ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

A Formal Analysis of the SecureDrop Protocol

Master Thesis

Luca Maier

13. January 2025

Supervisors: Prof. Dr. David Basin, Felix Linker, Shannon Veitch
Information Security Group
Department of Computer Science, ETH Zürich

Abstract

SecureDrop is a widely-used whistleblowing platform that enables secure communication between anonymous whistleblowers and journalists over a messaging server, traditionally hosted on-premises by the newsroom. To align with modern operational requirements, including cloud-based deployments, a new communication protocol has been designed to ensure confidentiality, authenticity, and anonymity even when an untrusted service provider hosts the messaging server.

This thesis formalizes the design goals, threat models, and security requirements of the newly developed protocol, addressing its critical role in ensuring the security guarantees in an era of pervasive surveillance. Using the Tamarin prover, we conduct a formal analysis of the protocol to uncover vulnerabilities and propose an improved design. The resulting protocol provides robust security guarantees, including message secrecy even against adversaries with quantum computing capabilities, as well as message authenticity—both of which were not ensured by the initial protocol.

Acknowledgments

I would like to express my gratitude to my two supervisors, Felix Linker and Shannon Veitch. Felix guided me to the right resources, taught me to love Tamarin, and provided direction throughout my research. Shannon stepped in mid-project and offered support on various cryptographic challenges.

I am also thankful to Prof. Kenny Paterson for showing interest in my thesis, discussing cryptographic topics, and directing Felix and me to the right experts (Shannon!) whenever ”rolling our own crypto” appeared tempting.

I also thank Prof. David Basin for the opportunity to conduct my research under his supervision.

Lastly, I wish to acknowledge the Freedom of the Press Foundation—especially Cory and Giulio—for helping me understand the SecureDrop protocol, patiently answering my questions, and defending their design choices when I tried to challenge them.

Contents

Chapter 1

Introduction

Whistleblowing is the act of disclosing information about unethical, illegal, or dangerous practices within an organization to journalists capable of addressing the wrongdoing. It is a deliberate act driven by ethical principles and a sense of public duty, often undertaken at significant personal risk. Whistleblowing is essential for democracy as it fosters transparency, exposes corruption, and holds powerful entities accountable. The act involves four primary actors: the whistleblower, referred to as the source in this thesis, who discloses the information; journalists, who investigate and disseminate it to the public and are accredited to and work for newsrooms; newsrooms, which provide the institutional framework and support for journalistic endeavors; and the public itself, which is the ultimate beneficiary of these revelations. By shining a light on hidden wrongdoing, whistleblowing plays a critical role in safeguarding democratic values and ensuring that governments and organizations operate with integrity.

In today's interconnected and highly surveilled world, whistleblowing faces unprecedented challenges. Modern technologies enable pervasive surveillance, making it increasingly difficult for whistleblowers to disclose information anonymously over digital channels. Governments and corporations have access to advanced tools that can trace communications, compromising the safety of whistleblowers and deterring others from coming forward. Despite these risks, the digital age has also introduced opportunities for secure communication and information sharing. Platforms leveraging encryption and anonymity offer a vital lifeline to whistleblowers, providing them with tools to safely expose wrongdoing while minimizing the risks of retaliation. For example, these tools eliminate the need for risky offline meetings, allowing whistleblowers to share information securely from a distance. This duality of technological threats and solutions highlights the critical need for secure and reliable whistleblowing mechanisms in a digital world.

SecureDrop, developed by the Freedom of the Press Foundation, is a platform designed to provide whistleblowers and journalists with a secure environment for confidential communication. Its mission is to protect the anonymity and safety of sources. SecureDrop aims to offer robust security guarantees, including the confidentiality and authenticity of all communications, as well as anonymity for sources. By removing identifiable digital traces, the platform empowers whistleblowers to share sensitive information without fear of retaliation. This commitment to security reflects the core values of transparency and accountability, ensuring that whistleblowing remains a viable mechanism for exposing wrongdoing in an increasingly complex digital landscape. SecureDrop is a platform with an underlying communication protocol. For the rest of the thesis, we focus on the protocol and refer to it as ”the SecureDrop protocol”.

Establishing certainty in the security guarantees offered by whistleblowing platforms is one of the foremost challenges in protecting whistleblowers. However, these systems’ complexity and the evolving nature of threats make it difficult to trust them fully. To address this concern, a formal analysis of the underlying protocols is required. It starts by formalizing their design goals—including intended functionality, threat models, and security requirements—and then analyze their guarantees. This process identifies and remedies potential vulnerabilities before deployment, providing a sound basis for evaluating the protocol’s strength against realistic adversaries. By setting a standard of assurance, formal analysis helps maintain the integrity of disclosures and safeguards the whistleblowers themselves.

1.1 Thesis Outline

In this thesis, we present a comprehensive specification of a renewed design for the SecureDrop protocol, which is currently being developed by the Freedom of the Press Foundation. The objective of this work is to formalize the protocol’s design goals, including its functionality, threat model, and security requirements, and to rigorously analyze its security using formal methods. By doing so, we aim to provide a solid foundation for assessing and improving the protocol’s ability to meet its confidentiality, authenticity, and anonymity objectives.

- Chapter 1: Overview of related work and their limitations. Additionally, this chapter outlines the specific contributions of the thesis.
- Chapter 2: Introduce necessary background, including cryptographic primitives, secure communication mechanisms such as Tor, and the protocol verification tool Tamarin.
- Chapter 3: Define design goals of the SecureDrop protocol.
- Chapter 4: Detailed specification of the newly designed SecureDrop protocol and present found vulnerabilities.

- Chapter 5: This chapter proposes modifications to address weaknesses of the design presented in the previous chapter and contains a full specification of the proposed protocol.
- Chapter 6: Formal analysis of the proposed protocol.
- Chapter 7: Discuss the analysis and design changes, highlight any remaining limitations.
- Chapter 8: Conclude the thesis and propose directions for future research.

1.2 Contributions

In this thesis, we contribute the following:

(a) **Formalization of the Design Goals.** We provide a comprehensive formal description of the SecureDrop protocol's design objectives. This includes laying out the threat model and elaborating the security requirements that guide the protocol's development and intended use.

(b) **Formal Analysis in Tamarin.** Using the Tamarin prover, we perform a formal security analysis of the SecureDrop protocol within the symbolic model.

(c) **Design Proposal.** Based on the findings from the formal analysis, we propose a redesign of the protocol with the following improvements:

  - *Message Agreement.* The initial design lacks message authenticity guarantees. To fix this, we propose a mechanism that derives a message-specific key from the sender's secret, ensuring explicit indication of who generated the message.
  - *Post-Quantum Secrecy.* Since classical encryption primitives may not guarantee long-term secrecy in the presence of a powerful (quantum) adversary, we recommend using a post-quantum secure KEM.

(d) **Full Specification and Security Proof of the Proposed Design.** We present a complete design specification of our recommended improvements and provide a full security proof in Tamarin. This demonstration shows how the proposed modifications fulfill the necessary security guarantees against malicious servers and users.

1.3 Related Work

In exploring related work, we first look at Signal, which provides secure messaging but faces limitations in anonymity. We then examine Mix Networks, which are known for robust anonymity but are less suited to practical whistleblowing demands. Finally, we consider CoverDrop, a newly introduced whistleblowing platform that takes a different technological approach. By evaluating these systems and their shortcomings, we underscore the need for SecureDrop, which aims to address the gaps in confidentiality, authenticity, anonymity, and usability that existing solutions leave unfilled.

1.3.1 Signal

Signal [14] is a widely used encrypted messaging application designed to provide secure communication between users. It employs advanced cryptographic primitives to ensure confidentiality and authenticity [10]. At its core, Signal relies on the Signal Protocol, which combines a Diffie-Hellman key exchange, symmetric encryption, and the double ratchet algorithm to secure communications.

Signal uses end-to-end encryption, ensuring that messages are encrypted on the sender's device and decrypted only on the recipient's device, leaving even Signal's servers unable to access the message content. Signal's double ratchet algorithm ensures that past and future messages remain secure even if one encryption key is compromised [3]. Additionally, Signal provides message authenticity by employing cryptographic signatures.

While Signal is a robust tool for secure personal communication, it is not well-suited as a whistleblowing platform. Its registration process requires a valid phone number, which is often linked to an individual's identity, undermining the anonymity whistleblowers need. Furthermore, Signal leaves identifiable traces on the user's device, and its server must know both sender and recipient phone numbers to route messages—introducing unacceptable metadata risks in whistleblowing contexts.

Although Signal offers strong security for personal messaging, its design and operational model do not meet the unique requirements of whistleblowing. Whistleblowers need robust anonymity, protection from traceability, and secure ways to handle sensitive information—capabilities Signal does not provide.

1.3.2 Mix Networks

Mix networks, or mix nets, are routing protocols designed to provide anonymous communication by obfuscating the relationship between senders and recipients of messages. First introduced by David Chaum [9], mix nets have become a cornerstone of anonymous communication systems. Chaum proposed a method in which messages are encrypted in multiple layers, each layer corresponding to a specific intermediary node, or ”mix”. As the messages pass through these nodes, they are decrypted layer by layer, shuffled, and re-encrypted before being forwarded to the next node. This process effectively breaks the link between the sender and the recipient, ensuring robust anonymity.

Mix networks provide robust anonymity through a combination of encryption, routing, and message processing techniques. Each message is encrypted in multiple layers using the public keys of the mix nodes it will traverse, a method often referred to as onion encryption. This ensures that only the intended node at each hop can decrypt its corresponding layer, preserving the confidentiality of the message's path. The sender specifies a path through multiple mix nodes, where at each node the message is decrypted to reveal the next destination, shuffled with other messages to obfuscate patterns, and then forwarded. This routing process effectively breaks the link between the sender and recipient.

To further enhance anonymity, mix networks process messages in batches, ensuring that the order of incoming messages does not match the order of outgoing messages. This approach hinders adversaries' attempts to correlate messages based on timing. By combining encryption, shuffling, and relaying, mix networks provide strong anonymity guarantees for both senders and recipients. Even if a mix node is compromised, it cannot link the sender and recipient without colluding from other path nodes. These features make mix networks effective in preserving privacy and resisting traffic analysis.

While mix nets can handle asynchronous messaging, they lack specialized mechanisms for secure, ongoing exchanges between whistleblowers and journalists. In particular, a whistleblower would need to set up a server to receive messages, undermining their deniability. Moreover, the technical and operational complexity of installing and managing a mix network makes it impractical for most news organizations. Mix nets serve as a strong foundational building block for secure communication, but they are not yet a complete system.

1.3.3 CoverDrop

CoverDrop [1] is a whistleblowing platform designed to enable secure and anonymous communication between whistleblowers and journalists. It is integrated directly into existing news applications, leveraging the app's user base to provide cover traffic and enhance anonymity. All app users contribute to this cover traffic by sending encrypted, indistinguishable messages at regular intervals, ensuring that any real whistleblower communication blends seamlessly with the activity of other users.

When a whistleblower sends a message, it is encrypted in two layers: an outer layer addressed to a mix node called the CoverNode and an inner layer intended for the journalist. The CoverNode, which partially operates within a Trusted Execution Environment (TEE), decrypts the outer layer to process the message without leaving persistent traces. This ensures that the whistleblower's communication remains secure and confidential.

CoverDrop supports two-way asynchronous communication, allowing journalists and whistleblowers to exchange messages while maintaining the whistleblower's anonymity. The system is designed to be easy to use, with low latency and message delivery confirmation, which helps establish trust between the parties. To further protect whistleblowers, all news app users have encrypted CoverDrop data on their devices, providing plausible deniability even if their devices are seized.

CoverDrop offers a promising approach for secure, anonymous communications between journalists and sources by leveraging existing news apps to generate substantial cover traffic, thus providing a large anonymity set. Its usability-focused design also addresses many of the hurdles platforms like SecureDrop face. However, several challenges remain. First, CoverDrop's reliance on TEEs is a strong assumption, which may not be universally feasible against strong adversaries (e.g., state-level attackers). Second, their security model considers only a passive global network adversary, leaving active attacks unaddressed. Finally, because CoverDrop depends on integrating with an existing news application, setting it up places a tangible technical burden on the newsroom and entails non-trivial costs.

Chapter 2

Preliminaries

This chapter provides the background for understanding the concepts discussed throughout this thesis. It introduces the essential principles of private-key and public-key cryptography, explores authentication mechanisms in the public-key setting, and provides an overview of Tor and Tamarin.

2.1 Symmetric Cryptography

This section introduces the symmetric cryptographic primitives and their security notions used throughout the thesis. Unless otherwise specified, definitions are based on [17], with occasional simplifications.

2.1.1 Symmetric Encryption

Symmetric encryption (SE) schemes are cryptographic methods where the same key is used to encrypt and decrypt data. They apply mathematical algorithms to plaintext, transform it into ciphertext using the shared key, and reverse the process to decrypt the data. However, a challenge in using symmetric encryption lies in securely distributing and managing the shared key between parties. While symmetric encryption is efficient and robust, it requires a separate key exchange process to securely establish the shared secret. If the key is intercepted or compromised during exchange or storage, the entire encryption system is rendered insecure, making key distribution a critical and often complex aspect of using symmetric encryption.

Definition 2.1 (Symmetric Encryption)**.** A SE scheme is a triple of algorithms $\mathsf{SE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ such that:

- The randomized key-generation algorithm K outputs a key k.
- The encryption algorithm Enc takes a key k and a plaintext message m as input, and outputs a ciphertext c. Enc may be randomized.

- The deterministic decryption algorithm Dec takes a key k and a ciphertext c as input, and outputs a message m.

For correctness, it is required that for every key k output by KGen, and every message m, it holds that $\mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m$

An SE scheme is secure under IND-CCA (indistinguishability under chosen-ciphertext attack) if an adversary, even with the ability to request decryptions of chosen ciphertexts (except the challenge ciphertext), cannot distinguish between the encryptions of two chosen plaintexts. Intuitively, even if an attacker has significant power, such as manipulating ciphertexts and observing their decrypted outputs (except for a specific target ciphertext), they still cannot learn anything meaningful about the encrypted message.

2.1.2 Authenticated Encryption

Authenticated Encryption with Associated Data (AEAD) schemes are cryptographic constructs that provide data confidentiality and integrity in a single, unified process. They encrypt plaintext to ensure its secrecy while generating an authentication tag verifying its integrity. AEAD schemes also support "associated data", which is not encrypted but is authenticated, enabling secure contextual information (such as headers or metadata) to be included. Hence, they can be seen as an extension of SE schemes.

Definition 2.2 (AEAD Scheme)**.** An AEAD scheme is a triple of algorithms $\mathsf{AEAD} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ such that:

- The randomized key-generation algorithm KGen outputs a key k.
- The encryption algorithm Enc takes a key k, a plaintext message m, and associated data ad as input. It outputs a ciphertext c. (Depending on the scheme, Enc may be randomized or require a nonce.)
- The decryption algorithm Dec takes a key k, a ciphertext c, and associated data ad as input. It outputs either a decrypted message m or an error $\perp$ (for instance, if the ciphertext is invalid or the authenticity check fails).

For correctness, it is required that for any key $k \leftarrow_{\$} \mathsf{KGen}()$, for all messages m, and for any associated data ad, it holds that $\mathsf{Dec}(k, \mathsf{Enc}(k, m, ad), ad) = m$.

An AEAD scheme is considered AE-secure if an adversary with a decryption oracle cannot distinguish the encryption of one message from another (IND-CCA security) *and* cannot generate any valid new ciphertexts that the decryption algorithm will accept as authentic.

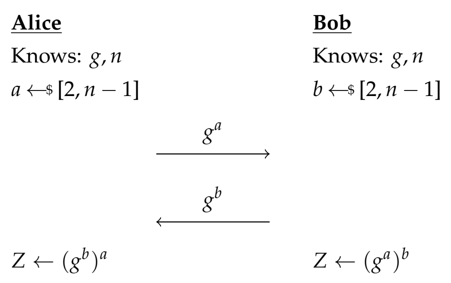

Figure 2.1: Two-party Diffie-Hellman key exchange.

2.1.3 Diffie-Hellman Key Exchange

The Diffie-Hellman key exchange (DHKE) protocol [12] is a technique that enables two parties to securely establish a shared secret key over a communication channel in the presence of a passive network adversary. Although DHKE is not itself a private-key cryptographic primitive, it is commonly paired with SE or AEAD schemes. Once established, the shared key derived through DHKE can be used to protect confidentiality and integrity in subsequent communications.

Both parties must agree on publicly known values g and n. Then, each party independently selects a random exponent x uniformly from the set $\{2, \ldots, n-1\}$, and computes the element $X = g^x$ which they then send to the other party. Upon receiving the value X' from the other party, each party computes the shared secret key Z by raising the received value X' to the power of their secret exponent x. A visual representation of the interaction is shown in Fig. 2.1.

The computational DH assumption (CDH) asserts that, given g, n, g^a, g^b for random exponents a and b, it is computationally hard for any adversary to compute the shared secret g^{ab}. Although this assumption has not been proven, there are many groups G where the CDH assumption is widely believed to hold under an adversary without quantum computing capabilities. However, in the presence of quantum computing, this assumption is known to be broken.

2.1.4 Keyed Pseudo Random Functions

A keyed pseudo random function (PRF) is a cryptographic algorithm that, given a key and a message as input, produces an output that is indistinguishable from random to anyone without the key. The PRF's deterministic nature ensures that the same input always yields the same output, while its randomness-like properties secure it against prediction or analysis.

Definition 2.3 (Keyed Pseudo Random Function)**.** Let $\mathcal{K}$ be a key space, and let $\mathcal{X}$ be the message space, and let $\mathcal{Y}$ be the function's range. A keyed pseudo random function (PRF) is a function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ with the syntax $F(k, x)$, where $k \in \mathcal{K}$ is the key and $x \in \mathcal{X}$ is the message input.

Intuitively, for a randomly chosen key k, the function $F(k, \cdot)$ behaves "like a truly random function" from $\mathcal{X}$ to $\mathcal{Y}$, as far as any unbounded adversary can tell.

One application for keyed PRFs are key derivation functions (KDF). KDFs are essential in many cryptographic protocols, mainly when multiple keys must be derived from a single shared secret or master key. By internally using keyed PRFs, KDFs ensure that each derived key appears indistinguishable from a truly random value to an outside observer. This property mitigates various key-related vulnerabilities and provides strong guarantees against cryptanalytic attacks. Furthermore, KDFs can be configured with parameters such as salt or context strings, enabling flexibility and domain separation for different use cases (e.g., deriving separate keys for encryption, and integrity).

2.2 Public-Key Cryptography

This section introduces the public-key cryptographic primitives and their security notions used throughout the thesis. Even though private-key cryptography is typically cheaper and faster than its public-key counterpart, public-key cryptography simplifies key distribution and allows secure communication without needing pre-shared secrets. Unless otherwise specified, definitions are based on [17], with occasional simplifications.

2.2.1 Public-Key Encryption

Public-key encryption (PKE) is a cryptographic method that uses a pair of keys: a public key, which is shared openly, and a private key, which is kept secret. When a sender encrypts data using the recipient's public key, only the recipient can correctly decrypt it using their private key. This asymmetric mechanism enables secure communication without the sender and recipient having previously exchanged a shared secret.

Definition 2.4 (Public-Key Encryption)**.** A PKE scheme is a triple of algorithms $\mathsf{PKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$ such that:

- The randomized key-generation algorithm KGen outputs a pair of keys (pk, sk). We refer to the first as the public key and the second as private key.

- The encryption algorithm Enc takes a public key pk and a plaintext message m as input, and outputs a ciphertext c. Enc may be randomized.

- The deterministic decryption algorithm Dec takes a private key sk and a ciphertext c as input, and outputs a message m or an error $\perp$.

For correctness, it is required that for every key pair (pk, sk) output by KGen and message m, it holds that $\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m$.

Like for SE schemes, a PKE scheme is IND-CCA secure (indistinguishability under chosen-ciphertext attack) if an adversary, even with the ability to request decryptions of chosen ciphertexts (except the challenge ciphertext), cannot distinguish between the encryptions of two chosen plaintexts.

2.2.2 Digital Signatures

Digital signatures are cryptographic mechanisms that provide integrity and non-repudiation for digital messages or documents. Using a private key, the signer generates a signature that anyone with the corresponding public key can verify. This ensures the claimed sender created the message and has not been altered.

Definition 2.5 (Digital Signature Scheme)**.** A digital signature scheme is a triple of algorithms $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vfy})$ such that:

- The randomized key-generation algorithm KGen outputs a pair of keys (pk, sk). We refer to pk as the public key and sk as the private key.
- The signing algorithm Sign takes a private key sk and a message m as input, and outputs a signature σ. Sign may be randomized.
- The deterministic verification algorithm Vfy takes a public key pk, a message m, and a signature σ as input, and outputs a bit $b \in \{0, 1\}$, where $b = 1$ indicates acceptance and $b = 0$ indicates rejection.

For correctness, it is required that for every key pair (pk, sk) output by KGen, and every message m, it holds that $\mathsf{Vfy}(pk, m, \mathsf{Sign}(sk, m)) = 1$.

The security of a digital signature scheme is typically defined by its resistance to forgery, particularly under chosen-message attacks (existential unforgeability, EUF-CMA). This means that an adversary, even after observing signatures on messages of their choice, cannot create a valid signature for any new message not explicitly signed by the legitimate key holder. This ensures the scheme is robust against impersonation and tampering, maintaining trust in the signed data.

2.2.3 Key Encapsulation Mechanisms

Key Encapsulation Mechanisms (KEMs) are cryptographic protocols designed to securely generate and exchange symmetric keys using asymmetric encryption. A sender uses the recipient's public key to encapsulate a shared secret, producing ciphertext that only the recipient, with their private key, can decapsulate to recover the shared secret.

Definition 2.6 (KEM)**.** A key encapsulation mechanism is a triple of algorithms $\mathsf{KEM} = (\mathsf{KGen}, \mathsf{Encap}, \mathsf{Decap})$ such that:

- The randomized key-generation algorithm KGen outputs a pair of keys (pk, sk). We refer to the first as the public key and the second as private key.
- The encapsulation algorithm Encap takes a public key pk as input. It outputs a ciphertext c and a key k. Encap may be randomized.
- The deterministic decapsulation algorithm Decap takes a secret key sk and a ciphertext c as input, and outputs a key k or an error $\perp$.

For correctness, it is required that for every key pair (pk, sk) output by KGen, it holds that if $\mathsf{Encap}(pk)$ outputs (c, k) then $\mathsf{Decap}(sk, c)$ outputs k. It is acceptable that this requirement does not hold with negligible probability.

The security of a KEM is defined by its ability to resist attacks aimed at recovering the shared key, even under chosen-ciphertext attacks (IND-CCA). This means an adversary cannot distinguish the shared key from a random value even with the ability to manipulate ciphertexts and observe their decapsulation results (except for the target ciphertext). This enables secure key exchange, ensuring the confidentiality of the derived symmetric keys.

KEM combiners A KEM combiner merges the outputs of multiple KEMs into a single key that remains secure if at least one constituent KEM is secure. This design hedges against uncertainty in the underlying cryptographic assumptions.

Two constructions presented in [15] are the hash-based and PRF-based approaches. Let n KEMs produce ciphertext-key pairs (c_i, k_i).

- **Hash-Based Combiner (Random Oracle Model)**: Under the assumption that H behaves as a random oracle,
$$K = H(k_1, \ldots, k_n, c_1, \ldots, c_n) \tag{2.1}$$
- **PRF-Based Combiner (Standard Model)**: Let $c = c_1 \parallel \ldots \parallel c_n$ be the concatenation of all ciphertexts. If F is a secure PRF,
$$K = F(k_1, c) \oplus \ldots \oplus F(k_n, c) \tag{2.2}$$
is secure without relying on the random oracle model.

Thus, the hash-based construction offers simplicity at the cost of the random oracle assumption, while the PRF-based construction provides a standard-model guarantee.

2.2.4 Hybrid Encryption

Hybrid encryption combines the strengths of public-key and symmetric cryptography: a public-key scheme securely establishes a one-time session key, while a symmetric scheme efficiently encrypts the actual data. This approach allows for fast encryption and decryption, typical of symmetric algorithms, while benefiting from the key distribution advantages of public-key techniques. A generic construction can be built by pairing a KEM to derive the session key with a SE scheme to protect the plaintext—also known as data encapsulation mechanism (DEM).

Construction 2.7 (KEM/DEM paradigm)**.** Let $\mathsf{KEM} = (\mathsf{KEM.KGen}, \mathsf{KEM.Encap}, \mathsf{KEM.Decap})$ be a KEM, and let $\mathsf{SE} = (\mathsf{SE.KGen}, \mathsf{SE.Enc}, \mathsf{SE.Dec})$ be a SE scheme. Construct a public-key encryption scheme $\mathsf{PKE} = (\mathsf{PKE.KGen}, \mathsf{PKE.Enc}, \mathsf{PKE.Dec})$ as follows:

- $\mathsf{PKE.KGen}$: run $\mathsf{KEM.KGen}()$ and use the public and private keys (pk, sk) that are output.
- $\mathsf{PKE.Enc}$: on input a public key pk and a message m do:
  (a) Compute $(c, k) \leftarrow_\$ \mathsf{KEM.Encap}(pk)$
  (b) Compute $c' \leftarrow_\$ \mathsf{SE.Enc}(k, m)$
  (c) Output the ciphertext $c \parallel c'$
- $\mathsf{PKE.Dec}$: on input a private key sk and a ciphertext $c \parallel c'$ do:
  (a) Compute $k \leftarrow \mathsf{KEM.Decap}(sk, c)$
  (b) Output the message $m \leftarrow \mathsf{SE.Dec}(k, c')$

Theorem 2.8 (IND-CCA of KEM/DEM construction [11])**.** If KEM is an IND-CCA secure KEM and SE is an IND-CCA secure symmetric encryption scheme (or AE-secure AEAD scheme), then PKE as in Construction 2.7 is an IND-CCA secure PKE scheme.

2.3 Authentication in Public-Key Encryption

Having discussed AEAD schemes in the private-key setting and the building blocks that provide confidentiality in the public-key setting, we now focus on the need for authentication in a hybrid encryption context. While hybrid encryption efficiently combines symmetric encryption with a KEM, it still lacks explicit mechanisms to verify the authenticity of the communicating parties. This section explores how to achieve confidentiality and authentication in hybrid setups, highlighting protocols and constructions that ensure secure key exchange and message integrity.

This section explores the constructions, definitions, and security properties of authentication in the hybrid setting as formalized in RFC 9180 [5] and its security analysis [2]. It focuses on how these tools are applied to establish trust and prevent impersonation or forgery, in a public-key setting.

2.3.1 Authenticated Key Encapsulation Mechanisms

An authenticated key encapsulation mechanism (AKEM) enhances a standard KEM by adding sender authentication to the key encapsulation process. Unlike a standard KEM, which focuses solely on confidentiality, an AKEM incorporates the sender's private key to generate the encapsulated key and shared secret. The recipient uses both their private key and the sender's public key to verify the origin of the encapsulated key, ensuring both confidentiality and sender authenticity in a single operation. This integration simplifies system design and is particularly useful in secure communication scenarios where message authenticity needs to be achieved.

Definition 2.9 (AKEM)**.** An authenticated key encapsulation mechanism AKEM consists of three algorithms:

- The randomized key-generation algorithm KGen outputs a pair of keys (pk, sk). We refer to the first as the public key and the second as private key.
- AuthEncap takes a (sender) private key sk and a (receiver) public key pk as input, and outputs a ciphertext c and a shared secret $K \in \mathcal{K}$.
- Deterministic AuthDecap takes a (receiver) private key sk, a (sender) public key pk as input, and a ciphertext c, and outputs a shared secret $K \in \mathcal{K}$.

For correctness, it is required that for all keys (sk_1, pk_1) and (sk_2, pk_2) output by KGen, it holds that $\mathsf{AuthDecap}(sk_2, pk_1, c) = K$ if $(c, K) \leftarrow\$ \mathsf{AuthEncap}(sk_1, pk_2)$. It is acceptable that this requirement does not hold with negligible probability. Note, (sk_1, pk_1) is the sender's key pair and (sk_2, pk_2) the receiver's key pair.

Security Notions Privacy ensures that ciphertexts encapsulated for an honest receiver do not leak any information about the shared secret or private keys. Two primary notions of privacy are considered: *Outsider-CCA* and *Insider-CCA* security. In *Outsider-CCA* security, the adversary is an external party with no knowledge of private keys and is limited to interacting with encapsulation and decapsulation oracles. On the other hand, in *Insider-CCA* security, the adversary is granted the additional capability of choosing the sender's private key, making it significantly stronger. Both definitions require the adversary to be unable to distinguish the shared secret from random under chosen-ciphertext attacks ensure robust privacy against both external and internal threats.

Authenticity ensures that an adversary cannot forge a valid ciphertext to impersonate an honest sender. In the *Outsider-Auth* setting, the adversary, without access to private keys, must not be able to create a ciphertext that an honest receiver would accept as originating from a legitimate sender. *Insider-Auth*, where the adversary knows the recipient's private key, is not defined, as it cannot be achieved in the following constructions.

DH-AKEM Next, we present the AKEM construction DH-AKEM [5]. In DH-AKEM, the sender generates an ephemeral key pair and uses the recipient's static public key and their ephemeral private key to compute a shared secret via the Diffie-Hellman key-exchange. The sender's static private key is also used to compute another shared secret with the recipient's static public key to authenticate the sender. These shared secrets are combined to derive the final encapsulated key and the corresponding ciphertext, which includes the sender's ephemeral public key. Upon receiving the ciphertext, the recipient uses their static private key and the sender's public keys (both static and ephemeral) to reconstruct the shared secrets and derive the same encapsulated key. This mechanism ensures that the recipient can verify the sender's identity and establish a confidential shared key in a single, integrated operation. This construction provides *Outsider-CCA*, *Insider-CCA*, and *Outsider-Auth* [2].

Construction 2.10 (DH-AKEM [5])**.** DH-AKEM is constructed from a group $\mathbb{G} = \{G, g, q\}$ and a key derivation function $\mathsf{KDF} : \{0,1\}^* \to \mathcal{K}$. This AKEM consists of the following algorithms:

```
$\mathsf{KGen}()$
    $x \leftarrow\$ \mathbb{Z}_q$
    return $(x, g^x)$

$\mathsf{AuthEncap}(skS, pkR)$
    $(skE, pkE) \leftarrow\$ \mathsf{KGen}()$
    $context \leftarrow pkE \,\|\, pkR \,\|\, g^{skS}$
    $dh \leftarrow pkR^{skE} \,\|\, pkR^{skS}$
    $K \leftarrow \mathsf{KDF}(dh \,\|\, context)$
    return $(pkE, K)$

$\mathsf{AuthDecap}(skR, pkS, pkE)$
    $context \leftarrow pkE \,\|\, g^{skR} \,\|\, pkS$
    $dh \leftarrow pkE^{skR} \,\|\, pkS^{skR}$
    $K \leftarrow \mathsf{KDF}(dh \,\|\, context)$
    return $K$
```

2.3.2 Authenticated Public-Key Encryption

Authenticated Public-Key Encryption (APKE) combines encryption and authentication in a single cryptographic scheme, ensuring both the confidentiality of messages and the authenticity of their origin. Unlike standard public key encryption, APKE allows the receiver to verify that the ciphertext was generated by a specific sender, preventing impersonation attacks. This is achieved by integrating the sender's private key into the encryption process, ensuring that only a legitimate sender can produce valid ciphertexts. APKE is particularly useful in scenarios requiring both secure communication and sender authentication, such as secure messaging and key exchange protocols.

Definition 2.11 (APKE)**.** A authenticated public key encryption scheme APKE consists of the following three algorithms:

- KGen outputs a key pair (sk, pk).
- AuthEnc takes a (sender) private key sk, a (receiver) public key pk, a message m, associated data ad, a bitstring $info$ as input, and outputs a ciphertext c.
- Deterministic AuthDec takes a (receiver) private key sk, a (sender) public key pk, a ciphertext c, associated data ad and a bitstring $info$ as input, and outputs a message m.

For correctness, it is required that for all keys (sk_1, pk_1) and (sk_2, pk_2) output by KGen, and all messages m, ad, and $info$, it holds that

$$\mathsf{AuthDec}(\mathsf{sk}_2, \mathsf{pk}_1, (\mathsf{AuthEnc}(\mathsf{sk}_1, \mathsf{pk}_2, \mathsf{m}, \mathsf{ad}, \mathsf{info})), \mathsf{ad}, \mathsf{info}) = \mathsf{m}.$$

Note, (sk_1, pk_1) is the sender's key pair and (sk_2, pk_2) the receiver's key pair.

Security Notions Privacy is defined similarly to that in AKEMs, encompassing both *Outsider-CCA* and *Insider-CCA* security notions. In the *Outsider-CCA* setting, an adversary without access to private keys must be unable to distinguish between a ciphertext encrypting a specific message and a random ciphertext, even with access to encryption and decryption oracles. In the stronger *Insider-CCA* setting, the adversary may have access to the sender's private key but must still fail to gain any information about the encrypted message beyond what is trivially deducible.

Authenticity ensures that an adversary cannot forge a valid ciphertext, associated data, and related metadata tuple for a receiver. In the *Outsider-Auth* setting, the adversary must not be able to create such a forgery for any honest receiver private key, assuming the sender's public key is valid. The stronger *Insider-Auth* setting considers adversaries who may possess a potentially compromised or malicious receiver private key. Even under these conditions, the scheme guarantees that only ciphertexts from the legitimate sender can be verified as authentic by the receiver.

Generic Construction APKE schemes can be constructed generically using an AKEM, a keyed PRF F, and an AEAD scheme. The AKEM establishes a shared secret key K, which, together with optional metadata $info$, serves as the input to F. In turn, F outputs an AEAD key k and an initialization vector $nonce$, from which the AEAD's nonces are subsequently derived.

In this construction, the AKEM handles establishing a confidential and authenticated shared key, while the AEAD scheme ensures secure encryption and integrity of the message with associated data. By leveraging this modular approach, APKE inherits the components' confidentiality and authenticity guarantees, resulting in a robust and flexible encryption mechanism suitable for various applications.

Construction 2.12 ($\mathsf{APKE}[\mathsf{AKEM}, \mathsf{F}, \mathsf{AEAD}]$)**.** Let F be a keyed PRF, AKEM an AKEM, and AEAD an AEAD scheme. An APKE scheme $\mathsf{APKE}[\mathsf{AKEM}, \mathsf{F}, \mathsf{AEAD}]$ can be constructed as follows (with $\mathsf{APKE.KGen} = \mathsf{AKEM.KGen}$):

```
$\underline{\mathsf{AuthEnc}(sk, pk, m, ad, info)}$
$(c_1, K) \leftarrow\$ \mathsf{AuthEncap}(sk, pk)$
$k \parallel nonce \leftarrow \mathsf{F}(K, info)$
$c_2 \leftarrow\$ \mathsf{AEAD.Enc}(k, m, ad, nonce)$
return $(c_1, c_2)$

$\underline{\mathsf{AuthDec}(sk, pk, (c_1, c_2), ad, info)}$
$K \leftarrow\$ \mathsf{AuthDecap}(sk, pk, c_1)$
$k \parallel nonce \leftarrow \mathsf{F}(K, info)$
$m \leftarrow \mathsf{AEAD.Dec}(k, c_2, ad, nonce)$
return $m$
```

The following security guarantees are shown in [2]:

Theorem 2.13 (Outsider-CCA)**.** If AKEM is Outsider-CCA secure, F is a secure PRF, and AEAD is AE-secure, then APKE as constructed in Construction 2.12 is Outsider-CCA secure.

Theorem 2.14 (Insider-CCA)**.** If AKEM is Insider-CCA secure, F is a secure PRF, and AEAD is AE-secure, then APKE as constructed in Construction 2.12 is Insider-CCA secure.

Theorem 2.15 (Outsider-Auth)**.** If AKEM is Outsider-CCA and Outsider-Auth secure, F is a secure PRF, and AEAD is AE-secure, then APKE as constructed in Construction 2.12 is Outsider-Auth secure.

However, Construction 2.12 does not achieve *Insider-Auth* security (i.e., bad receiver key) because the KEM's ciphertext and AEAD's ciphertext are not linked. That is, an adversary can reuse a KEM ciphertext and the AEAD encryption is exchanged with any other message.

Specific Construction Because DH-AKEM is an Outsider-CCA, Insider-CCA, and Outsider-Auth secure AKEM, one can construct an APKE scheme from DH-AKEM achieving the same three APKE security notions. This follows from the preceding three theorems.

Construction 2.16 ($\mathsf{HPKE}_{\mathsf{auth}}$)**.** RFC 9180 [5] presents the APKE scheme $\mathsf{HPKE}_{\mathsf{auth}} = \mathsf{APKE}[\mathsf{DH\text{-}AKEM}, \mathsf{F}, \mathsf{AEAD}]$ by applying the AKEM/DEM Construction 2.12 to DH-AKEM. The details regarding F, AEAD, and the KDF in DH-AKEM can be found in the RFC.

```
$\underline{\mathsf{KGen_S}()}$                                   $\underline{\mathsf{KGen_R}()}$
$(sk_{\mathsf{dh}}, pk_{\mathsf{dh}}) \leftarrow\!\$\ \mathsf{DH\text{-}AKEM.KGen}()$     $(sk_{\mathsf{dh}}, pk_{\mathsf{dh}}) \leftarrow\!\$\ \mathsf{DH\text{-}AKEM.KGen}()$
return $(sk_{\mathsf{dh}}, pk_{\mathsf{dh}})$                   return $(sk_{\mathsf{dh}}, pk_{\mathsf{dh}})$

$\underline{\mathsf{AuthEnc}(skS_{\mathsf{dh}}, pkR_{\mathsf{dh}}, m, aad, info)}$
$(c_1, k_1) \leftarrow\!\$\ \mathsf{DH\text{-}AKEM.AuthEncap}(skS_{\mathsf{dh}}, pkR_{\mathsf{dh}})$
$k \parallel nonce \leftarrow \mathsf{F}(k_1, info)$
$c \leftarrow\!\$\ \mathsf{AEAD.Enc}(k, m, aad, nonce)$
return $(c_1, c)$

$\underline{\mathsf{AuthDec}(skR_{\mathsf{dh}}, pkS_{\mathsf{dh}}, (c_1, c), aad, info)}$
$k_1 \leftarrow \mathsf{DH\text{-}AKEM.AuthDecap}(skR_{\mathsf{dh}}, pkS_{\mathsf{dh}}, c_1)$
$k \parallel nonce \leftarrow \mathsf{F}(k_1, info)$
$m \leftarrow \mathsf{AEAD.Dec}(k, c, aad, nonce)$
return $m$
```

For a secure keyed PRF F and an AE-secure AEAD scheme, the construction $\mathsf{HPKE_{auth}}$ is an APKE scheme guaranteeing *Outsider-CCA*, *Insider-CCA*, and *Outsider-Auth*.

2.4 Tor

This section introduces Tor's working principles. For more comprehensive information, we recommend consulting Tor's official documentation [25] and whitepaper [13].

Overview The Tor network is designed to enable anonymous communication and protect the confidentiality of users' activities online. The core concept of Tor is onion routing, where data is encrypted in multiple layers, akin to the layers of an onion. This encrypted data is then relayed through a series of volunteer-operated servers called relays, each peeling away a single layer of encryption, which obscures the data's origin and destination. This ensures anonymity by making it extremely difficult to trace the data back to the user. Additionally, Tor supports hidden services, allowing users to host websites and services that are accessible only within the Tor network, thus further enhancing privacy. These features collectively ensure that users can communicate and browse the Internet without exposing their identity or compromising their confidentiality.

Onion Routing When a user wants to send data, their Tor client software randomly selects a path through multiple nodes. The data is then encapsulated in layers of encryption, with each layer corresponding to one of the chosen nodes. The outermost layer is decrypted by the first node, which reveals the next node in the path, and the partially decrypted data is passed along. This process continues, with each node peeling off a single encryption layer until the data reaches its final destination in its original form. Importantly, each node only knows the identity of the previous and next node, but not the entire path, making it nearly impossible for any single node to trace the data back to its source. This multi-layered encryption scheme is what gives onion routing its name and ensures that the user's identity and data remain confidential and anonymous throughout the process, as long as not all nodes on the path collude.

Onion Services Onion services, also known as hidden services, operate within the Tor network, providing a means to host websites and services with enhanced privacy and security. Unlike regular websites, onion services use a special `.onion` address, which contains a string encoding of the service's public key. This ensures that the address is unique and directly tied to the service's identity. When a user wants to access an onion service, their Tor client establishes a path through the network by connecting to a series of relays. The client and the service agree on introduction points, which act as intermediaries. The client sends an encrypted challenge using the public key derived from the `.onion` address. Only the hidden service with the corresponding private key can decrypt this challenge and respond correctly, creating a direct communication channel to the legitimate site. For additional details and an example with SecureDrop as an onion service, please refer to [24].

Security Guarantees The threat model of Tor encompasses various scenarios where users' privacy and anonymity within the network may be compromised. Tor is designed to protect against adversaries attempting to monitor or interfere with a user's online activities, often by observing traffic patterns or attempting to deanonymize users. Threats include passive adversaries, such as government agencies or ISPs, who attempt traffic analysis to correlate users with their online actions. Active attackers, including malicious Tor nodes or compromised exit nodes, pose risks by attempting to intercept or modify traffic. There are also concerns about deanonymization attacks that exploit weaknesses in Tor's design or implementation, potentially revealing a user's actual IP address or compromising their identity. Furthermore, threats may arise from malware or phishing attacks targeting Tor users, aiming to compromise their devices or obtain sensitive information.

Tor provides several key security guarantees that are essential to maintaining anonymity and protecting user privacy within its network. Firstly, its multi-layered encryption ensures that no single relay knows the source and destination of the data, preserving user anonymity. Secondly, Tor's onion services employ end-to-end encryption for user traffic, protecting data from being intercepted or modified while in transit between the client and the destination server. Additionally, Tor's onion routing protocol ensures that even the entry and exit nodes in the network cannot link a user's IP address with their online activities, enhancing privacy. Moreover, Tor regularly rotates circuits and uses mechanisms like padding and traffic shaping to defend against traffic analysis attacks aimed at inferring user behavior. These combined measures provide robust security guarantees, shielding users from surveillance, censorship, and other online freedom and privacy threats.

2.5 Tamarin Prover

Tamarin Tamarin is an automated theorem prover that verifies protocols in the symbolic model [7, 6]. It aims to generate a formal proof by starting with the program's specification (i.e., a "theory") and systematically construct a proof for given properties. Properties of protocols are expressed as logical statements over traces, where traces represent sequences of events that capture the execution of the protocol. Tamarin supports two key verification modes: satisfiability, which checks whether a trace exists that satisfies a given property, and validity, which ensures the property holds for all possible traces. Proofs can be constructed automatically, leveraging Tamarin's heuristics to explore the state space, or interactively through a graphical user interface (GUI), allowing users to guide the proof process manually. This dual approach makes Tamarin a versatile tool for analyzing complex protocols. For more detailed information, we refer the reader to the official manual [26].

Terms, Functions, Equations In Tamarin, terms, functions, and equations are used to model the algebraic structures and operations that underpin protocols. Terms and functions represent cryptographic messages, and equational theories model how functions relate to each other. Tamarin provides several built-in equational theories (e.g., `symmetric-encryption`, `signing`, `diffie-hellman`, etc.). As an example, `asymmetric-encryption` defines the function symbols `aenc/2`, `adec/2`, and `pk/1` (the suffix `/2` denotes the number of input arguments), which are related by the equation `adec(aenc(m,pk(k)),k) = m`. Additionally, one can define custom function symbols and equational theories. In Tamarin, the type of a term is expressed using a prefix:

- `~x` denotes a fresh term
- `$x` denotes a global name (i.e., a fixed, publicly known constant)
- `#x` denotes a temporal variable

Facts, Rules, Actions Facts are composed of terms and model the state of a protocol. Linear facts can be consumed only once, while persistent facts can be consumed unlimited times. In Tamarin, persistent facts are prefixed with the symbol `!`, distinguishing them from linear facts. Rules describe the transitions between states in a protocol. Each rule has a left-hand side (LHS), called the premise, which specifies the facts that are consumed, and a right-hand side (RHS), called the conclusion, which specifies the facts that are produced as a result. Additionally, rules can be labeled using action facts. The rules capture how a protocol behaves and evolves.

Network Messages and Adversary Tamarin's adversary represents a global, active network adversary (i.e., Dolev-Yao). In other words, the untrusted network is the adversary who can read, modify, insert, drop, reorder, and replay messages. The linear facts `Out(m)` and `In(m)` model the network.

To illustrate these concepts, let us consider the following example modeling how a sender sends an asymmetrically encrypted message to a known receiver `$R`:

```
builtins: asymmetric-encryption

// Registering a public-key
rule Register_pk:
    [ Fr(~ltk) ] --> [ !Ltk($A, ~ltk), !Pk($A, pk(~ltk)) ]

// The public key is sent to the network
rule Get_pk:
    [ !Pk($A, pubkey) ] --> [ Out(pubkey) ]

// The action fact LtkReveal($A) exists in a trace
// if and only if $A's secret key has been revealed
rule Reveal_ltk:
    [ !Ltk($A, ltk) ] --[ LtkReveal($A) ]-> [ Out(ltk) ]

rule Sender:
    let c = aenc(~msg, pkR) in
    [ Fr(~msg), !Pk($R, pkR) ] --[ Sent($R, ~msg) ]-> [ Out(c) ]

rule Receiver:
    let msg = adec(c, ~ltk) in
    [ In(c), !Ltk($R, ~ltk) ] --[ Received($R, msg) ]-> []
```

The sender generates a fresh message `~msg` using the special fact `Fr(~msg)` and encrypt the message using `$R`'s public key `pkR`. The receiver can decrypt the ciphertext with the secret key `~ltk` that matches the public key used to encrypt the message (see Fig. 2.2). Note, the "placeholder" term `$A` in the `Register_pk` rule is replaced with the term `$R` from the LHS of the `Sender` rule.

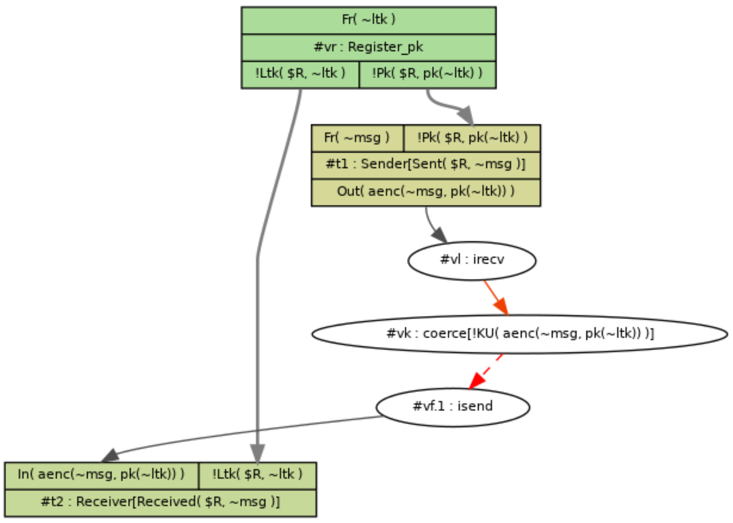
Fr(~ltk)
#vr : Register_pk
!Ltk($R, ~ltk)
!Pk($R, pk(~ltk))
Fr(~msg)
!Pk($R, pk(~ltk))
#t1 : Sender[Sent($R, ~msg)]
Out(aenc(~msg, pk(~ltk)))
#vl : irecv
#vk : coerce[!KU(aenc(~msg, pk(~ltk)))]
#vf.1 : isend
In(aenc(~msg, pk(~ltk)))
!Ltk($R, ~ltk)
#t2 : Receiver[Received($R, ~msg)]

Figure 2.2: Trace in Tamarin's GUI: Sender successfully sends an encrypted message to `$R`.

Lemmas Lemmas are used to specify desired properties which then can be proven by Tamarin. Satisfiability lemmas are generally used to sanity check the model by proving the existence of a trace representing the intended behavior. The trace in Fig. 2.2 was generated using the following lemma:

```
lemma Executability:
exists-trace
"Ex #t1 #t2 r msg.
        Sent(r, msg) @ #t1
      & Received(r, msg) @ #t2"
```

Recall, `#x` denotes a temporal variable. Therefore the property reads as follows: There exist timepoints `#t1` and `#t2`, and terms `r` and `msg`, such that there exists a trace with an action fact `Sent(r, msg)` at timepoint `#t1` and the action fact an action fact `Received(r, msg)` at timepoint `#t2`.

Tamarin tries to construct a trace that produces the required action facts to prove such a satisfiability lemma. In `Executability`, Tamarin starts with the `Sender` and `Receiver` rules to produce the required action facts. Both rules consume facts which must be produced by some other rule or are in the initial state (e.g., `Fr` can be produced "out of nowhere"). The input to `Receiver` is the output of `Sender`, and to derive the corresponding keys, the facts produced by a single `Register_pk` rule are consumed. This creates a valid trace with all facts being produced from the initial state.

On the other hand, validity lemmas are used to verify security properties (i.e., something holds for all traces). For the example protocol, we can check that the adversary can only know the sent message if the receiver's key was compromised. In other words, an action fact `LtkReveal($R)` must exist. This is captured by the lemma:

```
lemma Secrecy:
all-traces
"All #t1 #t2 r msg.
        Sent(r, msg) @ #t1
      & Received(r, msg) @ #t2
  ==>  (not Ex #x. K(msg) @ #x)
     | (Ex #x. LtkReveal(r) @ #x)"
```

The fact `K(msg)` models that the adversary knows the message `msg`. Lemmas of the form "`==> secure | X`" capture a threat model in which `secure` holds unless `X` happens (e.g., some key leakage). Tamarin negates the formula, transforming it into a satisfiability lemma, and checks whether a trace exists. The validity lemma is true if and only if Tamarin doesn't find a trace satisfying the corresponding satisfiability lemma. In our example, there is no trace without a `LtkReveal($R)` action fact where `msg` is leaked. We can see in Fig. 2.3 how the adversary can derive the receiver's secret key and how this introduces the action fact into the trace.

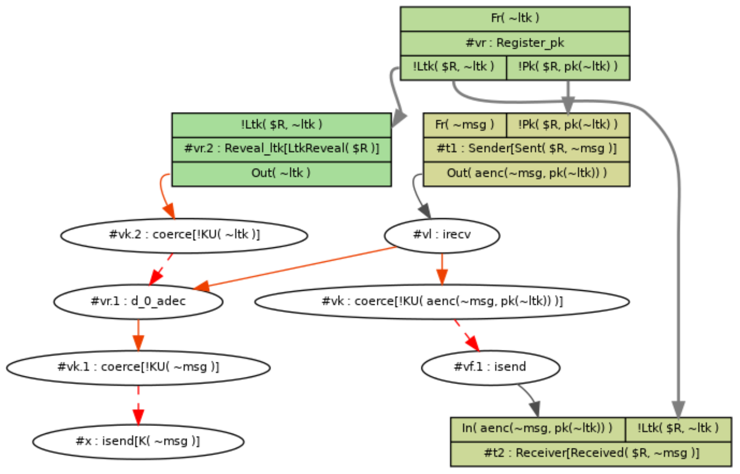
Fr(~ltk)
#vr : Register_pk
!Ltk($R, ~ltk)
!Pk($R, pk(~ltk))
!Ltk($R, ~ltk)
#vr.2 : Reveal_ltk[LtkReveal($R)]
Out(~ltk)
Fr(~msg)
!Pk($R, pk(~ltk))
#t1 : Sender[Sent($R, ~msg)]
Out(aenc(~msg, pk(~ltk)))
#vk.2 : coerce[!KU(~ltk)]
#vl : irecv
#vr.1 : d_0_adec
#vk : coerce[!KU(aenc(~msg, pk(~ltk)))]
#vk.1 : coerce[!KU(~msg)]
#vf.1 : isend
#x : isend[K(~msg)]
In(aenc(~msg, pk(~ltk)))
!Ltk($R, ~ltk)
#t2 : Receiver[Received($R, ~msg)]

Figure 2.3: The adversary knowing the encrypted message implies the existence of the action fact `LtkReveal($R)` in the trace (displayed is one such trace).

Validity lemmas can be transformed into auxiliary lemmas by annotating them with `[reuse]`. From this point onward, all the following lemmas assume this lemma to hold. Similarly, a source lemma (annotated with `[sources]`) is used for verification of all non-source lemmas. It is in general used to ″help″ Tamarin constructing the derivation rules when it fails to resolve where a fact must have come from.

Restrictions Restrictions can be used to restrict the set of traces that Tamarin takes into account when proving properties. In general, they come with action facts that need to be added to the protocol for the restrictions to have an impact. We present two common restrictions:

```
restriction Unique:
    "All x #i #j. Unique(x) @#i & Unique(x) @#j ==> #i = #j"
```

If the action fact `Unique(x)` appears twice with the same argument, the trace will only be considered if the two time points are identical (i.e., the facts can be ″merged″).

```
restriction Equality:
    "All x y #i. Eq(x,y) @#i ==> x = y"
```

The `Equality` restriction can be used in protocol rules to check equality of two inputs. That is, traces will only be considered if whenever an action fact `Eq(x,y)` occurs, `x = y` must hold.

Oracles Tamarin's heuristic is an optimization tool that guides the proof search process by prioritizing promising paths in the protocol's state space. Its goal is to reduce the computational effort required to verify properties by focusing on rules and constraints most relevant to the desired proof or counterexample. Users can influence the heuristic through oracles, which are independent programs used to rank proof methods, helping prioritize which method the prover should attempt first. The oracle takes a list of proof methods, provided in a default order by Tamarin, and outputs a reordered list of indices to guide the proof search. Users can specify the oracle program's path relative to the protocol file or working directory, as a default or through command-line options. If the oracle does not return a valid ranking, Tamarin defaults to the first method in the original order.

Chapter 3

Design Goals

This section outlines the rationale behind the design of the updated SecureDrop protocol and the guiding principles shaping its development. We begin by contextualizing why a new version of SecureDrop is necessary, even though an operational implementation already exists. Next, we explore the problem domain, identifying the key actors involved and the specific functionalities they require to fulfill their roles. Finally, we define the threat model, detailing the potential adversaries and risks, and establish the security requirements guaranteed by the updated SecureDrop design.

3.1 Historical Background

Chapter 1 discussed the need for a secure whistleblowing platform to enable confidential communication between anonymous sources and newsrooms via a messaging server. SecureDrop, developed by the Freedom of the Press Foundation (FPF), is actively used by newsrooms worldwide [1, 23]. However, FPF's interactions with newsrooms reveal that SecureDrop's architecture requires significant updates to align with evolving newsroom practices.

Traditionally, SecureDrop assumed its messaging server would be operated in a trusted environment, typically within a newsroom's on-premises technical infrastructure. However, with the widespread migration of modern newsrooms to cloud-based infrastructure, many no longer have the physical resources or technical expertise to manage such applications locally. As a result, SecureDrop must adapt to allow third-party service providers (e.g., AWS) to host its messaging server. Since the existing SecureDrop protocol was not designed for this context, it cannot be used without significant changes. For the rest of the thesis, the "SecureDrop protocol" refers to the renewed design which addresses the evolving newsroom practices.

3.2 Domain Model

This section provides a structured representation of the problem domain, introducing the various actors involved in the SecureDrop protocol. We also discuss the role of the FPF, which develops the protocol, though it is not itself an actor within the system.

Freedom of the Press Foundation The *Freedom of Press Foundation* designs and maintains the SecureDrop protocol, ensuring its security and usability for newsrooms. While the FPF offers technical assistance for setup and operation, its involvement is not required; newsrooms can independently deploy and manage SecureDrop based on their specific needs.

Newsrooms and Journalists *Newsrooms* represent the organizational entity responsible for operating a SecureDrop instance and overseeing its interactions with other actors. They manage the enrollment and removal of *journalists*, who act as trusted representatives within the newsroom's SecureDrop instance. Journalists are individuals affiliated with a newsroom, communicating securely with sources as part of their investigative roles. This relationship defines the newsroom as the central authority in the system, with journalists operating under its governance.

Sources *Sources* are individuals seeking to communicate securely with newsrooms while remaining fully anonymous. They use SecureDrop to share sensitive information or documents while aiming to keep their identity hidden, ensuring confidentiality and protection from potential surveillance or retaliation.

Messaging Server The *messaging server* enables asynchronous communication by securely storing and delivering messages between *clients* (i.e., sources, and journalists). The newsroom sets up and manages the server, ensuring control over its operation and integration into the newsroom's technical infrastructure.

3.3 Functional Requirements

Setup Newsrooms can choose to set up a SecureDrop instance, that is the messaging server, either on-premises or hosted by an untrusted third-party service provider (e.g., AWS). Newsrooms can enroll or remove trusted journalists. Enrollment involves verifying journalists over a secure channel (e.g., an in-person meeting) and equipping them with credentials to authenticate with the messaging server as journalists.

Sources, on the other hand, do not require any setup. By choosing a passphrase, a persistent anonymous identity is established within SecureDrop. This passphrase is the only persistent state associated with a source—there is no client software, no local storage, only a passphrase that can be memorized.

Messaging Sources can submit messages to a newsroom, and the submission is accessible to all journalists currently enrolled in that newsroom. The Freedom of the Press Foundation expects this to include only a small group of journalists per newsroom. Journalists can read a source's submission and reply, with the response sent to the initiating source and all other enrolled journalists. Sources can resume communication using the passphrase, allowing them to maintain continuity in the conversation. From a protocol perspective, a source's follow-up messages are treated as indistinguishable from initial submissions, though journalists can still recognize them as part of an ongoing conversation.

3.4 Threat Model

Clients communicate with the messaging server over Tor via an onion service. This SSL-like channel ensures end-to-end encryption, authentication, bidirectional anonymity, and session unlinkability. Through this secure channel, protocol participants can retrieve authentic newsroom key material, which acts as the root of trust. We consider the following two adversarial settings.

Malicious User Setting This adversary has the same access level as any honest client, and neither special network knowledge nor privileged access to the server.

Malicious Server Setting This adversary represents a strong adversary with full control over the messaging server (e.g., state-level adversary). This model also captures an active network adversary on the final hop between any protocol participant and the server, capable of actively engaging in the protocol.

We assume both adversaries may eventually gain access to quantum computing, enabling them to break cryptographic primitives used in the protocol, such as learning Diffie-Hellman keys or other encrypted secrets. In such a scenario, all honest clients and servers will halt the protocol immediately. This notion addresses the risk of "harvest now, decrypt later" quantum attacks.

3.5 Security Requirements

The three central security guarantees for SecureDrop concern message secrecy, message authenticity, and anonymity.

Secrecy SecureDrop is designed to ensure *message secrecy* and *forward secrecy*. Message secrecy means that an attacker cannot access their exchanged messages as long as neither participant's key material is exposed. Forward secrecy ensures that even if a long-term key is compromised, it will not affect the secrecy of messages sent or received in the past.

In the malicious server setting, SecureDrop ensures message secrecy for all messages and forward secrecy for messages sent to journalists. Forward secrecy cannot be achieved for messages received by sources because it is a functional requirement for sources not to have state. Message secrecy and forward secrecy are provided for all messages in the malicious user setting.

In our security analysis, we define secrecy lemmas that address message secrecy and forward secrecy and consider the impact of key compromises. A detailed explanation of these lemmas requires a deeper understanding of SecureDrop's key material, which we will cover later.

Authenticity *Message agreement* ensures that the sender and recipient have a mutual understanding of the exchanged messages. In SecureDrop, in both adversarial settings, the recipient can verify the message's integrity and authenticate the sender. This prevents an adversary from altering the message or impersonating a legitimate client.

In addition, *replay protection* is provided for messages sent to journalists. That is, messages intercepted and resent by an adversary cannot be accepted as a valid new messages by journalists.

Anonymity We define SecureDrop's anonymity guarantees in terms of accepted leakage.

A *malicious user* must not learn more than the number of journalists enrolled in the system and the messages for which it is the intended receiver. Furthermore, with access to a quantum computer, it can learn the number of ciphertexts stored at the server over time for all participants.

A *malicious server*, which controls the message delivery infrastructure, learns more than a malicious user. Additionally to the information learned by a malicious user, it can observe the number of ciphertexts per participant and identify this set of ciphertexts. Additionally, it learns the timing of each submission. In contrast to the malicious user, an active malicious server can infer which participant a given ciphertext is sent to without quantum computing capabilities.

Chapter 4

Initial SecureDrop Specification & Vulnerabilities

This chapter outlines the initial design of the SecureDrop protocol, as presented in [22], which serves as the successor to the existing SecureDrop protocol requiring a messaging server hosted on the newsroom's trusted infrastructure. We cover the setup of key material, the messaging protocol, and analyze design considerations made by the FPF. In Section 4.4, we discuss the protocol's vulnerabilities which will be addressed in our proposed specification (see Chapter 5).

4.1 Setup

This section describes the setup process of newsrooms, journalists, and sources. It additionally covers how key material is generated and distributed. An overview of the key material used in the protocol can be found in Table 4.1. For keys, we use the notation $X_{A,B}$, where X represents the key owner ($X \in \{NR, J, S\}$), A represents the key's usage ($A \in \{\mathsf{sig}, \mathsf{fetch}, \mathsf{dh}\}$), and is prefixed with an "e" if the key is ephemeral. B indicates whether the component is private or public. For Diffie-Hellman keys x, the public component is represented by the exponentiation $g^x = \mathsf{DH}(g, x)$.

Newsroom Setup The newsroom configures a messaging server as an onion service (see Section 3.3), generating a public-private keypair $(NR_{\mathsf{sig,pk}}, NR_{\mathsf{sig,sk}})$. This keypair authenticates enrolled journalists, with the public key $NR_{\mathsf{sig,pk}}$ made available to clients over Tor. Since SecureDrop operates as an onion service, the authenticity of the newsroom key $NR_{\mathsf{sig,pk}}$ is ensured if the SecureDrop instance's onion address is obtained from a trusted source, such as the newsroom's official webpage (see Section 2.4). Along with the newsroom keys, clients also retrieve authentic group variables, such as the generator g, which are used in Diffie-Hellman key exchanges.

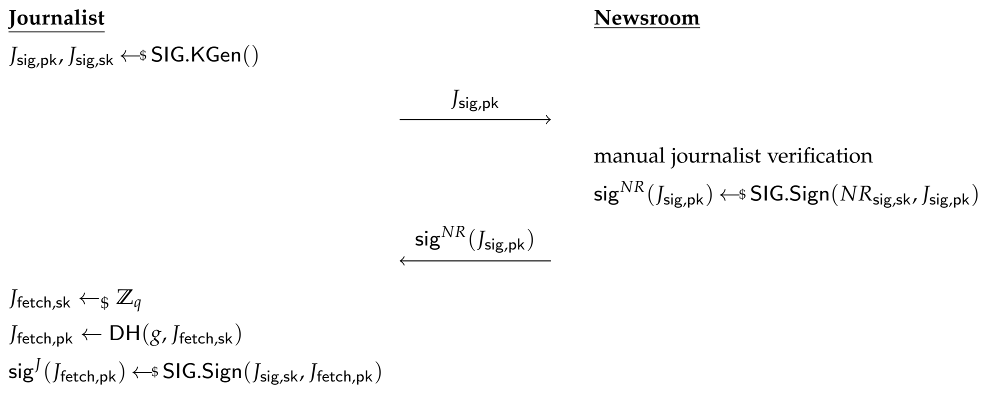
Journalist
Newsroom
$J_{\text{sig,pk}}, J_{\text{sig,sk}} \leftarrow\!\!\$\ \mathsf{SIG.KGen}()$
$J_{\text{sig,pk}}$
manual journalist verification
$\text{sig}^{NR}(J_{\text{sig,pk}}) \leftarrow\!\!\$\ \mathsf{SIG.Sign}(NR_{\text{sig,sk}}, J_{\text{sig,pk}})$
$\text{sig}^{NR}(J_{\text{sig,pk}})$
$J_{\text{fetch,sk}} \leftarrow\!\!\$\ \mathbb{Z}_q$
$J_{\text{fetch,pk}} \leftarrow \mathsf{DH}(g, J_{\text{fetch,sk}})$
$\text{sig}^{J}(J_{\text{fetch,pk}}) \leftarrow\!\!\$\ \mathsf{SIG.Sign}(J_{\text{sig,sk}}, J_{\text{fetch,pk}})$

Figure 4.1: Journalist enrollment into the newsroom over a secure channel.

Journalist Enrollment To enroll a journalist in a newsroom's SecureDrop instance, the journalist first generates a public-private keypair $(J_{\text{sig,pk}}, J_{\text{sig,sk}})$ and sends the public key $J_{\text{sig,pk}}$ to the newsroom. The newsroom manually verifies the journalist's identity through secure, offline means such as in-person meetings and signs $J_{\text{sig,pk}}$ with its signing key $NR_{\text{sig,sk}}$. The resulting signature, $\text{sig}^{NR}(J_{\text{sig,pk}})$, is returned to the journalist. All communication during this process occurs over a secure channel (e.g., offline). The journalist then generates a random fetching key $J_{\text{fetch,sk}}$ and signs its public component $J_{\text{fetch,pk}}$ with the private key $J_{\text{sig,sk}}$. Both public keys, along with their signatures, are stored on the messaging server. The interaction is shown in Fig. 4.1.

Ephemeral Keys Replenishment Journalists periodically generate ephemeral messaging keys $J_{\text{edh,sk}}$ and sign the corresponding public key $J_{\text{edh,pk}} = \mathsf{DH}(g, J_{\text{edh,sk}})$ with their signing key $J_{\text{sig,sk}}$. The public key $J_{\text{edh,pk}}$ and its signature are then stored on the messaging server. SecureDrop clients can request these ephemeral keys from the server, which are used to generate a shared key pair between the client and the journalist.

Sources Sources don't require any setup. Instead, they choose a *passphrase* which is used as input to a key derivation function KDF to derive the source's key material:

$$S_{\text{dh,sk}} \parallel S_{\text{fetch,sk}} = \mathsf{KDF}(\mathit{passphrase})$$

4.2 Messaging Protocol

High-Level Perspective When a source connects to a newsroom's SecureDrop instance, it fetches the signed key material required for secure communication with journalists and verifies its authenticity. The source then creates

Key	Type	Usage
$(NR_{\text{sig,sk}}, NR_{\text{sig,pk}})$	PPK	Signing
$(J_{\text{sig,sk}}, J_{\text{sig,pk}})$	PPK	Signing
$(J_{\text{fetch,sk}}, J_{\text{fetch,pk}})$	DH	Fetching
$(J_{\text{edh,sk}}, J_{\text{edh,pk}})$	Ephemeral DH	Messaging
$(S_{\text{fetch,sk}}, S_{\text{fetch,pk}})$	DH	Fetching
$(S_{\text{dh,sk}}, S_{\text{dh,pk}})$	DH	Messaging

Table 4.1: Key material in the SecureDrop protocol. (PPK := Public-Private Keypair)

a submission, which is broadcast to the newsroom and includes key material enabling journalists to reply. On a protocol level, the source's message is encrypted for each journalist enrolled in the newsroom, meaning the source sends a distinct ciphertext for every journalist.

Since SecureDrop clients are not constantly online, they periodically fetch messages from the server. To facilitate this, the server issues a challenge that only the intended recipient of the ciphertext can solve. Upon solving the challenge, the server provides the ciphertext to the client.

When a journalist receives a source's submission, it uses the included source identity keys to encrypt a response. To ensure a shared understanding of messages, the journalist retrieves from the server the key material of all the other journalists within the newsroom and uploads a copy of the response encrypted with each peer's key. From a protocol perspective, when sources fetch and receive messages, their response is treated as a new submission.

For the remainder of this section, we focus on the direct communication between one source and one journalist, as the broadcasting component is underspecified in the design provided by the Freedom of the Press Foundation. This represents a newsroom where a single journalist is enrolled.

Key Fetching SecureDrop clients retrieve the journalists' key material from the server and verify its authenticity, ensuring the journalist is enrolled in the correct newsroom for communication. The server returns the key material for each enrolled journalist, which is then used to send messages securely. When a client requests the key material for a specific journalist J, the server provides J's long-term keys and a randomly selected key from J's set of ephemeral messaging keys, which the journalist generated. The server then discards the ephemeral key and its signature to prevent reuse. The interaction is displayed in Fig. 4.2.

Source Submission After obtaining authentic key material for a journalist, the source symmetrically encrypts its message and key material, $S_{\text{dh,pk}}$, and $S_{\text{fetch,pk}}$, using a shared secret derived from a non-interactive DHKE between an ephemeral key share x and the journalist's public ephemeral

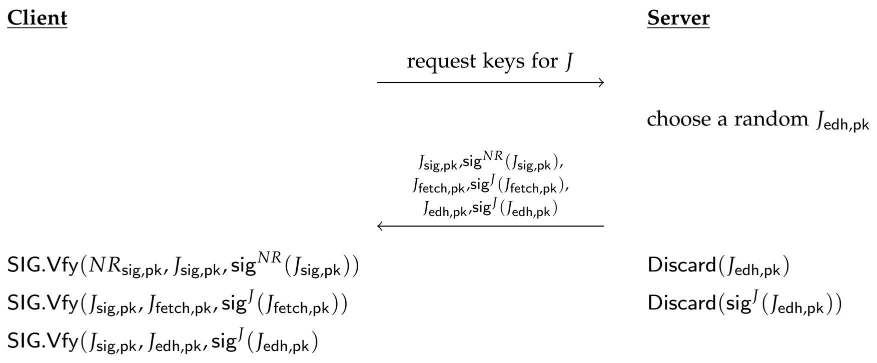
Client
Server
request keys for J
choose a random $J_{\mathsf{edh,pk}}$
$J_{\mathsf{sig,pk}}, \mathsf{sig}^{NR}(J_{\mathsf{sig,pk}})$, $J_{\mathsf{fetch,pk}}, \mathsf{sig}^{J}(J_{\mathsf{fetch,pk}})$, $J_{\mathsf{edh,pk}}, \mathsf{sig}^{J}(J_{\mathsf{edh,pk}})$
$\mathsf{SIG.Vfy}(NR_{\mathsf{sig,pk}}, J_{\mathsf{sig,pk}}, \mathsf{sig}^{NR}(J_{\mathsf{sig,pk}}))$
$\mathsf{SIG.Vfy}(J_{\mathsf{sig,pk}}, J_{\mathsf{fetch,pk}}, \mathsf{sig}^{J}(J_{\mathsf{fetch,pk}}))$
$\mathsf{SIG.Vfy}(J_{\mathsf{sig,pk}}, J_{\mathsf{edh,pk}}, \mathsf{sig}^{J}(J_{\mathsf{edh,pk}})$
$\mathsf{Discard}(J_{\mathsf{edh,pk}})$
$\mathsf{Discard}(\mathsf{sig}^{J}(J_{\mathsf{edh,pk}}))$

Figure 4.2: Client fetches key material and verifies the keys' authenticity.

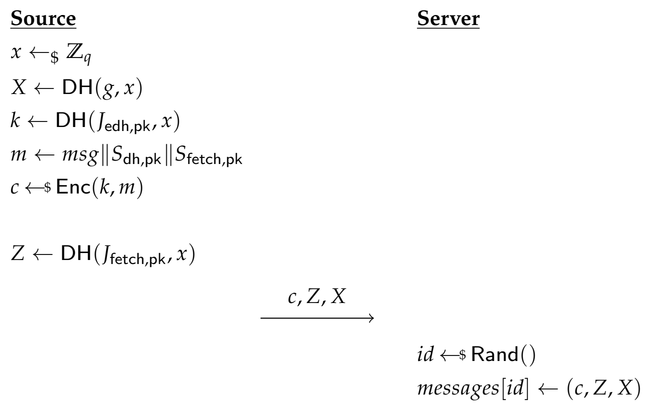
Source
Server
$x \leftarrow_{\$} \mathbb{Z}_q$
$X \leftarrow \mathsf{DH}(g, x)$
$k \leftarrow \mathsf{DH}(J_{\mathsf{edh,pk}}, x)$
$m \leftarrow msg \| S_{\mathsf{dh,pk}} \| S_{\mathsf{fetch,pk}}$
$c \leftarrow_{\$} \mathsf{Enc}(k, m)$
$Z \leftarrow \mathsf{DH}(J_{\mathsf{fetch,pk}}, x)$
c, Z, X
$id \leftarrow_{\$} \mathsf{Rand}()$
$messages[id] \leftarrow (c, Z, X)$

Figure 4.3: Source submits a message msg intended for journalist J to the server.

message encryption key $J_{\mathsf{edh,pk}}$. Because a source's only persistent state is its passphrase, it must fetch the journalist's key material as described in the preceding paragraph.

Alongside the ciphertext, the source sends $X = \mathsf{DH}(g, x)$ and a DH share, $Z = \mathsf{DH}(J_{\mathsf{fetch,pk}}, x)$, to the server. The server then generates a random message ID and stores the received information. The server will use the value Z to perform a three-party DHKE with the journalist and encrypt the newly generated message ID.

Journalist Message Fetching Journalists periodically request their messages from the server. To verify that a client is the intended recipient of a message, the server generates a challenge that only the client possessing the corresponding fetching key can solve. The interaction between a server and a journalist knowing $J_{\mathsf{fetch,sk}}$ is demonstrated in Fig. 4.4.

For every stored tuple (id, c_i, Z_i, X_i), the server creates a challenge by encrypting id with $\mathsf{DH}(Z_i, y)$ for a fresh value y. Note, if the sender intended to

```
Server                                                    Journalist
for $i = 0, \dots, \mathrm{len}(messages)$ :
  $id_i \leftarrow messages.\mathrm{keys}()[i]$
  $c_i, Z_i, X_i \leftarrow messages[id_i]$
  $y \leftarrow_\$ \mathbb{Z}_q$
  $k_i \leftarrow \mathsf{DH}(Z_i, y)$
  $Q_i \leftarrow \mathsf{DH}(X_i, y)$
  $cid_i \leftarrow_\$ \mathsf{Enc}(k_i, id_i)$
for $i = \mathrm{len}(messages), \dots, n$ :
  $Q_i \leftarrow_\$ \mathrm{Rand}()$
  $cid_i \leftarrow_\$ \mathrm{Rand}()$
                     $\xrightarrow{Q_{0,\dots,n}, cid_{0,\dots,n}}$
                                                          $ids \leftarrow \{\}$
                                                          for $i = 0, \dots, n$ :
                                                            $k_i \leftarrow \mathsf{DH}(Q_i, J_{\mathsf{fetch,sk}})$
                                                            $id_i \leftarrow \mathsf{Dec}(k_i, cid_i)$
                                                            if $id_i \neq \bot$ :
                                                              $ids \leftarrow ids \cup \{id_i\}$
                                                          return $ids$
```

Figure 4.4: The journalist requests the encrypted IDs and tries to decrypt them. This procedure returns all message IDs to the journalist that it successfully decrypted.

send the ciphertext to the journalist knowing $J_{\mathsf{fetch,sk}}$ then $Z_i = \mathsf{DH}(J_{\mathsf{fetch,pk}}, x_i)$ and $X_i = \mathsf{DH}(g, x_i)$ for a random x_i. Then, the server symmetrically encrypts id using $\mathsf{DH}(Z_i, y) = \mathsf{DH}(\mathsf{DH}(J_{\mathsf{fetch,pk}}, x_i), y) = \mathsf{DH}(J_{\mathsf{fetch,pk}}, x_i y)$, and returns the ciphertext cid_i and the public key $Q_i = \mathsf{DH}(X_i, y) = \mathsf{DH}(g, x_i y)$. To successfully decrypt the ciphertext, the recipient of the challenge needs to know $J_{\mathsf{fetch,sk}}$. The set of successfully decrypted IDs is returned to the journalist which then can query the server with the ID to retrieve the ciphertext.

To enhance privacy, the server encrypts the message IDs with a fresh key for each fetch request and includes random IDs alongside the real ones. As a result, the journalist receives a constant-sized list of encrypted message IDs that appears different with each fetch. This approach ensures that the response to a fetch request does not reveal any information about the server's internal state, such as the number of messages submitted since the last fetch request.

Journalist Read To read a particular message, the journalist queries the server using the previously decrypted message ID. The server responds with the ciphertext and the public key X associated with that message ID. The journalist then attempts to decrypt the ciphertext using all of its stored ephemeral message encryption keys $J_{\mathsf{edh,sk}}$.

The message and source's key material is returned to the journalist. The key material will be used in case the journalist intends to send a response to

Server **Journalist**

$$\xleftarrow{id}$$

$c, Z, X \leftarrow messages[id]$

$$\xrightarrow{c, X}$$

```
for all stored ephemeral keys $J_{\mathsf{edh,sk}}$ :
  $k \leftarrow \mathsf{DH}(X, J_{\mathsf{edh,sk}})$
  $m \leftarrow \mathsf{Dec}(k, c)$
  if $m \neq \perp$ :
    Parse $m$ as $msg \| S_{\mathsf{dh,pk}} \| S_{\mathsf{fetch,pk}}$
    $\mathsf{Discard}(J_{\mathsf{edh,sk}})$
    return $(msg, S_{\mathsf{dh,pk}}, S_{\mathsf{fetch,pk}})$
```

Figure 4.5: Journalist reads a source's message by querying the server with the decrypted message ID (assuming the ID exists).

the source. Furthermore, once an ephemeral key $J_{\mathsf{edh,sk}}$ is successfully used to decrypt a ciphertext, it is discarded. Before the server exhausts its supply of ephemeral keys, the journalist must rerun the key replenishment procedure. A visual representation of the interaction is shown in Fig. 4.5.

Journalist Reply The journalist replies to the source identified by its public keys $S_{\mathsf{dh,pk}}$ and $S_{\mathsf{fetch,pk}}$ from the preceding submission. The journalist generates a fresh ephemeral secret key share x to establish a shared secret key $k = \mathsf{DH}(S_{\mathsf{dh,pk}}, x)$, which is used to encrypt the response.

The ciphertext and a three-party DH key share $Z = \mathsf{DH}(S_{\mathsf{fetch,pk}}, x)$ and the journalist's ephemeral message encryption public key $X = \mathsf{DH}(g, x)$, are sent to the server. The server then generates a random message ID and stores the received information. The server will use the value Z to perform a three-party DHKE with the source and encrypt the newly generated message ID. This interaction closely mirrors a source's submission, as shown in Fig. 4.3, with the key difference being that the journalist retrieves the source's keys from the submission rather than from a key-fetching query (because sources are not registered).

To follow the group-messaging model, the journalist's response is shared with all other journalists. To achieve this, journalists fetch keys of their peers from the messaging server and forward the response to the others. This process uses the peer's messaging key $J_{\mathsf{edh,pk}}$ instead of $S_{\mathsf{dh,pk}}$ and the fetching key $J_{\mathsf{fetch,pk}}$ instead of $S_{\mathsf{fetch,pk}}$ (see Fig. 4.6).

Source Message Fetching The source's fetch operation mirrors the journalist's. The server processes the request similarly for sources and journalists, as it cannot distinguish between the two.

$$\begin{array}{lcl}
\underline{\textbf{Server}} & & \underline{\textbf{Journalist}} \\
 & & x \leftarrow_{\$} \mathbb{Z}_q \\
 & & X \leftarrow \mathsf{DH}(g, x) \\
 & & k \leftarrow \mathsf{DH}(S_{\mathsf{dh,pk}}, x) \\
 & & c \leftarrow_{\$} \mathsf{Enc}(k, msg) \\
 & & Z \leftarrow \mathsf{DH}(S_{\mathsf{fetch,pk}}, x) \\
 & \xleftarrow{c, Z, X} & \\
id \leftarrow_{\$} \mathsf{Rand}() & & \\
messages[id] \leftarrow (c, Z, X) & &
\end{array}$$

Figure 4.6: Journalist replies to a message.

$$\begin{array}{lcl}
\underline{\textbf{Source}} & & \underline{\textbf{Server}} \\
 & \xrightarrow{id} & \\
 & & c, Z, X \leftarrow messages[id] \\
 & \xleftarrow{c, X} & \\
k \leftarrow \mathsf{DH}(X, S_{\mathsf{dh,sk}}) & & \\
msg \leftarrow \mathsf{Dec}(k, c) & & \\
\text{return } msg & &
\end{array}$$

Figure 4.7: Source reads a message by querying the server with its decrypted message ID.

Source Read To read a particular message, the source queries the server using the previously decrypted message ID. The server responds with the ciphertext and the public key X associated with that message ID. The source then decrypts the ciphertext using its long-term messaging key $S_{\mathsf{dh,sk}}$. The interaction is shown in Fig. 4.7.

Source Reply The source replies by creating a new submission.

4.3 Design Considerations

Protocol Symmetry To maintain symmetry in the protocol, the journalist does not store the source's messages or key material after fetching. This ensures that the interaction appears identical from the server's perspective, regardless of whether a source or a journalist initiates it. By having journalists fetch the other journalists' keys, the system mimics the behavior of a stateless source, which does not retain keys or message history. While journalists could technically store each other's keys, this design choice ensures that the protocol remains uniform and the server's role is consistent for both parties. This symmetry helps prevent the server from distinguishing whether a message originates from a journalist or a source, enhancing the system's privacy.

Message Fetching Message fetching is implemented to conceal the server's internal state and protect the communication patterns from malicious users. Without this mechanism, a trial decryption approach or a bulletin board-like construction could reveal all communication patterns to an outsider, potentially compromising the anonymity of the source and journalist. By requiring message fetching, the protocol shields the communication process from malicious entities attempting to infer the relationships or timing of exchanges, thus enhancing overall security and privacy.

4.4 Vulnerabilities

In this section, we present the vulnerabilities identified in this design. These vulnerabilities were discovered while attempting to formally prove the protocol's security guarantees. The following chapter introduces an updated protocol design that addresses these issues.

Key Leakage (Key Tags) Both in the protocol specification and the prototype implementation, none of the encrypted nor signed messages were accompanied with message tags. By attaching a message tag, the sending and receiving party can agree on a specific message type.

In SecureDrop, missing message tags can allow an active network adversary to replace messages and elicit a key reveal. Before sources submit a message, the server sends $J_{\mathsf{fetch,pk}}$ and $J_{\mathsf{edh,pk}}$ including their signatures to the source. If the active network adversary removes $J_{\mathsf{fetch,pk}}$ and instead sends $J_{\mathsf{edh,pk}}$ a second time (alongside its signature), the verification of the signature succeeds and the source outputs $Z = \mathsf{DH}(J_{\mathsf{edh,pk}}, x) = k$. That is, it leaks the symmetric encryption key which the network adversary can use to decrypt the submission ciphertext c.

PQ Secrecy In a post-quantum scenario, any protocol relying on classical DHKE becomes insecure because quantum computers can efficiently break the discrete logarithm problem that underpins its security. This means a quantum-capable attacker can derive the shared secret from public information, effectively decrypting all previously captured traffic (if stored) and rendering the confidentiality the key exchange provides obsolete. In other words, once quantum computing resources are available, the core assumption that ensures the secrecy of DH-generated keys no longer holds, making the entire protocol vulnerable.

Message Agreement Currently, the SecureDrop protocol does not guarantee message agreement because there is no mechanism for sender authentication. Consequently, an adversary can impersonate any source S by including S's publicly known key in a message. Journalists attempt to identify sending sources by checking the key $S_{\mathsf{dh,pk}}$ embedded in the message, assuming that only someone holding the corresponding secret key $S_{\mathsf{dh,sk}}$ could have generated it. They then encrypt their response using $S_{\mathsf{dh,pk}}$ and $S_{\mathsf{fetch,pk}}$, believing only a party possessing both $S_{\mathsf{dh,sk}}$ and $S_{\mathsf{fetch,sk}}$ can decrypt it. However, without a reliable mechanism to ensure the rightful owner of a key is the only one who can generate a message, these assumptions are unfounded. Similarly, journalists do not include their own identity, so the source does not know who is responding.

Establishing message agreement guarantees that if a recipient believes a sender transmitted a message, the sender indeed intended to communicate with that recipient, and they agree upon their view of the message (i.e., this view includes the plaintext of the message itself). This property is a typical formalization of message authenticity [20]. Its absence is a fundamental flaw in the design and could lead to impersonation attacks, undermining the protocol's security and trustworthiness.

Chapter 5

Proposed SecureDrop Specification

In this chapter we address the vulnerabilities found in the initial SecureDrop design (see Section 4.4) and recommend changes to the specification that address the issues. The result is a complete protocol specification of the proposed design.

5.1 Authentication in the Public-Key Setting

To achieve sender authentication, we replace the message encryption using non-interactive DHKE with an authenticated public-key encryption (APKE) scheme. This change ensures that messages are both confidential and authentic. In contrast to standard public-key encryption, which only provides confidentiality, APKE allows the recipient to verify that a message was encrypted by a party holding a valid secret key corresponding to a known public key, which is used to identify the sender. This property protects against impersonation attacks and ensures that both the sender and the recipient have cryptographic proof of the message's authenticity.

Specifically, we adopt the APKE scheme $\mathsf{HPKE}_{\mathsf{auth}}$ (see Construction 2.16), which relies on DH-AKEM for its underlying AKEM. In this construction, both sources and journalists are identified by their long-term DH public keys. For sources, we use the existing public key $S_{\mathsf{dh,pk}}$. For journalists, we introduce a new long-term DH key pair $(J_{\mathsf{dh,sk}}, J_{\mathsf{dh,pk}})$, which is signed by the newsroom to ensure its authenticity. By leveraging $\mathsf{HPKE}_{\mathsf{auth}}$, we provide robust guarantees that messages are confidential and guaranteed to originate from the intended sender.

5.2 PQ Secrecy

Having achieved sender authentication, we now address post-quantum (PQ) secrecy. $\mathsf{HPKE}_{\mathsf{auth}}$ does not natively provide PQ secrecy. To overcome this limitation, we propose a scheme that integrates a PQ-secure KEM into $\mathsf{HPKE}_{\mathsf{auth}}$'s underlying Authenticated KEM (AKEM) DH-AKEM.

PQ-secure KEMs PQ secure KEMs are cryptographic protocols designed to establish shared keys which are resistant to attacks by quantum-capable adversaries. Unlike traditional key agreement methods—such as Diffie-Hellman, which rely on number-theoretic assumptions easily compromised by a sufficiently powerful quantum computer—PQ-secure KEMs employ hard mathematical problems from domains like lattices that currently have no known efficient quantum attacks (e.g., CRYSTALS-Kyber [4]).

PQ-secure AKEMs Ideally, we would replace DH-AKEM in $\mathsf{HPKE}_{\mathsf{auth}}$ with a PQ-secure AKEM. The security proofs for the compositional APKE scheme (see Construction 2.12) used to construct $\mathsf{HPKE}_{\mathsf{auth}}$ only rely on assumptions that hold against quantum attackers. Hence, DH-AKEM in $\mathsf{HPKE}_{\mathsf{auth}}$ can be replaced by any PQ-secure AKEM. However, to our knowledge, no such PQ-secure AKEM currently exists.[1] We therefore propose a straightforward construction $\mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}$ that combines DH-AKEM, a PQ-secure KEM, and a PRF-based KEM combiner.

[1]Authenticated key exchange using Kyber is interactive, making it unsuitable for our use case [4].

Construction 5.1 ($\mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}$)**.** Let $\mathsf{KEM}_{\mathsf{pq}}$ be a PQ-secure KEM and $\hat{F}$ a secure PRF. The other primitives are identical to the ones used in $\mathsf{HPKE}_{\mathsf{auth}}$. We define the APKE scheme $\mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}$ as follows:

```
$\underline{\mathsf{Gen_S}()}$
$(skS_{\mathsf{dh}}, pkS_{\mathsf{dh}}) \leftarrow\!\$\ \mathsf{DH\text{-}AKEM.KGen}()$
return $(skS_{\mathsf{dh}}, pkS_{\mathsf{dh}})$

$\underline{\mathsf{Gen_R}()}$
$(skR_{\mathsf{dh}}, pkR_{\mathsf{dh}}) \leftarrow\!\$\ \mathsf{DH\text{-}AKEM.KGen}()$
$(skR_{\mathsf{kem}}, pkR_{\mathsf{kem}}) \leftarrow\!\$\ \mathsf{KEM_{pq}.KGen}()$
return $((skR_{\mathsf{dh}}, skR_{\mathsf{kem}}), (pkR_{\mathsf{dh}}, pkR_{\mathsf{kem}}))$

$\underline{\mathsf{AuthEnc}(skS_{\mathsf{dh}}, (pkR_{\mathsf{dh}}, pkR_{\mathsf{kem}}), m, aad, info)}$
$(c_1, k_1) \leftarrow\!\$\ \mathsf{DH\text{-}AKEM.AuthEncap}(skS_{\mathsf{dh}}, pkR_{\mathsf{dh}})$
$(c_2, k_2) \leftarrow\!\$\ \mathsf{KEM_{pq}.Encap}(pkR_{\mathsf{kem}})$
$K \leftarrow \hat{F}(k_1, c_1 \parallel c_2) \oplus \hat{F}(k_2, c_1 \parallel c_2)$    // PRF-based KEM combiner
$k \parallel nonce \leftarrow \mathsf{F}(K, info)$
$c \leftarrow\!\$\ \mathsf{AEAD.Enc}(k, m, aad, nonce)$
return $((c_1, c_2), c)$

$\underline{\mathsf{AuthDec}((skR_{\mathsf{dh}}, skR_{\mathsf{kem}}), pkS_{\mathsf{dh}}, ((c_1, c_2), c), aad, info)}$
$k_1 \leftarrow \mathsf{DH\text{-}AKEM.AuthDecap}(skR_{\mathsf{dh}}, pkS_{\mathsf{dh}}, c_1)$
$k_2 \leftarrow \mathsf{KEM_{pq}.Decap}(skR_{\mathsf{kem}}, c_2)$
$K \leftarrow \hat{F}(k_1, c_1 \parallel c_2) \oplus \hat{F}(k_2, c_1 \parallel c_2)$    // PRF-based KEM combiner
$k \parallel nonce \leftarrow \mathsf{F}(K, info)$
$m \leftarrow \mathsf{AEAD.Dec}(k, c, aad, nonce)$
return $m$
```

Key	S2J	J2S
$(skS_{\mathsf{dh}}, pkS_{\mathsf{dh}})$	$(S_{\mathsf{dh,sk}}, S_{\mathsf{dh,pk}})$	$(J_{\mathsf{dh,sk}}, J_{\mathsf{dh,pk}})$
$(skR_{\mathsf{dh}}, pkR_{\mathsf{dh}})$	$(J_{\mathsf{edh,sk}}, J_{\mathsf{edh,pk}})$	$(S_{\mathsf{dh,sk}}, S_{\mathsf{dh,pk}})$
$(skR_{\mathsf{kem}}, pkR_{\mathsf{kem}})$	$(J_{\mathsf{ekem,sk}}, J_{\mathsf{ekem,pk}})$	$(S_{\mathsf{kem,sk}}, S_{\mathsf{kem,pk}})$

Table 5.1: $\mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}$ key material (S2J := source-to-journalist messages, J2S := journalist-to-source messages).

We conclude this section by mapping the keys used in $\mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}$ to those used in SecureDrop (see Table 5.1 for an overview). For messages sent from a source to a journalist, the source is identified by $S_{\mathsf{dh,pk}}$ and utilizes the ephemeral keys $J_{\mathsf{edh,pk}}$ and $J_{\mathsf{ekem,pk}}$ to encrypt its message. The journalist, in turn, authenticates itself using the new long-term key $J_{\mathsf{dh,pk}}$ and relies on the source's long-term keys $S_{\mathsf{dh,pk}}$ and $S_{\mathsf{kem,pk}}$ to encrypt messages back to the source securely.

5.3 Sender Anonymity

The construction $\mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}$ faces two issues regarding participants' anonymity. First, the recipient requires the sender's public key pkS_{dh} to decrypt the message. Second, if the underlying primitives DH-AKEM and $\mathsf{KEM}_{\mathsf{pq}}$ do not ensure key privacy, they may leak the recipient's identity through the public key used to generate the ciphertext[2].

We address these issues by encrypting all sensitive information under a PKE scheme that provides key privacy [8]. In SecureDrop, we choose to encrypt the sender's public key pkS_{dh} and the KEM encapsulations of $\mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}$ using a key-private PKE scheme $\mathsf{PKE}_{\mathsf{kp}}$. To achieve this, we introduce a long-term key pair $(S_{\mathsf{pke,sk}}, S_{\mathsf{pke,pk}})$ for the source and an ephemeral key pair $(J_{\mathsf{epke,sk}}, J_{\mathsf{epke,pk}})$ for the journalist.

5.4 Full Specification

This section presents the complete protocol specification that integrates authenticated public-key encryption into the SecureDrop design. While the fundamental workflow and message fetching mechanism remain unchanged, incorporating message agreement ensures that each message can be reliably attributed to its sender. We introduce the newly required key material, explain how it is incorporated into the message exchange, and provide a step-by-step overview of the resulting protocol. This full specification offers a more secure and verifiable communication channel between sources and journalists without altering the core interaction model.

5.4.1 Setup

This section describes the updated setup process of newsrooms, journalists, and sources. It additionally covers how key material is generated and distributed. An overview of the key material used in the protocol can be found in Table 5.2. For keys, we use the notation $X_{A,B}$, where X represents the key owner ($X \in \{NR, J, S\}$), A represents the key's usage ($A \in \{\mathsf{pke}, \mathsf{sig}, \mathsf{fetch}, \mathsf{dh}\}$), and is prefixed with an "e" if the key is ephemeral. B indicates whether the component is private or public. For Diffie-Hellman keys x, the public component is represented by the exponentiation $\mathsf{DH}(g, x)$.

Newsroom Setup The newsroom setup is identical to the initial protocol specification (see Section 4.1).

Journalist Enrollment In addition to the signing key pair and the fetching key, the journalist generates a long-term DH key share $J_{\mathsf{dh,sk}}$, used for authen-

[2]This key leakage is captured by the ANO-CCA security notion [16].

$$\begin{array}{lcl}
\underline{\textbf{Journalist}} & & \underline{\textbf{Newsroom}} \\
J_{\text{sig,pk}}, J_{\text{sig,sk}} \leftarrow\$ \ \mathsf{SIG.KGen}() & & \\
J_{\text{fetch,sk}} \leftarrow\$ \ \mathbb{Z}_q & & \\
J_{\text{fetch,pk}} \leftarrow \mathsf{DH}(g, J_{\text{fetch,sk}}) & & \\
J_{\text{dh,sk}} \leftarrow\$ \ \mathbb{Z}_q & & \\
J_{\text{dh,pk}} \leftarrow \mathsf{DH}(g, J_{\text{dh,sk}}) & & \\
& \xrightarrow{J_{\text{sig,pk}}, J_{\text{fetch,pk}}, J_{\text{dh,pk}}} & \\
& & \text{// manual journalist verification} \\
& & \sigma^{NR} \leftarrow\$ \ \mathsf{SIG.Sign}(NR_{\text{sig,sk}}, (J_{\text{sig,pk}}, J_{\text{fetch,pk}}, J_{\text{dh,pk}})) \\
& \xleftarrow{\sigma^{NR}} &
\end{array}$$

Figure 5.1: Journalist enrollment into the newsroom over a secure channel.

tication. The newsroom signs all three public components (i.e., $J_{\text{sig,pk}}$, $J_{\text{dh,pk}}$, and $J_{\text{fetch,pk}}$).

In contrast to the initial specification, the journalist's long-term fetching key $J_{\text{fetch,pk}}$ is now signed by the newsroom. This change does not introduce additional overhead for the newsroom and confines potential leakages that could compromise $J_{\text{fetch,sk}}$ to the newsroom's signing key. The journalist's signing key $J_{\text{sig,sk}}$ is only used to self-sign ephemeral keys. A visual representation of the interaction is shown in Fig. 5.1.

Ephemeral Keys Replenishment Journalists periodically generate ephemeral messaging keys $J_{\text{edh,sk}}$, KEM keys $J_{\text{ekem,sk}}$, and PKE keys $J_{\text{epke,sk}}$, and sign the corresponding public components $J_{\text{edh,pk}}$, $J_{\text{ekem,pk}}$, and $J_{\text{epke,pk}}$ with their signing key $J_{\text{sig,sk}}$. The public keys $J_{\text{edh,pk}}$,$J_{\text{ekem,pk}}$, and $J_{\text{epke,pk}}$, and corresponding signature $\text{sig}^J(J_{\text{edh,pk}}, J_{\text{ekem,pk}}, J_{\text{epke,pk}})$ are then stored by the messaging server. SecureDrop clients can request these ephemeral keys at the server. These keys are then verified and used to securely send a message to journalists.

Sources Sources do not require any setup. Instead, they choose a *passphrase* which is used as input to a key derivation function KDF to derive the source's key material:

$$S_{\text{dh,sk}} \parallel S_{\text{fetch,sk}} \parallel S_{\text{pke,sk}} \parallel S_{\text{kem,sk}} = \mathsf{KDF}(passphrase)$$

The public key $S_{\text{pke,pk}}$, used for the asymmetric encryption, and the public KEM key $S_{\text{kem,pk}}$, are derived from a suitable public-key generation procedure.

5.4.2 Messaging Protocol

In the proposed design, the overall communication pattern remains consistent with the initial specification, and the message fetching mechanism

Key	Type	Usage
$(NR_{\mathsf{sig,pk}}, NR_{\mathsf{sig,sk}})$	PPK	Signing
$(J_{\mathsf{sig,pk}}, J_{\mathsf{sig,sk}})$	PPK	Signing
$(J_{\mathsf{fetch,sk}}, J_{\mathsf{fetch,pk}})$	DH	Fetching
$(J_{\mathsf{dh,sk}}, J_{\mathsf{dh,pk}})$	DH	DH-AKEM
$(J_{\mathsf{ekem,sk}}, J_{\mathsf{ekem,pk}})$	Ephemeral PPK	$\mathsf{KEM}_{\mathsf{pq}}$
$(J_{\mathsf{epke,sk}}, J_{\mathsf{epke,pk}})$	Ephemeral PPK	PKE
$(J_{\mathsf{edh,sk}}, J_{\mathsf{edh,pk}})$	Ephemeral DH	DH-AKEM
$(S_{\mathsf{fetch,sk}}, S_{\mathsf{fetch,pk}})$	DH	Fetching
$(S_{\mathsf{dh,sk}}, S_{\mathsf{dh,pk}})$	DH	DH-AKEM
$(S_{\mathsf{kem,sk}}, S_{\mathsf{kem,pk}})$	PPK	$\mathsf{KEM}_{\mathsf{pq}}$
$(S_{\mathsf{pke,sk}}, S_{\mathsf{pke,pk}})$	PPK	PKE

Table 5.2: Key material in the proposed SecureDrop protocol with hybrid PKE extension. (PPK := Public-Private Keypair)

is unchanged and therefore not repeated here. Instead, we focus on the integration of authenticated public-key encryption. In particular, we detail how the $\mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}$ scheme and the $\mathsf{PKE}_{\mathsf{kp}}$ scheme work together to provide both confidentiality and authenticity, while preserving the anonymity of participants. This approach ensures that each party can reliably verify the origin of messages, maintain the original workflow, and safeguard users' identities.

Key Fetching SecureDrop clients retrieve the journalists' key material from the server and verify its authenticity, ensuring that the journalist is enrolled in the correct newsroom for communication. The server returns the key material for each enrolled journalist, which is then used to securely send messages. When a client requests the key material for a specific journalist J, the server provides J's long-term keys and a randomly selected subset of keys from J's set of ephemeral messaging keys, which the journalist generated. This subset includes the journalist's ephemeral DH public key $J_{\mathsf{edh,pk}}$, ephemeral KEM key $J_{\mathsf{ekem,pk}}$, and ephemeral PKE key $J_{\mathsf{epke,pk}}$, all signed by the journalist. The server then discards these ephemeral keys and their signatures to prevent reuse. The interaction is shown in Fig. 5.2.

Source Submission After retrieving the journalist's keys, the source sends a message containing the source's own key material for the journalist's response. The source includes the intended recipients—specifically, the newsroom and the journalist—identified by the newsroom's identifier and the journalist's signing key $J_{\mathsf{sig,pk}}$. Using the journalist's ephemeral keys, the source encrypts the message to ensure confidentiality. The long-term key $S_{\mathsf{dh,pk}}$ identifies the sending source and is incorporated into $\mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}$ to guarantee authenticity. To prevent identity leakage, the source asymmetrically

Client | **Server**

$\xrightarrow{\text{request keys for } J}$

choose ephemeral keys

$\xleftarrow[]{J_{\mathsf{sig,pk}}, J_{\mathsf{fetch,pk}}, J_{\mathsf{dh,pk}}, \sigma^{NR} \; J_{\mathsf{edh,pk}}, J_{\mathsf{ekem,pk}}, J_{\mathsf{epke,pk}}, \; \mathsf{sig}^{J}(J_{\mathsf{edh,pk}}, J_{\mathsf{ekem,pk}}, J_{\mathsf{epke,pk}})}$

verify the key material using $NR_{\mathsf{sig,pk}}$

discard the sent ephemeral keys

Figure 5.2: Client fetches key material of journalist J and verifies the keys' authenticity.

Source | **Server**

$m \leftarrow msg \| S_{\mathsf{dh,pk}} \| S_{\mathsf{pke,pk}} \| S_{\mathsf{kem,pk}} \| S_{\mathsf{fetch,pk}} \| J_{\mathsf{sig,pk}} \| NR$

$((c_1, c_2), C'') \leftarrow\$ \mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}.\mathsf{AuthEnc}(S_{\mathsf{dh,sk}}, (J_{\mathsf{edh,pk}}, J_{\mathsf{ekem,pk}}), m, \text{""}, \text{""})$

$C' \leftarrow\$ \mathsf{PKE}_{\mathsf{kp}}.\mathsf{Enc}(J_{\mathsf{epke,pk}}, S_{\mathsf{dh,pk}} \| c_1 \| c_2)$

$C \leftarrow C' \| C''$

$x \leftarrow_\$ \mathbb{Z}_q$

$X \leftarrow \mathsf{DH}(g, x)$

$Z \leftarrow \mathsf{DH}(J_{\mathsf{fetch,pk}}, x)$

$\xrightarrow{C, Z, X}$

$id \leftarrow\$ \mathsf{Rand}()$

$messages[id] \leftarrow (C, Z, X)$

Figure 5.3: The source submits a message msg to the server that is intended for newsroom NR's journalist J owning $J_{\mathsf{sig,pk}}$.

encrypts both the KEM encapsulations from $\mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}$ and $S_{\mathsf{dh,pk}}$ itself using $\mathsf{PKE}_{\mathsf{kp}}$. Encrypting these values ensures that neither the source's nor the journalist's identity is revealed to the server. As in the initial design, the server stores the resulting ciphertext and the components Z and X for the message fetching mechanism without learning the participants' identities. See Fig. 5.3 for a visual representation of the interaction.

Journalist Message Fetching The fetch operation for journalists is identical to the operation in the initial protocol specification (see Section 4.2).

Journalist Read After successfully decrypting a message ID with the journalist's fetching key $J_{\mathsf{fetch,sk}}$, the server is queried using this ID. The journalist attempts decryption with each of its ephemeral keys until the correct key is found, which is discarded. The decrypted message and the source's key material are returned to the journalist. The latter will be used when encrypting a response.

Server | **Journalist**

$$\xleftarrow{id}$$

$C, Z, X \leftarrow messages[id]$

$$\xrightarrow{C}$$

```
for all stored ephemeral keys $(J_{\text{edh,sk}}, J_{\text{ekem,sk}}, J_{\text{epke,sk}})$ :
  Parse $C$ as $C' \| C''$
  $\widetilde{M} \leftarrow \mathsf{PKE}_{\text{kp}}.\mathsf{Dec}(J_{\text{epke,sk}}, C')$   // check $\widetilde{M} \neq \bot$
  Parse $\widetilde{M}$ as $S \| c_1 \| c_2$
  $m \leftarrow \mathsf{HPKE}^{\text{pq}}_{\text{auth}}.\mathsf{AuthDec}((J_{\text{edh,sk}}, J_{\text{ekem,sk}}), S, ((c_1, c_2), C''), \text{""}, \text{""})$   // check $m \neq \bot$
  Parse $m$ as $msg \| \widetilde{S} \| S_{\text{pke,pk}} \| S_{\text{kem,pk}} \| S_{\text{fetch,pk}} \| \widetilde{J} \| \widetilde{NR}$
  // check $NR = \widetilde{NR}$, $J_{\text{sig,pk}} = \widetilde{J}$, and $S = \widetilde{S}$
  $\mathsf{Discard}(J_{\text{edh,sk}}, J_{\text{ekem,sk}}, J_{\text{epke,sk}})$
  return $msg \| S_{\text{dh,pk}} \| S_{\text{kem,pk}} \| S_{\text{fetch,pk}}$
```

Figure 5.4: Journalist reads a source's message by querying the server with a decrypted message ID (assuming the ID exists).

During the decryption process, the journalist enrolled in a particular newsroom NR verifies that they are the intended recipient. Since sources explicitly include the target journalist's signing key $\widetilde{J}$ and newsroom identifier, the journalist uses this information to confirm that the message was indeed intended for them. This interaction is shown in Fig. 5.4.

Journalist Reply Journalists reply to sources using the key material obtained from the preceding submission. To authenticate themselves, journalists include their long-term keys $J_{\text{sig,pk}}$, $J_{\text{fetch,pk}}$, and $J_{\text{dh,pk}}$, along with the newsroom's signature σ^{NR}. This ensures that sources do not need to fetch key material upon receiving a reply, preserving a communication pattern consistent with the scenario in which journalists receive messages. Furthermore, the same long-term DH key $J_{\text{dh,pk}}$ is employed in encryption using $\mathsf{HPKE}^{\text{pq}}_{\text{auth}}$, ensuring the origin authenticity of the communication. As in the initial design, the server stores the resulting ciphertext and the components Z and X for the message fetching mechanism without learning the participants' identities. A visual representation of the interaction is shown in Fig. 5.5.

Source Message Fetching The fetch operation for sources is identical to the operation in the initial protocol specification (see Section 4.2).

Source Read The source's read operation closely mirrors the journalist's read procedure (see Fig. 5.6). However, instead of using ephemeral keys, the source relies on its long-term keys, which are derived from its passphrase—

Server | **Journalist**

$m \leftarrow msg \| S \| J_{\mathsf{sig,pk}} \| J_{\mathsf{fetch,pk}} \| J_{\mathsf{dh,pk}} \| \sigma^{NR} \| NR$

$((c_1, c_2), C'') \leftarrow\!\!\$\; \mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}.\mathsf{AuthEnc}(J_{\mathsf{dh,sk}}, (S, S_{\mathsf{kem,pk}}), m, \text{""}, \text{""})$

$C' \leftarrow\!\!\$\; \mathsf{PKE}_{\mathsf{kp}}.\mathsf{Enc}(S_{\mathsf{pke,pk}}, J_{\mathsf{dh,pk}} \| c_1 \| c_2 \| c)$

$C \leftarrow C' \| C''$

$x \leftarrow_{\$} \mathbb{Z}_q$

$Z \leftarrow \mathsf{DH}(S_{\mathsf{fetch,pk}}, x)$

$X \leftarrow \mathsf{DH}(g, x)$

$\xleftarrow{C, Z, X}$

$id \leftarrow\!\!\$\; \mathrm{Rand}()$

$messages[id] \leftarrow (C, Z, X)$

Figure 5.5: Journalist replies to a message.

Source | **Server**

$\xrightarrow{id}$

$C, Z, X \leftarrow messages[id]$

$\xleftarrow{C, X}$

Parse C as $C' \| C''$

$\widetilde{M} \leftarrow \mathsf{PKE}_{\mathsf{kp}}.\mathsf{Dec}(S_{\mathsf{pke,sk}}, C')$ // check $\widetilde{M} \neq \bot$

Parse $\widetilde{M}$ as $J \| c_1 \| c_2$

$m \leftarrow \mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}.\mathsf{AuthDec}((S_{\mathsf{dh,sk}}, S_{\mathsf{kem,sk}}), J, ((c_1, c_2), C''), \text{""}, \text{""})$ // check $m \neq \bot$

Parse m as $msg \| \widetilde{S} \| J_1 \| J_2 \| J_3 \| \sigma \| \widetilde{NR}$

$\mathsf{SIG.Vfy}(NR_{\mathsf{sig,pk}}, \sigma, J_1 \| J_2 \| J_3)$

// check $NR = \widetilde{NR}$, $J = J_3$, and $S_{\mathsf{dh,pk}} = \widetilde{S}$

return $msg \| J \| NR$

Figure 5.6: Source reads a message by querying the server with its decrypted message ID.

its only stored state. This eliminates the need for trial decryption attempts. After decrypting the message, the source verifies the sender's authenticity by confirming that the journalist's keys match those of an enrolled journalist from the expected newsroom. This ensures that the source is indeed communicating with the correct recipient and maintains the integrity of the interaction.

Source Reply The source replies by creating a new submission to the newsroom.

Chapter 6

Security Analysis

This chapter presents the security analysis of the proposed SecureDrop specification. The analysis is conducted within the symbolic model using Tamarin and covers two distinct adversarial settings: a malicious server and a malicious user. For each setting, we provide the Tamarin model and the security guarantees that have been formally proven. We then discuss the limitations of our formal analysis and conclude with the manual analysis of anonymity guarantees. The Tamarin models and proofs can be found here [21]. The results and insights from this chapter will be interpreted and discussed in the subsequent chapter.

6.1 Malicious Server Setting

In the malicious server setting, the adversary fully compromises the messaging server, meaning it can read all messages and actively participate in the communication. Since the adversary has complete control over the network and can see all ciphertexts, the focus of the analysis shifts away from the message fetching mechanism. Instead, we concentrate on the secrecy and the integrity of messages. Additionally, we consider the possibility that the adversary may gain access to a quantum computer in the future. However, in this case, all honest participants are assumed to halt the protocol, thereby preventing the adversary from retrieving any ciphertexts.

6.1.1 Protocol Model

Here, the messaging server is not explicitly modeled. Instead, the network adversary functions as the messaging server, meaning that all communication between honest participants passes through the adversary's control. As a result, the adversary can read and modify any (encrypted) message transmitted between sources and journalists.

```
// PQ Attacker
rule PQAttackerStart:
  [] --[ PQAttack() ]-> [ !PQAttacker() ]

rule PQAttacker:
  [ !PQAttacker(), !NonPQSecKey(k) ] --> [ Out(k) ]

restriction PQAttacker:
  "All #compr #t.
        PQAttack() @ #compr & PrePQAttack() @ #t ==> #t < #compr"
```

Figure 6.1: PQ Attacker [19].

To secure communication, sources and journalists encrypt and authenticate their messages using the APKE scheme $\mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}$, which ensures the confidentiality and integrity of the transmitted data. However, the message fetching mechanism—which prevents unintended receivers from accessing ciphertexts—is not modeled because an adversary acting as the server already holds all ciphertexts, making that mechanism irrelevant to restricting the adversary's capabilities. Instead, honest participants engage in trial decryption to simulate the process of message retrieval. Therefore, there are only four communication rules: `Source_Send`, `Journalist_Recv`, `Journalist_Send`, and `Source_Recv`.

The protocol is modeled such that once the network adversary gains access to quantum computing capabilities, all honest participants (i.e., sources, journalists, and newsrooms) cease using the protocol. This is modeled by the `PQAttack()` action fact in the rules modeling quantum capabilities and a `PrePQAttack()` action fact in all honest participants' rules. The Tamarin rules modeling the PQ Attacker are shown in Fig. 6.1.

Note, all keys `k` which a PQ attacker can derive need to produce the fact `!NonPQSecKey(k)` (e.g., a DH secret share x if g^x is publicly known). The restriction limits the set of traces to the ones where no `PrePQAttack()` action fact occurs after a `!PQAttack()` action fact.

6.1.2 Security Guarantees

Sources' Message Secrecy In the malicious server setting, a source's message may be leaked to the network adversary in two ways. First, the adversary can act as the intended recipient by either compromising the newsroom to enroll a malicious journalist or by compromising an existing journalist's signing key. The adversary could then register malicious ephemeral keys, which the source uses to encrypt the message. In both cases, the adversary must register this malicious material before the source sends the message, ensuring that only the adversary can decrypt the ciphertext.

The second way the message can be leaked is if the journalist's ephemeral key material, which is used by the source to encrypt the message, is exposed.

```
All #t s j j_edh_pk j_ekem_pk nr m.
        Source_Sent(s, j, j_edh_pk, j_ekem_pk, nr, m) @ #t
  ==>   (not Ex #x. K(m) @ #x)
      | (Ex #x. Reveal_Journalist_SIG_Key(j) @ #x & #x < #t)
      | (Ex #x. Reveal_Newsroom_Key(nr) @ #x & #x < #t)
      | (  (  (Ex #x. Reveal_Journalist_EDH_Key(j_edh_pk) @ #x)
            | (Ex #x. PQAttack() @ #x))
         & (Ex #x. Reveal_Journalist_EKEM_Key(j_ekem_pk) @ #x))
```

Figure 6.2: Secrecy lemma of messages sent by sources.

$\text{HPKE}^{\text{pq}}_{\text{auth}}$ provides post-quantum security, so the journalist's ephemeral KEM key must be leaked. To break the DH-AKEM key component, the journalist's ephemeral DH key must be leaked, either explicitly or through quantum computing capabilities. If both journalist ephemeral keys are compromised, the adversary can decrypt the message encrypted under those keys. Note, this captures the notion of forward secrecy because the compromise of these ephemeral keys does not leak previously sent messages. The Tamarin lemma can be found in Fig. 6.2.

Sources' Message Agreement Message agreement ensures that whenever a journalist receives a message from a source, it is confirmed that the particular source sent that message to the journalist (see Fig. 6.3). The authenticity of this agreement is provided by the APKE scheme $\text{HPKE}^{\text{pq}}_{\text{auth}}$, which ensures that the message is properly authenticated using the sender's long-term DH key (i.e., for sources $S_{\text{dh,pk}}$). However, because $\text{HPKE}^{\text{pq}}_{\text{auth}}$'s underlying AKEM DH-AKEM does not offer Insider-Auth, the adversary can impersonate a source if the source's long-term DH key or the journalist's ephemeral DH key is leaked. This could also occur if the journalist's signing key is compromised, allowing the adversary to register leaked ephemeral key material, enabling impersonation.

Additionally, the protocol guarantees injective agreement due to the use of ephemeral recipient keys. This means that messages sent by sources cannot be replayed, as journalists use their ephemeral keys only once to decrypt messages. This ensures that each message is uniquely tied to the moment it was sent and prevents replay attacks.

Journalists' Message Secrecy Secrecy for responses from journalists to honest sources relies on the authenticity of the initial submission, which includes the source's key material used by the journalist to secure the response. If such a response is sent using the source's long-term keys, the message remains secret unless both keys of $\text{HPKE}^{\text{pq}}_{\text{auth}}$'s two underlying KEMs are leaked. Specifically, a DH-AKEM key is leaked either explicitly or via quantum computing capabilities, while the PQ-secure KEM key must be leaked explicitly for the message to be exposed. In contrast to messages

```
All #t1 #t2 j j_edh_pk nr s m.
        Journalist_Received(j, j_edh_pk, nr, s, m) @ #t1
      & Register_Source(s) @ #t2
  ==>   (Ex j_ekem_pk #x1.
                Source_Sent(s, j, j_edh_pk, j_ekem_pk, nr, m) @ #x1
              & #x1 < #t1
              & not (Ex j_2 j_edh_pk_2 nr_2 s_2 #x2.
                          Journalist_Received(j_2, j_edh_pk_2, nr_2,
                                                s_2, m) @ #x2
                        & not (#x2 = #t1)))
      | (Ex #x. Reveal_Source_DH_Key(s) @ #x)
      | (Ex #x. Reveal_Journalist_SIG_Key(j) @ #x)
      | (Ex #x. Reveal_Journalist_EDH_Key(j_edh_pk) @ #x)
```

Figure 6.3: Injective agreement lemma of messages sent by sources.

```
All #t1 #t2 #t3 s s_kem_pk j j_edh_pk j_ekem_pk j_dh_pk nr m_old m.
        Source_Sent(s, j, j_edh_pk, j_ekem_pk, nr, m_old) @ #t1
      & Journalist_Received(j, j_edh_pk, nr, s, m_old) @ #t2
      & Journalist_Response(j, j_dh_pk, nr, s, s_kem_pk, m_old, m) @ #t3
  ==>   (not Ex #x. K(m) @ #x)
      | (  (Ex #x. Reveal_Source_KEM_Key(s_kem_pk) @ #x)
         & (  (Ex #x. Reveal_Source_DH_Key(s) @ #x)
            | (Ex #x. PQAttack() @ #x)))
```

Figure 6.4: Secrecy lemma of messages sent by journalists.

sent *to* journalists, the compromise of these (long-term) keys also leaks past messages (i.e., there is no forward secrecy). The Tamarin lemma can be found in Fig. 6.4.

Journalists' Message Agreement Message agreement for communication from journalists to sources is defined similarly to that for communication from sources to journalists (see Fig. 6.5). The journalist is identified by its long-term DH key (recall that messages sent to the journalist use ephemeral DH keys). The adversary can impersonate a journalist if the newsroom key is leaked, allowing the adversary to enroll a malicious journalist, or if the journalist's long-term DH secret is leaked. Also, because $\mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}$ does not provide Insider-Auth, the leakage of the receiving source's long-term DH keys enables the adversary to impersonate the journalist.

Since the source's key material is long-term, injectivity for messages is not guaranteed. This means the network adversary can replay messages, as there is no mechanism to prevent the retransmission of previously sent messages between journalists and sources.

```
All #t1 s j nr m.
        Source_Received(s, j_dh_pk, nr, m) @ #t1
  ==>   (Ex j s_kem_pk m_old #x.
              Journalist_Response(j, j_dh_pk, nr, s,
                                  s_kem_pk, m_old, m) @ #x
            & #x < #t1)
      | (Ex #x. Reveal_Newsroom_Key(nr) @ #x)
      | (Ex #x. Reveal_Journalist_DH_Key(j_dh_pk) @ #x)
      | (Ex #x. Reveal_Source_DH_Key(s) @ #x)
```

Figure 6.5: Non-injective agreement lemma of messages sent by journalists.

6.2 Malicious User Setting

In this section, we analyze the security of the proposed SecureDrop protocol when faced with a malicious user who possesses quantum computing capabilities in the future. In this scenario, the messaging server is assumed to act honestly, with no malicious intent or compromise.

A malicious server poses a strictly stronger threat than a malicious user, as it has control over the messaging infrastructure and can potentially manipulate or intercept messages. Therefore, all security guarantees established in the malicious server setting also apply to the malicious user setting. In this analysis, we specifically focus on SecureDrop's message fetching mechanism, which differentiates the protocol from a simple bulletin board system. Unlike systems that rely on trial decryption, SecureDrop ensures that messages are only accessible to the intended recipient, providing an additional layer of security against malicious users.

6.2.1 Protocol Model

The protocol model in the malicious user setting assumes an honest server. In this setup, the server is explicitly modeled, and participants send their messages to the server over a confidential and authenticated channel, relying on the strong security guarantees provided by Tor to protect the communication from eavesdropping or tampering by external adversaries.

When a publicly accessible messaging server `$Server` is instantiated, clients communicate with it over a channel by selecting a fresh session ID *~sess*. If this session ID is leaked, the adversary gains the ability to read the messages. Honest clients send messages to the server using the `Client_Out` fact and receive messages using the `Client_In` fact. Similarly, honest servers send messages using the `Server_Out` fact and receive messages using the `Server_In` fact. The channel rules are depicted in Fig. 6.6.

Rather than the bulletin board-like construction of the previous model, the server generates a challenge that only the intended recipient of a particular ciphertext, with the appropriate fetching key, can solve. Therefore, this model has for each client a message fetching rule `Fetch_Messages` and a rule solving

```
rule ClientOut:
  [ Client_Out(~sess, $Server, msg), !Message_Server($Server) ]
  -->
  [ !Client(~sess, $Server, msg) ]

rule AdversaryClientOut:
  [ In(~sess), In(msg), !Message_Server($Server) ]
  -->
  [ !Client(~sess, $Server, msg) ]

rule ServerOut:
  [ Server_Out(~sess, $Server, msg), !Message_Server($Server) ]
  -->
  [ !Server(~sess, $Server, msg) ]

rule ClientToServer:
  [ !Client(~sess, $Server, msg) ]
  -->
  [ Server_In(~sess, $Server, msg) ]

rule ServerToClient:
  [ !Server(~sess, $Server, msg) ]
  -->
  [ Client_In(~sess, $Server, msg) ]

rule ServerToAdversaryClient:
  [ In(adversarySess), !Server(adversarySess, $Server, msg) ]
  -->
  [ Out(msg) ]
```

Figure 6.6: Secure channel rules [18].

the received challenge Solve_Fetch_Challenge.

As the secrecy of plaintext messages is not the focus in this model, participants send "dummy" messages instead of ciphertexts encrypted using the authenticated PKE scheme $\mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}$. This simplification allows us to focus on the core aspect of the protocol: secure message retrieval by the intended recipient. Therefore, the following security properties are solely concerned with the leakage of these "dummy" ciphertexts.

Again, the protocol is modeled such that once the adversary gains access to quantum computing capabilities, all honest participants (i.e., sources, journalists, newsrooms, and messaging servers) cease using the protocol.

6.2.2 Security Guarantees

Sources' Ciphertext Secrecy Journalists' fetching keys are signed by the newsroom, ensuring their authenticity. Suppose a source sends a ciphertext to a journalist enrolled in a particular newsroom. In that case, the ciphertext remains secret unless the journalist's fetching key is revealed or the adversary

```
All #t s j nr c.
        Source_Sent(s, j, nr, c) @ #t
  ==>   (not Ex #x. K(c) @ #x)
      | (  (Ex #x. Reveal_Journalist_Fetch_Key(j) @ #x)
         | (Ex #x. Reveal_Newsroom_Key(nr) @ #x & #x < #t))
```

Figure 6.7: Secrecy lemma of messages sent by sources.

```
All #t1 #t2 s j c_old c.
        Journalist_Response(j, s, c_old, c) @ #t1
      & Register_Source(s) @ #t2
  ==>   (not Ex #x. K(c) @ #x)
      | (Ex #x. Reveal_Source_Fetch_Key(s) @ #x)
```

Figure 6.8: Secrecy lemma of messages sent by journalists.

successfully enrolls a malicious journalist (see Fig. 6.7). In the latter case, the newsroom's key must first be leaked before the source has sent the ciphertext. This setup ensures that the secrecy of the ciphertext is maintained, as long as the fetching keys remain secure and the adversary cannot compromise the enrollment process.

Journalists' Ciphertext Secrecy On the other hand, if the journalist responds to an honest source (as captured by the `Register_Source` fact, ensuring that the malicious user is not the intended recipient), the ciphertext remains secret unless the source's fetching key is compromised (see Fig. 6.8). It is important to note that journalists receive a source's message fetching key via the initial submission. Therefore, the authenticity of this key depends on the message agreement for source-to-journalist communication (see Fig. 6.3).

Impact of PQ Capabilities The presence of post-quantum capabilities in the malicious user setting does not affect the security guarantees of the message fetching mechanism. Although the fetching mechanism relies on a Diffie–Hellman key exchange (which a quantum adversary could break), retrieving ciphertexts requires active participation by the adversary *after* it gains quantum capabilities. Specifically, with a quantum computer, the adversary can solve the discrete logarithm problem for the public fetching key g^x, recover the secret exponent x, and thus fetch messages from the server or solve any challenge intended for the legitimate client holding x. This attack can be performed for any client in the system, allowing the adversary to retrieve all ciphertexts. However, our threat model assumes that once a quantum adversary appears, all honest participants cease using the protocol, meaning the adversary can no longer rely on the honest server to serve further requests.

6.3 Model Limitations

Hash-Based KEM Combiner In our analysis, we modeled the KEM combiner in $\mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}$ using a hash-based construction instead of the proposed PRF-based construction (see Eq. (2.1)). This adjustment was necessary because the associativity and commutativity properties of the XOR operation blew up the size of Tamarin's search space. As a result, we were unable to run the proofs with the original PRF-based construction, leading to the use of a simpler hash-based alternative. This limitation may affect the accuracy of our security guarantees in the proposed PRF-based design.

Partition of Tamarin Models Having two separate Tamarin models for the malicious user and malicious server settings is not ideal. The need to separate the DH-AKEM and message fetching mechanisms arose because both rely on the `diffie-hellman` equational theory. However, due to the state space explosion resulting from commutativity in this theory, the computational resources available (250 GB of RAM) were insufficient to model both mechanisms simultaneously within the same Tamarin model. As a result, we had to treat them separately, which introduces some limitations in the scope and completeness of the analysis.

Nevertheless, the separation does not affect the overall validity of the security guarantees because the messaging and message fetching mechanisms rely on separate keys. Introducing the message fetching mechanism for the malicious server setting would not grant the adversary additional capabilities to break the confidentiality or message agreement of $\mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}$. The cryptographic operations and keys for fetching messages are distinct from those for encrypting and delivering messages, ensuring that their security properties remain isolated and unaffected by their separation in the model.

Similarly, for the malicious user setting, replacing the "dummy" ciphertext in the current model with a ciphertext output by $\mathsf{HPKE}^{\mathsf{pq}}\mathsf{auth}$ would not impact ciphertext secrecy for either sources or journalists. The cryptographic guarantees of $\mathsf{HPKE}^{\mathsf{pq}}_{\mathsf{auth}}$ ensure that the adversary cannot gain any additional insights or compromise message secrecy when this substitution is applied. Therefore, while the partition of models is suboptimal from a modeling perspective, it does not undermine the core security assurances provided by the protocol in either adversarial setting.

6.4 Anonymity Guarantees

In this section, we analyze SecureDrop's anonymity guarantees by examining the information an adversary may learn. Although the protocol's security requirements outline what an attacker is allowed to learn, this discussion does not constitute formal proof of anonymity guarantees—i.e., what the attacker cannot learn. We will discuss the implications of these findings in the next chapter.

Malicious User A malicious user can interact with the server like any honest client, which means it is only able to retrieve those ciphertexts intended for itself. This does not violate any anonymity assumptions since the legitimate recipient is, by definition of sender authentication, allowed to learn who sent the message. However, if the malicious user acquires quantum computing capabilities, it can observe number of ciphertexts stored on the server of all (honest) participants. By periodically fetching messages from the server, it obtains the challenges. Each set of challenges differs from one retrieval to the next. The adversary can solve the discrete logarithm problem with quantum computing resources to derive other participants' secret fetching keys. Because decryption fails when a party tries to decrypt a challenge not intended for its corresponding secret key, the adversary effectively learns the number of ciphertexts stored at the server over time.

Passive Malicious Server A passive malicious server with quantum computing capabilities can learn who the intended receiver of a ciphertext is by unmasking the masked receiver's fetching key. Specifically, it can solve the discrete logarithm problem on the values Z and X shown in Fig. 5.3 and Fig. 5.5, thereby revealing who each ciphertext was intended for.

Active Malicious Server Unlike the malicious user, a malicious server doesn't necessarily need access to a quantum computer to learn the intended recipient given a particular ciphertext. Instead, it can actively exploit the message fetching mechanism.

Assume the server receives a ciphertext c, a fetching share Z, and an encryption public key $X = \mathsf{DH}(g, x)$. The malicious server's goal is to identify the intended recipient of this ciphertext. Recall that Z is the intended recipient's masked public fetching key K. In other words, $Z = \mathsf{DH}(K, x)$.

To learn the intended recipient, the server can attempt to test whether the ciphertext is meant for a client associated with the public fetching key K. When a client begins fetching messages, the server can generate two message identifiers, ID_1 and ID_2, and proceed as follows:

- Encrypt ID_1 using Z, then send the resulting ciphertext alongside X to the requesting party.
- Encrypt ID_2 using $\mathsf{DH}(K, y)$ (where y is a fresh value), then send the resulting ciphertext along with $Y = \mathsf{DH}(g, y)$.

By linking ID_2 to the fetching client (and thus to the public key K) and ID_1 to the ciphertext, the server can observe that if it later receives a challenge response for both ID_1 and ID_2, the ciphertext c was intended for the client associated with K.

The analysis presented in this section highlights the varying degrees of information an adversary can extract under different threat scenarios. While SecureDrop adheres to its defined security requirements, these attacks demonstrate that it does not achieve stronger anonymity guarantees than those explicitly stated. Notably, all attacks align with the threat model.

Chapter 7

Discussion

This section reflects on the thesis, addressing the challenges encountered, the decisions made, and the key findings from the formal analysis of the SecureDrop protocol. It examines the implications of these findings for SecureDrop's security and development.

In the following paragraphs, we first outline how the shift to untrusted hosting impacts the threat model, then discuss our prioritization of secrecy, integrity, and anonymity. We also highlight key results regarding message secrecy and source authentication, before concluding with a look at anonymity and the potential complexities of group messaging. Following this discussion, we propose the most promising directions for future work in the next chapter.

Threat Model In the redesigned SecureDrop protocol, the critical shift lies in hosting the messaging server with an untrusted service provider. While the functional requirements remain unchanged from the currently deployed SecureDrop platform, this shift necessitates an expanded threat model. Specifically, beyond malicious users, we now account for a potentially malicious server—an actor with significantly greater access to message flows and metadata than typical outside adversaries. This consideration distinguishes SecureDrop from other protocols (e.g., Signal), which generally assume some trust in the server.

Different threats arise from passive versus active malicious servers. A passive malicious server can observe traffic patterns and potentially infer recipient identities, yet has limited capacity for direct interference. Conversely, an active malicious server can manipulate the challenge-response steps in the message fetching mechanism to learn even more about message recipients. While detection of such active behavior is in principle possible (e.g., by monitoring the number of challenges issued), our analysis reveals at least one subtle attack that remains undetectable from the client's perspective if the server does not deviate from its expected number of challenges returned (see Section 6.4). Thus, distinguishing between passive and active server threats is crucial for designing countermeasures and accurately assessing SecureDrop's anonymity guarantees. In hindsight, we would have refined our threat model to more comprehensively account for the nuanced behaviors of both passive and active malicious servers.

Lastly, in our threat model we assumed that all protocol participants halt the protocol when quantum computers become available. Given this passive PQ attacker model, our proposed design may no longer suffice to protect participants and must be reevaluated.

Prioritizing Security Guarantees SecureDrop aims to fulfill three core security requirements—message secrecy, authentication, and anonymity—none outweighs the others. However, given the well-defined nature of secrecy and authentication in this protocol, the limited time available to conduct this thesis, and due to Tamarin's established strengths in verifying these properties, our analysis focused on achieving demonstrable guarantees in these areas. This effort proved fruitful, resulting in significant improvements in understanding and securing SecureDrop's data confidentiality and message authenticity. Nevertheless, anonymity remains a critical pillar of SecureDrop—particularly in its role as a whistleblowing platform—and deserves further, more in-depth formal analysis. Future work should concentrate on formalizing and verifying anonymity guarantees to align with SecureDrop's ultimate goal of protecting whistleblower identities.

Message Secrecy Message secrecy is indispensable in SecureDrop, ensuring only intended recipients can access communications. The initial system design proposed by the Freedom of Press Foundation achieved secrecy against all classical adversaries by relying on Diffie-Hellman key exchanges, but it lacked security against PQ attackers. We improved the protocol with a PQ-secure KEM integrated via a KEM combiner to address this shortcoming. In this approach, the PQ-secure KEM key must be obtained by the adversary (classical or quantum) to access the plaintext of the underlying message.

SecureDrop employs ephemeral, message-specific keys for messages sent to journalists, thereby ensuring forward secrecy. Even if a journalist's long-term key material is later leaked, past communications remain secure because a unique, short-lived key protects each message. By contrast, messages sent to sources are encrypted under long-term keys due to the stateless design on the source side. While this functional requirements limits forensic evidence on the source's side, it forgoes forward secrecy: compromising the source's long-term keys compromises the confidentiality of all previously sent messages.

Source Authentication Message agreement—ensuring both parties agree on who sent what to whom—is critical in SecureDrop. Without agreement, impersonation attacks are possible, leading to potentially severe consequences.

For example, a source could receive a response from a journalist referencing statements the source never made, resulting from their impersonation by an attacker.

When sender keys are known beforehand (as with journalists), the protocol can combine public-key encryption with a digital signature scheme for straightforward authentication. However, sources in SecureDrop are anonymous and identified only by the key they use to encrypt their message (which is included in the message). In this setting, confidentiality and sender authentication of an unknown sender are jointly achieved through an authenticated public-key encryption scheme. Journalists use the same cryptographic primitives to achieve protocol symmetry.

Notably, our proposed design achieves message agreement (including sender authentication) in the symbolic model without assuming that public keys remain hidden; even if an adversary learns an honest participant's public key, it cannot impersonate that participant. This approach contrasts with the initial SecureDrop design, which relied on the strong and generally untenable assumption that these keys were effectively non-public. In cases where a source uses the same passphrase across submissions, public key leakage becomes a significant risk.

Although we have validated authenticity in the symbolic model, we have not provided a game-based proof of this property. While secrecy guarantees for KEM combiners are well researched, authenticity guarantees are less explored/standardized, making them a promising avenue for future work.

Anonymity In this thesis, we formalize anonymity guarantees by specifying the precise information an adversary may obtain and illustrating how they learn it. What remains unproven is that an adversary cannot learn anything beyond what we have identified. Due to time constraints, we did not invest more time into the analysis of these guarantees. For future work, we recommend analyzing the cryptographic primitives introduced here to understand their information leakage better. Moreover, the protocol must be assessed holistically—accounting for broader attacks such as timing—to ensure robust anonymity guarantees in real-world deployments. Ultimately, these anonymity properties make SecureDrop unique among secure communication platforms, underscoring their critical importance.

Group Messaging Model The Freedom of the Press Foundation envisioned SecureDrop as a group messaging platform, facilitating communication between sources and all journalists within a newsroom. However, this model raises concerns.

From a whistleblower's perspective, interacting with multiple journalists increases the potential points of failure and might complicate trust establishment. The inherent complexity of such a model also complicates protocol analysis. To address these challenges, this thesis focused on a one-to-one communication model, where sources initially select a journalist randomly and maintain consistent interaction with the chosen journalist.

If SecureDrop were to introduce an implicit group messaging model, its design goals and security analysis would require revisions. In particular, message agreement would become more complex, as multiple recipients must confirm to each other that each member of the group receives the same message. This would introduce additional layers of protocol design and verification that go beyond the scope of our current one-to-one analysis.

Chapter 8

Conclusion

This thesis presented a formal analysis of the SecureDrop protocol under a newly defined set of design goals, including a comprehensive threat model and explicit security requirements. By modeling the protocol in Tamarin, we identified critical vulnerabilities—notably the absence of message agreement—and formulated a revised design that addresses these issues. As part of this solution, we incorporated post-quantum secrecy to account for state-level adversaries, thereby strengthening the protocol against emerging cryptographic threats.

Our work also exposed certain limitations in Tamarin's handling of multiple Diffie–Hellman key exchanges within a single protocol model. While we provided substantial confidence in the security of our design against malicious server and malicious user adversaries, a complete unified model covering all these scenarios remains challenging and is an avenue for further research.

Future Work

Several open questions and improvements arise naturally from this research:

- **Authenticated Public Key Encryption.** Our revised protocol heavily relies on authenticated public key encryption schemes, such as those proposed in RFC 9180. How authentication influences security in KEM combiners remains an under-explored area, and further theoretical and practical studies are needed.

- **PQ-Secure Authenticated KEM.** To the best of our knowledge, no post-quantum secure authenticated KEM has been standardized or rigorously proven secure under game-based models. Our construction lacks a game-based proof, though the symbolic analysis in Tamarin offers partial assurance.

- **Anonymity Guarantees.** While we have analyzed confidentiality and authenticity properties, anonymity guarantees remain outside the main scope of this thesis. Future work could focus on formalizing and verifying anonymity properties.

- **Implementation and Integration.** Finally, an additional security analysis would be advisable once the proposed protocol is implemented. Multiple cryptographic primitives operate in tandem, and their interplay can introduce subtle vulnerabilities that formal methods may not capture in isolation.

Bibliography

[1] 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*, pages 47–67, 2022.

[2] Joël Alwen, Bruno Blanchet, Eduard Hauck, Eike Kiltz, Benjamin Lipp, and Doreen Riepel. Analysing the hpke standard. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 87–116. Springer, 2021.

[3] Joël Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: security notions, proofs, and modularization for the signal protocol. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 129–158. Springer, 2019.

[4] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M Schanck, Peter Schwabe, Gregor Seiler, and Damien Stehlé. Crystals-kyber algorithm specifications and supporting documentation. *NIST PQC Round*, 2(4):1–43, 2019.

[5] Richard Barnes, Karthikeyan Bhargavan, Benjamin Lipp, and Christopher A. Wood. Hybrid Public Key Encryption. RFC 9180, February 2022.

[6] David Basin, Cas Cremers, Jannik Dreier, and Ralf Sasse. Symbolically analyzing security protocols using tamarin. *ACM SIGLOG News*, 4(4):19–30, 2017.

[7] David Basin, Cas Cremers, Jannik Dreier, and Ralf Sasse. Tamarin: Verification of Large-Scale, Real World, Cryptographic Protocols. *IEEE Security and Privacy Magazine*, 2022.

[8] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 566–582. Springer, 2001.

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

[10] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. *Journal of Cryptology*, 33:1914–1983, 2020.

[11] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

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

[13] Roger Dingledine, Nick Mathewson, Paul F Syverson, et al. Tor: The second-generation onion router. In *USENIX security symposium*, volume 4, pages 303–320, 2004.

[14] Signal Foundation. Signal messenger. `https://signal.org/`. Accessed: 2025-01-06.

[15] Federico Giacon, Felix Heuer, and Bertram Poettering. Kem combiners. In *Public-Key Cryptography–PKC 2018: 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part I 21*, pages 190–218. Springer, 2018.

[16] Paul Grubbs, Varun Maram, and Kenneth G Paterson. Anonymous, robust post-quantum public key encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 402–432. Springer, 2022.

[17] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography: principles and protocols*. Chapman and hall/CRC, third edition, 2007.

[18] Felix Linker and David Basin. Soap: A social authentication protocol. *arXiv preprint arXiv:2402.03199*, 2024.

[19] Felix Linker, Ralf Sasse, and David Basin. A formal analysis of apple's imessage pq3 protocol. *Cryptology ePrint Archive*, 2024.

[20] Gavin Lowe. A hierarchy of authentication specifications. In *Proceedings 10th computer security foundations workshop*, pages 31–43. IEEE, 1997.

[21] Luca Maier. Github:a formal analysis of the securedrop protocol. https://github.com/lumaier/securedrop-formalanalysis. Accessed: 2025-01-13.

[22] Freedom of the Press Foundation. Securedrop (github). https://github.com/freedomofpress/securedrop-protocol. Accessed: 2024-11-25.

[23] Freedom of the Press Foundation. Securedrop (homepage). https://securedrop.org/. Accessed: 2024-11-26.

[24] Tor Project. Onion services. https://community.torproject.org/onion-services/overview/. Accessed: 2024-07-10.

[25] Tor Project. Tor specifications. https://spec.torproject.org/. Accessed: 2024-07-10.

[26] Tamarin. Manual. https://tamarin-prover.com/manual/master/tex/tamarin-manual.pdf. Accessed: 2024-11-20.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every written paper or thesis authored during the course of studies. In consultation with the supervisor, one of the following three options must be selected:

○ I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used no generative artificial intelligence technologies[1].

○ I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used and cited generative artificial intelligence technologies[2].

[✓ I confirm that I authored the work in question independently and in my own words, i.e. that no one helped me to author it. Suggestions from the supervisor regarding language and content are excepted. I used generative artificial intelligence technologies[3]. In consultation with the supervisor, I did not cite them.]

Title of paper or thesis:

A Formal Analysis of the SecureDrop Protocol

Authored by:
If the work was compiled in a group, the names of all authors are required.

Last name(s):	First name(s):
Maier	Luca

With my signature I confirm the following:
- I have adhered to the rules set out in the Citation Guide.
- I have documented all methods, data and processes truthfully and fully.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for originality.

Place, date	Signature(s)
Zurich, 13.01.2025	[illegible]

If the work was compiled in a group, the names of all authors are required. Through their signatures they vouch jointly for the entire content of the written work.

[1] E.g. ChatGPT, DALL E 2, Google Bard
[2] E.g. ChatGPT, DALL E 2, Google Bard
[3] E.g. ChatGPT, DALL E 2, Google Bard