{A}} \in \mathbb{Z}_p$ and uses her long-term public key $g^{x_{\mathsf{A}}}$ to compute a blinded public key $bpk_{\mathsf{A}} = (g^{x_{\mathsf{A}}})^{y_{\mathsf{A}}}$, which she includes in M_{init}. Alice then picks a discovery node and sends it a message via the anonymity network that contains M_{init} as a payload.

2. The discovery node that receives Alice's message sends M_{init} through the anonymity network. It is routed to Bob using the SURB created by the discovery nodes.

3. When Bob receives M_{init}, he computes $K_e = \mathsf{KDF}((g^a)^{x_{\mathsf{B}} y_{\mathsf{B}}} \parallel \text{“init key”})$ using his private key x_{B} and the blinding key y_{B} received in LOOKUP Step 7, and then decrypts the rest of the message. Based on Alice's codeword he can either decide to reply to it, or ignore it. If Bob decides to reply, he proceeds to the ADDFRIEND protocol specified below. If Bob decides to ignore the message, Alice does not learn if ID_{B} is registered to Pudding or not.

4. If Alice does not receive a reply from Bob within some timeout, she retries by sending M_{init} through a different discovery node up to $f + 1$ times to route around faulty discovery nodes.

ADDFRIEND **protocol.** If Bob decides to accept Alice's contact request, the users need to establish an authenticated communication channel: Alice needs to ensure that she is indeed talking to the user registered under the username ID_{B}, and if Alice supplied her username, Bob needs to check that he is indeed talking to ID_{A}.

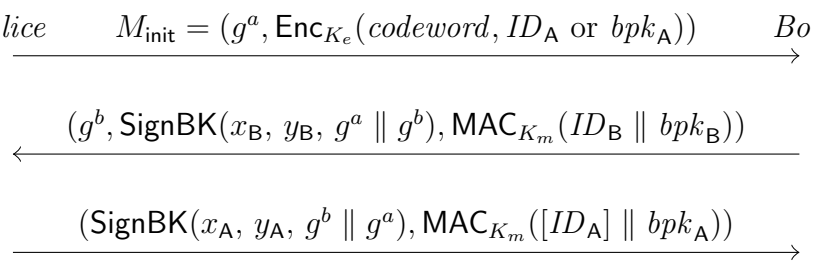


Figure 3.2: Summary of the AddFriend protocol. Square brackets denote that including Alice's username is optional. Omitted from the diagram: each message also includes the nonce from Alice's original Lookup request (to identify messages belonging to the same protocol run) and a SURB that allows the other party to send a reply message; Bob's message to Alice also includes the nonce he used in his optional Lookup run for Alice's username.

We achieve mutual authentication using the SIGMA authenticated key exchange protocol [123], adapted to support blinded-key signatures (see §3.3.2). We use SIGMA because it is fairly simple and proven correct [41], but another authenticated key exchange protocol could be substituted in its place. The AddFriend protocol is shown in Figure 3.2.

1. Bob obtains a blinded public key bpk_A for Alice: (a) if M_{init} does not include Alice's username, Bob uses the bpk_A in this message; (b) if M_{init} includes ID_A, Bob obtains a blinded public key $bpk_\mathsf{A} = (g^{x_\mathsf{A}})^{y_\mathsf{A}}$ for Alice by running Lookup with input ID_A and using the value returned by at least $f + 1$ discovery nodes.

2. Bob uses the blinding key y_B he received from the discovery servers (Lookup Step 7) and his public key g^{x_B} to obtain his own blinded public key $bpk_\mathsf{B} = (g^{x_\mathsf{B}})^{y_\mathsf{B}}$.

3. Bob generates an ephemeral Diffie-Hellman private key $b \in \mathbb{Z}_p$ and public key g^b, and calculates the MAC key $K_m = \mathsf{KDF}((g^a)^b \parallel \text{“MAC key”})$ using the value g^a from M_{init}. He uses K_m to calculate the message authentication code $\mathsf{MAC}_{K_m}(ID_\mathsf{B} \parallel bpk_\mathsf{B})$. Bob also generates $\mathsf{SignBK}(x_\mathsf{B}, y_\mathsf{B}, g^a \parallel g^b)$ using the key-blinded signature scheme [66]. Finally, he sends a message to Alice using the SURB in M_{init}, containing g^b, the MAC, the signature, and a SURB for

Alice to reply. If he ran LOOKUP for ID_{A} in Step 1, Bob also includes the nonce he used in this LOOKUP.

4. If Alice included her username ID_{A} in M_{init}, she waits until she has received a blinding key y_{A} from at least $f + 1$ distinct discovery nodes, as a result of Bob's LOOKUP run at Step 1 of ADDFRIEND. She uses the nonce in Bob's message to identify the blinding key. If Alice chose to remain anonymous, she uses the blinding key y_{A} that she generated at CONTACTINIT Step 1.

5. When Alice receives Bob's message, she checks that the signature over $g^a \parallel g^b$ is valid using the blinded public key bpk_{B} she received from the discovery nodes. She then calculates the MAC key $K_m = \mathsf{KDF}((g^b)^a \parallel \text{“MAC key”})$ and uses it to check the MAC over ID_{B} (the username she originally searched for) and the blinded public key bpk_{B} from the discovery nodes.

6. If all checks pass, Alice sends a reply to Bob containing a key-blinded signature $\mathsf{SignBK}(x_{\mathsf{A}}, y_{\mathsf{A}}, g^b \parallel g^a)$ computed from her private key x_{A} and the blinding key y_{A} from Step 4. If Alice sent her username in M_{init}, the message also includes $\mathsf{MAC}_{K_m}(ID_{\mathsf{A}} \parallel bpk_{\mathsf{A}})$, where $bpk_{\mathsf{A}} = g^{x_{\mathsf{A}} y_{\mathsf{A}}}$. If Alice is anonymous, the message includes $\mathsf{MAC}_{K_m}(bpk_{\mathsf{A}})$.

7. When Bob receives Alice's message, he checks the signature using Alice's blinded public key that he obtained in Step 1, and checks that the MAC over Alice's blinded public key (and optionally username) is valid using key K_m. If these checks succeed, Alice and Bob have fully authenticated each other.

8. Alice and Bob can then derive a shared session key $K_s = \mathsf{KDF}(g^{ab} \parallel \text{“session key”})$, and use an authenticated encryption scheme to secure their further communication (which may include sharing their contact information with each other).

3.4 Registering to Pudding

A user must register to Pudding with a username to be discoverable through that username. However, lookups are possible without registration. We describe the

Pudding registration protocol in this section.

3.4.1 Authentication with DKIM

The conventional way of checking whether a user owns a particular email address is by sending a random token to that address and asking the user to send that token back to the server, typically by clicking an HTTPS link in the email. However, since Pudding has multiple discovery servers, using traditional email address verification would cause complications. The first option to use traditional email authentication in Pudding would be for discovery nodes to trust each other, such that one node performs the verification and the other nodes trust its verdict. This option does not meet Pudding's security goals, since we assume that up to f discovery nodes may be Byzantine and hence not perform the email verification truthfully. The second option would be for each discovery node to perform its own email address verification, meaning the user would have to click n links in n separate emails. This is how Alpenhorn handles email verification for its users [131]. However, we believe that this method is inconvenient for users. We therefore propose a different method for checking whether the user owns an email address.

DomainKeys Identified Email (DKIM) [132] uses a digital signature in a header to confirm the origin of an email. The recipient can retrieve the public verification key that the domain owner has published as a DNS TXT record and use it to verify the DKIM signature. The signature confirms that the email was sent via an SMTP server authorised by the owner of the sender's domain [57]. Assuming that this SMTP server has authenticated the user (the part of the email address before the '@' sign), DKIM allows us to confirm that the sender of a message has not been spoofed.

In Pudding, one discovery node collects challenge values from all discovery nodes, and sends these challenges in a single email to the email address picked as the username. To confirm receipt, the registering user responds to this email using the reply function of their email client. Since email applications attach the original message to the response, and servers and automatically sign the email with the domain's DKIM key, the user typically does not have to type or copy-paste anything. The discovery node that receives the reply then shares it with all other

nodes. This mechanism allows each discovery node to independently confirm that the user's response email is (1) a genuine registration request (by checking if its challenge value is included in the email) and (2) sent by the owner of the email address (by checking the validity of the DKIM signature).

Substitutability of DKIM. Although this implementation uses DKIM, it is not the only viable verification method. For instance, if DKIM verification fails, the system can fall back to a method where each discovery node sends the user a verification link, with the usability downside of requiring multiple confirmation emails and link clicks.

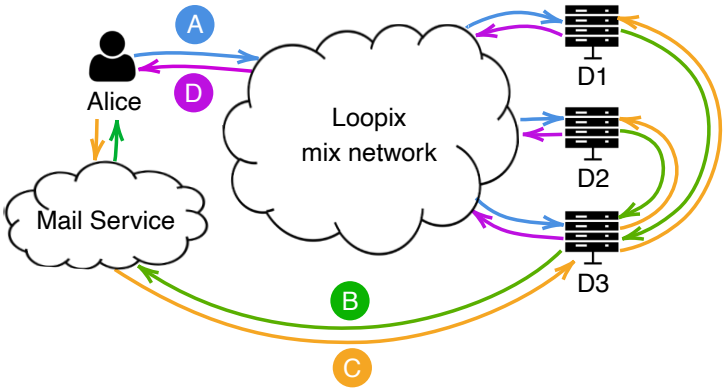


Figure 3.3: Schematic of user Alice registering to a Pudding setup with three discovery nodes (*D1, D2, D3*). (A) Alice sends to the discovery nodes a message containing her email address, contact information, and the ID of a randomly-selected discovery node D_{auth}. (B) D_{auth}, (*D3* in this case), collects challenge values from all discovery nodes and sends Alice an email including these challenges, along with her contact information and username. (C) Alice replies to the email through an email service; *D3* receives Alice's DKIM-signed reply email and sends it to the other discovery nodes. (D) Each discovery node verifies the DKIM signature and its own challenge's presence in the email. Once confirmed, they register Alice and send her a confirmation message.

3.4.2 Registration protocol

The user registration protocol REGISTER is described below and illustrated in Figure 3.3.

1. The user device joins the Loopix network by signing up to a provider node. This process involves creating a public/private key pair for secure communication over the network and obtaining the network topology from the provider (including the set of discovery servers and their public keys). The public key, along with the provider details and any other necessary information, constitutes the user's contact information Δ. The user also chooses an email address that she controls as her username ID.

2. The user device randomly picks a discovery node D_{auth} as the node responsible for sending the verification email. The user then sends a *registration message* to all discovery nodes, which contains ID, the contact information Δ, and the identifier of D_{auth}.

3. After receiving a registration message, each discovery node checks whether it has ID already registered, in which case it marks the registration request as invalid. Regardless, the discovery node generates a random challenge value.

4. Any discovery node that is not D_{auth} sends its challenge value and ID to D_{auth}. Meanwhile, D_{auth} waits until it has challenges from at least $2f + 1$ distinct discovery nodes, including itself (see §3.2.3). D_{auth} then sends a *verification email* to the email address provided as ID, which contains all the challenges and Alice's contact information Δ. Including Δ in the email prevents a malicious D_{auth} from providing a $\Delta_{\mathsf{bad}} \neq \Delta$ to the remaining discovery nodes.

5. If the user receives the email within a predetermined timeout period, she verifies that the included contact information Δ is correct, and responds with an email that includes all challenge values[4]. Otherwise, the user returns to Step 2 to try registering again with a new D_{auth}.

6. When D_{auth} receives the user's response, D_{auth} forwards the email response verbatim, including all headers, to all other discovery nodes. Each discovery

[4]This process can be automated in practice, for example by including a link in the email, which redirects the user from the email app to the anonymous messaging app. After automatically checking that the Δ is correct, the anonymous messaging app can use the OS features to automatically generate a reply email that contains all the email content and headers, and allow user to reply with the click of a button.

node independently checks if (1) the challenge it generated was included in the email, (2) the DKIM signature is valid, using the public key stored in DNS for the domain of ID, and (3) the registration request was not marked as invalid at Step 3. If any of these fail, the discovery node aborts the registration process.

7. Each discovery node that successfully completes the previous steps sends a signed *confirmation message* to all other discovery nodes, which includes Alice's ID and Δ. Once a discovery node has received messages from $2f$ discovery nodes for Alice's ID, it stores the link between ID and Δ in its database. Waiting for confirmation from $2f$ distinct discovery nodes ensures all correct discovery nodes register the same Δ, even if D_{auth} is malicious. Each discovery node then sends a message via Loopix to the user's contact information, confirming registration.

8. If the user device receives at least $2f+1$ confirmations within a predetermined timeout period, the user knows that she is successfully registered in Pudding.

If D_{auth} is malicious, it is possible with this protocol that some honest discovery nodes do not complete the registration. This can be avoided by disseminating (ID, Δ) using a Byzantine reliable broadcast protocol [38, 40].

3.5 Evaluation

To evaluate the Pudding protocol, we compare it against our security goals (§3.5.1), discuss its limitations (§6.3.5), and present findings from implementing Pudding over Nym (§3.5.3).

3.5.1 Security analysis

We begin by formalising the goals **G1**...**G4** from §3.2.3 as security games. We denote the adversary as $\mathcal{A}$ and challenger as $\mathcal{C}$. $U_{\mathcal{C}}$ and $U_{\mathcal{A}}$ refer to users controlled by the challenger and adversary, respectively.

Unlinkability game Exp_{G1}: The adversary $\mathcal{A}$ controls up to f discovery nodes, the user $U_{\mathcal{A}}$, the mix nodes in all but one of the Loopix network layers, and the

network between all of the nodes. The challenger $\mathcal{C}$ controls users $U_{\mathcal{C}}$, U_0, and U_1 chosen by the adversary, and generates a random bit $b \in_R \{0,1\}$. $U_{\mathcal{C}}$ is registered to the discovery nodes with the username $ID_{\mathcal{C}}$. U_b runs the discovery protocol (Lookup, ContactInit, and AddFriend) with input $ID_{\mathcal{C}}$, and $U_{\mathcal{C}}$ decides to respond in ContactInit Step 3. Finally, $\mathcal{A}$ outputs b'. The adversary wins if $b = b'$.

Exp_{G1} **security arguments:** We claim that the adversary cannot win Exp_{G1} with a significantly higher probability than the chance of breaking sender-receiver unlinkability in the underlying Loopix network (which depends on the size of the anonymity set). The reason is threefold. (1) An adversary performing passive or active attacks on the network links between nodes or adversary-controlled mix nodes is limited by the unlinkability property of Loopix. (2) Controlling discovery nodes does not confer the adversary an advantage, since all communication between U_b and those nodes is done via SURBs, which hide the identity of the recipient. (3) The potentially identifying information in M_{init} is encrypted with the blinded public key of $U_{\mathcal{C}}$, which $\mathcal{A}$ does not control. Appendix A.1 provides a more detailed discussion.

Security against impersonation game Exp_{G2}**:** There are three users: $U_{\mathcal{C}}$, $U_{\mathcal{A}}$, and an honest user $U_{\mathcal{H}}$ controlled by $\mathcal{C}$. $U_{\mathcal{H}}$ registers using ID, which is an email address not controlled by $\mathcal{A}$ and known to $\mathcal{C}$. $\mathcal{A}$ controls up to f discovery nodes, any number of mix nodes and provider nodes, and the network between all of the nodes. $\mathcal{A}$ chooses two messages m_0, m_1 of the same length and reveals them to $\mathcal{C}$. $\mathcal{C}$ generates a random bit $b \in_R \{0,1\}$. $U_{\mathcal{C}}$ runs the discovery protocol with input ID, and once AddFriend completes, $U_{\mathcal{C}}$ sends a message m_b encrypted with the session key K_s. Meanwhile, the adversary may inject arbitrary messages into the network. $\mathcal{A}$ outputs b', and wins the game if $b' = b$.

Exp_{G2} **security arguments:** We claim that the adversary has a negligible advantage over a random guess. (1) $U_{\mathcal{C}}$ only accepts one response per discovery node per Lookup, since responses are signed and linked with $U_{\mathcal{C}}$'s nonce. Moreover, $\mathcal{A}$ cannot modify the Lookup responses from the honest discovery nodes, since the Sphinx packet format ensures integrity [59]. Therefore, $U_{\mathcal{C}}$ will not reach the threshold of $f + 1$ responses required to accept the adversary's responses, and

therefore the SURB and blinded public key used by $U_{\mathcal{C}}$ will be the one generated by the honest discovery nodes using the contact information of $U_{\mathcal{H}}$. (2) The adversary knows the nonce and SURBs for $U_{\mathcal{C}}$, and can therefore send fake AddFriend protocol messages to $U_{\mathcal{C}}$ and $U_{\mathcal{H}}$. However, due to the existential unforgeability of the key-blinded signature scheme, the adversary cannot forge a signature that $U_{\mathcal{C}}$ will accept as valid for $U_{\mathcal{H}}$'s blinded public key (as produced by the honest discovery nodes). Therefore, the computation of K_m at AddFriend Step 5 will only be performed using the g^b sent by $U_{\mathcal{H}}$, and therefore $\mathcal{A}$ does not learn K_s. Even though the adversary can trick $U_{\mathcal{H}}$ into sending the ciphertext of m_b to $U_{\mathcal{A}}$ by replacing the SURB in AddFriend protocol messages, by the IND-CPA property of the symmetric cipher used to encrypt m_b, the adversary has only negligible advantage over guessing b. See Appendix A.2 for further details.

External identity verification game Exp_{G3}: $\mathcal{A}$ controls f discovery nodes out of $3f + 1$, as well as the user $U_{\mathcal{A}}$. $\mathcal{C}$ controls the remaining $2f + 1$ discovery nodes. ID is an email address not controlled by $\mathcal{A}$. Per our threat model for **G3** (§3.2.3), the email service used for identity verification is modelled as an oracle, guaranteeing that an email originating from an email address includes a DKIM signature that corresponds to that particular email address. $U_{\mathcal{A}}$ runs the Register protocol for ID. $\mathcal{A}$ wins the game if any of the $2f + 1$ challenger-controlled discovery nodes registers ID.

Exp_{G3} security arguments: We claim that the adversary has a negligible probability of winning game Exp_{G3}. In order to successfully complete the Register protocol for ID, $\mathcal{A}$ would have to forge a digital signature and pass the checks in Register Step 6 for any honest discovery node.

Membership unobservability game Exp_{G4}: $\mathcal{A}$ controls an arbitrary number of users $U_{\mathcal{A}0}, U_{\mathcal{A}1}, \ldots$, but $\mathcal{A}$ does not have access to the internal states of any discovery or provider nodes, as per our threat model for **G4** (§3.2.3). ID_0 and ID_1 are initially unregistered usernames known to both $\mathcal{A}$ and $\mathcal{C}$. $\mathcal{C}$ controls user $U_{\mathcal{C}}$, generates a random bit $b \in_R \{0, 1\}$, and runs the Register protocol to register $(U_{\mathcal{C}} : ID_b, \Delta)$ for some arbitrary Δ not controlled by $\mathcal{A}$. $\mathcal{A}$ then runs the discovery protocol for ID_0 and ID_1 from any adversary-controlled users arbitrarily many times, but $U_{\mathcal{C}}$ does not respond to any of those contact requests. Finally, $\mathcal{A}$ outputs

b' and wins the game if $b = b'$.

Exp_{G4} **security arguments:** We claim that $\mathcal{A}$ has a negligible advantage over a random guess. Since $U_{\mathcal{C}}$ does not respond to M_{init}, the only messages received by the adversary are the discovery node responses. These messages contain the searcher's nonce, a SURB, a blinded public key, and a signature. The nonce and signature do not depend on whether the username is registered; the SURB is computationally indistinguishable from a uniform random bit string [59]; and the key-blinding scheme we use ensures that the blinded public key for real contact information is indistinguishable from that for Δ_{fake} [66].

3.5.2 Discussion and limitations

Email service provider. There are two attacks that an adversary who controls a corrupted email service provider can perform. (1) An adversary who controls the user's SMTP server, or who can intercept unencrypted SMTP traffic, can register with Pudding on behalf of any email address on that server. (2) DKIM works at the *domain level*: it does not strictly verify that the part of the email address before the '@' sign is correct, but that an email has been signed by an email server that holds the private key for a particular domain [57]. Therefore, if the adversary has access to the domain's signing key, the adversary can forge emails with valid DKIM signatures, which appear to be sent from any email address hosted by that domain.

Email traffic. Even if the attacker does not control an email service provider, she can try to learn if a user has registered to Pudding with a certain email address by observing the network traffic between the user device and an email service provider, or between D_{auth} and the email service provider (e.g. for personal domains). This attack is outside our threat model.

Malicious Loopix providers. A Loopix provider node receives messages on behalf of its users while the users are offline, and allows users to download those messages when they next connect to the provider. This design is advantageous for mobile devices that are frequently offline, but it has the disadvantage that a malicious provider can learn the number of messages received over time by each of its users (but not their origin). Some messaging patterns, such as receiving $3f + 1$

messages in short succession, could indicate that the user has just performed a LOOKUP request. By linking the timing of such message receipt patterns with the timing of LOOKUP requests at a colluding discovery node, an adversary can make statistical estimates of which users might be searching for each other. This attack can be mitigated by ensuring that users receive sufficient (real or cover) traffic, and by increasing the size of the anonymity set by having more users submitting LOOKUP requests.

Spam. Spam is usually detected by looking at message contents. However, end-to-end encrypted messaging apps can reduce spam by detecting high volumes of automated messages coming from the same source and directed at a high number of users [115]. Since Pudding does not have access to such metadata, we should use other measures to prevent spam. One strict yet effective measure can be dropping the message if the person is not in your contact address book. Another measure could be making spamming expensive; for example, Nym requires its users to purchase bandwidth credentials.

Pseudonymous usernames. It may seem that Pudding should support pseudonymous identities for users who do not want to tie a non-anonymous identifier, such as an email address, to their anonymity network presence. However, removing external verification from registration also eliminates any chance of achieving membership unobservability. If anybody can register any username, an adversary can check whether username *ID* is registered by attempting to register *ID*, and then using another device to discover *ID* and send a message to it. If the adversary receives her own message, then *ID* was not previously registered. If the adversary does not receive her own message, even after several retries, then *ID* must have already been registered.

Consequently, in a seemingly paradoxical manner, tying anonymity network accounts to external identities such as email addresses provides *better* privacy than using short pseudonymous identities, in terms of hiding membership of the network. We have therefore chosen not to include pseudonymous usernames as a registration option in Pudding. However, we do not require users to register to discover others or send messages. Only one party in a conversation needs to register with Pudding in order to initiate a two-way conversation.

Online / offline patterns. Loopix hides communication patterns, however the provider nodes can still determine whether a user device is online or offline. If device pairs transition between online and offline states in a correlated fashion, it can potentially be used to infer that two users are communicating. However, this type of intersection attack is not unique to Loopix, but rather a concern in all anonymity networks [60]. This vulnerability is also acknowledged by other private user discovery protocols, as highlighted in Alpenhorn [131] and UDM [45].

Usability. In this work, we base Pudding's usability traits upon previous work. We argue that enabling user discovery through usernames increases usability, because previous research shows that users are not good at generating and exchanging information needed to establish communication over an encrypted channel [216, 178, 183]. Using email addresses as usernames increases usability, based on the fact that all popular encrypted messaging apps allow users to find their friends through identifiers that already exist in their address books. The requirement to allow user discovery without leaving the network draws from research showing that authentication ceremonies of encrypted messaging apps have poor usability [210, 94]. In such ceremonies, users share public keys or verification codes manually with each other via QR code, near-field communication (NFC), or through speech [206]. Although during registration to Pudding the user has to leave the anonymity network to respond to a verification email, this only happens once for each username registration, and the user typically does not need to take any action other than hitting the reply button. Further work is required to measure the specific usability improvement that our protocol presents.

3.5.3 Performance evaluation

We have implemented a prototype of Pudding on Nym to demonstrate that our protocol is practical. The prototype is implemented in 3000 lines of Rust code using the Nym SDK[5]. We also modified the SDK and the `sphinx-packet` crate to expose and modify internal methods that allow us to create deterministic SURBs and access the user's secret keys. The changes are contained in a `.patch` file with less than 2000 lines. Importantly, no modifications of Nym infrastructure (mix

[5] `https://nymtech.net/docs/sdk/rust.html`

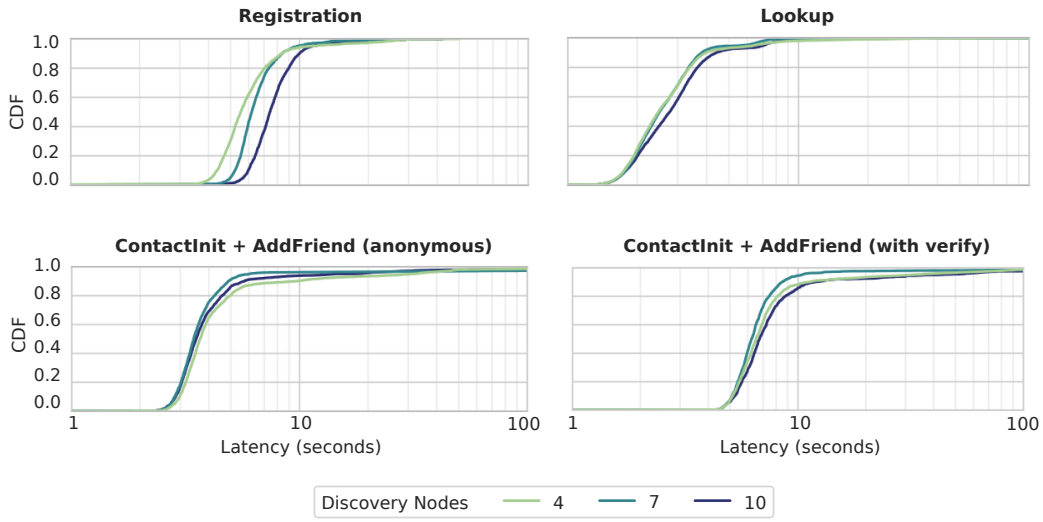


Figure 3.4: Latency of the Pudding sub-protocols represented as CDFs for different numbers of discovery nodes. Note that the x-Axis is log-scaled. See also Table 3.1.

nodes, provider nodes, or protocols) are required. As such, all our experiments run on the general main net and share the same anonymity set as existing Nym users. We have chosen Nym because it is an operational anonymity network running on hundreds of nodes, and it does not currently provide a user discovery mechanism. All source code and SDK modifications are available in our open-source repository under an MIT license: `https://github.com/ckocaogullar/pudding-protocol`.

The Nym SDK manages SURBs when sending messages to other recipients using their Nym address. When sending a message from a user U_{sender} to another user $U_{\text{recipient}}$, the Nym client transparently includes SURBs so that $U_{\text{recipient}}$ can reply without knowing U_{sender}'s address. As messages are exchanged, $U_{\text{recipient}}$ requests more SURBs from U_{sender} when her stock runs low. This mechanism is used in our protocol when the user communicates with the discovery servers. However, when using our deterministic SURBs (§3.3.1) both parties are anonymous towards each other and we have to make sure ourselves that the responses include sufficiently many SURBs.

In our evaluation, all clients run on a machine with an 8-core CPU (Intel Xeon Skylake) in a commercial data center. During an initialisation phase, the test program creates ephemeral clients with randomly chosen gateways for the users

# Discovery Nodes	Latency [s]			
	mean	p_{50}	p_{90}	p_{95}
Registration				
4	6.48	5.54	8.42	11.01
7	6.74	6.18	8.35	9.74
10	8.04	7.44	10.02	11.37
Lookup				
4	3.26	2.48	3.97	6.45
7	3.00	2.51	3.81	5.73
10	3.44	2.77	4.41	7.08
ContactInit + AddFriend (anonymous)				
4	8.77	3.66	9.47	30.44
7	10.58	3.43	4.87	6.01
10	8.21	3.51	5.70	14.21
ContactInit + AddFriend (with verify)				
4	11.51	6.52	11.16	32.06
7	10.00	6.23	8.55	10.35
10	12.34	6.75	11.86	38.07

Table 3.1: Registration and discovery requests latency over Nym for different numbers of discovery nodes.

and discovery servers. All clients execute in a multi-threaded asynchronous runtime inside the same process. However, they do not share any information out-of-band once started. The DKIM verification step is simulated by another Nym client that replies with a signature immediately, effectively excluding the time taken by the user to receive and reply to an email.

In this evaluation, we are interested in the latency distributions for the Pudding sub-protocols. Specifically, we are interested in four metrics: latency of the Register protocol (§3.4), latency of the Lookup protocol (§3.3.3), and the latency for the combination of the ContactInit and AddFriend protocols in both the anonymous and non-anonymous variant. Since Lookup is required for ContactInit, these protocols are executed as part of the same scenario. Therefore, we execute three different scenarios: a registration scenario, an anonymous discovery (Lookup +ContactInit +AddFriend) scenario, and a non-anonymous discovery scenario. We run each scenario for 10 minutes and repeat it 6 times to account for outliers, such as unlucky choice of slow gateway nodes. We run configurations with 20 client nodes and a varying number of $n = \{4, 7, 10\}$ discovery nodes.

For the Register scenario, each client starts a new registration attempt

every 30 seconds after an initial random pause. Similarly, for the other scenarios every client sends a LOOKUP request for a randomly chosen other user every 30 seconds and proceeds then with the CONTACTINIT and ADDFRIEND steps. For the discovery scenarios, all clients are pre-registered with the discovery nodes such that the results are not influenced by pending registration operations. We stop the clock for the REGISTER operation once the user receives $2f + 1$ confirmations, and for the LOOKUP operation once the searcher receives $f + 1$ confirmations. The combination of CONTACTINIT + ADDFRIEND is measured from the time when the searcher sends her first message to receiving the confirmation from the searchee, at which point the searcher and searchee can send messages to each other.

We have verified that the number of users (up to the maximum that our test bed can support) has negligible impact on the latency distributions. We also noted occasional instances where some messages were lost or nodes experienced temporary disconnections from the Nym network. Since these cases are automatically covered by the fault-tolerance of our protocol, we observed little difference between executions with failed nodes and those without.

Registration and discovery over Nym are fast. Our results are presented as cumulative distributions in Figure 3.4, while detailed numerical values can be found in Table 3.1. All operations have mean latencies below 13 seconds. We therefore believe that both registration and discovery latency are suitable for real-world usage.

For this paragraph, we consider the scenario with $n = 10$ discovery servers. The mean latency for REGISTER is 8.04 seconds ($p_{90} = 10.02$ s). These numbers are lower for the $n = 4$ configuration. This is because, as n increases, the client has to wait for a larger number of replies which increases the chance of having to wait for a message with large mix node delays. The mean LOOKUP latency is 3.44 seconds ($p_{90} = 4.41$ s). It is similar for other n as there are no "synchronization barriers", where a single discovery node has to collect message from the others. The latency for the combined CONTACTINIT and ADDFRIEND step differs between the anonymous and non-anonymous variants. For the former, the mean latency is 8.21 seconds ($p_{90} = 5.70$ s) while the non-anonymous variant has a mean latency of 12.34 seconds ($p_{90} = 11.86$ s). The non-anonymous variant is expected to be slower, because the searchee has to perform an extra communication round with

the discovery servers.

Pudding's performance is influenced by the anonymity network's throughput limits. Under high load, the latencies can exhibit significant outliers. This is because Nym clients send data at a fixed rate (adding cover traffic if necessary) to prevent traffic analysis. Consequently, sending more messages increases the expected queueing delay. This effect is especially pronounced for the discovery requests, where the responses from the discovery nodes exceed the size limit of a single payload in Nym. As a result, each response to the searcher must be split up into multiple Nym packets (in our implementation we need up to ten).

In a real deployment, we anticipate that the queuing delay on user devices will be small, because registration and discovery requests are sent rarely—occurring only once during app installation for registration and once during initial communication with a friend for discovery. However, the discovery nodes may need to handle a high rate of requests and responses. This can be addressed by running multiple instances of the Nym client at each discovery node, and load-balancing messages across them. For this, the list of discovery nodes may contain multiple Nym identifiers for each, and clients randomly pick one when sending a message to that node.

3.6 Related work

In this section, I discuss existing user discovery schemes, as well as cryptographic and hardware solutions that can support private user discovery. Finally, I outline the primary differences between the Pudding protocol introduced in this chapter and the protocol I presented in my MPhil dissertation on Advanced Computer Science.

3.6.1 Private user discovery schemes

A relatively small body of literature is concerned with providing user discovery for anonymity networks. We reviewed six systems (see Table 3.2), which are the only existing schemes we are aware of that provide user discovery while hiding who is communicating with whom: Alpenhorn [131], UDM [45], PROUD [167], OUTOPIA [168], Arke [156], and Apres [126].

Protocol	Ref.	Fault Tolerance	Traffic Analysis Protection	Out-of-Band Exchange	Membership Unobservability	External Identity Verification	Key Limitations
Alpenhorn	[131]	○	●	○	●	○	Requires large mailbox downloads, vulnerable to delays and server failures
UDM	[45]	○	◐	○	○	○	No default fault tolerance; weak passive adversary protection
PROUD	[167]	○	○	●	○	○	Requires out-of-band key exchange; no traffic analysis protection
OUTOPIA	[168]	○	○	●	○	○	Requires out-of-band exchange; not user-friendly
Arke	[156]	●	◐	○	○	○	Mutual interest required; unsuitable for unilateral contact
Apres	[126]	○	●	●	○	○	Pairwise IDs needed; secrecy and hard-to-guess IDs required

Table 3.2: Table comparing some security, privacy and usability properties of the existing private user discovery schemes.

Alpenhorn [131] is based on the Vuvuzela [208] anonymity network, and therefore provides a high level of metadata privacy. Alpenhorn has a round-based system that requires users to periodically download large inboxes rather than accessing individual messages on demand. As a result, this scheme has high bandwidth costs, which is especially challenging for devices relying on cellular networks. For instance, with one million users, each user device needs to download a 7.5 MB mailbox, which exceeds the allowance of many cellular data plans [131]. The alternative of delaying mailbox downloads until WiFi is available can increase the lookup latency. Furthermore, if a user device fails to participate in a round where a lookup request was initiated for that user, the request will be lost. Moreover, Alpenhorn cannot remain operational if a single server crashes, goes offline or otherwise does not follow the protocol. Pudding meanwhile, is fault-tolerant and fast; it works by exchanging small messages rather than large mailboxes.

UDM [45] aims to hide contact relationships and protect users' external public identities from being linked to their private network identities. Compared to

Pudding, UDM does not provide *membership unobservability* (**G4**) or *external identity verification* (**G3**). In addition, UDM does not prevent an adversary from deducing contact relationships by passively observing network patterns. Lastly, UDM does not provide fault tolerance by default, and discusses relevant measures as extensions to the current model.

PROUD [167] is a DNS-based private user discovery system that does not provide traffic analysis protection and requires users to exchange public keys out-of-band. Similarly, *OUTOPIA* [168] requires users to exchange out-of-band information such as public keys and "long hard-to-guess names", and is therefore not a usable discovery mechanism. Neither of these schemes meet our usability goals.

Arke [156] is a contact discovery protocol based on an unlinkable handshake mechanism, built atop an identity-based non-interactive key exchange. While it provides Byzantine fault tolerance like Pudding, Arke does not offer inherent unlinkability against passive adversaries observing network traffic. Also notably, Arke requires mutual interest for contact initiation, making it unsuitable for scenarios where one party seeks to contact another unilaterally, such as a whistleblower reaching out to a journalist.

Apres [126] allows users to find and message each other privately, providing unlikability and online unobservability. However, Apres requires each pair of users to communicate out-of-band and determine a pairwise introduction ID, which must remain secret and therefore also be hard-to-guess. In contrast, Pudding users have a single non-secret ID.

3.6.2 Cryptographic and hardware solutions

Private discovery of network entities is relevant for many types of anonymous and other communication networks. Below, we detail some prominent techniques and systems used for various private discovery purposes.

Private Set Intersection (PSI) protocols allow two or more parties to find the intersection of sets without learning anything else about each other's sets, except maybe their sizes [64]. PSI is a suitable solution for privately determining the intersection of the contacts in a user's address book and the users registered in the

communication system [116]. However, PSI is not directly applicable to retrieving contact information.

Private Information Retrieval (PIR) protocols allow querying items from a server without revealing the query to the server [24]. Demmler et al. used a technique combining PIR with PSI to privately find which users in someone's contact address book are registered in a messaging system [64]. However, this technique does not achieve *membership unobservability* (**G4**). DP5 [37] also aims to solve a related but different problem: privately learning the online status of one's contacts. DP5 also does not provide *membership unobservability* (**G4**), and it requires users to exchange private keys out-of-band.

Trusted Execution Environments (TEEs) are used for protecting the confidentiality of data *in-use*. In the user discovery context, Signal uses Intel Software Guard Extensions (SGX) to privately find the intersection of the user's address book and the users registered to the messenger service [141]. TEEs can also be integrated into Pudding discovery nodes, to reduce the level of trust associated with discovery node providers.

Private DNS solutions including DNSCrypt [164], DNSCurve [65], Confidential DNS [217] and Private DNS [93], focus on protecting DNS queries and responses from being read or modified by unauthorised parties. These mechanisms are not suitable for private user discovery in anonymity networks, since they do not prevent the DNS servers from linking users with their queries. More recent approaches like Oblivious DNS [180], DNS for Tor [179], and distributed DNS [96] are not resistant against global passive adversaries.

Distributed Hash Tables (DHTs) allow peers in the system to search for data objects using the keys associated with them. As with DNS, most DHT implementations do not provide protection against linking users with queries. The ones that aim to provide this property, including Salsa [161], AP3 [153], NISAN [166], Torsk [142] are susceptible to information leakage, as illustrated by numerous studies [166, 154, 211, 61].

3.6.3 Comparison to the old Pudding protocol

This section analyses the key distinctions between the Pudding protocol presented in my MPhil in Advanced Computer Science dissertation and the one described in this chapter. Notably, the earlier protocol did not provide membership unobservability, lacked a mechanism for establishing long-term contact after discovery, and relied on Shamir secret sharing for discovery, which offered fewer privacy benefits compared to the new deterministic SURB generation approach. Additionally, the earlier protocol was not evaluated for its performance.

Membership unobservability. One of the major improvements in the new Pudding protocol is achieving membership unobservability (**G4**).

In the older protocol (including both the Incognito and ID-Verified Pudding variants), an attacker could easily attempt to register a username and determine whether it was already taken based on the response from the discovery nodes. In contrast, the new protocol completely eliminates this issue in registration by ensuring that discovery nodes no longer provide clear responses to registration attempts.

In the discovery phase, the new protocol also protects membership unobservability by having the discovery nodes return a valid SURB for every query, regardless of whether the username exists. If the username is unregistered, the SURB points to a "black hole" destination. This makes it impossible for the searcher to tell whether the queried username is registered or not.

The old version of Pudding attempted to hide usernames by using a technique that combined salts and an Oblivious Pseudorandom Function (OPRF). An OPRF is a pseudorandom function, where a *key holder* (k) and *input provider* (x) jointly compute $F_k(x) = y$, where the provider learns y, and the key holder learns nothing [80]. Salts were used to create pseudorandom handles for usernames, which were stored in the discovery nodes. When discovering a user, the searcher had to interact with multiple discovery nodes to recreate the handle using the OPRF. While this method worked against offline attacks where a malicious discovery node searched its database without interacting with others, it could not stop online attacks. In other words, discovery nodes could cooperate and query each other's databases to identify usernames. Although rate limiting was suggested as a potential fix, it is

difficult to enforce in real-world scenarios [92]. The new protocol simplifies this process and ensures better protection against these vulnerabilities.

Transition to deterministic SURBs. In the old Pudding protocol, users registered to discovery nodes with Shamir secret shares [181] of their contact information, instead of the actual contact information. Discovery was achieved by querying a threshold number of discovery nodes, which respond with Shamir secret shares of the contact information, and reconstructing the contact information. Discovery nodes could similarly legitimately query other discovery nodes to extract contact information associated with registered usernames. Therefore, this mechanism did not bring any significant privacy benefits.

In the new Pudding protocol, we replace Shamir secret sharing with a deterministic SURB generation mechanism. In this approach, each discovery node independently generates identical SURBs for the same query; but the SURB is different for different queries even though the contact information used for generating the SURB is the same. This prevents a searcher from detecting if a contact information linked to a username changes, which can be useful information, signalling change of network location, provider, or registering to network (i.e. contact information changing from Δ_{fake} to Δ).

Establishing long-term contact. The old Pudding protocol focused solely on user discovery and lacked a mechanism to establish persistent communication. The new protocol addresses this limitation by introducing the AddFriend protocol (§3.3.3), which facilitates long-term communication in a secure and privacy-preserving manner. This mechanism allows users to exchange cryptographic keys while keeping their contact information hidden.

Using key-blinded signatures, the protocol ensures that the searcher sees only a pseudorandom value derived from the contact information and a blinding key, rather than the actual contact information itself. This approach not only prevents the exposure of sensitive information during key exchanges but also ensures that any future changes to a user's contact information remain hidden from the searcher, which can provide privacy benefits as described above in *Transition to deterministic SURBs*.

Evaluation. The evaluation of the old Pudding was based a model-checking

approach using a custom discrete-event simulator. Although this method was effective for verifying the protocol's correctness in terms of completeness and liveness, it did not provide insights into its performance. In contrast, the new Pudding protocol includes a full implementation and a comprehensive performance evaluation conducted on the Nym anonymity network. The results demonstrate the practicality and scalability of the updated protocol, and showcase its readiness for deployment in real-world applications.

3.7 Summary

In this chapter, I described Pudding, a novel private user discovery protocol that hides contact relationships, prevents impersonation, and provides fault-tolerance. Pudding also provides membership unobservability against malicious users using a new deterministic single-use reply block (SURB) generation mechanism. Users register to Pudding with their email address. We verify email ownership of through a mechanism based on DomainKey Identified Mails (DKIM). A messaging app can then automatically search for the friends of the user by using the email addresses of friends stored in her address book, confident that the recipient is in control of the associated email inbox.

Pudding is an application-layer protocol; it can be implemented as an overlay network on top of any anonymity network that supports SURBs. We have implemented our protocol, and evaluated its performance on Nym anonymity network. Our evaluation shows that Pudding is scalable, practical, and lightweight, achieving mean latencies of under 4 seconds for lookup, 13 seconds for initiating contact, and 8.5 seconds for registration—numbers significantly influenced by the delays in the underlying anonymity network.

In the next chapter, I introduce the Confidential Computing Transparency framework, which enables users to build trust in Confidential Computing systems. While Pudding focuses on establishing trust between users in privacy-sensitive environments such as anonymity networks, users may also need to trust the underlying infrastructure of highly secure and private systems. The Confidential Computing Transparency framework addresses this challenge by providing mechanisms to verify the integrity and trustworthiness of non-human components within these systems.

Chapter 4

A transparency framework for Confidential Computing

A fundamental challenge in Confidential Computing is that it requires a certain level of user trust. Trust here refers to the user's confidence in the secure and correct behaviour of the Confidential Computing system's components. While attestation verifies that the intended software is running on an intended TEE, it does not guarantee the absence of vulnerabilities or backdoors within a Confidential Computing system. Therefore, unless the user builds and installs the entire software stack that runs on top of the TEE by themselves, and the hardware is correct and reviewable, attestation alone does not provide enough evidence to the user to trust a Confidential Computing system (§4.1.1).

Transparency offers a complementary approach to overcome these inherent limitations of attestation by allowing scrutiny from those other than the ones who have built a Confidential Computing system, which can lead to quicker detection and resolution of issues. Companies that produce TEEs for Confidential Computing are exploring different transparency strategies to reduce reliance on blind trust (§2.5.1). However, the lack of standardisation makes it difficult to assess and compare their effectiveness. Moreover, the high implementation costs and uncertain immediate benefits pose challenges, particularly for smaller organisations, limiting broader adoption.

To address this challenge, this chapter presents the Confidential Computing

Transparency framework, which systematises the landscape of possible solutions (§4.2) for establishing human-to-machine trust in Confidential Computing. This new framework defines an end-to-end trust chain using methods such as open-source code, reproducible builds and provenance or endorsement statements. These methods are underpinned by digital certificates and verifiable transparency logs to allow user attestation. Our framework has three transparency levels, each defined by different agents of transparency, who can be first-party, third-party or community reviewers. This tiered structure allows organisations to progressively improve their transparency practices as their capabilities and resources allow. Although focused on Confidential Computing, the framework's principles may also extend to broader hardware and binary transparency contexts.

This chapter draws from the part of our paper "A Confidential Computing Transparency Framework for a Comprehensive Trust Chain" [119] that presents the framework. The parts of the paper related to the user study will be covered separately in Chapter 5. I led the development of the transparency framework, shaping its core principles and structure, analysing existing transparency practices, and outlining the foundational concepts, scope, and key considerations of transparency in Confidential Computing, and was the primary author of the text. Ivan, Ben, Al, and Christoph provided insights on industry practices, while Alastair, along with these co-authors, contributed to refining the framework's conceptual foundations and shaping the overall idea development and presentation.

4.1 Defining transparency

In this section, we answer five important questions about transparency in Confidential Computing: what is it and why is it important (§4.1.1), what is subject to transparency (§4.1.2), who stands to benefit from it (§4.1.4), and who facilitates it (§4.1.3). We defer answering the question of how transparency can be achieved to §4.2.

4.1.1 Necessity of transparency

Attestation is a tool that provides information on the trustworthiness of a TEE. It often involves a hardware-signed proof of some information about the origin and current state of a TEE [53]. There are three types of claims that may be derived from this signed proof:

1. ***Authenticity:*** Attestation allows users to verify the origin of the TEE and the workload running inside it. It involves verifying that the TEE has been produced by the expected manufacturer, and that it runs the expected workload.

2. ***Integrity:*** Attestation can provide users with claims about whether software components like firmware, bootloader, operating system and workload, have been manipulated.

3. ***Runtime configuration:*** Attestation can additionally provide partial insights into the runtime configuration of the current state of the TEE, such as the execution mode.

Although attestation is essential for building trust in the authenticity and integrity of a TEE [53], it has limitations in providing comprehensive security assurance. Notably, it cannot detect vulnerabilities or backdoors within attested components. For instance, a backdoor in the TEE's firmware or hardware Trusted Computing Base (TCB) could allow unauthorised access to user data without affecting the measurable authenticity or integrity verified by attestation, and the limited runtime information included in attestation evidence.

A transparent approach with thorough review processes is crucial for uncovering vulnerabilities or backdoors. Such reviews can include inspecting the source code of critical binaries, evaluating the tools used in generating the binaries from the source code, and conducting comprehensive assessments of the system's architecture (§4.1.2). The review methodologies may involve manual inspections, automated testing, formal reasoning (see §4.3.4), and other systematic techniques. Embracing this kind of transparency can mitigate hidden threats, empower users and stakeholders to make informed decisions, and ultimately provide a high level of assurance about the security of a Confidential Computing system.

4.1.2 Scope of transparency

Transparency in Confidential Computing must extend beyond the TEE. Communication channels with a TEE, where users load private data and code, must be secured, for example, by using a cryptographic protocol. Similarly, when multiple TEEs (e.g. those on a CPU and a GPU) collaborate, the communication between them must also be secure. Therefore, we consider the entire end-to-end Confidential Computing systems rather than focusing only on TEEs.

In an end-to-end Confidential Computing system, we classify components that could jeopardise the confidentiality or integrity of user data if compromised as *sensitive components*. These include the TEE TCB for security and integrity (e.g. the components that process plaintext user data, generate or store keys), as well as endpoints and protocols used in TEE-to-user or TEE-to-TEE communication. While transparency should ideally extend to all sensitive components, its benefits depend on three key prerequisites:

1. ***Reviewability:*** Security insights about a component should be achievable through inspection. This goal is inherently subjective and must be considered case-by-case. For example, the ideal approach to achieve reviewability for a software component is granting reviewers access to the source code. However, even with source code access, nuances exist [195]; access to the documentation, architecture, reference manuals, or expert guidance might impact reviewability. The same applies to granting access to the commit history versus a snapshot of source code. In cases where providing source code access is not feasible, other forms of reviewability can also be achieved, for example by executing a binary within a confined environment and analysing its behaviour, as Apple did with PCC [39]. While the highest level of reviewability is ideal, practical constraints may call for a best-effort approach.

2. ***Certifiability:*** It should be possible to generate and publish a digitally signed statement for a reviewed component. Certifiability is crucial for maintaining the integrity and transparency of the review process, potentially enhancing the quality of reviews, aiding in compliance with legal and regulatory requirements, providing evidence in disputes, etc. Forms of certifiability

vary, such as providing a signed statement for reviewed source code or, in the case of a frequently fine-tuned machine learning model, certifying the architecture instead of the weights.

3. ***Attestability:*** While transparency addresses limitations of attestation in providing security assurance, attestability itself serves as a prerequisite for transparency. This paradoxical need arises because without attestability, users cannot confirm whether a specific instance corresponds to a reviewed version of a component, and certifiers cannot be unequivocally held accountable for their assessments (§4.1.1). Therefore, transparency cannot enhance a component's security assurance in an externally verifiable way without attestability.

We call the sensitive components that meet all three criteria as *transparency-enhanced* sensitive components.

Hardware transparency. A notable category of sensitive components that may not fit the *transparency-enhanced* description is hardware. While hardware designs can be open source, like OpenTitan [135], reviewed, and even formally verified [117], there appears no scalable method for verifying that any piece of hardware matches a particular design, with the same technical guarantees as software.

Methods like supply chain auditing and monitoring can provide some level of insight into hardware integrity. Apple PCC counters targeted hardware attacks by using high-resolution imaging in the manufacturing chain, and auditing hardware at the data centres under third-party oversight [74]. Another approach can be inspecting random hardware samples to verify that they correctly correspond to the design. However, these methods fall short of the certainty provided by reproducible software builds. As a result, *reviewability* and *certifiability* of hardware components present significant challenges. The same applies to *attestability*, as current attestation protocols focus on software elements like firmware and drivers, offering little or no assurance about hardware.

These fundamentally restrict a user's ability to assess whether the hardware can support Confidential Computing; let alone its correctness. Given these challenges, this paper focuses on the software components of Confidential Computing systems. If these challenges are resolved, our transparency framework can be used for

hardware as well. For now, the security properties gained by our transparency framework (§4.2) rest on the correctness of the hardware.

Non-sensitive components. Transparency for *non-sensitive components*, which do not threaten the confidentiality or integrity of user data if compromised, requires establishing a well-defined trust boundary. This trust boundary acts as a sensitive component, protecting integrity and confidentiality of data. Transparency can then be used to externally validate that this trust boundary prevents the non-sensitive components from compromising confidentiality and integrity of data. Assurance in this context can be achieved through methods like hardware isolation or memory encryption. If the trust boundary is transparency-enhanced, non-sensitive components can be excluded from the transparency discussions. Otherwise, non-sensitive components should be treated as sensitive and included in the transparency efforts.

4.1.3 Agents of transparency

A Confidential Computing system achieves transparency when users can review its sensitive components directly or delegate this responsibility to trusted parties. These reviewers must remain accountable to users by providing verifiable evaluations and justifying their decisions. Accountability begins with establishing a verifiable link between a review and its reviewer. We implement this link through digitally signed review certificates, which uniquely identify both the reviewed component and the reviewer. Therefore, we call the reviewers *certifiers*, who act as agents of transparency by providing traceable evaluations.

This work only focuses on establishing the link between reviews and reviewers, with broader mechanisms for detecting and enforcing accountability left for future exploration. While institutional reviewers benefit from legal frameworks and governance structures, achieving accountability for individual reviewers requires structured systems, including verifiable digital identities, reputation mechanisms, and transparent review histories.

Traits: Each certifier has traits relating to their *methodology* and *motivation*.

In terms of *methodology*, certifiers can be either reporting or alerting. A certifier's methodology influences the certificate content.

- **Reporting certifiers** may uncover both the strengths and shortcomings of the component under review. The code owner may provide a signed public review plan to guide certifiers, similar to Protection Profiles in Common Criteria [28]. They issue certificates, that are similar to Security Assessment Reports (SAR), including review scope and findings. Format of these certificates can be standardised by the code owner or an independent entity. Users may choose their own trusted reviewers, similar to the open-source Rust code review system `cargo-vet` [158], or form a web of trust, as in `cargo-crev` [173], another Rust review system. Also similar to `cargo-vet` [158], users can define policies to ensure that their trusted reviewers assess the code according to specific criteria.

- **Alerting certifiers** focus on finding bugs and vulnerabilities. They issue certificates detailing the discovered issues, including information such as the vulnerability type, root cause, impact description, and public references—similar to CVE records [58]. These certificates are useful for urging the code owner to fix the issues and holding them accountable if they fail to do so. After a fix alerting certifiers must assess and issue a follow-up certificate that includes their opinion on the solution. The number and content of alerting certificates, certifier credibility, and code owner's response serve as trust signals.

In terms of *motivation*, certifiers can either be independent, or they may be affiliated to the code owner.

- **Independent certifiers** have no vested interests in the code owner's outcomes and are not responsible to them. They may have a greater commitment to ensuring the component's quality and security, but can also be driven by self-interest: independent researchers seeking publication, open-source contributors who wants to gain experience, individuals affiliated with competing organisations who aim to highlight weaknesses in the code owner's product, or bug bounty hunters motivated by financial rewards.

- **Affiliated certifiers** have formal or informal associations with the code owner through contractual agreements, employment, partnerships, financial

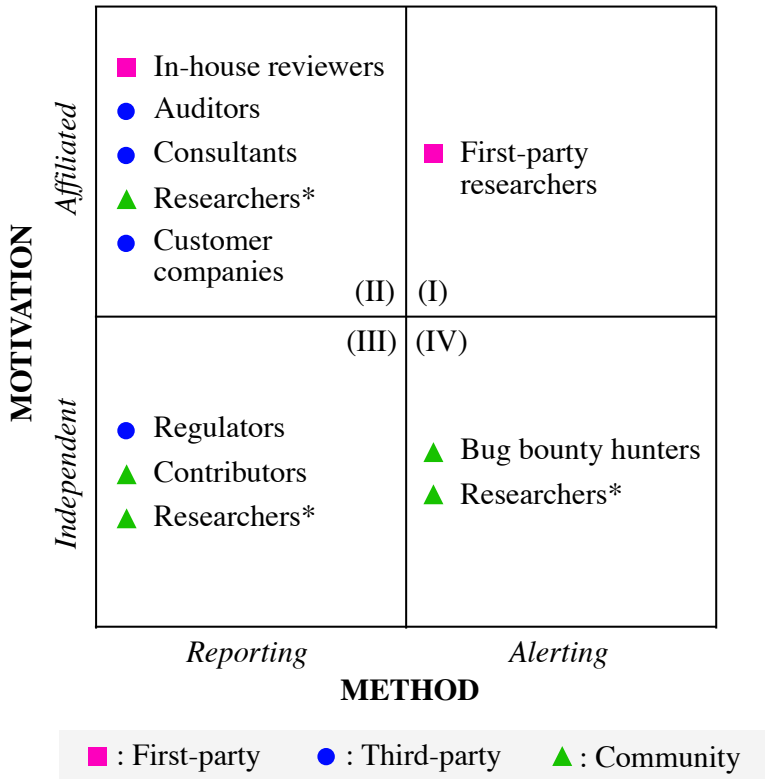


Figure 4.1: Graph showing all certifier trait combinations, with each quadrant including examples from the three reviewer categories where applicable. An asterisk (*) indicates that researchers may have varying motivations and methods, such as (II) those funded by code owners, (III) volunteer researchers in community efforts, and (IV) those focused on finding vulnerabilities and publishing papers.

ties, etc. While affiliation introduces potential impact on their assessments, they are incentivised to maintain high standards due to accountability for their certificates and the risk to their reputation. A structured reputation system tracking review quality, consistency, and peer feedback can further encourage thorough and truthful assessments.

Categories: We categorise certifiers into three groups based on their access methods to the reviewed components: *first-party*, *third-party*, and *community certifiers*. As Figure 4.1 shows, the categories of reviewers are orthogonal to their traits.

- **First-party certifiers** are internal to the *code owner*, the entity responsible

for producing the reviewed component. They are typically involved in the component's development or oversight and have the authority to address issues directly. All first-party certifiers are affiliated, and most likely reporting in methodology. However, there are also examples of alerting first-party certifiers, including first-party security research teams like Google Project Zero [84], Microsoft Security Response Center (MSRC) [148], and IBM X-Force [102], which identify and disclose vulnerabilities in their own closed-source [5, 6] or open-source [4, 7] software.

- **Third-party certifiers** are granted exclusive access to the components for review. The ideal third-party certifiers are independent experts, including regulators, and to some degree, customer companies. For instance, a cloud platform purchasing hardware may review the firmware and drivers, or a company outsourcing software development may review the components to ensure quality. Customer companies lack full independence due to business dynamics, such as financial ties or concerns about the performance impact on purchased components. Third-party certifiers may also be affiliated, such as auditors or consultants paid by the code owner. Third-party certifiers might be obligated to maintain confidentiality of the reviewed components.

- **Community certifiers** access components through open sourcing and include individuals or groups with relevant expertise, such as academics, independent researchers, bug bounty hunters, and external adopters. Notable existing community certification programmes include Rust `cargo-crev` [173] and `cargo-vet` [158]. Professional research teams such as Google Project Zero [84], MSRC [148], and IBM X-Force [102] also act as alerting community certifiers when reviewing code produced by other code owners [150, 151, 88, 89].

4.1.4 Beneficiaries of transparency

We identify four types of users who may benefit from transparency:

- **End users** have their data processed within a Confidential Computing system, whether on a remote resource, such as a cloud infrastructure, or their own

devices. These users' primary concern revolves around ensuring the privacy and integrity of their data.

- **Application developers** execute code within Confidential Computing systems, where their priority may be protecting the confidentiality of their code, their customers' data, or both. This category of users includes the application developers using cloud-based services such as Google Cloud Confidential Computing [50], Microsoft Azure Confidential Computing [147], and AWS Nitro [15]. It also includes users who leverage hardware of end-devices to execute their code, be it an operating system, a hypervisor, or other lower-level components.

- **Application providers** execute third-party code, i.e. code they have not authored, within Confidential Computing systems. Their primary concern is ensuring the security and trustworthiness of the code they run.

- **Platform providers** include cloud providers, software-as-a-service providers, CPU manufacturers as platform providers for executing code in enclaves, Android with Private Compute Core [139] or Android Virtualisation Framework (AVF) [23], and other entities offering infrastructure or services for Confidential Computing. Their main objective is to establish and maintain a trustworthy Confidential Computing system for end users and application developers, ensuring the integrity and security of the systems and services they provide.

4.2 The Confidential Computing Transparency framework

Following the reasoning and discussions we provide in §4.1, we define Confidential Computing Transparency as follows.

Definition 4.2.1 (Transparency) *Transparency in Confidential Computing is the practice of allowing certifiers (as categorised in §4.1.3) to examine the security*

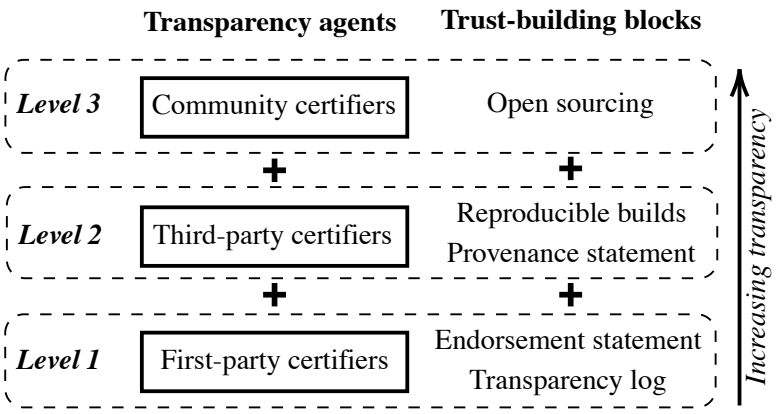


Figure 4.2: A depiction of the three transparency levels in our framework, showing the agents and trust-building blocks needed to achieve them. Each level builds on the previous one by adding more trust-building elements.

properties of the sensitive, reviewable, certifiable, and attestable (as defined in §4.1.2) components of a Confidential Computing system. These certifiers should be able to certify their comprehensive assessments in a manner that is publicly verifiable.

In this section, we present a framework to answer the question of *how* this transparency can be achieved in Confidential Computing. Our framework acknowledges the diverse transparency needs, limitations, and risk profiles of various system components. For instance, some transparency-enhanced sensitive components might not allow open sourcing or third-party certification due to IP restrictions or proprietary information. This complexity is more pronounced in real-world systems involving multiple stakeholders and components from different vendors. For example, even a single TEE can include microcontroller units from different vendors, each with dedicated firmware and drivers.

To address these complex requirements and challenges, our framework adopts a layered approach, defining three transparency levels, each incorporating a certifier category described in §4.1.3 and a set of trust-building blocks. As shown in Figure 4.2, each level builds on the previous one by adding more trust-building blocks and integrating an additional certifier category. Based on needs and limitations, system designers can apply different transparency levels to different components, though

we strongly advise aiming for the highest transparency level achievable.

4.2.1 Level 1 (L1)

Requirements. The foundational level of transparency is achieved through engaging *first-party certifiers* as accountable reviewers. This level relies on the following trust-building blocks to ensure transparency:

Endorsement statement is a digitally signed statement by affiliated and reporting first-party certifiers, including a unique identifier of a binary (e.g. hash), as defined and used by Google's open-source project Oak [85]. It can also include the time of issuance and the validity period. By issuing an endorsement statement for a specific binary, the certifier confirms that the binary was generated from specific source code and endorses its use in production for the validity period. After the validity period, the certifier may issue a new statement or passively revoke the existing one by taking no action (see §4.2.4).

Verifiable transparency log is a publicly available append-only ledger maintained on a trusted infrastructure by an independent party, separate from the code owner or the certifiers [127, 144, 49, 98]. Transparency logs do not inherently guarantee truthful and accurate operations, so additional measures are required to ensure their integrity and consistency, as discussed in §4.3.1.

Process. At this transparency level, the review and certification must happen prior to the binary's release. There is one exception to this rule: if first-party researchers identify a vulnerability in the binary after its release, they may issue alerting certificates. The first transparency level is achieved through the following steps.

1. First-party certifiers review the source code associated with the sensitive components, and approve it for release.

2. The code owner builds a binary from the source code. To ensure the integrity of the build process, it is essential that both the code owner and the first-party certifiers have established trust in the build toolchain and the infrastructure

used. This trust is ideally built through direct review of the tools and processes. If direct review is not feasible, it may be reasonable to implicitly trust open-source tools, or tools widely used across the company and maintained by a dedicated team. The code owner can then release the binary, e.g. by distributing it to cloud servers, end devices, or publishing it to an app store.

3. First-party certifiers generate an endorsement statement that they sign with their private signing keys (separately or using a multisignature scheme [111]). They publish this endorsement statement on a transparency log.

4. If first-party researchers discover a vulnerability after the binary's release, they generate a signed alerting certificate as described in §4.1.3, and publish it on the transparency log.

5. After receiving a binary, the user must verify that the transparency log contains a valid endorsement statement. This involves: (I) confirming the endorsement's binary hash[1] and validity period, and (II) verifying the endorsement's signature(s) using the certifiers' public keys, unless relying on third-party monitors (§4.3.1). In Binary Transparency, this verification is typically done by an *auditor*, usually represented as client-side software [13]. Inclusion proofs can be used for efficiently verifying that an endorsement statement is added to the transparency log (§4.3.2). The user (or auditor) must also periodically check the log for any alerting certificates, which may prompt them to stop using the binary or take relevant measures.

Discussion. At this transparency level (L1), beneficiaries cannot directly see the attested components, but generating and publishing endorsement statements holds the code owner accountable for their released binaries. This ensures the code owner cannot disown a particular release. Furthermore, this transparency level ensures that any insider or outsider attacks on the released binary do not go unnoticed, as new transparency log entries or their absence are detectable. This achieves

[1] If the binary is running on a server where the user is the client (as opposed to the user's end-device) the user must obtain the attestation evidence of the binary to retrieve the correct hash value.

'Signature Transparency', a concept implemented by Sigstore's Rekor [184] and Sigsum [174].

L1 also guards against covert targeted attacks, where an attacker serves a specific user a targeted malicious binary by allowing users to verify they are served the same binary as everyone else. However, this system is not immune to more overt, coordinated attacks. Even if the transparency log operations are monitored for correctness as described in §4.3.1, an adversary can manipulate the timing of population updates. For instance, the adversary might delay visibility of transparency log updates for certain IPs or intentionally cause out-of-band checks for update availability to fail for specific IPs. This way, the adversary can potentially create split-view release streams with 'good' and 'bad' binary versions. Even so, this transparency level can alert vigilant observers to anomalous activities like unusual release patterns or duplicate version numbers, serving as a deterrent and early warning mechanism against targeted attacks.

As all first-party certifiers are affiliated (see §4.1.3), reviewers in this transparency level are limited to the first and second quadrants of the certifier diagram (Figure 4.1). In other words, there is no variety of motivation among certifiers at this transparency level.

4.2.2 Level 2 (L2)

Requirements. The second transparency level is achieved by introducing *third-party certifiers* as accountable certifiers. First-party certifiers also participate at this level, performing reviews and issuing certificates similarly to how they operate in L1. This ensures that code owners remain accountable for the binaries they release. In addition to the trust-building blocks used in the first transparency level, the second level introduces the following ones.

Reproducible builds refers to a collection of tools and techniques that are used to ensure that every build for the same source code consistently generates the same bit-exact binary output [52].

Provenance statement includes all the necessary configuration details for building the source code into a specific binary, such as what toolchain, commands

and flags to use [85]. It is provided by the code owner to certifiers alongside the component under review.

Process. The build, release, and first-party review process of the binary happens as described in L1 (§4.2.1). To attain L2 transparency, the following additional steps must be followed. At this transparency level, the review and certification can happen before or after the binary's release.

1. Third-party certifiers review the source code of the sensitive components. The source code must be reproducibly buildable, enabling each certifier to build it and generate the same binary as other reviewers (see §4.3.3 for the alternative). By comparing the hash of this binary with the signed certificates in the transparency log, certifiers can ensure they have reviewed the same source code as others.

2. Each certifier uses the provenance statement to learn the build configurations for the component and independently compiles the source code into a binary on trusted infrastructure. Certifiers must establish trust in the build toolchain instructed by the provenance statement, achievable by applying transparency principles to the toolchain itself (§2.5.4).

3. Each third-party certifier generates a signed certificate (either reporting or alerting as described in §4.1.3) for the self-built binary, and publishes it on a transparency log.

4. Once the binary is received on an end-device (or the attestation evidence if the binary runs on a server), the user should check if a valid endorsement statement for it exists on a transparency log, as in L1 Step 5. Unlike endorsement statements, the existence and validity of certificates from third-party certifiers is not enough to provide trust to the beneficiaries; their contents matter (§4.1.3).

Discussion. The second transparency level (L2) significantly improves trust by incorporating both affiliated and independent third-party reviewers, unlocking the third quadrant and expanding the diversity of reviewer motivations (Figure 4.1).

From a practical standpoint, incorporating third-party certifiers may introduce potential delays in the review process, particularly in dynamic production environments where code updates are frequent. This creates a trust dilemma, especially in scenarios where a vulnerability is detected. The code owner may need to decide whether to quickly implement a patch without waiting for certification, or follow the full review process and delay until certification is received.

To mitigate these time-related challenges, one potential solution might be implementing a transparent source control mechanism to allow certifiers to review only the code changes. An alternative approach is issuing certificates after the binary is released. Although this might mean users are not immediately assured that third-party certifiers have vetted the binary processing their data, this mechanism disincentivises the code owner from releasing malicious or subpar code. To implement this approach, the code owner should include an explicit promise in the endorsement statement at Step 3 that there will be a third-party audit certificate to follow. This promise may also specify a timeline for certification, e.g. `to_be_certified_by: <date>`. Having a signed explicit promise like this eliminates ambiguity and allows for more automated verification of the promise at Step 4.

Making the source code reproducibly buildable can be challenging, or the reviewers might find it impractical to build the binary themselves. An alternative approach is using a trusted builder (see §4.3.3). However, this option comes with certain caveats, including the need for the certifiers and beneficiaries of transparency to trust the builder.

4.2.3 Level 3 (L3)

Requirements. The third and highest level of transparency employs *community certifiers* as accountable reviewers. As in L2, community certifiers do not replace the first-party and third-party certifiers from the previous levels. L3 must include first-party certifiers, and ideally, it should also incorporate third-party certifiers. In addition to the trust-building blocks used by the other levels, this transparency level introduces a new one:

Open sourcing in this context refers to making the source code of a component

publicly accessible on a platform, which the user does not need to trust.

Process. In this level, each community certifier follows the same process described in L2 (§4.2.2).

Discussion. This level of transparency significantly improves openness and accountability, driven by principles of public accessibility and community involvement. As we discuss in §4.1.3, signed certificates provide the foundation for accountability by creating verifiable links between reviews and their reviewers. While certificates establish identity, complete accountability for community certifiers requires broader mechanisms like reputation systems, peer audits, and enforcement protocols.

Similar to L2 (§4.2.2), this transparency level may not always guarantee real-time verification. Open sourcing, while a powerful tool for transparency, does not guarantee that all security issues will be detected immediately, as evidenced by past vulnerabilities in large-scale open-source projects with extensive communities including Heartbleed [188], POODLE [157], Log4Shell [25], and Shellshock [63]. Nonetheless, open sourcing remains a crucial step towards minimising the need for user trust. Moreover, unlike traditional open sourcing, this transparency level requires reviewers to certify their assessments, serving as an additional trust signal (§4.1.3).

Availability of source code substantially improves the ability of independent alerting certifiers to analyse code for defects (§4.1.3), fully unlocking the last and fourth quadrant of the certifier diagram (Figure 4.1). To incentivise such certifiers, the code owner may set up bug bounty programmes. The code owner may also set up coordinated vulnerability disclosure mechanisms to allow the alerting reviewers to report issues to the code owner and giving it some time before issuing a public certificate. In this case, if the issue is fixed before it is disclosed to the public, the certifiers may still issue their certificate after a patch has been released, similar to how first-party security research teams file CVE records after patch releases.

4.2.4 Revocation

Revocation is an essential part of the transparency framework, as the majority of software is ultimately revoked. Two main reasons to revoke an endorsement statement or a binary are:

Vulnerability identified: The code owner or a certifier finds a vulnerability in the endorsed binary.

Log inconsistencies: A monitor, witness (§4.3.1), or auditor (§4.2.1) detects anomalies in the transparency log or endorsement statements.

In both of these cases, there are decisions to make about *who* will make the revocation decision and *how* will the revocation be carried out. For vulnerabilities, the code owner can revoke the endorsement statement. One way to achieve this is through passive revocation by issuing short-lived endorsement statements and simply not issuing a new endorsement statement for the affected binary. This notifies users of the issue, allowing them to stop using the binary if feasible or take appropriate mitigation steps within their organisational processes. Alternatively, code owner can actively revoke a statement by publishing its unique identifier on a publicly accessible *certificate revocation list (CRL)* [55, 219].

For both revocation reasons above, users can independently choose to stop using a binary based on information they receive from auditors, monitors, or witnesses. One way to implement this is through a policy in the client-side auditor software [13], similar to the policies in Rust `cargo-vet` [158]. This policy can alert the client about inconsistencies on a transparency log or the existence of alerting certificates about a binary. Alternatively, monitors or witnesses can collectively issue global revocation statements, which they can then publish on CRLs.

4.3 Additional considerations

In this section, we describe optimisations and other modifications that can be applied to our transparency framework to accommodate different needs and constraints.

4.3.1 Monitoring transparency logs

Publicly available transparency logs are not inherently truthful and accurate. Monitors can watch logs for correct behavior, such as ensuring the log is append-only, hashes are valid, and the log does not present split views to different observers [129]. To avoid split views, monitors can share their current and previous views

of tranpsarency logs via *gossiping* [163, 49, 143], gaining a comprehensive understanding of the log's state and prevent propagation of inaccuracies. Approaches like collective signatures from witnesses have been proposed to mitigate potential attacks on gossiping [189].

Even if the transparency log behaves correctly, the mere presence of an endorsement statement in a log does not confirm its validity; it only means that the information is discoverable. Without validating the endorsement statement, the trust relies on the expectation that someone will eventually detect any inaccuracies and raise an alarm. So far, we have described this validity check as a client-side responsibility (see L1 Step 5 and L2/L3 Step 4), which may be impractical due to resource constraints, technical expertise, or the large volume of data in transparency logs. Additionally, clients would need certifier public keys, which can be challenging to manage.

Beneficiaries of transparency can instead delegate this validation responsibility to monitors, who can verify the correctness of statements and certificate signatures within the transparency log [129, 13, 189]. Ideally, the code owner should offer an open-source monitoring mechanism, enabling individuals to scrutinise its source code and deploy them on their trusted machines. The trustworthiness of the monitor code itself can be increased by using our transparency framework.

4.3.2 Inclusion proofs

When a user receives a binary or attestation evidence for a binary, they should verify the presence of a corresponding valid endorsement statement in a transparency log (see L1 Step 5 and L2/L3 Step 4). This can be done efficiently with *inclusion proofs*, similar to how they are used in Certificate Transparency to verify the inclusion of a certificate on a transparency log [86, 186, 13].

To be able to have inclusion proofs, the transparency log is constructed as a Merkle tree [146], where the tree head is signed by the log operator [129]. An inclusion proof contains the shortest list of hashes needed to compute the tree head given a leaf node. With an inclusion proof and the hash of an endorsement statement (as the leaf node), if the client can compute the tree head hash and also verify the signature of the tree head, this confirms that the endorsement statement

has indeed been included in the transparency log (conditional upon the client having the correct public key to verify the signature). To keep inclusion proofs scalable, the log operator periodically issues a signed commitment to the state of the log, sometimes called a *checkpoint*, which can be used instead of the original tree head [14]. This approach reduces the verification complexity from linear to logarithmic in the number of certificates.

Certificate Transparency uses inclusion promises, signed commitments from log maintainers guaranteeing future entry addition, to address merge delays. Unlike inclusion proofs, these promises require full trust in the log and additional monitoring [14]. In Confidential Computing Transparency, where some latency at publish time is likely tolerable, we do not recommend inclusion promises to be used. Alternative transparency log designs like Sunlight [2] can help reduce merge delays.

4.3.3 Trusted builders

In L2 and L3 (§4.2.2 and §4.2.3), the certifiers are responsible for building the binary to ensure the binary they issue a certificate for is generated from the source code they have reviewed. Alternatively, a *trusted builder*, a dedicated tool for building binaries, can take on this task. Using a trusted builder presents an important tradeoff: the certifiers are relieved from the responsibility of building the binary themselves, but they need to trust the builder. Despite this tradeoff, a trusted builder can be helpful especially in L3, where community reviewers may lack resources. Moreover, using a trusted builder removes the requirement for the source code to be reproducibly buildable, which can be a difficult task for the code owner to achieve and maintain. However, ideally, the source code should still be reproducibly buildable to allow independent verification.

A key difference with using a builder is that for each build, the trusted builder should generate a *signed* provenance statement and post it on a transparency log. This statement is a verifiable and attributable claim that the binary was built honestly on a trusted, uncompromised platform, allowing the certifiers to gain insight into the build environment. This is especially important since the code does not have to be reproducibly buildable, and the certifiers might not be able to build the binary themselves to compare it with the binary that the trusted builder

produced.

To establish trust in a builder, users must first trust the build toolchain. This can be achieved by using an open-source build toolchain and applying the transparency principles to it (also see §2.5.4). Second, the toolchain must also operate on trusted infrastructure, ideally controlled by an actor unlikely to collude with the code owner. For instance, Google's open-source Confidential Computing project Oak [85] has a trusted builder that uses the open-source SLSA build stack [192] and runs on GitHub, which is currently owned by Microsoft. The level of assurance can be further increased by involving multiple independent builders. For example, Oak runs a trusted builder instance on Google Cloud in addition to GitHub. Another way of increasing the trustworthiness of a builder is to run the build toolchain's TCB inside a TEE that is different from the one under scrutiny. Using a hardened build platform with strong isolation and protection for the provenance signing key can also increase trust on the builder. This idea is used in SLSA V1.0 as well [193].

If a trusted builder is used in L2 or L3, the release process for the binary is revised as follows:

1. The code owner initiates a build of the source code using a trusted builder.

2. The trusted builder generates a signed provenance statement about the binary and the build process, publishes it on a transparency log, builds the code into a binary and returns it to the code owner. If there are multiple trusted builders, then all builders do the same.

3. The code owner verifies the signature of the provenance statement, and confirms that the information it contains about the build process is correct. If there are multiple trusted builders, the code owner does these checks for each build. If the binary is reproducibly buildable, the code owner also checks that all binaries are identical.

4. If the checks about the build are successful, the code owner generates an endorsement statement about the binary. If the code is not reproducibly buildable and there are multiple trusted builders, the code owner generates an endorsement statement for each unique binary. These endorsement statements

include a unique pointer to the provenance statement generated and published by the trusted builder in Step 2.

Following this build and release process, the third-party or community certifier in L2 or L3 review the source code. If the source code is not reproducibly buildable, Step 2 of L2 and L3 is replaced by the certifier verifying that a certain binary correctly corresponds to the reviewed source code. To do so, the certifier gets the endorsement statement for that particular binary version from the transparency log, and checks that the signatures on both the endorsement and provenance statements are valid. The certifier also checks if endorsement statement includes the correct binary hash. If the source code is reproducibly buildable, the certifier can additionally build it on its own trusted infrastructure, as in Step 2 of L2 and L3.

4.3.4 Automated certifiers

A certifier can be replaced by automated processes, introducing a fourth certifier category, which we call *automated certifiers*. An automated certifier can be constructed by building a transparent mechanism that generates formal proofs of security properties of the component. The transparency of the generation mechanism gives assurance to the beneficiaries (§4.1.2) that the proofs are generated correctly. This alternative certifier type can be used for achieving transparency in a similar way in L2 and L3, also using a trusted builder as we describe in §4.3.3. The level of transparency provided depends on the assurance of the generated proof. This alternative certifier type can both be seen as an optimisation, since automated proofs may work more efficiently than human reviewers, and as a privacy enhancement, as the code owner can potentially generate proofs that do not reveal proprietary information, e.g. by using zero-knowledge proofs [137].

4.4 Related work

The Confidential Computing Transparency framework shares certain similarities with supply chain security, especially to SLSA (Supply-chain Levels for Software Artefacts) [192]. SLSA is a framework incorporating Binary Transparency concepts

to address software supply chain threats. Like our transparency framework, SLSA uses a levelled structure and can complement our approach in enhancing Confidential Computing systems' resilience against supply chain attacks.

SLSA v0.1 offers general supply-chain security guidelines, while v1.0 focuses on the Build track (other planned tracks being Source and Dependencies). Our framework's L2 and L3 align with SLSA v0.1 L4, emphasising reproducible builds and meeting the two-person review requirement through affiliated reporting first-party certifiers (Figure 4.1, quadrant II). Implementing our framework's L2 and L3 with a trusted builder (§4.3) achieves SLSA Build L3, (keeping in mind that L3 requires a hardened builder with strong isolation and provenance signing key protection). Therefore, with specific implementation choices, our transparency framework can provide high levels of supply-chain security.

4.5 Summary

In this chapter, I introduced the Confidential Computing Transparency framework designed to systematise transparency required by Trusted Execution Environments (TEEs) to reduce reliance on user trust (§4.2). The framework defines three progressive transparency levels, engaging first-party, third-party, and open-source reviewers. It extends existing industry practices by adding reviewer accountability and a robust trust chain with verifiable transparency logs, signed statements, and reproducible builds. The tiered structure provides organisations with a practical pathway for incrementally enhancing the transparency of complex Confidential Computing systems in real-world deployments. We also address key questions about transparency in Confidential Computing, including its definition, importance, scope, beneficiaries, and facilitators (§4.1).

Although the framework we proposed in this chapter focuses on Confidential Computing due to the unique *attestability* of TEEs (§4.1.2), our transparency framework can also be applied to various Binary Transparency scenarios (§2.5.4). For binaries in remote TEEs, attestation ensures integrity, while in controlled environments users can directly validate binaries via signature checks and transparency log inclusion proofs. While we used software as our motivating example throughout, our approach is applicable to computer hardware too as long as it is within scope

(§4.1.2).

In the next chapter, I present our user study, which evaluates the impact of the proposed transparency framework on end-user trust. The study examines how different transparency levels influence users' perceptions of Confidential Computing and their willingness to share sensitive data. These findings provide empirical insights into the effectiveness of transparency measures and inform the broader discussion on designing more trustworthy Confidential Computing systems.

Chapter 5

Evaluating user perceptions of Confidential Computing Transparency

The Confidential Computing Transparency framework, introduced in Chapter 4, is designed to build user trust in the reliable and secure functioning of Confidential Computing systems' underlying hardware and software components. To evaluate the effectiveness of this framework, we conducted a large-scale user study examining how transparency influences trust and data-sharing behaviour in Confidential Computing systems. This chapter presents the study, which focuses on two key aspects: (1) how different levels of transparency affect end-user trust and (2) how transparency influences the types of data users are willing to share. Our results show that higher transparency levels correlate with increased user trust, though the type of data being shared also plays a role.

To further explore these effects, we conducted two study variants. The first, a *low-detail* version, provided a high-level explanation of Confidential Computing, transparency, and the proposed framework to the study participant, playing the role of an end-user of a Confidential Computing system. The second, a *high-detail* version, offered a more in-depth explanation and directly addressed common misconceptions observed in the low-detail variant. We found that clearer explanations and targeted corrections of misconceptions led to greater comfort with

increasing transparency levels and reduced misunderstandings.

This chapter is based on the evaluation sections of our paper "A Confidential Computing Transparency Framework for a Comprehensive Trust Chain" [119]. Tina and I collaborated equally in designing the user study. Tina led the data analysis presented in the paper, while I conducted additional analysis for this chapter. Ivan, Alice, and Alastair contributed to both refining the study design and shaping the presentation of findings. This user study was supported by an unrestricted gift from Google.

5.1 Methodology

This study was designed to answer the research questions outlined in §5.1.1. To explore these questions, we developed two study variants, which are described in §5.1.2. The study was structured into three main components, which I elaborate in this section. The first is background section (§5.1.3), where participants received information about Confidential Computing and transparency, and answered comprehension questions. The second is the core questionnaire, which explored their trust perceptions and preferences (§5.1.4). Finally, the study included a set of additional questions for further insights (§5.1.5). The full user study script can be found in Appendix B. We conducted the user study as an online survey. §5.1.6 explains how we recruited participants and collected the data.

5.1.1 Research questions

We aimed to answer two main research questions with this user study:

RQ1. How do different transparency levels of the Confidential Computing Transparency framework shape the end-user sense of trust?

RQ2. What types of personal data are end users comfortable sharing at different transparency levels?

Our first research question, **RQ1**, is directly tied to the Confidential Computing Transparency framework's levels (§4.2). It tests the hypothesis that people are more comfortable sharing their data with apps that implement stronger transparency measures.

Our second research question, **RQ2**, is based on prior studies indicating that users perceive different types of data as having varying levels of sensitivity [48, 152, 177, 169, 185]. Given that transparency is meant to enhance trust in data handling, we wanted to examine whether users demand higher transparency for data they personally categorise as more sensitive. This research question tests the hypothesis that the level of transparency users expect depends not only on the system processing the data but also on the type of data being processed. Understanding this relationship provides insight into whether transparency preferences are uniform or context-dependent.

5.1.2 Detail variants

We conducted two versions of the study. Initially, we provided the users with a concise explanation about Confidential Computing and transparency (*low-detail*). This script presented information about Confidential Computing and transparency in a relatively abstract manner. We took this approach as we wanted to mimic the real-life scenario where non-experts are exposed to technical concepts through high-level summaries, for example in an advertisement.

The results of this first study indicated some common misconceptions among the participants, which we discuss in §5.3.2. Motivated by this, we conducted a second study (*high-detail*) where we increased the level of detail in our explanation, as well as the thoroughness of the comprehension questions.

The only difference between the two study variants is the instructional scripts provided in the background section of the study, and the comprehension questions following them. The differences between the two variants are detailed in the next section (§5.1.3).

5.1.3 Background information

At the beginning of the survey, participants were first introduced to the core concepts of Confidential Computing through an informational script, available both as a video[1] and as written text (Appendix B.3). The two instructional scripts,

[1]Link to videos used in the *low-detail* and *high-detail* studies `https://www.cl.cam.ac.uk/research/security/datasets/cct-user-study/`.

Aspect	*Low-detail* script	*High-detail* script
Conceptual Approach	Uses a vault and keypad analogy to explain Confidential Computing in a relatable way	Provides a technical description of how Confidential Computing protects data in cloud environments
Data Protection Explanation	Focuses on the general idea that user data should remain private from app developers	Explains how data is processed securely within a protected computing area and what could go wrong
Transparency & Trust	Mentions that systems can be reviewed but does not explain the importance of transparency	Defines transparency as an alternative to blind trust, explaining how public reviews create accountability
Types of Reviewers	Introduces three categories of experts but does not specify their access levels	States that all reviewers receive the same level of access to system components but cannot access user data
Security Risks & Limitations	Briefly acknowledges that secure design is important but does not elaborate	Explains that any flaws in system components could compromise privacy, leading to data exposure
Comprehension Questions	Three true/false questions, testing basic understanding of Confidential Computing, reviewer accountability, and data access	Ten multiple-choice questions, testing deeper understanding of data security risks, transparency mechanisms, and reviewer access levels, including a GIF-based matching task

Table 5.1: Comparison of the *low-detail* and *high-detail* background sections.

low-detail and *high-detail* , were designed to introduce participants to the concept of Confidential Computing and the role of transparency in ensuring its trustworthiness. Both scripts conveyed the same core message but differed in the level of detail and complexity of explanations. Both scripts were written with guidance from expert Confidential Computing researchers in the industry and academia, including the co-authors of the paper [119]. Table 5.1 summarises the key differences between the two scripts.

The *low-detail* script introduced Confidential Computing using a simplified analogy—a vault with a keypad—to explain how the system protects user data. In contrast, the *high-detail* script provided a more precise explanation, describing a cloud computing scenario and explaining why data can become exposed during processing. The *high-detail* version explicitly outlined how Confidential Computing isolates data within a protected area of a computer.

When discussing transparency, the *low-detail* script kept the explanation relatively high-level. Building on the vault analogy, this script highlighted that one can only trust a vault if they trust that it was designed and built securely. The *high-detail* script, however, explained why transparency matters more explicitly, highlighting the risks and limitations of Confidential Computing. Specifically, it stated that flaws in any part of the system could compromise privacy, allowing unauthorised access to user data. It framed transparency as an alternative to blind trust and emphasised that making reviews publicly available can incentivise accurate and honest assessments.

Finally, both scripts presented a list of potential reviewers, but the *high-detail* version provided additional clarification: all reviewers receive the same level of access to the system components, but cannot see, access, or modify user data. The *high-detail* version also made a few subtle changes in the phrasing of the certifiers (§4.1.3), which are discussed in detail in §5.3.2.

The two instructional scripts were followed by distinct sets of comprehension questions, matching the level of detail the instructional scripts. In the *low-detail* variant, participants answered three true/false questions, which tested basic understanding of Confidential Computing, reviewer accountability, and data access. These questions were designed to be straightforward, requiring minimal cognitive effort to verify fundamental concepts. In contrast, the *high-detail* variant included ten multiple-choice questions, assessing a deeper understanding of data security risks, transparency mechanisms, and reviewer access levels. This included a matching task, pairing illustrative GIFs with the corresponding transparency levels.

(Six versions of the following screen were presented to the participants, varying the data types and transparency level descriptions as listed after the questions.)

Virtual assistant app `[LETTER IN THE RANGE A-F]` requires permission from you to access your `[DATA TYPE, TABLE 5.2]`. This includes `[EXAMPLES OF DATA TYPE, TABLE 5.2]`. In order to assess if the Confidential Computing system is designed and built securely, the system was `[TRANSPARENCY LEVEL, TABLE 5.3]`.

(Participants rated their comfort on a 101-point Likert-type scale (0 = "Not at all comfortable/confident," 50 = "Neutral", 100 = "Extremely comfortable/confident").)

Q1 How comfortable would you feel using this app?

Q2 How confident would you feel about this app's ability to keep your `[DATA TYPE]` data safe?

Figure 5.1: Structure of the core questionnaire.

5.1.4 Core questionnaire

We designed the core part of the study around a multi-purpose virtual assistant app, a familiar software category that can justify access to a wide range of personal data. The core questionnaire was formulated as described in Figure 5.1.

Following the background section, participants were presented with six different versions of the virtual assistant app. Each app required access to a different type of personal data, which we categorised into: (i) social-economic, (ii) lifestyle-behaviour, (iii) tracking, (iv) financial, (v) authenticating, and (vi) medical-health data, drawing from the work of Chua et al. [48]. This paper suggested many examples for each data category. To keep the study concise, we selected a representative subset of examples for each category, as listed in Table 5.2. Each app was randomly assigned a specific transparency level L1, L2, or L3 from our framework (§4.2), or no transparency at all. We described each transparency level using neutral language and labelled all certifier types as *experts*.[2] Table 5.3 shows the exact wordings we used to describe the transparency for both *low-detail* and *high-detail* variants. For each app, we also included an animated GIF illustrating the corresponding transparency level, taken from the informative video. Participants sequentially saw the apps and rated their comfort levels using the app, and their confidence in the ability of the app to protect their data.

[2]While our framework incorporates first-party certifiers across all transparency levels, we omitted this detail from the study to maintain clear distinctions between levels, avoiding potential confusion and to not imply any hierarchy.

Data type	Selected examples
Lifestyle-behavior	Religious and political beliefs, interests and preferences, browsing habits, family and relationships.
Social-economic	Ethnicity, physical characteristics (from pictures and videos), age and gender, professional career.
Tracking	Email address, physical address, phone number, phone recordings and text messages, geolocation, IP address.
Financial	Credit history, financial assets, bank information and credit card numbers, purchases, transactions, taxes.
Authenticating	Passwords, passport data, government ID data, usernames.
Medical-health	Personal health history and diagnoses, prescriptions, mental health records, disabilities, genetic data.

Table 5.2: Six data type categories and selected examples from Chua et al. [48]

Transp. Level	Low-Detail Variant	High-Detail Variant
In order to assess if the Confidential Computing system is designed and built securely, the system was reviewed by ...		
L1	... experts working for the app developer company.	... experts working for the app developer.
L2	... experts who are granted exclusive access to the code for reviewing it. They may include consultants and auditors hired by the developer company, as well as regulatory authorities.	... experts directly authorised by the app developer, such as hired consultants and auditors, as well as regulatory authorities.
L3	... experts from the broader software engineering community. The system is made publicly available for review by academics, independent researchers, individuals who get rewards for finding bugs, or anyone interested in reviewing the code.	... experts from the broader software engineering community, such as academics, independent researchers, individuals who get rewards for finding bugs, or anyone who is interested in reviewing the code.
No transp.	The Confidential Computing system was not formally reviewed.	The Confidential Computing system was not formally reviewed.

Table 5.3: Transparency levels and their corresponding descriptions in the user study script for both the *low-detail* and *high-detail* variants.

For each of these six virtual assistant app scenarios, participants were asked to evaluate their level of comfort using the app and their confidence in its ability to protect their data. Responses were recorded on a a 101-point unipolar Likert-type scale, where 0 labelled as "Not at all comfortable (or confident)", 50 as "Neutral"

midpoint, and 100 as "Extremely comfortable (or confident)". Notably, the survey software did not display the slider until participants clicked to ensure that responses were not biased by the initial position of the slider.

5.1.5 Additional questions

In the final section of the study, participants answered a set of additional questions designed to explore their attitudes toward technology, privacy concerns, and transparency preferences. The set of questions presented in this section were the same in both *high-detail* and *low-detail* study variants.

In this section, participants rated their approach to adopting new technology, their likelihood of using virtual assistants, and their concern about unauthorised data access. They also assessed their level of concern in the event of a data breach, considering different data types, and indicated their comfort with various review processes for virtual assistants. The participants answered these questions using a 101-point unipolar Likert-type scale, similarly labeled at the 0, 50 and 100 points as in the core questionnaire (§5.1.4). Additionally, participants provided an open-ended explanation of how they assigned scores to the review options, providing insight into their reasoning. Lastly, demographic information was collected, including age, gender, education level, field of study/work, and primary country of residence. The appendices B.6 and B.7 provide more details about this part of the study.

5.1.6 Data collection, participant selection and ethics

We conducted the study as an online survey using Qualtrics and recruited participants via the Prolific academic platform [165], as recommended by relevant research [190]. We obtained ethics approval from the University of Cambridge, Department of Computer Science and Technology ethics review board. We recruited 817 participants, split roughly equally between the two variants, in small batches between August and November 2024.[3] The participants were compensated for their time at a rate equivalent to the UK living wage per hour. The median time taken

[3]The sample size was calculated through a power analysis based on pilot data. The batches were spread over time and time zones to minimise the effect of time of the day, and day of the week.

to finish the survey was ∼15 minutes, and the participants were compensated at a fixed rate of £3 or ∼£12/h. The payment was made directly to the participant using the Prolific platform. Participants who did not complete the study or were identified as bots were excluded and automatically replaced by Prolific. Our final analysis after the removals was performed on 758 participants.

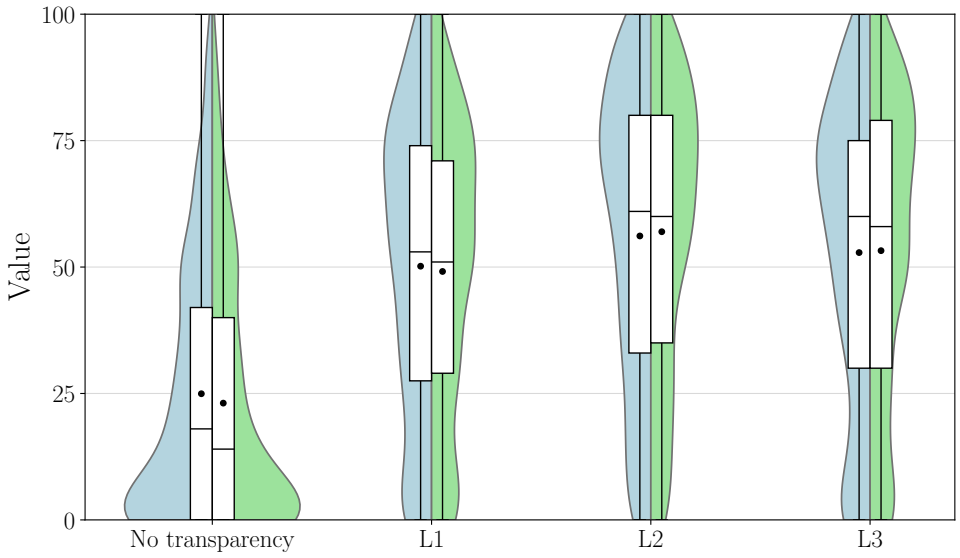
Given that our study involved human participants, there were several important ethical considerations. We obtained informed consent from all users, with the option to withdraw from the study at any point or not answer any or all questions. We also provided direct contact with the researchers for any questions or concerns. The data was anonymised and as such no single individual can be identified. We collected minimal demographics information, with all data analysed on an aggregate level. All data was stored in electronic form on encrypted disks or a secure server. Only researchers involved in the project have direct access to raw data. We did not anticipate any risks or harms to participants or researchers involved in the study or to the general public.

5.2 Quantitative results

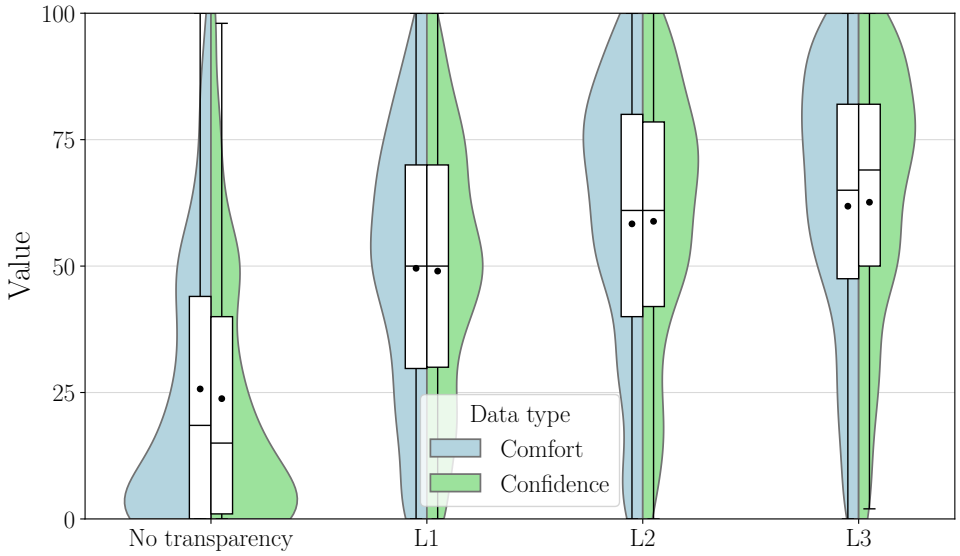
We analysed participant trust using two key metrics. The first one is participant comfort levels in the core phase of the study, aggregated across all data types. The second key metric is comfort with the virtual assistant app under given transparency options, irrespective of data type, which we gather from the last phase of the study (§5.1.5). We report here results based on the first metric for three reasons. First, results based on both metrics are similar. Second, the first metric allows us to address both research questions **RQ1** and **RQ2** using the same primary outcome. Third, tests based on the first metric are more conservative, meaning any results reported hold, and are even stronger with the second metric.

Similarly, although we collect data on both comfort and confidence levels, we only report on our analysis for the comfort levels, as confidence levels yield very similar results. Figure 5.2 shows both the user comfort and confidence levels when sharing their data in the *low-detail* and the *high-detail* variants.

We formally test the effect of transparency on the comfort of participants through a two-way ANOVA, and reject the null hypothesis that all transparency



(a) *low-detail* variant



(b) *high-detail* variant

The violin plots show the median of each transparency group as a horizontal line segment within each box plot, and the mean as a dot.

Figure 5.2: Violin plots of participants' comfort and confidence levels, aggregated over all data types.

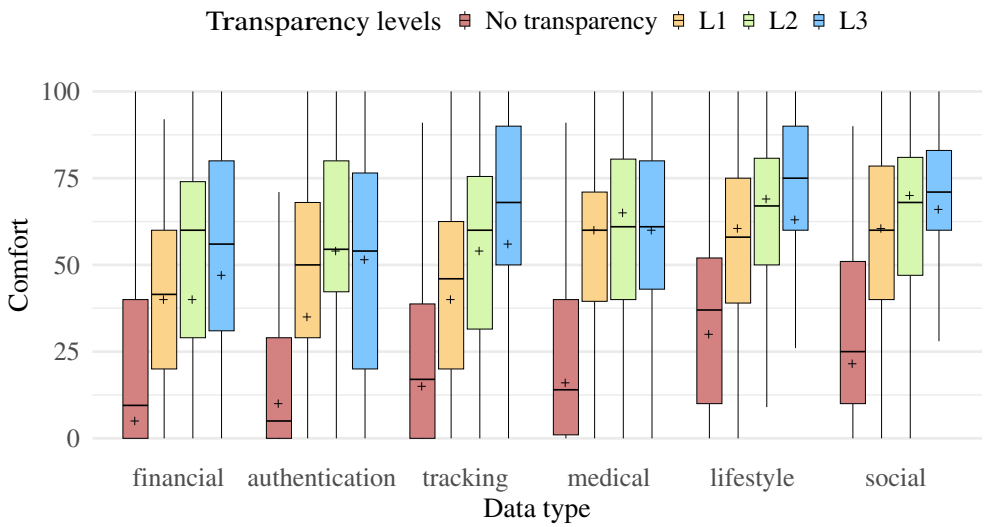
group means are equal (*low-detail* variant: $F = 169.116, p < 0.001, N = 2,419$; *high-detail* variant: $F = 218.023, p < 0.001, N = 2,334$). We further investigate which transparency group means differ from each other using the Tukey pairwise comparison test. Across both the *low-detail* and *high-detail* variants, participants report the lowest comfort levels when no transparency is provided. However, the preferred transparency levels differ between the two detail variants. In the *low-detail* variant, L2 yields the highest comfort, with L1 and L3 being perceived similarly. In contrast, in the *high-detail* variant, comfort levels increase consistently across L1 to L3, forming a clear ranking. While the first three transparency options remain relatively stable across the two detail variants, the clear ranking in the *high-detail* variant is driven by an increase in comfort with L3.

At the individual level, we analyse the preference for L3 over L2. In the *low-detail* variant, L3 and L2 are chosen at roughly equal rates. However, in the *high-detail* variant, L3 is preferred by approximately three-quarters of participants, reinforcing the impact of additional information.

Beyond transparency, we examine the effect of data type on participants' comfort, independent of transparency level. Participants are least comfortable sharing financial and authentication data, followed by tracking and medical data; they are most comfortable sharing socioeconomic and lifestyle-behavioural data. ANOVA confirms the significant effect of data type (*low-detail* variant: $F = 21.761, p < 0.001, N = 2,419$; *high-detail* variant: $F = 18.567, p < 0.001, N = 2,334$). These findings largely align with those of Chua et al. [48], suggesting a consistent pattern of privacy concerns across studies.

Next, we investigate interaction effects between transparency and data type to address **RQ2** (see Figure 5.3). Our findings hold within most data categories: in the *low-detail* variant, L2 is generally preferred, while in the *high-detail* variant, L3 becomes the dominant choice. A notable exception is financial data, where the preference between L2 and L3 reverses in both variants.

We also investigate the impact of providing more detailed information on participants' comfort levels for different transparency options within each data type. For less sensitive data types (on the right side of Figure 5.3), additional information has little to no effect on comfort levels, possibly due to ceiling effects, as comfort was already high in the *low-detail* variant. In contrast, for more sensitive



The boxplots show median comfort levels in the *high-detail* variant across data types and transparency options. Black dots represent medians in *low-detail* variant.

Figure 5.3: Boxplots of participants' comfort levels.

data types, increased detail improves comfort across all transparency levels. When we examine the data types individually, we find that medical and authentication data remain unaffected by additional information, with comfort levels remaining similar across L1, L2, and L3. The opposite trend is observed for tracking and lifestyle data, where higher transparency levels consistently increase comfort in the *high-detail* variant.

Finally, we examine how the level of detail in the introductory explanation influences participants' preferences for different transparency levels across data types. For social, medical, and authentication data, third-party and community certifiers are perceived as roughly equivalent between *low-detail* and *high-detail* variants, and providing more information does not significantly impact comfort. In contrast, for tracking and lifestyle data, participants show a stronger preference for community certifiers in the *high-detail* variant.

5.2.1 Demographic analysis

An exploratory analysis of comfort levels across demographic variables reveals no statistically significant effects. In this section, we present some noteworthy patterns. Three participant categories showed greater comfort in sharing data with our fictitious apps: younger participants, participants more willing to explore new technology, or participants from the Middle East and Africa. However, comfort levels were not correlated with participant gender. We also observe some differences in the most preferred transparency option. University graduates and participants in arts/humanities, economics and STEM fields prefer L3, whereas participants in social or health sciences and participants with no university education either prefer L2 or are indifferent between the two.

5.2.2 Additional insights

In this section, we examine whether participants' technology adoption tendencies, virtual assistant usage, and privacy concerns impact their comfort with different transparency levels, based on their answers to the relevant questions in the study (see §5.1.5 and Appendix B.6). Understanding these relationships provides insight into how personal experiences and risk perceptions shape trust in Confidential Computing.

This analysis reveals that affinity to new technology and virtual assistant use positively influence comfort levels, with effects varying by transparency level and detail variant. Technology affinity consistently predicts higher comfort, with the effect strengthening as transparency increases and being more pronounced in the *high-detail* study. Virtual assistant use shows a weaker and less consistent effect, only becoming significant at higher transparency levels. L3 consistently exhibits the strongest relationship, suggesting that tech-affine users benefit most from high transparency, particularly when detailed explanations are provided. Conversely, at no transparency, effects are weakest, indicating that in the absence of transparency, other factors (e.g., trust, privacy concerns) likely influence comfort more than tech familiarity. The *high-detail* study amplifies the effect of tech familiarity, suggesting that providing more background information enhances users' ability to leverage transparency for comfort. These findings highlight that while transparency is

beneficial, clear communication is crucial to maximise trust and comfort, especially for users less familiar with technology.

This analysis also reveals that concern about unauthorised data access negatively impacts comfort levels, but the effect varies by transparency level and detail variant. In both detail variants, higher concern correlates with lower comfort when no transparency is provided (*low-detail* variant: $r = -0.147, p < 0.001, N = 2,419$; *high-detail* variant: $r = -0.100, p = 0.019, N = 2,334$). In the *high-detail* study, transparency largely mitigates this effect, with no significant correlations at L1, L2 or L3, suggesting that detailed explanations help alleviate concerns. In contrast, in the *low-detail* study, concern remains a significant negative predictor at L3 ($r = -0.184, p < 0.001$; $\beta = -0.2421, p < 0.001$), implying that without sufficient information, users may still feel uneasy despite high transparency. L1 and L2 show no strong effects in either detail variant. This analysis once again indicates that although transparency alone improves comfort, providing detailed explanations is crucial for fully addressing pre-existing security concerns, particularly at higher transparency levels.

5.3 Qualitative results

To complement our quantitative analysis, we explored participants' personal views on transparency levels. We included an open-ended question at the end of the survey encouraging them to articulate their thought process, providing the flexibility to discuss any or all transparency options in detail. In this section, I discuss some findings of our analysis of this qualitative data (§5.3.1) and provide details about the misconceptions we identified after the *low-detail* study and how we addressed them in the *high-detail* study (§5.3.2).

5.3.1 User perceptions

We manually analysed the answers to the open-ended question, first filtering out non-informative answers that did not address the question. These included vague responses (e.g."I was honest", "Personal experience") and general statements unrelated to transparency levels (e.g. "All seem very safe", "I would rather trust

my data to machines than to people").

Starting with 738 responses, we removed 452 empty or non-informative answers, leaving 286 responses for manual annotation. Using an iterative inductive approach, we identified emerging themes until saturation was reached, i.e. no new themes appeared. Two independent annotators then classified responses into thematic categories, achieving high inter-rater reliability (Cohen's $\kappa = 0.82$, almost perfect agreement). Any disagreements were discussed and resolved.

Participants in both *low-detail* and *high-detail* variants provided similar reasoning, with roughly equal frequencies, so we present them jointly unless otherwise noted. The qualitative responses supported the quantitative findings, with 35% of annotated responses strongly opposing no transparency, citing concerns that unreviewed systems might have security flaws or allow the app developer to conceal issues. No participants explicitly defended unreviewed systems, although some acknowledged that reviewing is imperfect due to human error or expressed satisfaction with no review.

While there was broad consensus against no transparency, preferences for L1, L2, and L3 varied. The most prominent theme was objectivity and impartiality (60% of responses). Participants expressed distrust in first-party certifiers, believing they faced conflicts of interest, employer pressure, and self-serving bias. In contrast, third-party certifiers were perceived as more neutral and trustworthy, as they were thought to be less influenced by the app developer. Many participants viewed both community and third-party certifiers as more impartial (43%). However, a smaller portion believed that only community certifiers can be truly independent. This sentiment was stronger in the *high-detail* variant (42%) compared to the *low-detail* variant (14%).

Beyond objectivity, participants raised varied and sometimes conflicting concerns about different certifier types. Some felt first-party certifiers had the best understanding of the app, giving them an advantage in reviewing (5%). Others believed third-party certifiers were more precise and specialised in security auditing (7%). This belief was especially pronounced in the *low-detail* variant, where 11% of responses explicitly referred to third-party certifiers as experts or assumed they had better access to sensitive components compared to community certifiers. However, some expressed scepticism about third-party certifiers (8%), believing they could

still be pressured by or inclined to favour the app developer, although these concerns were less pronounced than for first-party certifiers.

Opinions on community certifiers were divided. Some participants valued their diversity of perspectives, innovative problem-solving, and unique insights (5%). Others saw the benefit of having more reviewers, increasing the chances of identifying security flaws (13%). However, concerns also emerged, particularly among *low-detail* participants (6%), who feared that open access to source code could attract malicious actors. Additionally, a new concern surfaced in the *high-detail* variant (5%), where some participants questioned whether individual contributors in open-source communities were truly qualified, as they were not formally hired or vetted.

5.3.2 Misconceptions and detail variants

We identified two prominent misconceptions among participants in the *low-detail* variant. First, despite clear explanations in the introduction and comprehension questions, 18% of participants incorrectly believed that certifiers could access or tamper with personal data. This misconception particularly impacted user perceptions for community certifiers, as some participants mistakenly assumed their personal data would be visible to anyone. Second, as discussed in §5.3.1, some participants (11%) mistakenly believed that only third-party certifiers had expertise in the area or had better access to the system. This was despite the fact that the script provided no indication of differing access levels, and all certifiers were explicitly labelled as experts to avoid implying any hierarchy between them.

To address these misconceptions in the *high-detail* variant, we clarified that all reviewers have an equal level of access to the system. We also changed the wording for L2 from reviewers having "exclusive access to the code" to them being "directly authorised by the app developer", as we noticed that the former wording led some participants to infer that third-party reviewers had higher access or expertise. We also shortened the explanation for L3 to make it closer length to the explanations about the other two levels. Additionally, we expanded the introductory explanation with details on concepts like Confidential Computing, its limitations, the importance of transparency, and the review process. To encourage deeper engagement with these explanations, we replaced the three true/false comprehension questions with

ten multiple-choice questions. Since the *low-detail* variant already clarified that reviewers cannot access user data, no changes were made in the *high-detail* variant regarding this misconception.

These changes significantly reduced misconceptions in the *high-detail* variant. The proportion of participants who incorrectly believed that reviewers had different levels of access to user data dropped from 18% to 6%. Misconceptions about differing expertise or system access disappeared entirely from the previous 11%. However, as previously mentioned, some participants still expressed concerns about community certifiers, questioning whether independent contributors were sufficiently vetted or formally hired for the role. Overall, participants in the *high-detail* variant exhibited greater comfort with community certifiers, suggesting that a clearer understanding of the system positively influences trust in higher levels of transparency.

5.4 Limitations

Although user study offers valuable insights into participants' perceptions of transparency in Confidential Computing, several limitations should be acknowledged. First, the study relies on self-reported comfort levels, which may not fully capture actual user behaviour in real-world settings. Participants' stated preferences might differ from their choices when interacting with real applications, particularly when faced with tradeoffs between security, usability, and convenience.

Second, the study uses hypothetical scenarios rather than real applications, which may influence responses. While we aimed to provide clear and engaging descriptions, participants' interpretations of these scenarios might not fully align with how they would evaluate transparency mechanisms in practice.

Third, despite efforts to ensure clarity with a second *high-detail* study, some misconceptions persisted, particularly regarding reviewer access levels. This suggests that security-related concepts can be difficult to convey effectively, even with refined explanations and comprehension checks.

As previously noted, the actual transparency framework builds upon successive levels, with each adding to the previous one. However, to prevent confusion, this study presented them as mutually exclusive and independent. While this approach clarifies distinctions between transparency levels, it may lead participants to view

them as isolated options rather than a progressive system.

Lastly, while we recruited participants from a broad sample, the study population may not be fully representative of all users. Differences in technical literacy, privacy attitudes, or cultural perspectives on transparency could influence how users perceive and evaluate certification mechanisms.

Future work could address these limitations by incorporating behavioural data from real-world interactions, using interactive prototypes instead of static scenarios, conducting qualitative follow-up interviews, and expanding recruitment to include a more diverse participant pool.

5.5 Summary

The Confidential Computing Transparency framework introduced in Chapter 4 is designed to enhance user trust. This chapter has presented a user study with over 800 participants, aiming to assess the impact of this framework on user perceptions. Our findings have implications for both academia and industry.

Overall, participants favoured higher transparency over unreviewed systems, with comfort levels varying across transparency levels depending on the type of personal data involved. Specifically, third-party and community certifiers were perceived as the most trustworthy, suggesting that systems incorporating these transparency mechanisms may be more appealing to end-users. The primary reason for this preference is the distrust in internal reviews, as participants viewed first-party certifiers as less reliable due to a perceived conflict of interest (§5.3).

The study also highlights how limited information about transparency can lead to misconceptions. For instance, some participants mistakenly believed that open source access to code means making user data also public. Others assumed that if the same system component were reviewed by both third-party and community reviewers, third-party reviewers must have superior access or expertise. Our results suggest that effectively communicating how Confidential Computing works, the role of transparency, and the review process can help mitigate these misconceptions, ultimately fostering greater user trust.

This chapter, along with the previous one, examined how transparency and certification mechanisms can enhance human trust in Confidential Computing

systems. However, trust is not only a human concern—an equally important challenge is how machines verify and authenticate each other to collaborate in confidentiality-sensitive tasks. The next chapter shifts focus to this machine-to-machine trust establishment, describing a protocol for achieving it efficiently in distributed Confidential Computing systems.

Chapter 6

Establishing trust in decentralised Confidential Computing systems

As the adoption of mobile and IoT devices, cloud computing, industrial control systems, and ambient computing continues to increase, the secure deployment and maintenance of large, dynamic, self-organising peer-to-peer networks becomes increasingly critical. These networks can perform a wide range of security-, privacy-, or safety-critical tasks, such as preventing collision in self-driving cars, processing private data in the cloud, and securely sharing sensor data among IoT devices. It is therefore essential to ensure that each node carries out its computation tasks securely and privately. This can be achieved with Confidential Computing, equipping each node with a TEE.

For peer-to-peer networks to collaborate on critical tasks, nodes must be able to establish trust. Here, trust refers to the assurance that one machine has in another's secure operation and adherence to confidentiality and integrity standards enabled by TEEs. Remote attestation enables this by allowing a TEE to prove its authenticity, integrity, and runtime status to other nodes. However, if every node in the network attests every other node, the resulting quadratic cost quickly becomes unscalable. This challenge is compounded in heterogeneous networks, such as those made up of smart cars from different manufacturers, where nodes may use different TEEs and incompatible attestation protocols. In such cases, peer-to-peer trust cannot be established universally. The challenge is further compounded in dynamic

networks, where nodes frequently go offline, making it impossible to establish trust with unavailable peers.

To address these limitations, we propose Careful Whisper, which is a machine-to-machine trust establishment mechanism for peer-to-peer TEE networks. Careful Whisper supports networks with intermittent connectivity and diverse TEEs. It uses gossiping to disseminate trust information efficiently, based on the idea that trust can be transitive when established and sustained within a trusted computing base. Gossiping avoids the need for routing, topology awareness, or constraints on network changes. Unlike collective attestation protocols, Careful Whisper supports ad hoc trust decisions, eliminating the need for central authorities during deployment or maintenance.

This work is a collaboration with Gustavo Petri, Dominic P. Mulligan, Derek Miller, Hugo J. M. Vincent, Shale Xiong, and Alastair Beresford. The overarching idea for gossip-based attestation and the related patent application came from Gustavo, Dominic, Derek, and Hugo. I used this core idea to create a complete protocol, authored the text, and developed a complementary protocol for a stronger adversarial model. Shale and Alastair contributed to the development and presentation of the protocols.

6.1 Peer-to-peer trust

This section first details the naive approach in establishing a peer-to-peer network of mutually trusting nodes. In this approach, each node is required to verify the attestation report generated by every other node in the network. We adopt this base-case approach to illustrate the problem we aim to address, since existing collective attestation protocols do not cater to the establishment of peer-to-peer networks, as discussed in §6.6. We then move on to describing limitations of the naive approach in dynamic and heterogeneous peer-to-peer networks.

6.1.1 Naive peer-to-peer attestation

In peer-to-peer networks where nodes process sensitive data and share it with other nodes for collaborative computation tasks, it is crucial to ensure that the data

is kept privately at all times, including while in use. Consider, for example, a network of self-driving cars that collaboratively train machine learning models to improve their contextual awareness in a specific area, using data collected from various vehicles. In such a setting, without Confidential Computing or another method for protecting data in use, a malicious node can learn sensitive information about the drivers using data from other nodes, such as their driving habits, where they live and work, and behavioural patterns. Even if each node encrypts its data while storing and shares them with others through secure channels, this does not guarantee privacy during computation, as the data must be decrypted and processed in plaintext. This vulnerability persists even in federated learning, where each node trains on its own local data while contributing to a global model. Despite appearing privacy-preserving, federated learning has been shown to leak sensitive information through model updates and gradients [220, 31, 95]. Equipping nodes with Trusted Execution Environments (TEEs) offers a solution by enabling computations to take place in isolated environments that keep data confidential even while it is being processed.

As discussed in §2.4.1, TEEs can generate attestation evidence about their trustworthiness. This information can be verified by another party to determine whether or not to trust the TEE. This is a significant improvement over authenticating nodes using a traditional digital certificate scheme such as TLS, which verifies the identity of the node. Remote attestation introduces a deeper layer of trust by allowing a verifier to evaluate the authenticity, software integrity, and certain runtime properties of the prover.

The verifier in a remote attestation scheme can be a TEE; therefore, two TEEs can engage in mutual attestation, each acting as both prover and verifier. This means that, TEE nodes can establish reciprocal trust in a peer-to-peer setting. Although this seems like a promising foundation for building trusted peer-to-peer systems, the naive approach where each node attests every other node has several fundamental limitations. The next section outlines these shortcomings.

6.1.2 Problem definition

We now turn to analysing the limitations of naive peer-to-peer attestation in the context of dynamic and heterogeneous peer-to-peer networks.

Naive peer-to-peer attestation is not scalable

Consider a set N of nodes employing the naive protocol described above. With $|N|$ nodes in the network, there are $C(|N|, 2)$ node pairs, and in every pair, both nodes must prove their state and verify the other's. This leads to a total of $\binom{|N|}{2} * 2 = |N|^2 - |N|$ attestations, resulting in quadratic complexity ($O(|N|^2)$). As the number of nodes increases, the number of attestation interactions grows quadratically, quickly becoming a bottleneck.

This scalability issue is further exacerbated when remote attestation protocols require interaction with trusted third-party services. For example, Intel SGX's attestation mechanism involves communication with the Intel Attestation Service (IAS) [114, 22], introducing additional latency and network overhead. Even non-interactive remote attestation protocols are not immune to this problem, as they still require each pair of nodes to exchange at least one message [18, 104, 43]. Therefore, onboarding a new node with the naive peer-to-peer attestation into the system requires it to individually attest to and be attested by every existing node, leading to an explosion in communication and verification costs as the network scales.

Naive peer-to-peer attestation does not support heterogeneous networks

In real-world peer-to-peer networks, it is common for nodes to be manufactured by different vendors or run different versions of software and hardware platforms. This diversity can lead to incompatibilities between attestation protocols, making it impossible for some nodes to mutually attest each other.

The naive peer-to-peer attestation approach fails to establish trust between nodes that do not implement the same remote attestation protocol. This problem has not been resolved by the collective attestation literature either (see §6.6). While some collective attestation schemes claim support for different integrity measurement mechanisms, they typically rely on non-interactive attestation protocols where

the prover can generate attestation evidence without engaging directly with the verifier [121]. However, many attestation protocols used by real-world TEEs, such as Arm TrustZone-A and Intel SGX, require the prover to interact with the verifier to create the attestation proof [145]. This creates a significant interoperability barrier in heterogeneous networks.

Naive peer-to-peer attestation is not resilient to unstable connectivity

Another key limitation of the naive approach is its assumption of stable and consistent network connectivity. In dynamic environments, such as edge deployments or mobile ad hoc networks, this assumption does not hold.

If a node cannot reach another node due to temporary disconnection or network instability, it cannot complete mutual attestation, and trust between the two cannot be established. Moreover, some attestation schemes require access to external services to validate attestation reports. For example, Intel SGX's IAS must be contacted by the verifier to confirm report authenticity [114, 22]. If a verifier cannot connect to such a service, the attestation process fails. This shortcoming makes naive peer-to-peer attestation unsuitable for deployments where network conditions are unpredictable or infrastructure-independent trust is essential.

6.2 System model and goals

Having identified the limitations of the naive attestation approach, we present the objectives of our proposed protocol in this section.

6.2.1 System goals

To describe our protocol's objectives and core mechanism, we must first establish the transitivity of trust between TEEs through attestation.

Remark 6.2.1 (Transitivity of Trust) *Let* $\mathbf{T}$ *be a binary trust relation over a set* N *of TEE-enabled nodes in a peer-to-peer network. Then:*

$$\forall n_A, n_B, n_C \in N.\ (n_A \mathbf{T} n_B \wedge n_B \mathbf{T} n_C) \Rightarrow n_A \mathbf{T} n_C$$

That is, trust is transitive over the network. If node n_A trusts node n_B, and n_B trusts node n_C, n_A also trusts n_C. To prove that this is correct, consider any $n_A, n_B, n_C \in N$, where $n_A \mathbf{T} n_B$ and $n_B \mathbf{T} n_C$. In other words, we presume that n_A trusts n_B and n_B trusts n_C. We need to show that $n_A \mathbf{T} n_C$, or n_A trusts n_C. Now, assuming that attestation establishes trust on the attested TEE, if n_B has received an evidence of attestation from n_C, it can propagate this evidence to n_A having attested n_B. That is, if n_A attests n_B, and n_B has evidence that n_C has been attested (either by n_B itself, or recursively through this procedure), then n_A can rely on n_B's account of n_C being attested, and it does not need to further attest B. Therefore, we see that $n_A \mathbf{T} n_C$, as required.

Next, we establish the efficiency goal of Careful Whisper by discussing the lower bound of complexity for an ideal peer-to-peer attestation protocol.

Remark 6.2.2 (Optimal Validation Complexity) *In a network of $|N|$ nodes, the best-case time complexity for validating attestations from all nodes in N is $O(|N|)$.*

Each node needs to be attested by at least one node to be trusted through attestation. Therefore, we cannot establish trust among a network of nodes with a better complexity then $O(|N|)$. Following Remark 6.2.1, we can achieve this ideal time as follows. We add each of the nodes in N to the peer-to-peer network one by one, where each added node attests itself to one arbitrary node, and validates the attestation of one arbitrary node. A new node is onboarded only after the new node's successful attestation is communicated to all other nodes in the network. As each incoming node is involved in 2 attestation operations, end-to-end trust is established in the network with $2 * |N|$, hence $O(|N|)$ attestations.

Goal 1: Careful Whisper aims to achieve peer-to-peer attestation in between $O(|N|)$ and $O(|N|^2)$ time.

In a peer-to-peer network, nodes joining the network can do so at any time and in any order. Ensuring that all nodes join the network sequentially as described in Remark 6.2.2 is not always feasible. As a result, a node may need to attest to multiple nodes if it does not receive an attestation report for a newly contacted node. This means that real-world peer-to-peer attestation cannot always achieve

$O(|N|)$ time complexity. In the worst case scenario, every node needs to attest to every other node in the network, which is the case in the naive protocol outlined in §6.1.2. This can occur even in a protocol that uses transitivity of trust, particularly if the network is highly fragmented. Therefore, our protocol aims to operate in amortised time between $O(|N|)$ and $O(|N|^2)$, given these upper and lower bounds.

Remark 6.2.3 (Heterogeneous Transitivity) *Let p and q be the only attestation protocols supported by nodes in the network N, and let P and Q be the sets defined as:*

$$P = \{n \in N \mid n \textit{ supports } p\}, \quad Q = \{n \in N \mid n \textit{ supports } q\}$$

Then, $\forall n_A \in P - Q.\ \forall n_B \in P \cap Q.\ \forall n_C \in Q - P.$ it holds that:

$$(n_A \mathbf{T} n_B \wedge n_B \mathbf{T} n_C) \iff n_A \mathbf{T} n_C$$

That is, nodes that do not support a common attestation algorithm can still establish trust among them, if and only if there exists a node that shares a supported attestation algorithm with each node. Consider node n_A supporting attestation protocol p, n_C supporting q, and n_B supporting both p and q. Through transitivity of trust (Remark 6.2.1), we get $(n_A \mathbf{T} n_B \wedge n_B \mathbf{T} n_C \Rightarrow n_A \mathbf{T} n_C)$. For the other direction, since n_A and n_C do not share a common supported attestation protocol, they cannot attest each other directly. Given that we assume that a node needs to be attested by at least one node to be trusted, and n_A trusts n_C, this trust must have been established transitively through n_B.

Goal 2: Careful Whisper aims to accommodate networks with nodes that support various attestation protocols.

Remark 6.2.3 highlights that trust can be established between nodes that do not share a common attestation protocol by leveraging the transitivity of trust. This is made possible through intermediary nodes that support multiple attestation protocols, effectively acting as bridges for trust propagation. By enabling trust formation in heterogeneous networks, Careful Whisper aims to support collaborative systems involving diverse stakeholders, such as those found in smart cities or networks of self-driving vehicles from different manufacturers. Moreover, the

protocol's ability to accommodate various attestation schemes promotes backwards compatibility and interoperability, making it easier to integrate legacy systems and heterogeneous devices into the network. This is especially important in real-world deployments, where the use of specialised hardware for TEEs can make it impractical or costly to maintain uniformity across all nodes.

Remark 6.2.4 (Transitive Trust under Attestation Failure) *Let t be a given point in time, and let N be the set of TEE-enabled nodes. Suppose that node $n_C \in N$ cannot be attested at time t. Then, for all $n_A, n_B \in N$:*

$$(n_A \mathbf{T} n_B \wedge n_B \mathbf{T} n_C) \iff n_A \mathbf{T} n_C$$

In simpler terms, if node n_C cannot be attested at a given time, the only way for node n_A to establish trust in n_C at that time is by obtaining trust information from a third node n_B, which itself trusts n_C. The proof follows similarly to that of Remark 6.2.3. By the transitivity of trust (Remark 6.2.1), n_A can derive trust in n_C through n_B, provided that n_A already trusts n_B and n_B trusts n_C. Therefore, we have $(n_A \mathbf{T} n_B \wedge n_B \mathbf{T} n_C \Rightarrow n_A \mathbf{T} n_C)$. For the other direction, since n_A cannot directly attest n_C directly at time t, n_A must establish trust with n_C transitively through n_B.

Goal 3: Careful Whisper aims to enable trust establishment in networks with unreliable connectivity.

Consider a scenario where a node requires internet access to complete its attestation, but temporarily loses connectivity to external services essential to the protocol. Alternatively, a node may become unreachable by its peers, preventing it from being attested altogether. In such cases, it becomes difficult to establish trust in a timely or reliable manner. As discussed in Remark 6.2.4, Careful Whisper addresses these challenges by allowing nodes to continue making informed trust decisions, even when another node cannot be attested within a given time window. This goal is particularly relevant in environments with intermittent connectivity, such as delay-tolerant networks, or in highly dynamic systems where nodes frequently join and leave the network.

6.2.2 System model

We focus on peer-to-peer networks with no central operator to deploy nodes with certain features or enforce attestation criteria. We consider both 'unpermissioned' and 'permissioned' versions of such networks. In unpermissioned networks, any node can join or leave the network freely. In contrast, permissioned networks only allow nodes owned or operated by authorised parties to join. The permissioned setup may be desirable in real-world systems like self-driving car networks or cloud computing networks, where control over network participants is necessary. In permissioned networks, we assume that nodes have the knowledge of allowed manufacturers and their public keys for verifying digital certificates.

We assume that nodes may support any attestation protocol of their choosing. Furthermore, we consider that the network is dynamic, allowing nodes to join and leave at any time. We also consider that nodes may experience intermittent connectivity to other nodes in the network or to external services. We do not make any assumptions about the number of nodes in the network at any given time or the topology of the network. However, we assume that any node willing to join the network has knowledge of the set of hash functions necessary to construct and query Bloom filters and to hash trusted node information.

6.2.3 Security considerations

Trust assumptions. We assume all nodes are equipped with Trusted Execution Environments (TEEs) that can generate attestation reports to confirm the status of their hardware or software. These reports can be verified directly by other devices or through an external attestation validation service. We view attestation as a sufficient way of establishing complete trust in the TEE being attested, assuming that the code executed by each TEE being free of vulnerabilities. In the case of permissioned networks, where only allowed parties can join (see §6.2.2), we assume that all parties on the list of allowed parties are trustworthy and protect their private signing keys from unauthorised third parties.

Adversary model. The base version of Careful Whisper assumes a software-only adversary, denoted Adv_{SW}. This adversary can compromise the software of any node in the network, allowing them to observe and modify the memory and

execution state that is not protected by a TEE. Moreover, Adv_{SW} has complete control over the communication links between the nodes, and can eavesdrop on and interfere with all exchanged messages. This adversary model is common in remote attestation literature, however, some collective attestation protocols (see §6.6), such as SALAD [121], SANA [17], and DARPA [103] aim to provide some protection against hardware adversaries. In §6.4, we discuss a protocol extension that provides a similar degree level of protection against hardware adversaries.

Security goals. Our protocol aims to prevent Adv_{SW} from deceiving any honest or compromised node into believing an unattested node n as trustworthy, regardless of whether the Adv_{SW} controls n or not. As is common in collective attestation protocols (see §6.6) [121, 9], we consider denial-of-service attacks to be out of scope of this protocol, as Adv_{SW} has full control over the communication between nodes and can discard any messages. Nevertheless, we aim to limit the impact of denial-of-service attacks to only targeted nodes in permissioned networks (see §6.3.4).

6.3 The Careful Whisper protocol

In this section, describe some building blocks of the Careful Whisper protocol (§6.3.1), describe the base protocol (§6.3.3), discuss some additional features that can be achieved by making some implementation decisions (§6.3.4), finally discussing some limitations (Section 6.3.5).

6.3.1 Preliminaries

Node IDs. In order to ensure that each node in the network can be distinctly identified, it is necessary for each node to have a unique ID that is stored securely in the TEEs. One possible approach to generate IDs is to use a keyed pseudorandom function that is seeded by each TEE using its own public key or a key derived from it. This method ensures that clashes between node IDs are prevented without requiring coordination between nodes. Although this approach requires each node to know which keyed pseudorandom function to use, this information can be made publicly available, and only needs to be set up by the node owner once. Alternatively, if

the TEE already has a unique ID assigned by its manufacturer or owner, it can use that instead.

Policies. As we discuss in §6.2.2, we build our protocol for networks with no central operator to deploy nodes with certain features or enforce attestation criteria. Furthermore, each node can support any attestation protocol of its choice. This brings out the need for a mechanism to ensure that each node can establish trust transitively as discussed in §6.2.1, while verifying the exact properties that it would check for if it attested those nodes directly. We propose a policy framework to achieve this.

We define a policy as a data structure that contains information about the criteria that a verifier has attested the prover against. A policy can also include additional information, such as an attestation timestamp or expiry period (see §6.3.4). Upon completing an attestation, each verifier generates a policy for the prover and records it in its list of trusted nodes. When sharing lists of trusted nodes, nodes also share the recorded policies for each node in the list. To ensure consistent interpretation of policies across heterogeneous nodes, a predefined set of rules must be established. All nodes joining the network are expected to follow these rules when creating or interpreting policies. For example, such rules could be defined using a domain-specific language, or by specifying a clear mapping between criteria and their possible values.

Trusted nodes list. Each node in the network should maintain a list of trusted nodes, where each entry in the list is a mapping of a node ID and a policy. Trusted nodes lists may have multiple entries for the same node ID, each with different policies. To ensure the security and integrity of the trusted nodes list, it should be stored in a secure manner. This can be achieved by storing the list in the secure memory inside the TEE. However, since the list size grows with the networks size and no assumptions are made about the secure memory requirements, the list may exceed the capacity of secure memory. In such a case, the list may be encrypted with a key stored in the secure memory using an integrity-protected encryption method, and stored outside the TEE.

Trusted node Bloom filters. To be able to establish transitive trust relationships, nodes must be able to communicate their trusted nodes with other nodes. However,

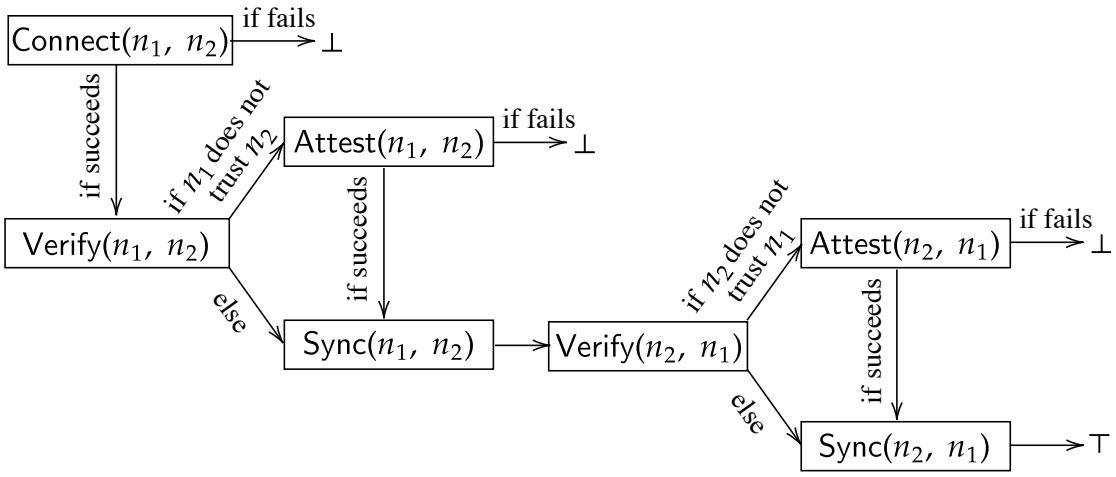


Figure 6.1: The diagram of Careful Whisper's subprotocols.

exchanging full lists of node IDs can increase network overhead in large networks. To address this issue, nodes can use Bloom filters [35] instead of exchanging full lists.

A Bloom filter is a probabilistic data structure that is used for testing if an element is member of a set or not. It is a memory-efficient construct, consisting of a bit array of length m, initially all set to 0. To insert an element, it is hashed k times with independent hash functions, and the resulting positions in the array are set to 1. To check membership, the same k hashes are computed; if all corresponding bits are set to 1, the element is possibly in the set, otherwise it is definitely not. In other words, a Bloom filter can give false positives, but not false negatives. The false positive probability after inserting n elements is $\sim \left(1 - e^{-kn/m}\right)^k$. Bloom filters do not inherently support deletion.

In Careful Whisper, each node n_A shares a Bloom filter with n_B as a compact representation of its trusted node list. Node n_B can then use this Bloom filter to identify entries in its own trusted list that are likely not present in n_A's list. With high probability, this allows n_B to share only the missing entries with n_A, rather than its entire trusted node list.

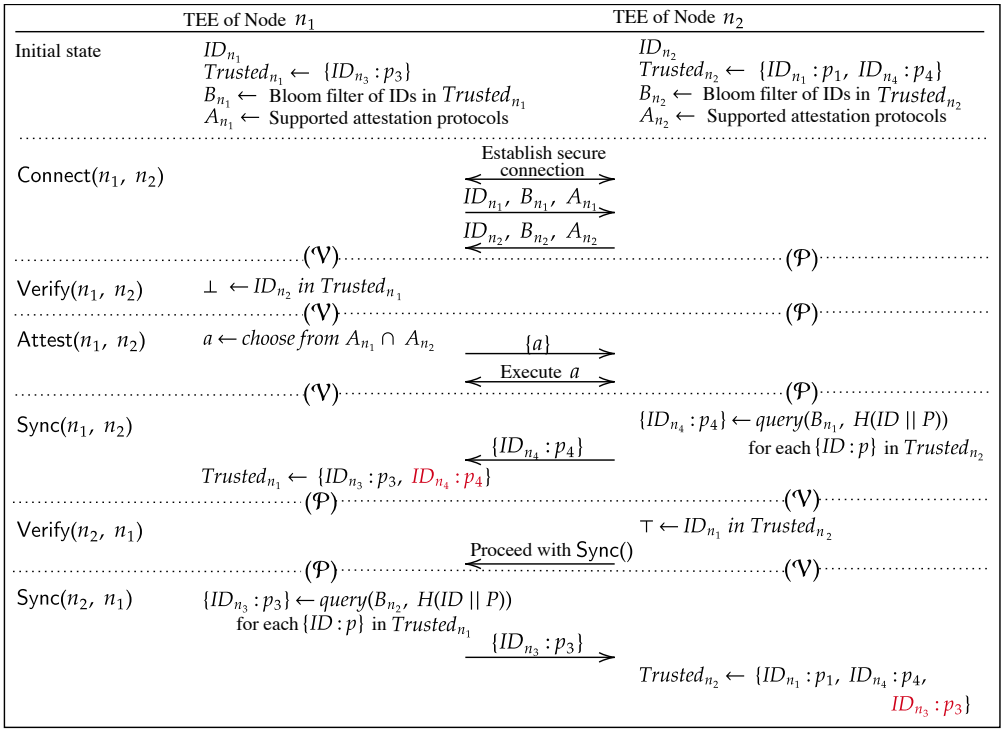


Figure 6.2: An example Careful Whisper protocol flow between nodes n_1 and n_2, where n_1 does not have prior trust on n_2, while n_2 already trusts n_1. The prover and verifier roles of the two nodes for Verify(), Sync(), and Attest() subprotocols are denoted as $\mathcal{P}$ and $\mathcal{V}$, respectively.

6.3.2 Protocol overview

The Careful Whisper protocol consists of four subprotocols, namely Connect(), Verify(), Sync(), Attest(), which we describe in detail later in this section. These subprotocols together form the Careful Whisper protocol as follows, and as described in Figure 6.3.1.

1. Initially, two nodes establish secure connection between their TEEs (Connect()).
2. One node assumes the role of the verifier, and checks if the other node, the prover is already in its list of trusted nodes (Verify()).
   (a) If the verifier is not yet trusted, the prover employs a remote attestation protocol to prove its trustworthiness to the verifier (Attest()).

(b) If the prover is already trusted or if the attestation from Step 2a is successful, the verifier expands its list of trusted nodes with the entries from the prover's list (Sync()).

3. Once the one-sided trust establishment is complete, the roles of the nodes switch, and the process is repeated from Step 2.

6.3.3 The subprotocols

After outlining the Careful Whisper protocol flow, we describe the details of the subprotocols below. The operations included in all these subprotocols occur inside the TEEs, and in our descriptions in this section, we use the word 'node' to refer to the TEE.

Connect(n_1, n_2) This subprotocol establishes a secure communication channel between two nodes and exchanges basic trust-related metadata.

1. TEEs of the two nodes n_1 and n_2 establish a secure communication channel between each other. For permissionless networks (see §6.2.2), a non-authenticated key agreement protocol like Diffie-Hellman can be used for secure connection establishment. On the other hand, for permissioned networks, an authenticated protocol such as TLS can be employed.

2. Once the secure communication channel is established, each node sends a message to the other one with three pieces of information: (1) its ID, (2) a Bloom filter pre-populated with hashes of the node ID-policy mappings in its trusted nodes list, (3) the set of remote attestation protocols it supports.

3. Once this subprotocol succeeds, any one of the nodes initiates the Verify() protocol, and assumes the role of the initial prover.

Verify(n_1, n_2) This subprotocol determines whether the prover (n_2) is already trusted by the verifier (n_1) and decides whether to proceed with attestation or synchronisation.

1. n_1 checks its list of trusted nodes to determine whether it already trusts n_2.

(a) If so, n_1 sends a message to n_2, signalling it to initiate Sync().

(b) Otherwise, n_1 initiates Attest().

Sync(n_1, n_2) This subprotocol allows the verifier (n_1) to expand its list of trusted nodes with the prover's (n_2) list of trusted nodes by selectively exchanging missing entries.

1. n_2 queries the Bloom filter provided by n_1 in Connect() subprotocol, using each of its trusted node ID - policy pairs. This allows n_2 to identify which entries are likely missing from n_1's trusted nodes list.

2. n_2 then sends a message to the verifier containing a list of these potentially missing entries.

3. n_1 updates its list of trusted nodes with the list of nodes it has received from n_2. If an entry with the same node ID already exists, the verifier may choose to retain only one version, based on factors such as memory constraints or preference for newer attestation timestamps.

4. If n_2 has not acted as a verifier throughout the current Careful Whisper protocol flow, it runs Verify(n_2, n_1). This effectively switches the verifier and prover roles for the rest of the protocol flow. If both nodes have already acted as both prover and verifier, the protocol run for n_1 and n_2 is complete.

Attest(n_1, n_2) This subprotocol performs remote attestation between the prover (n_2) and verifier (n_1), and records a policy if successful.

1. n_1 compares its supported attestation protocols with those of n_2, as received during Connect(). This step is necessary even for non-interactive protocols, since the verifier must inform the prover whether to proceed with attestation.

   (a) If the two nodes know at least one common attestation protocol, n_1 picks any protocol from the intersection set, and sends a message to n_2 to trigger the attestation protocol. If the chosen attestation protocol is interactive and requires any additional data, such as a challenge value, from the verifier, n_1 also provides this data in this message.

i. If the attestation succeeds, n_1 first creates a policy instance for n_2. This policy includes information about what criteria n_2 was attested against. After this, n_1 runs $\mathsf{Sync}(n_1, n_2)$.

ii. Otherwise, the Careful Whisper protocol flow is terminated. n_1's trust establishment with n_2 fails.

(b) If the nodes do not know any common attestation protocols or the attestation protocol fails, both this subprotocol and Careful Whisper protocol terminates.

6.3.4 Additional considerations

Denial-of-Service resistance. While our protocol does not guarantee perfect denial-of-service protection, it limits an attacker's ability to perform such attacks in permissioned networks (see §6.2.2). In these networks, nodes can share certificates obtained from their owner or manufacturer in the Connect() phase, allowing the verifier to confirm whether a node is authorised to join the network and blacklist it if necessary. As we assume software-only attackers who cannot modify the protocol code inside the TEE, this measure prevents the attacker from using the protocol to launch denial-of-service attacks against nodes that it does not control. Despite this, our protocol cannot provide full denial-of-service protection since the attacker can still drop messages, as described in §6.2.3.

Removing nodes. Our protocol includes a mechanism to detect nodes that do not maintain the criteria they were attested against and revoke trust accordingly. This is achieved by including an attestation timestamp or expiry date in the prover's policy, which is used by each verifier to decide when to expire a node from its trusted nodes list. Upon receiving the list of node IDs and policies in Sync(), each node checks if the attestation is still valid before adding each entry to its trusted nodes list. Apart from this, the nodes should also keep track of their trusted node lists for any expired trust relationships. Since it is not possible to remove any elements from Bloom filters, nodes need to repopulate their Bloom filter with the updated trusted nodes list upon removing a node. If maintaining trust with an expired node is deemed urgent, the node that removed it from the trusted nodes list can immediately execute Attest() subprotocol to reestablish trust.

6.3.5 Limitations

Our protocol makes no assumptions about the attestation protocols supported by the nodes in the network. Therefore, there may be nodes that cannot join the network because they do not support any of the attestation protocols used by other nodes. Another limtation is that, in permissionless networks, the attacker can join the network with a node that runs malicious code inside the TEE and perform denial-of-service attacks.

Since Bloom filters may produce false positives [35], the prover may incorrectly classify a node as already trusted by the verifier in Sync(). This does not cause any security issues, but decreases the efficiency of the protocol, since each false positive will result in the verifier losing the opportunity to establish transitive trust with a node. The likelihood of false positives increases with the number of entries in a trusted nodes list. However, the probability of false positives can be decreased by choosing number of hash functions used in the Bloom filter and the filter size appropriately [35, 159].

Lastly, this protocol does not provide protection against powerful adversaries with hardware access. We discuss a protocol extension to mitigate this issue in the next section.

6.4 Protection against hardware adversary

The Careful Whisper protocol described in Section 6.3 assumes a software-only adversary and relies on the integrity of the TEEs used on the nodes, without accounting for potential hardware-level compromises (see Section 6.2.3). However, a stronger adversary with physical access may be able to bypass TEE protections by exploiting hardware vulnerabilities, such as side-channel attacks or fault injection [198, 207], to forge a "trusted nodes list" that includes malicious nodes, causing honest nodes to trust them based on transitivity of trust.

In this section, we introduce an extension to the base Careful Whisper protocol (Section 6.3) designed to significantly restrict the impact of such physical adversaries in permissioned networks. This enhancement trades off increased memory and computational overhead for stronger security. We begin by detailing the extended

adversary model and security goals (Section 6.4.1), then provide a high-level overview of the protocol's design (Section 6.4.2), and finally describe the full protocol in detail (Section 6.4.3).

6.4.1 Security considerations

Adversary model. In this protocol extension, we consider a hardware adversary Adv_{HW}, who can bypass the protection of the TEE. Adv_{HW} can access and modify keys, the protocol code and trusted node list inside the TEE.

Security goals. Since Adv_{HW} has access to all keys and protocol code, it is not possible to prevent this adversary from forging an attestation report for the compromised node. Therefore, we aim to prevent the adversary from forging attestation proofs for uncompromised nodes. Specifically, consider a node n that is not compromised by Adv_{HW} and for which the adversary has no attestation proof. This extension aims to prevent Adv_{HW} from deceiving any node into believing that n is a trustworthy node. In other words, the aim is that the adversary can only convince a node that has been physically compromised that an untrusted node is indeed trustworthy, but cannot disseminate this misinformation to any other uncompromised nodes. This aim corresponds to a similar level of security that many collective attestation protocols provide against physical attackers [121, 17, 42, 103, 120].

6.4.2 Extension overview

To mitigate the risks posed by a Adv_{HW}, the protocol extension introduces cryptographic proofs that bind each attestation to its origin. Specifically, every node generates a verifiable proof of authenticity alongside its attestation data. This proof is shared with its verifier during direct attestation and subsequently propagated to other peers through trust sharing.

The goal is to ensure that when a node receives transitive trust information, it can independently verify that the underlying attestation originated from a genuine TEE and was not fabricated by a compromised or malicious party. By enforcing proof verification as a prerequisite for establishing transitive trust, the protocol

significantly limits the ability of physically compromised nodes to inject false trust into the network.

Key distribution problem. Introducing authentication to our protocol presents a key distribution problem, which is particularly challenging to address in our peer-to-peer decentralised network setup.

Message authentication codes (MACs) are not suitable for authentication in our setting. Since every node in our protocol can act as a verifier, it would be impractical for each node to maintain a separate secret key with every other node. SALAD [121] uses MACs to protect against hardware attacks, but it relies on a central verifier that possesses all the necessary keys for MAC verification. One of the core goals of our protocol is to maintain a peer-to-peer system and therefore avoid a central verifier.

Traditional digital signatures are also not feasible for authenticating attestations in our case, because they require each node to know the public key of every other node. Collective attestation protocols such as SANA [17] and PASTA [122] use digital signatures without addressing the key distribution problem, presumably because these protocols assume a single network owner that can distribute the public keys. We consider a dynamic network setup, where the set of nodes joining the network is not predetermined and connections are unreliable. Therefore, using a public key infrastructure (PKI) to distribute public keys is not feasible either, as nodes may be unable to reach the key servers for extended periods, causing delays in verifying the authenticity of attestations and even allowing a malicious attacker to launch denial-of-service attacks. Sharing public keys between nodes is similarly problematic both due to high communication costs, the goal to be resilient to unstable connectivity, and the high memory cost of storing public keys for all trusted nodes for periodical attestation renewals (see §6.3.4).

Identity-Based Signatures. To overcome the key distribution challenge in Careful Whisper's decentralised peer-to-peer setup, we use identity-based signatures [182] for authenticating attestations. In an identity-based signature scheme, a public key generator (PKG) publishes a master public key. Participants obtain their individual private keys associated their authorised identities by contacting the PKG. Identity-based signatures resolve the key distribution problem, since

anyone who knows the master public key and the identity of any signer can verify signatures of that signer [182].

However, using identity-based signatures in our setup introduces other challenges. First, the need for a trusted PKG poses a challenge, as our network setup does not have a central owner or maintainer. Nonetheless, since we propose this extension for permissioned networks, it is reasonable to assume that a consortium of manufacturers or deployers can establish a trusted PKG.

Using an identity-based signature scheme also raises issues with key revocation, since a node's key cannot be revoked without changing its identity, and revoking the master key will revoke the private keys for all nodes [90]. This is an issue, because if a hardware-compromised node is detected, we would like to be able to permanently disallow that node from participating in the protocol. To address this, we do not use node IDs as the identities for the signature scheme, but a combination of the node IDs and an epoch timestamp, which is a technique used in puncturable encryption [90]. The PKG keeps a blacklist for nodes to be removed from the network, and to remove a node, the PKG does not issue a private key for that node in the coming epochs.

The tradeoff, however, is that this approach requires nodes to periodically reconnect to the PKG to obtain updated keys. This may be impractical in networks with unstable or intermittent connectivity. One potential mitigation is to adjust the epoch length based on the expected threat model and desired availability. Another option is to allow nodes to prefetch keys for upcoming epochs within a configurable time window, reducing the likelihood of disruption.

6.4.3 Extension protocol

The first change to the core protocol described in §6.3.3 is that we define a new subprotocol GetKey(). This subprotocol should be executed by each node before joining the network and also periodically before the identity-based signature private keys for each epoch expires.

GetKey(ID, *epoch*) This subprotocol allows a node to request its epoch-specific private key from the PKG in order to participate in the extended protocol.

1. The node sends a request to the PKG containing its node ID (ID), the target

epoch timestamp (*epoch*), and its certificate.

2. The PKG verifies the node using the node's certificate; the same way that the nodes verify each other in the permissioned network.

3. The PKG also checks if ID is included in the blacklist.

4. If the node's certificate is verified and ID is not in the blacklist, PKG sends the node its private key for the requested *epoch*.

In addition to this new subprotocol, we also make two minor modifications to the Attest() and Sync() subprotocols as follows.

Changes to Attest(n_1, n_2).

1. At the end of a successful attestation in Step 1(a)i, the verifier shares the policy it has created with the prover (n_2).

2. Upon receiving the policy, n_2 signs it using its identity-based signature private key and sends the signature to n_1.

3. n_1 derives a public key from the PKG's master public key using a combination of n_2's node ID and the current epoch.

4. n_1 verifies the validity of n_2's signature using this derived public key, and stores the signature in its trusted nodes list along with n_2's ID and the corresponding policy.

Changes to Sync(n_1, n_2).

1. In Step 2, in addition to sharing a list of node IDs and policies with the verifier (n_1), the prover (n_2) also includes the policy signatures generated during the modified Attest() subprotocol.

2. In Step 3, n_1 verifies each received signature before adding the corresponding entries to its trusted nodes list. It does so by deriving a public key from the master public key using each node ID and corresponding epoch, then checking that the signature is valid for the given policy using that derived public key.

6.5 Evaluation

We evaluate Careful Whisper's performance through extensive simulation, focusing on three research questions:

RQ1. How does the protocol scale with increasing network sizes and different network topologies? (§6.5.4)

RQ2. How quickly and effectively does trust propagate across the network under different protocol strategies? (§6.5.5)

RQ3. How resilient is Careful Whisper to attestation failures, and how do failure rates affect trust convergence? (§6.5.6)

We address these questions through a series of discrete-event simulations (§6.5.3) under varying network topologies (§6.5.2). We compare Careful Whisper's performance against alternative approaches (§6.5.1).

6.5.1 Evaluated approaches

To understand the design tradeoffs of Careful Whisper, we compare three approaches. The first one is the original protocol described in §6.3.

We also evaluate a variation of the protocol, where nodes share entire lists of trusted nodes without performing any filtering via Bloom filters. We include this variant to establish an upper bound on propagation speed in the absence of any memory-saving mechanism.

Finally, we include the naive approach we used as a baseline in designing our protocol (§6.1.1), where nodes only establish trust through direct peer-to-peer attestation and do not gossip trust information beyond what they attest themselves.

6.5.2 Network topologies

To assess the effectiveness of Careful Whisper under different network conditions, we evaluate the protocol across four well-established network topologies:

- An **Erdős–Rényi graph** [75] is a random graph, where each edge between a pair of nodes is formed independently with probability p. This topology

is useful in modelling sparse, random, distributed networks, such as ad-hoc wireless networks or partially connected peer-to-peer systems.

- A **Watts–Strogatz graph** [213] is used for modelling small-world networks, where each node is initially connected to its k nearest neighbours, and each link is randomly rewired with probability p. It mimics networks with local clustering and short path lengths, such as proximity-based networks.

- A **Barabási–Albert graph** [32] models a scale-free network, where most nodes have few connections, while a small number of nodes have many. In this model, nodes are added one at a time and connect to m existing nodes with a probability proportional to their degree. Therefore, new nodes prefer to connect to already well-connected nodes. With a few highly connected hubs and many low-degree nodes, this topology resembles the Internet and some social networks.

- A **complete graph** represents a fully connected network where every node has a direct link to every other node. While unrealistic for large-scale systems, this topology serves as an upper-bound reference for trust propagation in ideal conditions.

We selected the following parameter values for each topology to ensure moderate connectivity and meaningful trust propagation within a limited number of simulation rounds. For the Erdős–Rényi model, we set the edge probability to $p = 0.05$, which keeps the network sparse but connected. In the Watts–Strogatz model, we chose $k = 4$ for the initial lattice degree and $p = 0.1$ for the rewiring probability, producing small-world graphs with high clustering and some long-range shortcuts, which balances local and global trust propagation. For the Barabási–Albert model, we set $m = 2$, meaning each new node connects to two existing nodes, generating a scale-free network with sufficient sparsity to challenge trust propagation, while still forming effective hubs. These values were selected empirically to reflect realistic network conditions without making the network overly dense or trivially connected.

6.5.3 Implementation details

We built a custom round-based simulator in Python, using `networkx` for topology generation and modelling. Each simulation runs for 500 synchronous rounds, during which 100 node pairs are randomly selected (with replacement) to simulate pairwise interactions. These interactions may include remote attestation, optional trust list sharing, and list updates depending on the protocol variant.

To enable efficient filtering of trust entries during synchronisation, we use a Bloom filter implemented with the `bitarray` library for compact bit storage and `mmh3` (MurmurHash3) for fast, non-cryptographic hashing. Each Bloom filter is configured to use 512 bits (64 bytes) and three independent hash functions, each derived from `mmh3` with a different seed.

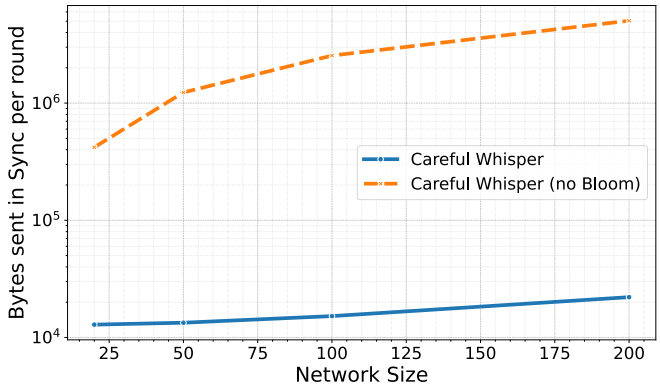
We assume each trust entry is 128 bytes, based on a conservative breakdown of components: 8 bytes for a node ID, 40 bytes for a policy (e.g., timestamps, flags), 64 bytes for a signature (e.g., Ed25519), and 16 bytes of encoding, padding or other optional data. These assumptions are used to estimate the communication overhead of synchronisation in each round.

We track three primary metrics: average trust (the average number of peers each node trusts per round), network overhead (bytes exchanged during synchronisation), and runtime (simulation time per round). These measurements allow us to compare propagation efficiency and cost across variants and conditions.

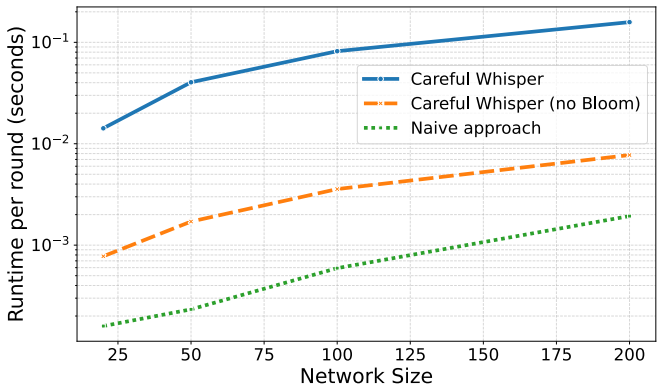
6.5.4 Resource efficiency

To evaluate the protocol's efficiency, we recorded the runtime and the total number of bytes exchanged per round. These metrics were collected across all protocol variants and topologies, each tested at multiple network sizes ($n \in 20, 50, 100, 200$) with five trials per configuration to ensure consistency. Each trial consists of 500 rounds of pairwise node interactions. In each round, 100 randomly selected node pairs (with replacement) interact according to the corresponding approaches' logic, performing attestation and optionally exchanging trusted node lists or Bloom filters.

Figure 6.3a shows that the original Careful Whisper protocol maintains a nearly constant message size per round across different network sizes. In contrast, the



(a) Average peer-to-peer message size per round.



(b) Average runtime per round.

Figure 6.3: Resource usage of Careful Whisper and other approaches as a function of network size.

no-Bloom-filter variant shows a linear increase in communication overhead, as nodes share their full trust lists. This confirms the effectiveness of Bloom filters in limiting communication to only likely-missing entries.

Figure 6.3b shows that while all variants experience longer per-round runtime with larger networks, the original Careful Whisper protocol consistently incurs more computational cost than the no-Bloom-filter version. This is because to scan a verifier's Bloom filter to identify the nodes it potentially misses in its trusted list, the verifier needs to compute multiple hashes (in our case 3) for each of its trusted nodes. As expected, the naive protocol has the lowest runtime, since it

does not involve any steps other than attestation, but at the cost of much slower trust propagation (see Figure 6.4) and higher network overhead (see Figure 6.3a).

6.5.5 Trust propagation

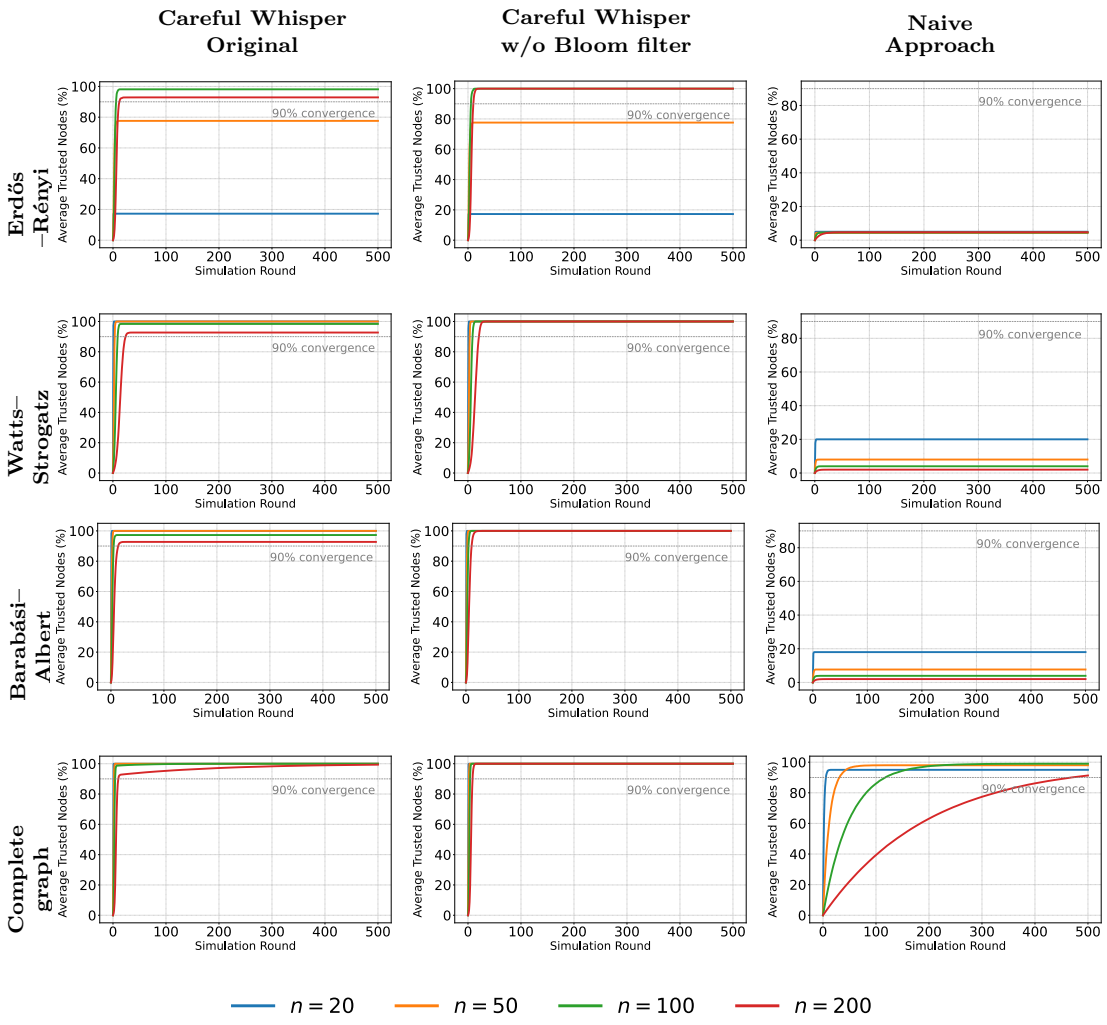


Figure 6.4: Trust propagation across different topologies (rows) and protocol strategies (columns).

To evaluate how quickly trust propagates, we simulate 500 rounds of pairwise interactions across four network sizes ($n \in \{20, 50, 100, 200\}$), with five trials per configuration. In each round, 100 random edges from the network are sampled

(with replacement) to interact. After each round, we compute the average number of nodes that each node trusts. This metric, which we refer to as *average trust*, serves as a proxy for how widely trust has propagated throughout the network.

Figure 6.4 shows trust propagation across network topologies (rows) and protocol variants (columns). Careful Whisper enables rapid trust propagation across all settings, consistently approaching high trust levels. However, in sparser topologies like Erdős–Rényi, particularly at smaller network sizes, limited connectivity reduces the chances of nodes encountering enough trusted peers, causing trust propagation to plateau slightly below 90%.

The no-Bloom-filter variant of the protocol performs nearly identically in terms of propagation speed, confirming that Bloom filters provide substantial bandwidth savings (as shown in Section 6.5.4) without compromising convergence performance. The slight difference between the original Careful Whisper protocol and the no-Bloom-filter variant is due to the tradeoff between efficiency and completeness in trust synchronisation. Bloom filters introduce a controlled false positive rate, meaning that a prover may incorrectly assume the verifier already knows certain trust entries and omit them during synchronisation. This can slightly delay the spread of trust, particularly in the early rounds and in sparse network topologies where each piece of shared information has greater influence. In contrast, the no-Bloom-filter variant always shares the full trust list, guaranteeing that all missing entries are delivered. This results in marginally faster trust propagation. Despite this, the original Bloom-filtered version converges nearly as quickly, and offers a balance between propagation speed and bandwidth efficiency.

The naive approach, where nodes only establish trust through direct attestation and do not propagate trust transitively, struggles to scale. Trust growth quickly plateaus at low levels, especially in sparser graphs, as nodes can only trust those they directly meet. This limitation becomes especially apparent in sparser or less connected topologies, like Erdős-–Rényi, where random pairwise interactions may not cover all node pairs even after many rounds. Some nodes simply never encounter each other directly, and because no indirect trust propagation occurs, their trust lists remain incomplete. This behaviour reflects the scalability and coverage limitations of the naive peer-to-peer attestation approach.

Network topology influences convergence speed. Complete graphs converge

quickly due to full connectivity, while Barabási–Albert and Watts–Strogatz topologies benefit from hubs and shortcut links. Despite topological differences, Careful Whisper provides robust and scalable propagation across all conditions.

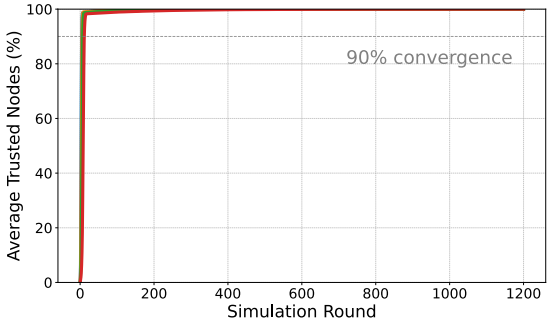
6.5.6 Attestation failure resilience

In real-world settings, remote attestation can intermittently fail due to a number of issues, such as network instability (e.g., dropped packets, latency), hardware issues (e.g., TEE faults or reboots), and software inconsistencies (e.g., outdated drivers, certificate mismatches). Service unavailability, such as downed attestation servers or reliance on third-party infrastructure, can also disrupt attestation. These challenges highlight the importance of designing protocols that remain robust in the face of occasional attestation failures. We assess the protocol's resilience by simulating a range of attestation success rates (ASR) ranging from 100% to 25%. We focus on the complete graph topology as it presents the ideal network conditions. Figure 6.5 shows the results.

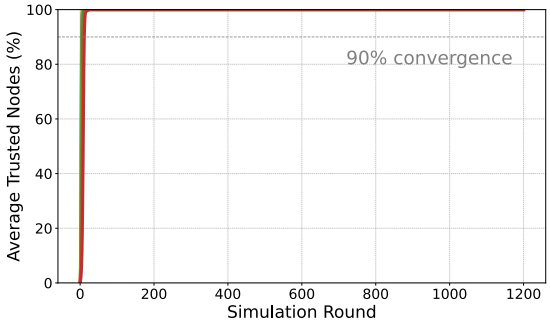
The original Careful Whisper maintains relatively high levels of trust even when attestation is unreliable. Trust continues to grow steadily at 75% ASR and remains effective even at 50% and 25%, although more slowly. This shows that the protocol successfully compensates for missed attestations through gossiping with peers. The no-Bloom-filter variant of Careful Whisper behaves similarly to the original protocol in terms of trust propagation. As both versions rely on the same gossip-based mechanism to spread trust; the absence of Bloom filters mainly affects bandwidth efficiency rather than trust propagation under attestation failure. On the other hand, the naive approach struggles at moderate to high failure rates.

6.5.7 Overview of findings

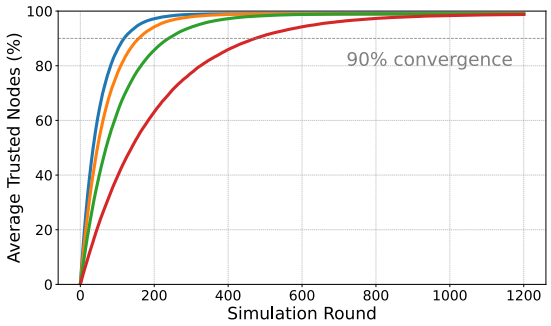
Our evaluation demonstrates that Careful Whisper effectively balances scalability, trust propagation speed, and resilience under attestation failure. It achieves quick trust propagation while significantly reducing communication overhead, maintaining nearly constant sync sizes even as network size increases. Although the no-Bloom-filter variant of the protocol converges slightly faster due to sharing full trust



(a) Careful Whisper



(b) Careful Whisper (no Bloom filter)



(c) Naive approach

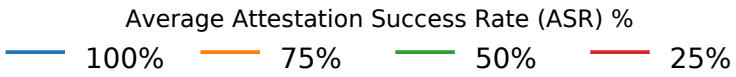


Figure 6.5: Impact of attestation failure on trust propagation in complete graph topology.

lists, it incurs higher network overhead, confirming the value of Bloom filtering for bandwidth efficiency.

Across all tested topologies, Careful Whisper exhibits consistent performance, with minimal sensitivity to network structure. In contrast, the naive approach leads to incomplete trust propagation, especially in sparse or randomly connected networks.

When attestation becomes unreliable, Careful Whisper continues to perform robustly, sustaining trust growth even at low attestation success rates. This resilience is due to its gossip-based trust dissemination, which helps compensate for missed direct peer-to-peer attestations. The naive baseline, lacking this redundancy, quickly stagnates under failure.

Overall, the results shows that Careful Whisper provides scalable, resource-efficient, and resilient trust propagation, even in challenging network conditions.

6.6 Related work

Collective (or swarm) attestation protocols aim to allow a trusted verifier to verify the trustworthiness of a swarm of provers, in a more efficient way than attesting every individual device. These protocols focus on a centralised model for attesting multiple devices.

The assumption of most collective attestation protocols is that the network is deployed by a trusted party; and also that a single entity, often the deployer, is responsible for verifying and obtaining information about the trustworthiness of the overall network or individual nodes [17, 42, 103, 120, 121]. Even schemes that allow multiple verifiers assume that the network is deployed and operated by a single party, and that all verifiers have knowledge of the public keys of all nodes in the network [122, 17]. As a result, these protocols are ill-suited for decentralised deployment and maintenance of large TEE networks.

In *SEDA* [29], the verifier initiates an attestation request that constructs a spanning tree of prover devices, with the verifier logically positioned as the root. Each prover in the tree attests its child nodes and forwards an aggregated attestation result to its parent. Ultimately, the verifier receives a single, cumulative report representing the integrity of the entire swarm. This report includes only the total number of devices that successfully completed attestation; it does not specify which individual devices passed or failed. While this design achieves scalability

and efficiency, it relies on a stable network topology to preserve the spanning tree throughout the attestation process, a requirement that limits its applicability in highly dynamic or mobile networks. Furthermore, the verifier only learns the total number of trustworthy devices, but not their individual identities, which may be insufficient in scenarios that require fine-grained trust decisions or localised remediation.

DARPA [103] is a supplementary protocol to existing single-verifier schemes, which aims to provide protection against physical attacks. It is based on the assumption that physical attacks typically require the targeted device to be temporarily disconnected or unreachable for a non-negligible amount of time. DARPA leverages this by implementing a heartbeat protocol: devices periodically broadcast authenticated heartbeats to prove their presence, and log the heartbeats of their peers. During the next attestation round, a central verifier collects these logs to detect any devices that were absent for longer than a defined threshold. However, DARPA assumes a centralised deployment model and still depends on a single trusted verifier for attestation coordination and final judgement. While it improves security in the presence of physical attacks, its reliance on centralised control and static heartbeat intervals limits its applicability in dynamic, decentralised environments.

SANA [17] introduces a novel Optimistic Aggregate Signature (OAS) scheme that enables publicly verifiable aggregate attestation reports with constant-time verification, regardless of network size. To mitigate denial-of-service (DoS) attacks, SANA incorporates an authorisation token mechanism that limits which verifiers can initiate the protocol. SANA provides granular visibility into which devices passed or failed attestation, including their software configurations. Despite its advantages, SANA shares a key limitation with SEDA: it requires full device connectivity during attestation and relies on the construction of a stable aggregation tree. Moreover, the OAS cryptographic operations impose significant computational overhead on resource-constrained devices, which can hinder its deployment in real-world IoT environments with strict energy and processing constraints. SANA also relies on a central verifier and assumes uniform attestation mechanisms, making it less suitable for decentralised or heterogeneous environments where nodes must establish trust autonomously.

DIAT [9] considers a setting where each prover is also a verifier. It uses a decentralised attestation mechanism, where each device is responsible for verifying the integrity of others based on their interactions. Although this does not require a device to directly communicate with every other node it intends to verify, the complexity of verification remains quadratic. This is because devices must still individually validate attestations tied to each device, and attestation results are not globally aggregated or reused. Compared to Careful Whisper, DIAT offers strong guarantees for data integrity in collaborative settings but lacks decentralised trust propagation and flexibility across heterogeneous TEE environments.

PASTA [122] uses tokens created with Schnorr multisignatures to achieve a decentralised, robust, and secure attestation scheme. In PASTA, provers do not exchange detailed attestation reports with one another. Instead, each device performs a self-attestation inside its TEE. If this self-attestation succeeds, each device contributes to the generation of a collective, unforgeable token. This token includes a timestamp, a list of participating device IDs, and a joint signature over this information. It does not contain any explicit evidence of the device's software configuration or runtime state. Devices can then assess the integrity of their peers by checking whether they have recently contributed to a valid token. Nevertheless, PASTA offers limited transparency and flexibility: it does not support heterogeneous attestation methods, nor does it allow fine-grained trust relationships or evidence inspection.

SALAD [121] is a collective attestation protocol that uses gossiping as its core mechanism, like Careful Whisper. However, SALAD significantly differs from our protocol in terms of its trust model, deployment assumptions, and attestation flow. SALAD relies on a central network operator to initiate attestation rounds, define valid software states, and ultimately verify the aggregated attestation results. In contrast, Careful Whisper does not rely on a single verifier; it supports ad hoc deployments where nodes can join without any central coordination, allowing for multiple or even no operators. SALAD's attestation proofs are based on MACs, and while the paper does not detail key distribution, it implies a pre-established trust through keys issued by the network operator during deployment. SALAD enforces a global definition of "healthy" software states determined by the operator, whereas Careful Whisper allows each device to evaluate attestation results according to its

own locally defined trust policies.

In addition to its centralised nature, SALAD lacks support for heterogeneous attestation schemes and does not provide transitive trust relationships. Moreover, SALAD focuses on maintaining trust in a network that has already been deployed, while Careful Whisper is concerned with establishing trust from an untrusted starting point. Careful Whisper supports both interactive and non-interactive remote attestation protocols, accommodating diverse TEEs and attestation mechanisms, whereas SALAD is effectively limited to non-interactive protocols.

6.7 Summary

This chapter presented Careful Whisper, a protocol for scalable, decentralised trust establishment in peer-to-peer networks of trusted execution environments (TEEs). It enables nodes to gradually build indirect trust through authenticated interactions and efficient gossip-based synchronisation, without requiring central infrastructure, global views, or stable connectivity.

We evaluated our protocol across a wide range of network conditions. Simulations showed that the protocol propagates trust rapidly, even in large and dynamic networks, while significantly reducing communication overhead by using Bloom filters. It remains effective under high attestation failure rates due to redundancy of gossiping. Compared to both a naive baseline and a variant without Bloom filters, Careful Whisper offers strong tradeoffs between propagation speed, bandwidth efficiency, and robustness, demonstrating its suitability for real-world deployment in heterogeneous, resource-constrained systems.

Chapter 7

Conclusion and future work

This dissertation examined the complex trust challenges in confidential computation and communication systems, and proposed practical, scalable, reliable and transparent approaches to address them. Chapter 3 introduced Pudding, a discovery protocol for user-to-user trust in anonymity networks that enables email-based discovery while preserving metadata privacy, securing usernames, and ensuring fault tolerance. Chapter 4 presented the Confidential Computing Transparency framework, which helps users make informed trust decisions by exposing relevant technical information in structured, accessible ways. Chapter 5 reported a large-scale user study showing that clear, well-communicated transparency significantly increases user trust and willingness to share sensitive data. Finally, Chapter 6 presented Careful Whisper, a gossip-based protocol that reduces the complexity of remote attestation in peer-to-peer TEE networks, enabling scalable, interoperable trust in dynamic and heterogeneous environments. Together, these contributions offer practical, privacy-preserving approaches to trust that reflect real-world complexity and bring confidential systems closer to scalable deployment.

On user-to-user trust in confidential communications

Secure mobile communication technologies have become significantly more user-friendly over time, a trend that mirrors their rising popularity. This change extends beyond improvements in the User Interface (UI) and User Experience (UX) design; it reflects deeper changes at the protocol level. In PGP, users needed to generate

and backup keys, securely share public keys, verify other users' identities through key fingerprints, and handle key revocation and expiration. These responsibilities were intrinsic to the protocol and could not be simplified by improving the UI and UX. End-to-end encrypted messaging apps provided a fundamental usability improvement at the protocol level by using centralised servers to handle most of these operations on behalf of the user. Pudding also provides a protocol-level user discovery solution for anonymity networks by automating user discovery with distributed discovery servers.

While end-to-end encrypted messaging apps build on usable protocols with polished interfaces, Pudding does not yet explore UI and UX design. This is a promising direction for future work, as we cannot directly apply the techniques that have been successful in end-to-end encrypted messaging apps. For example, Pudding requires users to send emails to discovery servers for registration, unlike messaging apps, which typically rely on phone numbers. Although replying to an email is often straightforward, it is still a different interaction. Another difference is that Pudding always returns a legitimate-looking response to discovery queries, unlike typical messaging apps. A user study could help identify how to design interfaces and interactions that improve usability in this unique context.

It is also important to note that although user discovery is a critical challenge in anonymity networks, it is not the only barrier to broader adoption. Public awareness also plays a critical role. As discussed in Chapter 5, even in a different context (transparency and Confidential Computing), our user study has shown that providing clear and detailed information about technical privacy concepts can meaningfully shift users' understanding and attitudes. Despite these standing challenges, Pudding represents a meaningful step toward more usable, accessible, and widely adopted metadata-private communication.

On systematising a mechanism for user-to-machine trust in Confidential Computing

This work originated during my internship at Google Research, which took place from May to December 2023. During this internship, I participated in discussions surrounding the design of a Confidential Computing system and engaged with

relevant teams about their approaches to transparency. Informal conversations evolved into a months-long project, which first focused on the specific Confidential Computing system in question and expanded into the generalised transparency framework.

I gained valuable insights from being involved in the design process of a real-world Confidential Computing system. First of all, many components in these systems are proprietary or IP-sensitive to the manufacturer, which is a challenge to transparency. Second, in many cases, the manufacturing company does not have full governance over all constituent components, as they use hardware and software purchased from third-party vendors. As a result, even the components designed and produced in-house may be subject to external constraints. For example, a compiler developed by the manufacturer might target an architecture that is IP-restricted, and releasing it as open source could violate contractual agreements with the IP holder. A third major insight was the scale of effort required to maintain transparency across the entire stack. Confidential Computing systems are complex, and the responsibilities that come with open source, such as responding to community-reported issues, require significant resources.

These challenges highlight why a rigid "open everything or nothing" approach can often result in no meaningful transparency at all. Instead, what is needed is a more gradual, flexible approach; something that lets system designers mix and match different transparency tools and techniques, depending on the specific constraints and needs of each component. This kind of an adaptive approach can better accommodate the realities of modern systems without abandoning the core goals of transparency and trust.

The proposed transparency framework is a step in this direction and can be extended by future work on systematising how to assess appropriate transparency levels across system components. As not all parts of a system require the same degree of transparency; decisions should account for trust assumptions, technical constraints, and user needs. Structured models and guidelines could help navigate these tradeoffs, enabling more consistent and trust-oriented design choices.

On measuring the impact of transparency in user-to-machine trust in Confidential Computing

Our user study demonstrates that people do care about transparency in Confidential Computing—crucially, once the concept is clearly explained to them. This may indicate that transparency resonates with users, but only when its relevance is made explicit. A promising direction for future research is to explore how users perceive and value transparency in other technical contexts, as well as how much they care prior to being introduced to the concept.

One major challenge in designing the user study was deciding how to communicate Confidential Computing as a concept to end users. We needed to explain both what Confidential Computing is and why transparency is important in this context. Only after establishing that foundation could we ask users meaningful questions to understand their perceptions and preferences. I believe there is a lot of work to do focusing only on the first task above: how to communicate what Confidential Computing can and cannot offer to end users. At the time of writing, I am not aware of any formal studies that have tackled this communication challenge directly. I believe such research can play an important role in building broader public understanding and awareness of the technology.

This communication challenge extends beyond end users. Although we did not study this formally, anecdotal conversations with industry professionals suggest that even decision-makers responsible for adopting these technologies often lack a full understanding of their capabilities and their limitations. Another key audience for these efforts is internal stakeholders: even experts who understand that the trust model in Confidential Computing relies heavily on transparency often need to persuade their own managers, who may be less familiar with this fact, to allocate resources accordingly. I hope the findings of this user study can be used as a motivator in such conversations all over the world in manufacturers, small and large.

On machine-to-machine trust in peer-to-peer Confidential Computing

As seen with the security and implementation challenges introduced by the EU Digital Markets Act's push for end-to-end encryption interoperability [8], achieving

interoperability in a security-sensitive context is far from straightforward. In Confidential Computing, although attestation frameworks like Security Protocols and Data Models (SPDM) aiming standardise attestation exist [70], looking at the current practices, it seems likely that a diversity of attestation protocols will continue to coexist. This reflects the reality that hardware vendors, cloud providers, and platform builders often have different assumptions and threat models, and no single protocol currently meets all needs. Careful Whisper, by supporting a variety of attestation protocols while still enabling interoperability, can be especially helpful.

While Careful Whisper demonstrates scalability and robustness in simulated dynamic peer-to-peer networks, future work could explore how these improvements manifest in practical deployments. It would also be valuable to study how the number and placement of multi-protocol bridge nodes affects convergence and resilience.

Careful Whisper uses policies as tools for nodes to choose which gossiped trusted nodes to trust. In the absence of a global trust anchor, these local trust decisions shape the overall trust graph. These policies must be a shared way to express and evaluate trust decisions. Future work can look into developing a common policy language or framework could make this possible by standardising how trust conditions are described, exchanged, and interpreted across implementations. Ideally, such a framework would be flexible enough to support different use cases, yet simple enough to be reliably implemented and audited.

References

[1] Tor rendezvous specification - version 3. Tor Project. URL https://gitweb.torproject.org/torspec.git/tree/rend-spec-v3.txt. (accessed 2021-09-02).

[2] Sunlight Certificate Transparency log. https://sunlight.dev/, accessed 2024-09-04.

[3] Fitness app Strava lights up staff at military bases. BBC News, January 2018. URL https://perma.cc/ZWS2-5VTC. (accessed 2021-08-27).

[4] CVE-2020-16009. Available from MITRE, CVE-ID CVE-2020-16009, July 2020. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-16009.

[5] CVE-2020-16875. Available from MITRE, CVE-ID CVE-2020-16875, August 2020. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-16875.

[6] CVE-2021-1223. Available from MITRE, CVE-ID CVE-2021-1223, January 2021. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-1223.

[7] CVE-2021-29976. Available from MITRE, CVE-ID CVE-2021-29976, April 2021. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-29976.

[8] Regulation (EU) 2022/1925 of the European Parliament and of the Council of 14 september 2022 on contestable and fair markets in the digital sector

and amending directives (eu) 2019/1937 and (eu) 2020/1828 (digital markets act). *Official Journal of the European Union*, L 265:1–66, 2022. URL https://eur-lex.europa.eu/eli/reg/2022/1925/oj.

[9] Tigist Abera, Raad Bahmani, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, and Matthias Schunter. DIAT: Data integrity attestation for resilient collaboration of autonomous systems. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS*, 2019.

[10] Adam Langley. Maintaining digital certificate security. Google Security Blog. https://security.googleblog.com/2014/07/maintaining-digital-certificate-security.html, accessed 2023-11-28.

[11] Heather Adkins. An update on attempted man-in-the-middle attacks. Google Security Blog. https://security.googleblog.com/2011/08/update-on-attempted-man-in-middle.html, accessed 2023-11-28.

[12] Mansoor Ahmed-Rengers, Diana A Vasile, Daniel Hugenroth, Alastair R Beresford, and Ross Anderson. Coverdrop: Blowing the whistle through a news app. *Proceedings on Privacy Enhancing Technologies*, (2):47–67, 2022.

[13] Mustafa Al-Bassam and Sarah Meiklejohn. Contour: A practical system for binary transparency. In *ESORICS 2018 International Workshops, DPM 2018 and CBT 2018*, 2018.

[14] Al Cutter. Trillian transparent logging: A guide. https://github.com/google/trillian/blob/6ee0d6b6a7823b02b11c3df2a81ccdfb8d9153bb/docs/TransparentLogging.md, accessed 2023-11-20.

[15] Amazon Web Services. AWS Nitro system. https://aws.amazon.com/ec2/nitro/, accessed 2023-11-29.

[16] Amazon Web Services. AWS Nitro system gets independent affirmation of its Confidential Compute capabilities. AWS Compute Blog, 5 2023. https://aws.amazon.com/blogs/compute/aws-nitro-system-gets-independent-affirmation-of-its-%

confidential-compute-capabilities/, author=Liguori, Anthony, accessed 2023-11-28.

[17] Moreno Ambrosin, Mauro Conti, Ahmad Ibrahim, Gregory Neven, Ahmad-Reza Sadeghi, and Matthias Schunter. Sana: secure and scalable aggregate network attestation. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 731–742, 2016.

[18] Moreno Ambrosin, Mauro Conti, Riccardo Lazzeretti, Md Masoom Rabbani, and Silvio Ranise. Pads: practical attestation for highly dynamic swarm topologies. In *2018 International Workshop on Secure Internet of Things (SIoT)*, pages 18–27. IEEE, 2018.

[19] AMD. AMD-ASPFW (source code). https://github.com/amd/AMD-ASPFW, accessed 2023-11-28.

[20] AMD. AMD Secure Encrypted Virtualization (SEV). https://www.amd.com/en/developer/sev.html, accessed 2024-08-20.

[21] AMD. Amd shares the technical details of technology powering confidential computing on 4th gen epyc processors, August 2023. URL https://ir.amd.com/news-events/press-releases/detail/1154/amd-shares-the-technical-details-of-technology-powering. accessed 2025-01-30.

[22] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, volume 13. ACM New York, NY, USA, 2013.

[23] Android. Android Virtualization Framework (AVF) overview. https://source.android.com/docs/core/virtualization, accessed 2024-08-19.

[24] Sebastian Angel and Srinath Setty. Unobservable communication over fully untrusted infrastructure. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, pages 551–569, 2016.

[25] Apache Software Foundation. CVE-2021-44228: Log4Shell. Available from MITRE, CVE-ID CVE-2021-44228. https://www.cve.org/CVERecord?id=CVE-2021-44228.

[26] Arm. Arm Confidential Compute Architecture (CCA). https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture, accessed 2024-08-20.

[27] Arm. Arm Trusted Firmware TF-RMM. https://www.trustedfirmware.org/, accessed 2025-01-30.

[28] CCRA (Common Criteria Recognition Arrangement). Common Criteria for Information Technology Security Evaluation (CC). Technical report, Common Criteria Recognition Arrangement (CCRA), 2022. URL https://www.commoncriteriaportal.org/. ISO/IEC 15408.

[29] Nadarajah Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. SEDA: Scalable embedded device attestation. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 964–975, 2015.

[30] Michael Backes, Aniket Kate, Praveen Manoharan, Sebastian Meiser, and Esfandiar Mohammadi. AnoA: A framework for analyzing anonymous communication protocols. In *IEEE Computer Security Foundations Symposium*, pages 163–178, 2013.

[31] Li Bai, Haibo Hu, Qingqing Ye, Haoyang Li, Leixia Wang, and Jianliang Xu. Membership inference attacks and defenses in federated learning: A survey. *ACM Computing Surveys*, 57(4):1–35, 2024.

[32] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[33] Henk Birkholz, Dave Thaler, Michael Richardson, Ned Smith, and Wei Pan. Remote ATtestation procedureS (RATS) Architecture, 2023. https://datatracker.ietf.org/doc/rfc9334/, accessed 2023-11-29.

[34] Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of A5/1 on a PC. In *Fast Software Encryption: 7th International Workshop (FSE)*, pages 1–18. Springer, 2001.

[35] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[36] Joseph Bonneau and Stuart Schechter. Towards reliable storage of 56-bit secrets in human memory. In *Proceedings of the 23rd USENIX Security Symposium (USENIX Security)*, pages 607–623, 2014.

[37] Nikita Borisov, George Danezis, and Ian Goldberg. DP5: A private presence service. In *Proceedings on Privacy Enhancing Technologies Symposium (PETS)*, 2015.

[38] Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[39] Posted by Apple Security Engineering and Architecture (SEAR). Security research on private cloud compute. Apple Security Research Blog, October 2024. `https://security.apple.com/blog/pcc-security-research/`, accessed 2024-11-22.

[40] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to Reliable and Secure Distributed Programming.* Springer, second edition, 2011. ISBN 9783642152597. doi: 10.1007/978-3-642-15260-3.

[41] Ran Canetti and Hugo Krawczyk. Security analysis of IKE's signature-based key-exchange protocol. In *Proceedings of the Annual International Cryptology Conference (CRYPTO)*, pages 143–161, 2002.

[42] Xavier Carpent, Karim ElDefrawy, Norrathep Rattanavipanon, and Gene Tsudik. Lightweight swarm attestation: a tale of two LISA-s. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, pages 86–100, 2017.

[43] Xavier Carpent, Gene Tsudik, and Norrathep Rattanavipanon. Erasmus: Efficient remote attestation via self-measurement for unattended settings. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1191–1194. IEEE, 2018.

[44] David Chaum. Blind signatures for untraceable payments. In *Proceedings of the Annual International Cryptology Conference (CRYPTO)*, pages 199–203. Springer, 1983.

[45] David Chaum, Mario Yaksetig, Alan T Sherman, and Joeri de Ruiter. UDM: Private user discovery with minimal information disclosure. *Cryptologia*, 46 (4):347–379, 2022. doi: 10.1080/01611194.2021.1911876.

[46] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[47] Raymond Cheng, William Scott, Elisaweta Masserova, Irene Zhang, Vipul Goyal, Thomas Anderson, Arvind Krishnamurthy, and Bryan Parno. Talek: Private group messaging with hidden access patterns. In *Proceedings of the 36th Annual Computer Security Applications Conference (ACSAC)*, pages 84–99, 2020.

[48] Hui Na Chua, Jie Sheng Ooi, and Anthony Herbland. The effects of different personal data categories on information privacy concern and disclosure. *Computers & Security*, 110:102453, 2021.

[49] Laurent Chuat, Pawel Szalachowski, Adrian Perrig, Ben Laurie, and Eran Messeri. Efficient gossip protocols for verifying the consistency of certificate logs. In *Proceedings of the IEEE Conference on Communications and Network Security (CNS)*, pages 415–423, 2015.

[50] Google Cloud. Confidential Computing. `https://cloud.google.com/confidential-computing`, accessed 2023-11-29.

[51] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol.

In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 451–466, 2017.

[52] Software Freedom Conservancy. Reproducible builds. `https://reproducible-builds.org/`, accessed 2023-11-28.

[53] Confidential Computing Consortium. A technical analysis of Confidential Computing. Technical report, Confidential Computing Consortium, November 2022. accessed 2023-11-28.

[54] Confidential Computing Consortium. Common terminology for Confidential Computing. Technical report, Confidential Computing Consortium, December 2022. accessed 2023-11-28.

[55] David Cooper, Stefan Santesson, Stephen Farrell, Sharon Boeyen, Russell Housley, and William Polk. RFC 5280: Internet X.509 public key infrastructure certificate and Certificate Revocation List (CRL) profile, 2008. `https://datatracker.ietf.org/doc/html/rfc5280`, accessed 2024-08-19.

[56] Russ Cox and Filippo Valsorda. Proposal: Secure the public go module ecosystem, April 2019. `https://go.googlesource.com/proposal/+/master/design/25530-sumdb.md`, accessed 2023-11-29.

[57] Dave Crocker, Tony Hansen, and Murray Kucherawy. DomainKeys Identified Mail (DKIM) signatures. RFC6376, 2011.

[58] CVE. CVE Process. `https://www.cve.org/About/Process`, accessed 2023-11-20.

[59] George Danezis and Ian Goldberg. Sphinx: A compact and provably secure mix format. In *Proceedings of the 30th IEEE Symposium on Security and Privacy (S&P)*, pages 269–282. IEEE, 2009.

[60] George Danezis and Andrei Serjantov. Statistical disclosure or intersection attacks on anonymity systems. In *International Workshop on Information Hiding*, pages 293–308. Springer, 2004.

[61] George Danezis and Paul Syverson. Bridging and fingerprinting: Epistemic attacks on route selection. In *Proceedings on Privacy Enhancing Technologies Symposium (PETS)*, pages 151–166. Springer, 2008.

[62] George Danezis, Roger Dingledine, and Nick Mathewson. Mixminion: Design of a type iii anonymous remailer protocol. In *IEEE Symposium on Security and Privacy (S&P)*, pages 2–15. IEEE, 2003.

[63] Debian GNU/Linux. CVE-2014-6271: ShellShock. Available from MITRE, CVE-ID CVE-2014-6271. `https://www.cve.org/CVERecord?id=CVE-2014-6271`.

[64] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. Pir-psi: scaling private contact discovery. *Proceedings on Privacy Enhancing Technologies Symposium (PETS)*, 2018.

[65] Matthew Dempsky. DNSCurve: Link-level security for the Domain Name System. IETF Internet-Draft, 2009. URL `https://tools.ietf.org/id/draft-dempsky-dnscurve-00.html`.

[66] Frank Denis, Edward Eaton, Tancrède Lepoint, and Christopher A. Wood. Key blinding for signature schemes. IETF Internet-Draft, 2023. URL `https://datatracker.ietf.org/doc/draft-irtf-cfrg-signature-key-blinding/`.

[67] Claudia Diaz, Harry Halpin, and Aggelos Kiayias. The Nym network: The next generation of privacy infrastructure. Technical report, Nym Technologies SA, February 2021. URL `https://nym.com/nym-whitepaper.pdf`.

[68] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[69] Roger Dingledine, Nick Mathewson, Paul F Syverson, et al. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium (USENIX Security)*, volume 4, pages 303–320, 2004.

[70] Distributed Management Task Force. Security Protocol and Data Model (SPDM) specification. Technical Report DSP0274, Distributed Management Task Force, June 2023. URL https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.3.0.pdf.

[71] Orr Dunkelman, Nathan Keller, and Adi Shamir. A practical-time attack on the a5/3 cryptosystem used in third generation gsm telephony. *Cryptology ePrint Archive*, 2010.

[72] Edward Eaton, Douglas Stebila, and Roy Stracovsky. Post-quantum key-blinding for authentication in anonymity networks. In *7th International Conference on Cryptology and Information Security in Latin America (LATINCRYPT)*, pages 67–87, 2021.

[73] Apple Security Engineering and Architecture (SEAR). Advancing iMessage security: iMessage contact key verification. Apple Security Research Blog, October 2023. https://security.apple.com/blog/imessage-contact-key-verification/, accessed 2023-11-28.

[74] Apple Security Engineering, Architecture (SEAR), User Privacy, Core Operating Systems (Core OS), Services Engineering (ASE), Machine Learning, and AI (AIML). Private Cloud Compute: A new frontier for AI privacy in the cloud. Apple Security Research Blog, June 2024. https://security.apple.com/blog/private-cloud-compute/, accessed 2024-08-06.

[75] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5:17–60, 1960.

[76] ETSI. Universal Mobile Telecommunications System (UMTS); 3G Security; Criteria for cryptographic algorithm design process. Technical Report ETSI TR 133 901 V3.0.0, European Telecommunications Standards Institute, 2000. URL https://www.etsi.org/deliver/etsi_tr/133900_133999/133901/03.00.00_60/tr_133901v030000p.pdf.

[77] European Telecommunications Standards Institute (ETSI). GSM Security Algorithms: A5/1, A5/2, and A5/3. Technical Report ETR 278, ETSI,

1996. URL https://www.etsi.org/deliver/etsi_etr/200_299/278/01_60/etr_278e01p.pdf.

[78] Armando Faz-Hernandez, Sam Scott, Nick Sullivan, Riad S. Wahby, and Christopher A. Wood. Hashing to elliptic curves. IETF Internet-Draft, 2023. URL https://datatracker.ietf.org/doc/draft-irtf-cfrg-hash-to-curve/.

[79] The Linux Foundation. Sigstore project. https://www.sigstore.dev/, accessed 2023-11-28.

[80] Michael J Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography Conference*, pages 303–324. Springer, 2005.

[81] Andres Freund. CVE-2024-3094: Xz backdoor. Available from MITRE, CVE-ID CVE-2024-3094, March 2024. https://www.cve.org/CVERecord?id=CVE-2024-3094.

[82] Ian Goldberg, David Wagner, and Lucky Green. The (real-time) cryptanalysis of a5/2. *Presented at the Rump session of Crypto*, 99:16, 1999.

[83] Google. Certificate Transparency. Chrome Policies. https://chromium.googlesource.com/chromium/src/+/master/net/docs/certificate-transparency.md#Chrome-Policies, accessed 2023-11-28.

[84] Google. Google Project Zero. https://googleprojectzero.blogspot.com/p/about-project-zero.html, accessed 2023-11-20.

[85] Google. Project Oak. https://github.com/project-oak/oak, accessed 2023-11-28.

[86] Google. Trillian: General transparency. https://github.com/google/trillian, accessed 2023-11-20.

[87] Google. Android Binary Transparency, April 2023. https://developers.google.com/android/binary_transparency/overview, accessed 2023-11-28.

[88] Google Project Zero. Issue 2383: XNU: NFSSVC root check bypass; use after free due to insufficient locking in upcall worker threads. `https://bugs.chromium.org/p/project-zero/issues/detail?id=2383`, November 2022. accessed 2024-04-22.

[89] Google Project Zero. Issue 2392: Windows Kernel multiple issues with subkeys of transactionally renamed registry keys. `https://bugs.chromium.org/p/project-zero/issues/detail?id=2392`, October 2022. accessed 2024-04-22.

[90] Matthew D Green and Ian Miers. Forward secure asynchronous messaging from puncturable encryption. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 305–320. IEEE, 2015.

[91] Peter Gutmann. *The Design and Verification of a Cryptographic Security Architecture*. Phd thesis, University of Auckland, New Zealand, 2000.

[92] Christoph Hagen, Christian Weinert, Christoph Sendner, Alexandra Dmitrienko, and Thomas Schneider. All the numbers are US: Large-scale abuse of contact discovery in mobile messengers. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS)*, 2021.

[93] Phillip Hallam-Baker. Private DNS. IETF Internet-Draft, 2014. URL `https://tools.ietf.org/html/draft-hallambaker-privatedns-00`.

[94] Amir Herzberg and Hemi Leibowitz. Can Johnny finally encrypt? Evaluating E2E-encryption in popular IM applications. In *Proceedings of the 6th Workshop on Socio-Technical Aspects in Security and Trust (STAST)*, pages 17–28, 2016.

[95] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 603–618, 2017.

[96] Nguyen Phong Hoang, Ivan Lin, Seyedhamed Ghavamnia, and Michalis Polychronakis. K-resolver: towards decentralizing encrypted dns resolution. *arXiv preprint arXiv:2001.08901*, 2020.

[97] Nicholas Hopper. Proving security of Tor's hidden service identity blinding protocol. Technical report, Tor project, 2013.

[98] Yuncong Hu, Kian Hooshmand, Harika Kalidhindi, Seung Jin Yang, and Raluca Ada Popa. Merkle^2: A low-latency transparency log system. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 285–303, 2021.

[99] Daniel Hugenroth, Ceren Kocaoğullar, and Alastair R Beresford. Choosing your friends: Shaping ethical use of anonymity networks. In *Cambridge International Workshop on Security Protocols*, pages 151–161. Springer, 2023.

[100] Ryan Hurst and Gary Belvin. Security through transparency. Google Security Blog, January 2017. `https://security.googleblog.com/2017/01/security-through-transparency.html`, accessed 2023-11-28.

[101] Anton A Huurdeman. *The worldwide history of telecommunications*. John Wiley & Sons, 2003.

[102] IBM. IBM X-Force. `https://www.ibm.com/x-force`, accessed 2023-11-20.

[103] Ahmad Ibrahim, Ahmad-Reza Sadeghi, Gene Tsudik, and Shaza Zeitouni. DARPA: Device attestation resilient to physical attacks. In *Proceedings of the 9th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, pages 171–182, 2016.

[104] Ahmad Ibrahim, Ahmad-Reza Sadeghi, and Shaza Zeitouni. SeED: secure non-interactive attestation for embedded devices. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSEC)*, pages 64–74, 2017.

[105] Apple Inc. Cloudattestation, 2024. URL `https://github.com/apple/security-pcc/tree/main/CloudAttestation/CloudAttestation`. (accessed 2025-04-01).

[106] Apple Inc. Thimble, 2024. URL https://github.com/apple/security-pcc/tree/main/Thimble. (accessed 2025-04-01).

[107] Apple Inc. srd_tools, 2024. URL https://github.com/apple/security-pcc/tree/main/srd_tools. (accessed 2025-04-01).

[108] Intel. Intel Software Guard Extensions (SGX), . https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html, accessed 2024-08-20.

[109] Intel. Intel Trust Domain Extensions (TDX), . https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/overview.html, accessed 2024-08-20.

[110] Intel. Intel Trust Domain Extension (Intel TDX) module. https://www.intel.com/content/www/us/en/download/738875/intel-trust-domain-extension-intel-tdx-module.html, accessed 2023-11-28.

[111] Kazuharu Itakura and K Nakamura. A public-key cryptosystem suitable for digital multisignature. *NEC Research and Development*, 71:1–8, 1983.

[112] Rob Jansen, Florian Tschorsch, Aaron Johnson, and Björn Scheuermann. The sniper attack: Anonymously deanonymizing and disabling the Tor network. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS*, 2014.

[113] Rob Jansen, Tavish Vaidya, and Micah Sherr. Point break: A study of bandwidth {Denial-of-Service} attacks against Tor. In *Proceedings of the 28th USENIX Security Symposium (USENIX Security)*, pages 1823–1840, 2019.

[114] Simon Johnson, Vinnie Scarlata, Carlos Rozas, Ernie Brickell, and Frank Mckeen. Intel software guard extensions: EPID provisioning and attestation services. *Whitepaper*, 1(1-10):119, 2016.

[115] Matt Jones. How WhatsApp reduced spam while launching end-to-end encryption. In *USENIX Enigma*, 2017.

[116] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *Proceedings of the 28th USENIX Security Symposium (USENIX Security '19)*, pages 1447–1464, 2019.

[117] Christoph Kern and Mark R Greenstreet. Formal verification in hardware design: a survey. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 1999.

[118] Ceren Kocaoğullar, Daniel Hugenroth, Martin Kleppmann, and Alastair R Beresford. Pudding: Private user discovery in anonymity networks. In *Proceedings of the 2024 IEEE Symposium on Security and Privacy (S&P)*, pages 3203–3220. IEEE, 2024.

[119] Ceren Kocaoğullar, Tina Marjanov, Ivan Petrov, Ben Laurie, Al Cutter, Christoph Kern, Alice Hutchings, and Alastair R Beresford. Confidential computing transparency. *arXiv preprint arXiv:2409.03720*, 2024.

[120] Florian Kohnhäuser, Niklas Büscher, Sebastian Gabmeyer, and Stefan Katzenbeisser. SCAPI: a scalable attestation protocol to detect software and physical attacks. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSEC)*, pages 75–86, 2017.

[121] Florian Kohnhäuser, Niklas Büscher, and Stefan Katzenbeisser. SALAD: Secure and lightweight attestation of highly dynamic and disruptive networks. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 329–342, 2018.

[122] Florian Kohnhäuser, Niklas Büscher, and Stefan Katzenbeisser. A practical attestation protocol for autonomous embedded systems. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 263–278. IEEE, 2019.

[123] Hugo Krawczyk. SIGMA: The 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE-protocols. In *Proceedings of the Annual International Cryptology Conference (CRYPTO)*, pages 400–425, 2003.

[124] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3): 382–401, July 1982.

[125] Donald C Latham. Department of defense trusted computer system evaluation criteria. *Department of Defense*, 1986.

[126] Ben Laurie. Apres-a system for anonymous presence. Technical report, 2004.

[127] Ben Laurie. Certificate Transparency. *Communications of the ACM*, 2014.

[128] Ben Laurie, Adam Langley, and Emilia Kasper. Rfc 6962: Certificate Transparency, 2013. `https://datatracker.ietf.org/doc/html/rfc6962`, accessed 2023-11-21.

[129] Ben Laurie, Adam Langley, Emilia Kasper, Eran Messeri, and Rob Stradling. Rfc 9162: Certificate Transparency version 2.0, 2021. `https://datatracker.ietf.org/doc/rfc9162/`, accessed 2024-02-22.

[130] Sean Lawlor and Kevin Lewi. Deploying key transparency at WhatsApp, April 2023. `https://engineering.fb.com/2023/04/13/security/whatsapp-key-transparency/`, accessed 2023-11-28.

[131] David Lazar and Nickolai Zeldovich. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (USENIX OSDI)*, pages 571–586, 2016.

[132] Barry Leiba and Jim Fenton. DomainKeys Identified Mail (DKIM): Using digital signatures for domain verification. In *Conference on Email and Anti-Spam*, 2007.

[133] Ian Levy and Crispin Robinson. Principles for a more informed exceptional access debate. `https://www.lawfareblog.com/principles-more-informed-exceptional-access-debate`, November 2018. `https://perma.cc/7RJK-FM32`.

[134] Mario Lins, René Mayrhofer, Michael Roland, and Alastair R Beresford. Mobile App Distribution Transparency (MADT): Design and evaluation of a system to mitigate necessary trust in mobile app distribution systems. In *Nordic Conference on Secure IT Systems*, 2023.

[135] lowRISC contributors. OpenTitan: Open source silicon root of trust RoT. `https://opentitan.org/`, accessed 2023-11-28.

[136] Joshua Lund. Technology preview: Sealed sender for Signal. Signal blog, October 2018. URL `https://perma.cc/6HC9-Y44E`. (accessed 2021-05-09).

[137] Ning Luo, Timos Antonopoulos, William R Harris, Ruzica Piskac, Eran Tromer, and Xiao Wang. Proving UNSAT in zero knowledge. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2203–2217, 2022.

[138] Verne H Mac Donald. Advanced Mobile Phone Service: The cellular concept. *The Bell System Technical Journal*, 58(1):15–41, 1979.

[139] Eugenio Marchiori, Sarah de Haas, Sergey Volnov, Ronnie Falcon, Roxanne Pinto, and Marco Zamarato. Android Private Compute Core architecture. *arXiv preprint arXiv:2209.10317*, 2022.

[140] Moxie Marlinspike. Signal protocol. *Signal Foundation*, 2016.

[141] Moxie Marlinspike. Technology preview: Private contact discovery for Signal. Signal blog, September 2017. URL `https://perma.cc/N3GF-8WD6`. (accessed 2021-09-02).

[142] Jon McLachlan, Andrew Tran, Nicholas Hopper, and Yongdae Kim. Scalable onion routing with Torsk. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, pages 590–599, 2009.

[143] Sarah Meiklejohn, Pavel Kalinnikov, Cindy S Lin, Martin Hutchinson, Gary Belvin, Mariana Raykova, and Al Cutter. Think global, act local: Gossip and client audits in verifiable data structures. *arXiv preprint arXiv:2011.04551*, 2020.

[144] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. CONIKS: Bringing key transparency to end users. In *Proceedings of the 24th USENIX Security Symposium (USENIX Security)*, pages 383–398, 2015.

[145] Jämes Ménétrey, Christian Göttel, Anum Khurshid, Marcelo Pasin, Pascal Felber, Valerio Schiavoni, and Shahid Raza. Attestation mechanisms for trusted execution environments demystified. In *IFIP International Conference on Distributed Applications and Interoperable Systems*, pages 95–113. Springer, 2022.

[146] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Proceedings of the Annual International Cryptology Conference (CRYPTO)*, pages 369–378, 1987.

[147] Microsoft. Confidential Computing on Azure, . `https://learn.microsoft.com/en-us/azure/confidential-computing/overview-azure-products`, accessed 2023-11-29.

[148] Microsoft. Microsoft Security Response Center (MSRC), . `https://msrc.microsoft.com/`, accessed 2023-11-20.

[149] Microsoft. Erroneous VeriSign-Issued Digital Certificates Pose Spoofing Hazard. Microsoft Security Bulletin MS01-017 - Critical. `https://learn.microsoft.com/en-us/security-updates/securitybulletins/2001/ms01-017`, accessed 2023-11-28.

[150] Microsoft Security Response Center. New macOS vulnerability "Migraine" could bypass system integrity protection. `https://www.microsoft.com/en-us/security/blog/2023/05/30/new-macos-vulnerability-migraine-could-bypass-system-integrity-protection/`, May 2023. accessed 2024-04-22.

[151] Microsoft Security Response Center. Patching Perforce perforations: Critical RCE vulnerability discovered in Perforce Helix Core server. `https://www.microsoft.com/en-us/security/blog/2023/12/15/`

patching-perforce-perforations-critical-rce-vulnerability-%discovered-in-perforce-helix-core-server/, December 2023. accessed 2024-04-22.

[152] George R Milne, George Pettinico, Fatima M Hajjat, and Ereni Markos. Information sensitivity typology: Mapping the degree and type of risk consumers perceive in personal data sharing. *Journal of Consumer Affairs*, 51 (1):133–161, 2017.

[153] Alan Mislove, Gaurav Oberoi, Ansley Post, Charles Reis, Peter Druschel, and Dan S Wallach. AP3: Cooperative, decentralized anonymous communication. In *ACM SIGOPS European Workshop*, pages 30–es, 2004.

[154] Prateek Mittal and Nikita Borisov. Information leaks in structured peer-to-peer anonymous communication systems. *ACM Transactions on Information and System Security (TISSEC)*, 15(1):1–28, 2012.

[155] U. Moeller, L. Cottrell, P. Palfrader, and L. Sassaman. Mixmaster Remailer Protocol – Type II. https://datatracker.ietf.org/doc/html/draft-sassaman-mixmaster-03, 2004. (accessed 2025-04-14).

[156] Nicolas Mohnblatt, Alberto Sonnino, Kobi Gurkan, and Philipp Jovanovic. Arke: Scalable and Byzantine fault tolerant privacy-preserving contact discovery. ACM SIGSAC Conference on Computer and Communications Security (CCS), 2024.

[157] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. This POODLE bites: exploiting the SSL 3.0 fallback. *Google Security Advisory*, 2014.

[158] Mozilla. Cargo Vet. https://mozilla.github.io/cargo-vet/index.html, accessed 2024-08-20.

[159] James K Mullin. A second look at Bloom filters. *Communications of the ACM*, 26(8):570–571, 1983.

[160] Steven J Murdoch and George Danezis. Low-cost traffic analysis of Tor. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 183–195. IEEE, 2005.

[161] Arjun Nambiar and Matthew Wright. Salsa: a structured approach to large-scale anonymity. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 17–26, 2006.

[162] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. CHAINIAC: Proactive Software-Update transparency via collectively signed skipchains and verified builds. In *Proceedings of the 26th USENIX Security Symposium (USENIX Security)*, pages 1271–1287, 2017.

[163] Linus Nordberg, Daniel Kahn Gillmor, and Tom Ritter. Gossiping in CT, 2018. `https://datatracker.ietf.org/doc/draft-ietf-trans-gossip/05/`, accessed 2023-11-29.

[164] OpenDNS. DNSCrypt. URL `https://perma.cc/UDW6-7HYA`. accessed 2021-01-07.

[165] Stefan Palan and Christian Schitter. Prolific.ac—a subject pool for online experiments. *Journal of behavioral and experimental finance*, 17:22–27, 2018.

[166] Andriy Panchenko, Stefan Richter, and Arne Rache. NISAN: network information service for anonymization networks. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS)*, pages 141–150, 2009.

[167] Panagiotis Papadopoulos, Antonios A Chariton, Elias Athanasopoulos, and Evangelos P Markatos. Where's Wally? How to privately discover your friends on the Internet. In *Proceedings of the Asia Conference on Computer and Communications Security (ASIACCS)*, pages 425–430, 2018.

[168] Panagiotis Papadopoulos, Antonios A Chariton, Michalis Pachilakis, and Evangelos P Markatos. OUTOPIA: private user discovery on the internet. In *Proceedings of the 15th European Workshop on Systems Security (EuroSec)*, pages 8–14, 2022.

[169] Minjung Park, Sangmi Chai, Naief G Azyabi, Liguo Lou, Joon Koh, J Park, and H Rho. The value of personal information: An exploratory study for types

of personal information and its value. *Asia Pacific Journal of Information Systems*, 28(3):154–166, 2018.

[170] Andrew Paverd, Andrew Martin, and Ian Brown. Modelling and automatically analysing privacy properties for honest-but-curious adversaries. Technical report, University of Oxford, 2014. URL `https://ajpaverd.org/publications/casper-privacy-report.pdf`.

[171] Slobodan Petrovic and Amparo Fuster-Sabater. Cryptanalysis of the A5/2 algorithm. *Cryptology EPrint Archive*, 2000.

[172] Ania M Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. The loopix anonymity system. In *Proceedings of the 26th USENIX Security Symposium (USENIX Security)*, pages 1199–1216, 2017.

[173] Crev project. CREV: Cryptographically verifiable, distributed reviews for Rust crates. `https://web.crev.dev/rust-reviews/`, accessed 2023-11-20.

[174] Signsum Project. Sigsum. `https://www.sigsum.org/`, accessed 2024-01-16.

[175] John Redford. Public report – AWS Nitro system API & security claims. NCC Group, 4 2023. `https://www.nccgroup.com/us/research-blog/public-report-aws-nitro-system-api-security-claims/`, accessed 2023-11-28.

[176] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[177] John MM Rumbold and Barbara K Pierscionek. What are data? a categorization of the data sensitivity spectrum. *Big Data Research*, 12:49–59, 2018.

[178] Scott Ruoti, Jeff Andersen, Daniel Zappala, and Kent Seamons. Why Johnny still, still can't encrypt: Evaluating the usability of a modern PGP client. *arXiv preprint arXiv:1510.08555*, 2015.

[179] Mahrud Sayrafi. Introducing DNS resolver for Tor. The Cloudflare Blog, May 2018. URL `https://perma.cc/NJY9-GDMD`. (accessed 2021-05-14).

[180] Paul Schmitt, Anne Edmundson, Allison Mankin, and Nick Feamster. Oblivious DNS: Practical privacy for DNS queries. In *Proceedings on Privacy Enhancing Technologies Symposium (PETS)*, pages 17–19, 2019.

[181] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11): 612–613, 1979.

[182] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Proceedings of the Annual International Cryptology Conference (CRYPTO)*, pages 47–53. Springer, 1985.

[183] Steve Sheng, Levi Broderick, Colleen Alison Koranda, and Jeremy J Hyland. Why johnny still can't encrypt: evaluating the usability of email encryption software. In *Proceedings of the ACM Symposium on Usable Privacy and Security (SOUPS)*, pages 3–4. ACM, 2006.

[184] Sigstore. Rekor. `https://docs.sigstore.dev/logging/overview/`, accessed 2024-01-16.

[185] Anya Skatova, Rebecca McDonald, Sinong Ma, and Carsten Maple. Unpacking privacy: Valuation of personal data protection. *PLoS ONE*, 18(5):e0284581, 2023.

[186] Emily Stark, Joe DeBlasio, and Devon O'Brien. Certificate Transparency in Google Chrome: Past, present, and future. *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 19(6):112–118, 2021.

[187] Martin Strohmeier, Matthew Smith, Vincent Lenders, and Ivan Martinovic. The real first class? inferring confidential corporate mergers and government relations from air traffic communication. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 107–121. IEEE, 2018.

[188] Synopsys Inc. The Heartbleed bug. `https://heartbleed.com/`. accessed 2023-11-28.

[189] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 526–545, 2016.

[190] Jenny Tang, Eleanor Birrell, and Ada Lerner. Replication: How well do my results generalize now? the external validity of online privacy and security surveys. In *Proceedings of the ACM Symposium on Usable Privacy and Security (SOUPS)*, pages 367–385, 2022.

[191] Bellingcat Investigation Team. Lucas villa: Cell phone data gives new insight into unsolved case of colombian protestor's killing. Bellingcat.com, 2023. URL `https://www.bellingcat.com/news/2023/05/04/lucas-villa-cell-phone-data-gives-new-insight-into-%investigation-of-colombian-protestors-killing/`.

[192] The Linux Foundation. SLSA—Supply-chain Levels for Software Artifacts, . `https://slsa.dev/`, accessed 2023-11-21.

[193] The Linux Foundation. SLSA version 1.0 Levels, . `https://slsa.dev/spec/v1.0/levels`, accessed 2023-11-21.

[194] The Tor Project. How often does tor change its paths?, 2025. URL `https://support.torproject.org/about/change-paths/`. (accessed 2025-04-13).

[195] Ken Thompson. Reflections on trusting trust. *Communications of the ACM*, 27(8):761–763, 1984.

[196] Aric Toler. Hunting the hunters: How we identified Navalny's FSB stalkers. Bellingcat.com, December 2020. URL `https://perma.cc/25JU-V4A4`. (accessed 2021-08-27).

[197] Binary Transparency. Building trust in the software supply chain, April 2023. `https://binary.transparency.dev/`, accessed 2023-11-28.

[198] Thomas Trippel, Orion Weisse, Warren Spilo, Nadia Redini, Christopher W. Fletcher, Shaz Qadeer, Sanjit A. Garg, Mathias Payer, and Mahmood Sharif. Clkscrew: Exposing the perils of security-oblivious energy management. In *Proceedings of the 26th USENIX Security Symposium (USENIX Security)*, pages 1057–1074, 2017.

[199] TrustedFirmware.org. Hafnium. `https://www.trustedfirmware.org/projects/hafnium/`, accessed 2025-01-30, .

[200] TrustedFirmware.org. Mcuboot. `https://www.trustedfirmware.org/projects/mcuboot/`, accessed 2025-01-30, .

[201] TrustedFirmware.org. Op-tee (open portable trusted execution environment). `https://www.trustedfirmware.org/projects/op-tee/`, accessed 2025-01-30, .

[202] TrustedFirmware.org. Psa crypto. `https://www.trustedfirmware.org/projects/psa-crypto/`, accessed 2025-01-30, .

[203] TrustedFirmware.org. Trusted firmware-a (tf-a). `https://www.trustedfirmware.org/projects/tf-a/`, accessed 2025-01-30, .

[204] TrustedFirmware.org. Trusted firmware-m (tf-m). `https://www.trustedfirmware.org/projects/tf-m/`, accessed 2025-01-30, .

[205] TrustedFirmware.org. Trusted services. `https://www.trustedfirmware.org/projects/trusted-services/`, accessed 2025-01-30, .

[206] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. Sok: secure messaging. In *IEEE Symposium on Security and Privacy (S&P)*, pages 232–249. IEEE, 2015.

[207] Jo Van Bulck, Orion Weisse, Marina Minkin, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Anders Fogh, Kaveh Razavi, and Yuval Yarom. Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Security Symposium (USENIX Security)*, pages 991–1008, 2018.

[208] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles (ACM SOSP)*, pages 137–152, 2015.

[209] Diana A. Vasile, Martin Kleppmann, Daniel R. Thomas, and Alastair R. Beresford. Ghost trace on the wire? Using key evidence for informed decisions. In *27th International Workshop on Security Protocols (SPW)*, volume LNCS 12287. Springer, 2019.

[210] Elham Vaziripour, Justin Wu, Mark O'Neill, Jordan Whitehead, Scott Heidbrink, Kent Seamons, and Daniel Zappala. Is that you, Alice? A usability study of the authentication ceremony of secure messaging applications. In *Proceedings of the ACM Symposium on Usable Privacy and Security (SOUPS)*, pages 29–47, 2017.

[211] Qiyan Wang, Prateek Mittal, and Nikita Borisov. In search of an anonymous and secure lookup: attacks on structured peer-to-peer anonymous communication systems. In *Proceedings of the 17th ACM conference on Computer and Communications Security (CCS)*, pages 308–318, 2010.

[212] Tao Wang, Xiang Cai, Rishab Nithyanand, Rob Johnson, and Ian Goldberg. Effective attacks and provable defenses for website fingerprinting. In *Proceedings of the 23rd USENIX Security Symposium (USENIX Security)*, pages 143–157, 2014.

[213] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[214] WhatsApp. WhatsApp key transparency overview. Technical report, Meta, August 2023.

[215] WhatsApp Ltd. WhatsApp privacy policy. URL `https://perma.cc/CC39-2CFT`. (accessed 2021-09-02).

[216] Alma Whitten and J Doug Tygar. Why Johnny can't encrypt: A usability evaluation of pgp 5.0. In *Proceedings of the USENIX Security Symposium (USENIX Security)*, volume 348, pages 169–184, 1999.

[217] Wouter Wijngaards and Glen Wiley. Confidential DNS. IETF Internet-Draft, 2015. URL `https://tools.ietf.org/html/draft-wijngaards-dnsop-confidentialdns-03`.

[218] Philipp Winter, Richard Köwer, Martin Mulazzani, Markus Huber, Sebastian Schrittwieser, Stefan Lindskog, and Edgar Weippl. Spoiled onions: Exposing malicious Tor exit relays. In *Proceedings on Privacy Enhancing Technologies Symposium (PETS)*, pages 304–331, 2014.

[219] Peter Yee. RFC 6818: Updates to the internet X.509 public key infrastructure certificate and Certificate Revocation List (CRL) profile, 2013. `https://datatracker.ietf.org/doc/html/rfc6818`, accessed 2024-08-19.

[220] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. *Advances in neural information processing systems*, 32, 2019.

[221] Philip Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.

Appendix A

Pudding security proof sketches

A.1 Unlinkability (G1)

Lemma A.1.1 *The adversary's advantage in winning Exp_{G1} from observing or performing active attacks on the network links between nodes, or from controlling mix nodes, can be reduced to the probability of breaking sender-receiver unlinkability in the underlying Loopix network, provided that at least one mix node on the path of every message is honest.*

All messages exchanged between the challenger's users and the adversary-controlled nodes are sent via Loopix [172], which provides *sender-receiver third-party unlinkability* under a threat model that includes a global passive adversary (GPA), and protection against $(n-1)$ active attacks. Analytical and empirical analysis of this unlinkability property of Loopix shows that given appropriate parameters are chosen for the network (i.e. at least 3 mix node layers, message sending rate to delay parameter ratio $\lambda/\mu \geq 2$), and assuming that there is a sufficient number of active Loopix users who are not controlled by the adversary to provide a large enough anonymity set, it is very unlikely that the adversary can identify which pairs of Loopix users are communicating by observing all traffic within the network and also controlling a portion of corrupt mix nodes [172]. If the adversary were to control all mix nodes on the path of a message between U_b and $U_{\mathcal{C}}$, she would be able to trace the message and thereby link sender and receiver; however, assuming at least one honest mix node on the path prevents this attack. Moreover, since the

Sphinx packet format [59] used by Loopix ensures integrity, any packets that are modified by the adversary are simply dropped, which prevents an active network adversary from modifying the responses from discovery nodes to a user.

Lemma A.1.2 *Assume the adversary controls up to f discovery nodes, any number of mix nodes, and the network between the nodes. When an honest user $U_{\mathcal{C}}$ performs a* Lookup*, the SURB and blinded public key she accepts at Step 6 of* Lookup *are those generated by the honest discovery nodes.*

$U_{\mathcal{C}}$ waits to receive identical responses from at least $f + 1$ distinct discovery nodes before accepting those responses. Adversary-controlled discovery nodes may send multiple responses to the same Lookup request in order to try to reach that threshold (for example, the adversary may save a SURB from one Lookup request, and use it to send two responses to a later Lookup request; alternatively, the adversary may guess the searcher's username, look up their contact information in the database, and send them Loopix messages directly). However, discovery nodes' responses are signed, $U_{\mathcal{C}}$ knows the public keys of the discovery nodes (which it obtains as part of creating their user account), and we assume the adversary cannot forge signatures from the honest discovery nodes with greater than negligible probability. $U_{\mathcal{C}}$ therefore only accepts one response per Lookup nonce per discovery node signing key. Moreover, any malicious mix nodes or active network attackers cannot modify the responses from honest discovery nodes without invalidating the signature. Thus, the adversary-generated responses are unable to reach the $f + 1$ threshold, and therefore if some SURB and blinded public key are accepted by $U_{\mathcal{C}}$, they must have been generated by honest discovery nodes.

Lemma A.1.3 *An adversary controlling up to f discovery nodes, and the mix nodes in all but one of the Loopix network layers, gains only a negligible advantage in winning Exp_{G1}.*

Let us consider all of the messages that the adversary-controlled nodes receive from the challenger. In Lookup, each adversary-controlled discovery node receives a fresh random nonce, the searched username $ID_{\mathcal{C}}$, and a SURB for replying to U_b. In ContactInit, an adversary-controlled discovery node may receive M_{init},

whose mix header is the SURB generated by discovery nodes, and whose payload is g^a, the same nonce, and a ciphertext encrypted with the blinded public key of $U_{\mathcal{C}}$. The adversary cannot learn any information from these messages that would increase its advantage in winning the game. The Sphinx packet format used by Loopix does not reveal anything about the sender of a message. The SURB for U_b is cryptographically indistinguishable from a uniform random bit string [59], and hence does not reveal its destination. Knowing that $ID_{\mathcal{C}}$ is the target of the search does not help distinguish whether U_0 or U_1 is the searcher, and the nonce and g^a are fresh random values.

By Lemma A.1.2, the blinded public key bpk_{B} that U_b uses to encrypt M_{init} is generated by the honest discovery nodes by blinding the public key in the contact information for $U_{\mathcal{C}}$, and the corresponding private key is not known to $\mathcal{A}$. In the case of a lookup of a nonexistent username, M_{init} is encrypted using the public key in Δ_{fake}, which is chosen so that no party knows the corresponding private key, as noted in §3.3.1. In either case, the ciphertext in M_{init} cannot be decrypted by the adversary except with negligible probability, and therefore the adversary does not learn the username of U_b or the codeword that U_b included in the message.

In the case where the adversary controls U_b's provider node or the first mix node along the path of M_{init}, the adversary would be able to recognise the mix header of M_{init} and link it to U_b if U_b were to place it in the network directly. However, by reflecting M_{init} off a discovery node, with the communication between U_b and the mix node going via Loopix, the adversary becomes unable to link M_{init} to any particular sender (even if the reflecting discovery node is controlled by the adversary).

Lemma A.1.4 *An adversary who controls up to f discovery nodes, and who performs an active attack on the* ContactInit *and* AddFriend *protocols, gains only a negligible advantage in winning Exp_{G1}.*

Lemma A.1.3 already shows that the adversary gains no advantage from the contents of M_{init}. The other two messages sent during the AddFriend protocol (from $U_{\mathcal{C}}$ to U_b in Step 3, and from U_b to $U_{\mathcal{C}}$ in Step 6) are not normally seen by the adversary (except in encrypted form travelling through the mix network). However, through an active attack, the adversary could inject its own SURBs into

a protocol run, and thereby $\mathcal{A}$ could trick $U_{\mathcal{C}}$ and U_b into sending these messages to $U_{\mathcal{A}}$ instead of each other.

Intercepting the messages in this way gives the adversary only negligible advantage in winning Exp_{G1}. Both messages contain only a key-blinded signature and a MAC, and the message from $U_{\mathcal{C}}$ to U_b additionally contains a Diffie-Hellman key g^b. The signature is unlinkable, i.e. the adversary cannot distinguish between two signatures produced from two separate signing keys, and two signatures produced from the same signing key but with different blinding keys [66]; since our protocol generates a different blinding key on every protocol run with overwhelming probability, this means that the signature cannot be linked to any particular user. The MAC is based on a key that is not known to $\mathcal{A}$ (see Lemma A.2.1), and therefore the adversary cannot distinguish it from a random string. And finally, g^b is a fresh random value. Therefore, even an active attack that redirects messages to the adversary does not impact unlinkability. (We examine active attacks on impersonation in Lemma A.2.1.)

Theorem A.1.5 *The adversary's advantage over a random guess in winning the unlinkability game Exp_{G1} can be reduced to the probability of breaking sender-receiver unlinkability in the underlying Loopix network.*

The adversary gains only negligible advantage from inspecting message contents or returning bad responses at the discovery nodes (Lemma A.1.3), or from performing active attacks on the authenticated key exchange protocol (Lemma A.1.4). This leaves observing or manipulating the traffic between nodes as the only remaining attack, through which the adversary can win Exp_{G1} only by breaking sender-receiver unlinkability in the underlying Loopix network (Lemma A.1.1).

A.2 Security against impersonation (G2)

Lemma A.2.1 *An adversary who controls up to f discovery nodes, and who performs an active attack on the* ContactInit *and* AddFriend *protocols, has only a negligible chance of obtaining the session key K_s in Exp_{G2}.*

By controlling at least one discovery node, the adversary learns the username ID that $U_{\mathcal{C}}$ is searching for, the nonce, the contact information $\Delta_{\mathcal{H}}$ for user $U_{\mathcal{H}}$,

one or more SURBs that allow $\mathcal{A}$ to send messages to $U_\mathcal{C}$ without knowing $U_\mathcal{C}$'s identity, the SURB for constructing M_{init}, the blinded public key bpk_B for $U_\mathcal{H}$, and the blinding key y_B for $U_\mathcal{H}$. If $U_\mathcal{C}$ picks an adversary-controlled discovery node as reflector, $\mathcal{A}$ additionally learns the Diffie-Hellman key g^a generated by $U_\mathcal{C}$ and the ciphertext in M_{init}. The adversary can use this information to inject modified versions of M_{init} and the AddFriend protocol messages (see Figure 3.2) into the network.

First consider the M_{init} message. $\mathcal{A}$ can construct her own version M'_{init} of this message, containing the nonce from $U_\mathcal{C}$. If she uses $U_\mathcal{C}$'s DH key g^a, her chance of constructing a modified ciphertext that is correctly authenticated under $K_e = \mathsf{KDF}((bpk_\mathsf{B})^a \parallel \text{“init key”})$ is negligible without knowledge of a; therefore she must either forward the unmodified ciphertext or replace g^a with a new DH key $g^{a'}$ (where $a' \in \mathbb{Z}_p$) and generate a new ciphertext. If she chooses a new DH key, $\mathcal{A}$ can include any username and/or codeword of her choice in the new ciphertext (but not the codeword in M_{init}, which by Lemma A.1.3 she cannot decrypt), and she can include a SURB that will cause $U_\mathcal{H}$'s reply to be routed back to $\mathcal{A}$.

$U_\mathcal{H}$ treats the adversary's M'_{init} like any other contact request, and decides whether to respond based on the username and/or codeword in the message. If $U_\mathcal{H}$ chooses to respond and the SURB in M'_{init} is routed to $\mathcal{A}$, the adversary learns g^b, a blinded-key signature over the Diffie-Hellman terms, and a MAC over $U_\mathcal{H}$'s identity. $\mathcal{A}$ can also let the response from $U_\mathcal{H}$ go directly to $U_\mathcal{C}$ without modification. However, if $\mathcal{A}$ replaced g^a with $g^{a'}$ in M'_{init}, and the response from $U_\mathcal{H}$ is unmodified, then $U_\mathcal{C}$ will reject the signature, and the active attack on M_{init} leads to an abort of the protocol.

The adversary could also inject an altered version of the message from $U_\mathcal{H}$ to $U_\mathcal{C}$ (AddFriend Step 3) into the network, using one of the SURBs for $U_\mathcal{C}$. However, $U_\mathcal{H}$ will only accept a message containing a signature that validates with the blinded public key bpk_B, which by Lemma A.1.2 is generated by honest discovery nodes and is not under the adversary's control. Due to the existential unforgeability of the key-blinded signature scheme, $\mathcal{A}$ has a negligible chance of generating a valid key-blinded signature on this message without knowing $U_\mathcal{H}$'s private key x_B. $\mathcal{A}$ could replay a signature and g^b Diffie-Hellman term from another protocol run, but in that case $\mathcal{A}$ would not know b, hence $\mathcal{A}$ cannot compute K_m, and hence $\mathcal{A}$ has

a negligible chance of forging the MAC on this message. In any case, the adversary cannot inject an altered version of this message that is accepted by $U_{\mathcal{C}}$.

Finally, the adversary could inject an altered version of the final message from $U_{\mathcal{C}}$ to $U_{\mathcal{H}}$ (AddFriend Step 6). If M_{init} was previously unaltered, $U_{\mathcal{H}}$ will accept this message only with a key-blinded signature that is validated with $U_{\mathcal{C}}$'s blinded public key bpk_{A} (which was either included unaltered in M_{init}, or which was obtained from $U_{\mathcal{C}}$'s username in M_{init} via Lookup, which by Lemma A.1.2 is generated by honest discovery nodes and is not under the adversary's control); by a similar argument to the previous paragraph, $\mathcal{A}$ cannot construct a modified message where both the signature and the MAC are valid.

If M_{init} was previously replaced by M'_{init}, then the last message in the AddFriend protocol must have also been generated by $\mathcal{A}$, since $U_{\mathcal{C}}$ would abort the protocol and would not generate such a message. In this case, since the username or blinded public key and the Diffie-Hellman term $g^{a'}$ in M'_{init} are under the adversary's control, $\mathcal{A}$ is able to produce a valid key-blinded signature and MAC for this message. However, the session key derived from this protocol run is $K'_s = \mathsf{KDF}(g^{a'b} \parallel \text{“session key”})$, whereas $U_{\mathcal{C}}$ computed the session key as $K_s = \mathsf{KDF}(g^{ab} \parallel \text{“session key”})$. K_s and K'_s are different with overwhelming probability, and $\mathcal{A}$ cannot compute K_s since she does not know a.

Theorem A.2.2 *The adversary cannot win the impersonation game Exp_{G2} with a significantly better chance than random guessing.*

By Lemma A.2.1, the adversary does not obtain the session key K_s. Therefore, when $U_{\mathcal{C}}$ encrypts m_b with K_s using a symmetric cipher that provides indistinguishability under chosen plaintext attack, $\mathcal{A}$ has only negligible advantage over randomly guessing b.

Appendix B

Confidential Computing Transparency user study script

B.1 Welcome screen

Thank you for participating in this study by *(anonymised for review)*. Through this study we hope to learn more about people's preferences surrounding data sharing in different transparency scenarios. You will receive a payment of £3 for participating. The study is expected to take no more than 20 minutes.

Your participation is confidential and your identity will not be stored with your data. You will not be asked to provide any personally identifying information. Your answers will be used only for research purposes and in ways that will not reveal who you are. The results will be reported in aggregate form only, and cannot be identified individually. Any information that could identify you will be removed before data is shared with other researchers or results are made public. We will keep the raw data for the duration of the study and delete it by April 2025. For reproducibility, we may release a public appendix containing some aggregate and processed data.

You must be at least 18 years old to participate. By participating in this study, you consent to the data being used for this purpose. Your participation in this research is entirely voluntary and you have the right to withdraw consent at any time by closing your browser. For any questions during or after the study, please

contact *(anonymised for review)*.

B.2 Consent

If you do not wish to participate in this study, please return your submission on Prolific by selecting the 'Stop without Completing'. Are you happy to continue? *(Note: Participant chooses between "Continue with the study" and "Stop without completing".)*

B.3 Informational scripts

Please watch the following informational video[0]. If you cannot watch it, click on the button below to see the text. You can proceed after the video finishes playing.

Low-detail variant instructional script (written and video)

In this study, you will be presented with a number of imaginary apps. These apps run on a **Confidential Computing system** that aims to protect your data in such a way that even the app developer should not be able to see your data. However, this Confidential Computing system can only be trusted if it has been designed and built securely.

Imagine this system like a safe where your data is stored. The safe has a keypad, where you can enter your PIN to lock and unlock it. In order to trust the safe with protecting your data, you should trust that the keypad does not secretly record your PIN or have a special mechanism that opens the safe.

Sometimes, the Confidential Computing system can be checked by reviewers who certify that the product is safe to use. If their certification turns out to be faulty the reviewers can be held accountable. The reviewers can be:

- experts working for the app developer company
- experts who are granted exclusive access to the code for reviewing it. They may include consultants and auditors hired by the developer company, as well as regulatory authorities.

- experts from the broader software engineering community. The system is made publicly available for reviewing by academics, independent researchers, individuals who get rewards for finding bugs, or anyone who is interested in reviewing the code.

Or the system may not be reviewed at all.

Remember! Reviewers only look at the Confidential Computing system and do not see users' personal data.

High-detail variant instructional script (written and video)

Many apps we use every day process our personal data in the cloud—a network of remote computers controlled by a cloud provider and the app developer. Apps protect data while it is stored or sent from one computer to another. However, data may become visible to the app developer and cloud provider while it's being processed, because some security protections must be disabled for processing.

Confidential Computing is a technology aimed at keeping our data private even while it is processed inside a computer. It does so by creating a special protected area within the computer, where data can be processed privately. This protected area keeps the data isolated from the rest of the computer, meaning that even the cloud provider or the app developer cannot see the data while it's being processed.

However, Confidential Computing isn't without limitations. Confidential Computing systems consist of multiple components, which need to work correctly together in order to provide privacy. If any part of the system has flaws or intentional weaknesses, it could allow the app developer, cloud provider, or another unauthorised person to see our private data.

One way to build confidence that a Confidential Computing system works correctly is to increase transparency. Transparency is an alternative to naively trusting that the system was built correctly and honestly. Transparency allows people to review how the system was built and publish their findings publicly. Though most people won't review the system themselves, a transparent process allows expert reviewers to check it instead. By making the reviewers' analysis and findings public, transparency can incentivise them to provide honest and good quality reviews.

These reviewers can be:

- experts working for the app developer
- experts directly authorised by the app developer, such as hired consultants and auditors, as well as regulatory authorities
- experts from the broader software engineering community, such as academics, independent researchers, individuals who get rewards for finding bugs, or anyone who is interested in reviewing the code

Or, the Confidential Computing system may not be formally reviewed at all.

Remember! All of these reviewers get the same level of access to the system. This means that they all see how the reviewed components are constructed in order to validate whether they were built correctly. It's important to note that reviewers cannot see, access or modify user data in any way.

B.4 Comprehension

Low-detail variant

Please answer the following questions. *(Note: The answers are multiple choice between True / False)*

Q1 A Confidential Computing system claiming to be secure is always secure, **regardless of how it was built**.

Q2 Reviewers who certify the system's safety **can be held accountable** if their certification is faulty.

Q3 Reviewers get access to the full system, **including users' personal data**.

High-detail variant

Please answer the following questions. *(Note: The answers are multiple choice)*

Q1 Why is our data more at risk while it's being processed, especially in the cloud?

(a) Because data protection is temporarily disabled while it's being processed

(b) Because cloud providers do not provide any security measures

(c) Because the data is typically not protected when it is stored or sent from one computer to another

(d) Because cloud providers and app developers always have full access to user data

Q2 What is the main purpose of Confidential Computing, and how does it protect data?

(a) To store data permanently in a secure location on a computer

(b) To protect data when it's being sent from one computer to another

(c) To keep data private during processing by creating a protected area where it can be processed privately

(d) To enhance computer processing speed by isolating certain data

Q3 How can Confidential Computing fail to protect your data?

(a) Confidential Computing never fails to protect data

(b) If a part of the Confidential Computing system fails to work correctly

(c) If there is not enough data

(d) If the user forgets their password

Q4 How can transparency help build confidence in Confidential Computing? (Select all that apply)

(a) By making the system available for review, it allows experts to check that the system was built correctly

(b) Transparency means that all users must review the system themselves to ensure it is secure

(c) Transparency can incentivise expert reviewers to provide good quality and honest reviews by making their findings public

(d) Transparency reduces the need for any review, as users can trust that the system is secure by design

Q5 - 8 Match the GIFs to reviewer types. `[GIFS AND DESCRIPTIONS OF THE FOUR TRANSPARENCY LEVELS]`

Q9 Which of the following reviewer categories gets a higher level of access to the system compared to others?

(a) Experts from the broader software engineering community, such as academics, independent researchers, individuals who get rewards for finding bugs, or anyone who is interested in reviewing the code

(b) Experts directly authorised by the app developer, such as hired consultants and auditors, as well as regulatory authorities

(c) Experts working for the app developer

(d) All of them get the same level of access

Q10 Do reviewers gain access to user data?

(a) Yes

(b) No

B.5 Instructions and the Core Study

You will now be presented with a number of **different multi-purpose virtual assistant apps** that help you with your everyday tasks. To operate efficiently, each app requires access to **different types of your data**. Each app runs on the previously described **Confidential Computing system**. The apps have been developed by a startup and are just about to be released.

For each of the scenarios, you will be asked about your comfort and confidence using the app under the specified conditions.

(Note: This instruction was followed by the core questionnaire as described in §5.1.4.)

B.6 Additional questions

Note: Participants answer each question using a slider ranging from 0 to 100.

Q1 How would you best describe your approach to purchasing or experimenting with new technology?

Q2 How likely are you to use a multi-purpose virtual assistant app (e.g. Siri, Alexa, Google Assistant)?

Q3 How concerned are you about other people getting access to your data without your consent?

Q4 Imagine your data was leaked or tampered with. How worried would you be for each of the data types listed below?

Q5 Regardless of the data type, how comfortable would you feel with the virtual assistant app being reviewed with each of the following options?

Note: The following is an open-ended question with a text box for the response.

Q11 Please explain how you assigned the scores to the review options.

B.7 Demographics

Note: We collect the following information: age, gender, level of schooling, field of work/study and country in which the participants lived the longes. We omit the full demographics questions in the interest of space.

B.8 End of survey and payment

Thank you for taking part in this study. Please click on the 'Finish' button below to be redirected back to Prolific and register your submission. (Note: *Upon clicking the 'Finish' button, the participants are redirected to Prolific platform and payment is made.*)