The SecureDrop Protocol: End-to-End Encrypted Whistleblowing for All

Giulio Berra
Freedom of the Press Foundation
giulio@freedom.press

Felix Linker
ETH Zurich
flinker@inf.ethz.ch

Luca Maier
ETH Zurich
luca.maier@gmx.com

Cory Francis Myers
Freedom of the Press Foundation
cory@freedom.press

Kenneth G. Paterson
ETH Zurich
kenny.paterson@inf.ethz.ch

Rowen Shane
Freedom of the Press Foundation
ro@freedom.press

Shannon Veitch
ETH Zurich
shannon.veitch@inf.ethz.ch

Abstract

Confidential sources are vital for investigative journalism and thus for holding those in power to account. However, sources often face great risks to their privacy and safety. SecureDrop is a system that enables sources to anonymously contact journalists, including at major news organisations around the world. Despite its widespread use, the current design requires physical servers hosted on premises. While cloud-based deployment would alleviate this burdensome requirement and improve SecureDrop's usability and accessibility, it would also introduce new threats to security that are not addressed by the current design. In particular, a lack of end-to-end encryption presents serious risks in the event that a cloud service provider is coerced into revealing information.

In this work, we present and formally analyse a new protocol for SecureDrop which addresses the challenges of off-premises deployment. Our protocol composes an encryption scheme with hybrid post-quantum guarantees and an identity-hiding message-fetching mechanism to provide strong anonymity guarantees. In contrast to existing systems, we minimise incriminating evidence against whistleblowers by providing message-level deniability and by having sources remain stateless. Our formal security analysis combines the Tamarin prover for symbolic analysis and game-based proofs for computational analysis. Finally, our benchmarks demonstrate that the protocol achieves practical levels of performance in a browser context. The Freedom of the Press Foundation plans to deploy the new protocol, with integration efforts beginning in 2026.

1 Introduction

Confidential sources are integral to journalism's ability to hold power to account, but they often do so at great personal risk. When they contact a journalist, sources risk loss of employment, legal prosecution, imprisonment, and threats to their physical safety and mental health [51, 127, 133]. These risks are present even in countries with legal protections for whistleblowers [95]. To mitigate these risks, a number of systems that provide secure and private communication generally [8, 83, 107, 113, 132] or source-to-journalist communication specifically [54, 60, 70] have been proposed.

SecureDrop [54], first deployed in 2013, is a system that lets sources anonymously and securely contact media organisations and NGOs. Open-source and maintained by Freedom of the Press Foundation (FPF), SecureDrop is used by major news organisations around the world, such as *The Guardian* in the United Kingdom, *The Globe and Mail* in Canada, *The New York Times* in the United States, *Der Spiegel* in Germany, and *Al Jazeera* in Qatar.

However, SecureDrop's current design has several limitations that hinder adoption. Most notably, news organisations must host a SecureDrop deployment on premises. This affords certain benefits: attempts to access confidential source material (e.g., via subpoena) must be directed to the news organisation itself [119]. The news organisation can then choose how to respond, and may invoke their jurisdiction's relevant legal protections [1]. However, maintaining and safeguarding this physical infrastructure is burdensome [80, 112]. In addition, some organisations face risks of physical search and seizure that preclude on-premises hosting entirely [110].

In this work, we present a new protocol for SecureDrop that removes trust assumptions from the SecureDrop deployment so that it can be hosted off premises, for example, in the cloud. Moving to off-premises hosting creates a hard-to-solve challenge. Despite enabling wider use of the system, it also brings severe new threats that the current architecture does not address. In particular, service providers can be coerced to compromise their infrastructure or to reveal information about any whistleblowing services they host. If a legal order is served directly to the provider, a news organisation may not know about it or be able to contest it [126, 130].

Addressing this challenge is far from trivial. SecureDrop's redesign must meet several constraints that are informed by more than a decade of real-world deployment experience, feedback from practitioners, and cumulative research [16, 18, 22, 66, 112]. These requirements rule out using existing systems, both theoretical and deployed. First, secure messaging systems such as Signal [96, 107] and Messaging Layer Security [8] require clients to publish long-term keys and maintain state. In the whistleblowing context, any persistent state, such as a client application, could be used as evidence against a source [16]. Second, systems that provide anonymity assuming a wide user base [38, 132] are not appropriate for low-traffic deployments like SecureDrop, nor are cover-traffic schemes that pose deanonymisation risks [137]. Third, anonymous messaging systems often require multiple servers, specialised hardware, or other complex or costly infrastructure [82, 132]. This poses the same challenge to resource-constrained organisations as an on-premises

design. Finally, for a system as sensitive as SecureDrop, we require well-understood, production-ready cryptographic implementations that will continue to receive long-term support. This rules out approaches that would otherwise be attractive, such as private information retrieval and oblivious message retrieval [29, 90].

All of these constraints have led FPF to a different solution: a protocol that can enable outsourcing the SecureDrop deployment to a third-party cloud server while preventing the server from learning anything about who is communicating with whom or about what. By addressing the privacy and message-interception risks of running a cloud-based whistleblowing system, such a protocol would provide an alternative to existing cloud-based systems and the current on-premises deployment model (see Section 3).

FPF has published two draft designs for such a protocol. The first [66] was based on Signal, with the limitations mentioned above. The second [18] discussed those limitations as motivations for a custom messaging protocol and presented a proof of concept. This work, a collaboration between FPF and ETH Zurich, presents an updated protocol design produced hand-in-hand with formal security analysis. We followed an iterative process combining both the Tamarin prover [116] for symbolic analysis of protocol-level confidentiality and authentication, and game-based, computational proofs to obtain guarantees of the underlying cryptographic primitives and the anonymity properties of the overall protocol. This twin approach is a state-of-the-art case study in combining symbolic and computational proof frameworks, allowing us to benefit from their complementary strengths. The concrete improvements of this work include (non-exhaustively): redesigning around HPKE [9] as a standardised cryptographic building block for encryption; incorporating ML-KEM [102] and X-Wing [7] to upgrade the protocol from classical-only to hybrid post-quantum/traditional security; adopting the Ristretto group to prevent small-order subgroup attacks; fixing protocol-level problems identified by analysis in Tamarin, such as missing domain separation and message agreement; and formalising the threat model (see Section 2.2).

The revised design combines two components. First, it uses a post-quantum/traditional hybrid encryption scheme built from the authenticated modes of HPKE [4] to provide confidential and authenticated messaging. HPKE's anonymity properties, which we establish here, prevent an adversarial server from identifying senders and receivers of messages. While sources being stateless precludes true forward secrecy, we still obtain a weaker forward secrecy-like property against external adversaries, assuming that the server deletes messages in a timely fashion. Second, the design uses a Diffie–Hellman-based message retrieval mechanism [18], which provides anonymity and unobservability: external adversaries (i.e., those who interact with the server using the same public interface available to all users) learn nothing about the messages held by the server. In contrast to existing systems, our design offers plausible deniability for whistleblowers, by having senders remain anonymous and stateless.

Along with the design and its formal analysis, we provide a proof-of-concept implementation that demonstrates the feasibility of our protocol in practice and that uses formally verified codebases wherever possible. Our native, Chrome, and Firefox benchmarks show that the message encryption, decryption, and retrieval mechanisms perform well even in a browser context. Finally, we discuss additional pre-deployment considerations, including verifiability and the need for a browser-based code-integrity mechanism, both of which address the problem of security against a malicious server.

While the new design for SecureDrop is a marked departure from the existing one, this has proven necessary in order to fulfil the twin challenges of enabling cloud deployment whilst maintaining suitable levels of security and anonymity. FPF has committed to implementing the new design, with integration work beginning in 2026 [53], alongside other parallel research and development efforts that will support the redesigned deployment [13, 21].

2 Design Goals

The new SecureDrop protocol provides asynchronous message exchange between journalists and stateless sources and should be easy to deploy. In this section, we present its security requirements and threat model, which were drawn from FPF's and others' prior research on SecureDrop [18, 22, 64] and from the literature on newsroom security and confidential journalistic sources [67, 99, 112, 120].

Critically, the new protocol must not require any on-premises hardware such as the physical servers in the original SecureDrop design: we require a single-server, cloud-deployable system [44, 67]. We refer to this server as the *SecureDrop instance* or the *server*. We refer to the system's users sometimes generically (*users*), sometimes by role (*sources*, *journalists*), and sometimes by what action they are currently taking (*senders*, *receivers*).

2.1 Security Requirements

We define SecureDrop's security requirements by reference to the protocol's functionality: the message exchange between sources and journalists.

Confidentiality: Message contents must remain confidential, and only the intended recipients can access messages.
Authentication: Message recipients must be able to identify a message's sender.
Unlinkability: The adversary must not be able to learn anything about the identities of the senders or recipients of messages.
Unobservability: The adversary must not be able to detect the presence of message exchange or the number of messages exchanged[1].
Deniability: SecureDrop must provide plausible deniability for sources. Sources must be able to deny having sent a particular message, which we call **message-level deniability**, and to deny having even used the system, which we call **source-existence deniability**. We require source-existence deniability even when a source's devices are seized.

Note that authentication and deniability are conflicting requirements and impose a trade-off. Whereas MLS [8] and iMessage PQ3 [89, 121] opt for stronger authentication guarantees, we opt for dishonest receiver deniability [57], similar to the Signal protocol [50, 135]. As is typical of non-interactive protocols [35], at the message level we address offline deniability, also called transcript deniability, discussed in Section 5.1.3. Additionally, requiring source-existence deniability (Section 5.3.2) after device seizure implies that sources must not maintain persistent client-side state [16].

[1] Our terminology of unobservability follows [113].

Sources may use general-purpose tooling (e.g., Tor Browser), but must not use a bespoke application or retain persistent key material other than a memorizable or generically-disguisable passphrase. Following [129], deniability is valued for what it *avoids*: evidence that strongly incriminates a source.

2.2 Adversary Model

The new SecureDrop protocol delivers messages via an intermediary server; the protocol is designed for deployment on top of an anonymizing transport, such as Tor, discussed further in Section 6.2.1. In general, we provide our security requirements against an active network adversary. In our security analysis (Section 5), we additionally consider specific stronger adversaries to precisely capture SecureDrop's security guarantees against certain compromise events. First, we consider that the adversary may learn any participant's key material. Second, we consider the case in which the adversary obtains a quantum computer in the future, which could help them to decrypt previously intercepted communication. Third, we refine the single, active network adversary into three distinct adversaries with increasing capabilities:

External adversary: An adversary that interacts with the server using the same public interface available to all users.

Next, we are particularly concerned about the compromise of the server. In cloud scenarios, compromise might occur via hypervisor-level monitoring or by obtaining disk or RAM snapshots. Disk and memory snapshots are part of normal, non-adversarial procedures, such as automated backups, and should be presumed routine [49]. Such snapshots are often undetectable by the tenant, and we must assume that the adversary can obtain them. Moreover, in the worst case, an adversary may obtain persistent access to the server. This leads to our second and third adversaries:

Passive server adversary (honest-but-curious): An adversary with passive access to the server, e.g., via disk or memory snapshots, or persistent monitoring.

Active server adversary: An adversary who completely compromises the server, potentially modifying data or execution arbitrarily.

We consider timing side-channels out of scope for this work and rely on the transport layer (Section 6.2.1) for network-level anonymity properties, but the risks of both timing side-channels and traffic analysis attacks can be mitigated at the application level. In particular, traffic analysis attacks on Tor [75] can be mitigated by upload size limits and delays, as well as deployment guidance, for instance avoiding use in regions with low Tor user density.

2.3 Use Cases

To illustrate how the requirements address different threat models, we present a set of example organisations operating under varying conditions of risk and deployment. Based on the system requirements, we provide per-organisation deployment recommendations.

Organisation A is a prominent news organisation with multiple physical offices in different cities and a history of publishing high-impact investigative work on the national and international stage. In cloud deployments, it may be targeted by a state-level adversary with *active server adversary* capabilities. Given its high-profile exposure and significant resources, including control of physical spaces, this organisation should deploy exclusively on premises, both for legal protection and to increase the cost of compromises.

Organisation B is a smaller newsroom focused on local corruption. It may operate remotely or share office space, with fewer technical and legal resources. It is primarily threatened by law enforcement agencies with *passive server adversary* capabilities. This organisation is more resource-constrained and less exposed to state-level adversaries than Org. A, and thus can deploy in the cloud.

Organisation C is a news organisation operating under heightened risk of physical raids or coercion in a region with limited protections for the press. When self-hosting on physical infrastructure, it faces potential *active server adversary* threats. However, if able to deploy via a foreign cloud infrastructure provider uncooperative with law enforcement in the news organisation's region, the threat may be downgraded to an *external adversary*. This scenario favours a cloud deployment.

Organisation D is a public institution or company subject to whistleblowing legislation. Since such regulations often do not mandate anonymity, and the primary threat is usually retaliation by an employer, the protocol's protections go beyond regulatory requirements, offering a higher standard by default. This organisation can safely opt for a cloud deployment.

Organisation E is an independent group of investigative journalists that publishes high-profile work. Its risk profile resembles Org. A, while its size and resources are more comparable to Org. B. In this case, deployment choices must be evaluated on a case-by-case basis; however, a confidential-computing-based cloud deployment, as discussed in Section 6.2, may be appropriate in this setting.

3 Related Work

The current incarnation of SecureDrop [54] is our baseline, from which we derive our goals and use cases. In the current SecureDrop design, the Tor network provides confidentiality and anonymity in transit, while messages and files are encrypted on disk via OpenPGP. This design precludes safe cloud-based deployment, since compromised server infrastructure may have access to key material, submitted data, and metadata.

SecureDrop's closest peer is GlobaLeaks [60], which is also designed expressly to facilitate whistleblower communications. However, because it targets a range of use cases, including some within organisations, its security model is more flexible than SecureDrop's. For example, it supports but does not require a Tor onion service for anonymous access via Tor Browser. Although it is designed to be deployed in the cloud, it shares SecureDrop's core limitation of server-side encryption of messages using server-generated keys.

CoverDrop [70] is also designed specifically for whistleblowing, with a messaging client embedded directly within a mobile app. However, anonymity depends on the app's implementation and user distribution, and a multi-server deployment is recommended, ideally with a trusted computing element. This conflicts with our desires for a single-server deployment and to avoid relying primarily on cover traffic, due to deanonymisation risks [137]. Furthermore, CoverDrop's message encryption is not quantum-safe.

FPF's first proof of concept for end-to-end encrypted SecureDrop [66] used the Signal protocol [96, 107]. This approach is especially appealing given the significant scrutiny these protocols have

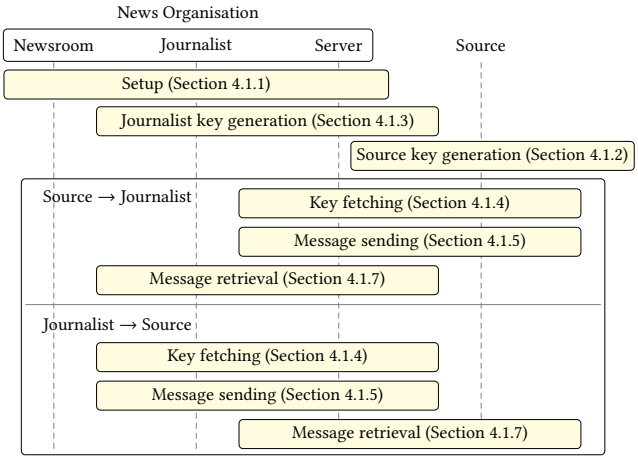


Figure 1: Protocol overview. After setup and key generation, the boxes "Source → Journalist" and "Journalist → Source" may repeat in any order for a long-lived conversation, unless the source opts for one-time drop mode (Section 4.1.6).

received [24, 33, 42] and their widespread use, including by journalists [120]. However, the Signal protocol requires that clients either maintain state locally, which violates our requirement that sources remain stateless; or cache their state at the server, which increases the metadata available on a compromised server [16]. Contact discovery in Signal [97] requires a trusted computing element, which may be difficult for some organisations to deploy.

The Messaging Layer Security standard [8] is intended to be a building-block protocol for secure messaging applications [71]. Its group semantics would accommodate one-to-many groups for each source and multiple journalists. However, it shares Signal's state requirements. In addition, members of MLS groups sign messages, which violates plausible deniability for sources.

Cwtch [83] is a metadata-minimising communication system that eliminates a centralised messaging server altogether by sending messages over connections between Tor onion services. However, Cwtch clients must maintain local state. In addition, the initial key exchange is synchronous, which is not viable for sources and journalists who may not be online at the same time.

In general, the systems we evaluated failed to satisfy our requirements because they (a) do not provide client-side message encryption [60, 131]; (b) require stateful clients [8, 66, 83, 103, 114]; (c) allow servers to see account metadata or count users [8, 60, 66, 114]; (d) require interactive or synchronous conversation bootstrapping [34, 83] and/or message exchange [81, 128, 132]; (e) rely on cover traffic or a minimum user base size [70, 81, 114, 128, 132]; (f) require or recommend deployment using multiple servers [34, 70, 74, 77, 81, 128, 132] or a trusted computing element [70, 97, 114]; or (g) do not provide quantum-resistant encryption [70, 103].

4 Protocol

In this section, we introduce the new SecureDrop protocol. We depict an overview in Figure 1. Newsrooms appoint a small group of journalists to manage their SecureDrop instance; communication between a source and the newsroom is carried out between the source and all journalists in that group. Journalists first enrol in a SecureDrop instance, which authenticates and stores their key material. To initiate communication, a source loads the Web application served by the SecureDrop instance, which generates their keys and obtains the keys of the journalists. A source includes their own public keys in their encrypted messages, enabling journalists to respond to them. When responding to a source, a journalist sends the same message to all other journalists enrolled in the same SecureDrop instance. This acts as a form of group messaging and ensures consistent message patterns between sources and journalists. Sources and journalists fetch messages via a challenge-response protocol that hides their identity from an honest-but-curious server. Clients communicate with the server via an anonymity network such as Tor [48] (see Section 6.2.1).

Building blocks. We use the following building blocks, further defined in Appendix A, and for which we provide specific instantiations in Section 4.2 and Section 4.3:

- A signature scheme $\mathsf{SIG} = (\mathsf{KGen}, \mathsf{Sign}, \mathsf{Vfy})$. We write $\sigma \xleftarrow{\$} \mathsf{Sign}(sk, m)$ for signing and $b \leftarrow \mathsf{Vfy}(vk, m, \sigma)$ for verification.
- A key derivation function $\mathsf{KDF}\colon \{0,1\}^{ikl} \times \{0,1\}^* \xrightarrow{\$} \{0,1\}^{okl}$ and a password-based key derivation function $\mathsf{PBKDF}\colon \{0,1\}^* \xrightarrow{\$} \{0,1\}^{pbkl}$.
- A nonce-based authenticated encryption scheme with associated data $\mathsf{AEAD} = (\mathsf{Enc}, \mathsf{Dec})$. We write $c \leftarrow \mathsf{Enc}(k, nonce, ad, m)$ to encrypt message $m \in \{0,1\}^*$ using key $k \in \mathcal{K}$, nonce $nonce \in \{0,1\}^{nl}$, and associated data $ad \in \{0,1\}^*$, and $m \leftarrow \mathsf{Dec}(k, nonce, ad, c)$ for the corresponding decryption of ciphertext c.
- A nominal group $\mathcal{N} = (\mathbb{G}, g, p, \mathcal{E}_H, \mathcal{E}_U)$, consisting of a group $\mathbb{G}$, a base element $g \in \mathbb{G}$, a prime p, and descriptions of a finite set of honest exponents $\mathcal{E}_H \subset \mathbb{Z}$ and a finite set of exponents $\mathcal{E}_U \subset \mathbb{Z} \setminus p\mathbb{Z}$. This is an abstraction of elliptic curve groups [3].
- A public key encryption scheme $\mathsf{PKE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{Dec})$. We write $c \xleftarrow{\$} \mathsf{Enc}(pk, m, ad, info)$ for encryption of a message $m \in \{0,1\}^*$ with public key pk, associated data $ad \in \{0,1\}^*$, and info $info \in \{0,1\}^*$, and $m \leftarrow \mathsf{Dec}(sk, c, ad, info)$ for the corresponding decryption. Associated data here comes from the fact that our PKE scheme is built using an underlying AEAD scheme.
- An authenticated public key encryption scheme $\mathsf{APKE} = (\mathsf{KGen}, \mathsf{AuthEnc}, \mathsf{AuthDec})$. We write $c \xleftarrow{\$} \mathsf{AuthEnc}(sk_S, pk_R, m, ad, info)$ for encryption of a message $m \in \{0,1\}^*$ with recipient public key pk_R, sender private key sk_S, associated data $ad \in \{0,1\}^*$, and info $info \in \{0,1\}^*$; and $m \leftarrow \mathsf{AuthDec}(sk_R, pk_S, c, ad, info)$ for the corresponding decryption. APKE schemes [3] are akin to signcryption [141]: these incorporate the sender's private key in encryption and the sender's public key in decryption to ensure sender authentication in addition to properties of PKE schemes.

4.1 Protocol Description

4.1.1 Setup. A newsroom setting up a SecureDrop instance first generates a signing and verification key, $(sk_{NR}^{sig}, vk_{NR}^{sig})$ via $\mathsf{SIG.KGen}()$. How sources can authenticate the SecureDrop instance and its vk_{NR}^{sig} is out of scope for this work, but FPF plans to provide support for complementary authentication mechanisms.

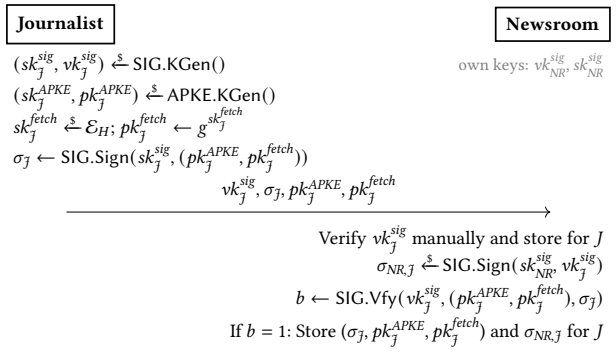


Figure 2: Journalist enrolment procedure

A journalist working with the newsroom enrols in the system by generating their own signing and verification key, which is then signed by the newsroom (Figure 2). Enrolment enables a journalist to later sign ephemeral key bundles for messaging, which can be verified by another user through the chain of signatures beginning with the newsroom's signature on the journalist's signing key. The journalist also generates and signs a long-term APKE key for replying to messages, and a long-term Diffie–Hellman fetching key for retrieving messages. The newsroom verifies the journalist-provided signatures and stores these keys and their corresponding signature.

4.1.2 Source Key Generation. The client application generates a passphrase for a source as a mnemonic phrase of sufficient entropy (e.g., [104, 111, 136]). The source derives a master key from their passphrase, using a password-based KDF: $mk \leftarrow \mathsf{PBKDF}(\texttt{passphrase})$. Then, they derive three private keys from the master key: one for fetching messages sk_S^{fetch}, and two (A)PKE keys for message and metadata encryption, sk_S^{APKE}, sk_S^{PKE}. Each key is derived from the master key mk and a label. Public keys are derived from private keys in the usual way. All source keys are long-term. Generating keys directly from a passphrase ensures that no source key material is stored on the server, contributing to source-existence deniability.

4.1.3 Journalist Key Generation. Journalists generate a number of ephemeral key bundles for messaging. Each key bundle includes an APKE and a PKE public key. The bundles are signed with the journalist's signing key (generated upon enrolment). The server stores the bundles and signatures, while the journalist stores the complete keypairs. This process is illustrated in Figure 3a.

4.1.4 Key Retrieval. Upon each visit to the SecureDrop instance, a user retrieves and verifies keys from the server (Figure 3b). The server selects one ephemeral key bundle per journalist to be sent to the user visiting the SecureDrop instance, and then deletes this from their store of key bundles. This ensures that each bundle is used for at most one message sent to the journalist (ensuring forward secrecy on messages from sources to journalists). The user verifies the received signatures using the newsroom's verification key (available via the application). This key retrieval is required for both sources and journalists so that (1) journalist/source message patterns are indistinguishable, and (2) all journalists managing the instance can be copied on messages sent by other journalists.

4.1.5 Message Sending. A user sends a message as follows (described in full in Figure 3c). Sources send messages to all journalists enrolled in the SecureDrop instance, while journalists send messages to a source while copying all other journalists. All users must send the same number of ciphertexts when sending a message to avoid trivial distinguishing attacks between sources and journalists. Figure 3c uses an index i to denote each recipient of the message. In the following, we describe the process of sending a message to a single recipient.

The sender creates two ciphertexts for the recipient. First, they use an APKE scheme to encrypt their plaintext message and their long-term fetching and PKE public keys (so that a recipient can reply). For this APKE ciphertext, the associated data is set to the `NewsroomID` to prevent replay of messages across different SecureDrop instances, and the info field is set to the recipient's fetching key to ensure that senders and recipients agree upon all keys used in this message exchange. Second, they use an (unauthenticated) PKE scheme to encrypt their APKE public key to the recipient so that the public key need not be sent in the clear, ensuring anonymity with respect to an adversarial server. Here, associated data and info parameters are not used (we maintain the parameters in our syntax for consistency with existing HPKE APIs).

Sources and journalists send messages using the journalists' ephemeral keys as recipient public keys in the APKE encryption. These are obtained in the prior key retrieval process. A source uses their long-term APKE private key derived from their passphrase as the sender private key in the APKE encryption. When replying to messages from a source, a journalist uses their long-term APKE private key (established during enrolment) for the sender private key, along with the source's long-term APKE key (included in a previous message) as the recipient public key.

In addition to standard confidentiality and authentication guarantees, we opt for APKE and PKE schemes that provide deniability (ensuring message-level deniability), and anonymity (which contributes to the unlinkability requirement). Moreover, we select schemes that provide hybrid (post-quantum/traditional) security.

The sender sends these two ciphertexts along with a *hint* to the server. The hint is derived from the fetching key of the recipient, pk_R^{fetch}, and enables recipients to fetch their messages without revealing their identity to the server, contributing to unlinkability. The hint consists of two components: a public key $X = g^x$ and a Diffie–Hellman share $Z = (pk_R^{fetch})^x$. The server stores this message (containing the PKE and APKE ciphertexts, and the hint) along with a random ID that it generates itself.

4.1.6 Extension: One-time Drop Mode. In some cases, sources may choose to use the protocol in a *one-time drop* mode, in which they encrypt a message submission for the intended recipient (e.g., a set of journalists) but do not generate or share any persistent key material (i.e., do not generate a passphrase). This configuration sacrifices the ability to receive replies or authenticate future messages as coming from the same sender, but maximizes deniability. Because the source retains no long-term credential or passphrase, there is no cryptographic evidence linking them to the submission. We do not describe this mode in detail in the protocol figures, as it is a strict subset of the previously described protocol.

Journalist — **Server**

Repeat for number of ephemeral key bundles

$(sk_{J,i}^{APKE_E}, pk_{J,i}^{APKE_E}) \xleftarrow{\$} \text{APKE.KGen}()$
$(sk_{J,i}^{PKE_E}, pk_{J,i}^{PKE_E}) \xleftarrow{\$} \text{PKE.KGen}()$
$\sigma_{J,i} \xleftarrow{\$} \text{SIG.Sign}(sk_J^{sig}, (pk_{J,i}^{APKE_E}, pk_{J,i}^{PKE_E}))$

$(\sigma_{J,i}, pk_{J,i}^{APKE_E}, pk_{J,i}^{PKE_E})$ →

$b \leftarrow \text{SIG.Vfy}(vk_J^{sig}, (pk_{J,i}^{APKE_E}, pk_{J,i}^{PKE_E}), \sigma_{J,i})$
If $b = 1$: store $(\sigma_{J,i}, pk_{J,i}^{APKE_E}, pk_{J,i}^{PKE_E})$ for J

(a) Journalist upload of key bundles

User — **Server**

knows key: vk_{NR}^{sig}

"RequestKeys" →

for all journalists J, select one key bundle, i, at random

$pks \leftarrow \{(vk_J^{sig}, pk_{J,i}^{APKE_E}, pk_{J,i}^{PKE_E}, pk_J^{fetch}, pk_J^{APKE})\}_{\text{for all } J}$
$sigs \leftarrow \{(\sigma_{NR,J}, \sigma_J, \sigma_{J,i})\}_{\text{for all } J}$

for all journalists J, remove key bundle i from storage

← $pks, sigs$

if $\text{SIG.Vfy}(vk_{NR}^{sig}, vk_J^{sig}, \sigma_{NR,J}) = 0$ for some J: abort
if $\text{SIG.Vfy}(vk_J^{sig}, (pk_J^{APKE}, pk_J^{fetch}), \sigma_J) = 0$ for some J: abort
if $\text{SIG.Vfy}(vk_J^{sig}, (pk_{J,i}^{APKE_E}, pk_{J,i}^{PKE_E}), \sigma_{J,i}) = 0$ for some J, i: abort

(b) Key Retrieval Process

Sender — **Server**

known from key retrieval: $pks, sigs$
own keys: $pk_S^{APKE}, pk_S^{PKE}, pk_S^{fetch}, sk_S^{APKE}, sk_S^{PKE}, sk_S^{fetch}$
may know from prior communication: $pk_R^{APKE}, pk_R^{PKE}, pk_R^{fetch}$

if sender is journalist J:
  remove key bundle of J from pks
  $pks \leftarrow pks \cup \{(-, pk_R^{APKE}, pk_R^{PKE}, pk_R^{fetch})\}$

for all $(-, pk_{R,i}^{APKE}, pk_{R,i}^{PKE}, pk_{R,i}^{fetch}) \in pks$

$pt \leftarrow m \| pk_S^{fetch} \| pk_S^{PKE}$
$ct^{APKE} \leftarrow \text{APKE.AuthEnc}(sk_S^{APKE}, pk_{R,i}^{APKE}, pt, \texttt{NewsroomID}, pk_{R,i}^{fetch})$
$ct^{PKE} \leftarrow \text{PKE.Enc}(pk_{R,i}^{PKE}, pk_S^{APKE}, -, -)$
$C_S \leftarrow (ct^{APKE}, ct^{PKE})$
$x \xleftarrow{\$} \mathcal{E}_H$
$X \leftarrow g^x$
$Z \leftarrow (pk_{R,i}^{fetch})^x$

(C_S, X, Z) →

$id \xleftarrow{\$} \{0,1\}^{il}$
Store (id, C_S, X, Z) in db

(c) Message Sending Protocol

Server — **Receiver**

known from key retrieval: $pks, sigs$
own keys if source: $pk_R^{APKE}, pk_R^{PKE}, sk_R^{APKE}, sk_R^{PKE}, pk_R^{fetch}, sk_R^{fetch}$
own keys if journalist: $\{pk_{R,i}^{APKE}, pk_{R,i}^{PKE}, sk_{R,i}^{APKE}, sk_{R,i}^{PKE}\}_{\text{for all } i}, pk_R^{fetch}, sk_R^{fetch}$

fetched ← ∅

← "RequestMessages"

challs ← ∅
for $k \in [0, \text{padsize}]$:
  $r_k \xleftarrow{\$} \mathcal{E}_H$
  if $C_k = (id_k, C_{Sk}, X_k, Z_k)$ in db:
    $Q_k \leftarrow X_k^{r_k}$
  else: // pad with random values up to padsize
    $z_k \xleftarrow{\$} \mathcal{E}_H$; $Z_k \leftarrow g^{z_k}$
    $x_k \xleftarrow{\$} \mathcal{E}_H$; $Q_k \leftarrow g^{x_k r_k}$; $id_k \xleftarrow{\$} \{0,1\}^{il}$
  $idk_k \leftarrow \text{KDF}(Z_k^{r_k}, \texttt{NewsroomID})$
  $eid_k \leftarrow \text{AEAD.Enc}(idk_k, 0^{nl}, -, id_k)$
  challs ← challs ∪ $\{(eid_k, Q_k)\}$

challs →

cids = ∅
for all $k \in [0, |\text{challs}| - 1]$:
  $(eid_k, Q_k) \leftarrow \text{challs}[k]$
  $tk_k \leftarrow \text{KDF}(Q_k^{sk_R^{fetch}}, \texttt{NewsroomID})$
  $res_k \leftarrow \text{AEAD.Dec}(tk_k, 0^{nl}, -, eid_k)$
  if $res_k \neq \perp$: cids ← cids ∪ $\{res_k\}$
tofetch ← cids \ fetched
if tofetch ≠ ∅ : $cid \leftarrow$ tofetch[0]

← cid

C_{Sk} s.t. $id_k = cid$ →

$(ct^{APKE}, ct^{PKE}) \leftarrow C_{Sk}$
$i \leftarrow 0$
do:
  $++i$; $sk_R^{PKE} \leftarrow sk_{R,i}^{PKE}$; $sk_R^{APKE} \leftarrow sk_{R,i}^{APKE}$
  $pk_S^{APKE} \leftarrow \text{PKE.Dec}(sk_R^{PKE}, ct^{PKE}, -, -)$
until $pk_S^{APKE} \neq \perp$
$pt \leftarrow \text{APKE.AuthDec}(sk_R^{APKE}, pk_S^{APKE}, ct^{APKE}, \texttt{NewsroomID}, pk_R^{fetch})$
$m \| pk_S^{fetch} \| pk_S^{PKE} \leftarrow pt$
if $(-, pk_S^{APKE}, -, -) \notin pks$: discard message
fetched ← fetched ∪ $\{cid\}$
if tofetch \ $\{cid\} \neq \emptyset$:
  repeat from "RequestMessages"

(d) Message Retrieval Protocol. Journalist-only code is highlighted in blue. Source-only code is highlighted in red.

Figure 3: Protocol Descriptions for Key Bundle Upload and Retrieval, and Message Sending and Retrieval

4.1.7 Message Retrieval. Clients can query the server for new messages sent to them (Figure 3d). The server will then run a challenge-response protocol using the hint included by the sender upon message submission. This protocol ensures that (1) clients only receive messages intended for them, while (2) the messaging server does not learn which client a message was intended for (enabling unlinkability), assuming it honestly follows the protocol.

The server generates a series of challenges containing encrypted IDs for each message it currently holds. For each ID, the server generates a new DH share g^r. Then, it encrypts the ID using a key derived from the fetching hint included in the message and its own ephemeral component, Z^r. The server's ephemeral component ensures that each response to a request for messages contains a new set of challenges. Furthermore, the server generates random, encrypted IDs up to a maximum number of messages (this number is denoted padsize). This ensures that an external adversary cannot observe differences in the number of ciphertexts on the server. In particular, the server sends only the encrypted IDs, and not the corresponding APKE/PKE ciphertexts, so an external adversary does not obtain ciphertexts that may leak information about the plaintext on future user compromise; this facilitates the unobservability goal.

The receiver attempts to decrypt each of these IDs, using a key derived from the DH share sent by the server, $Q = X^r$, and their own fetching key, sk_R^{fetch}. If an ID decrypts correctly, they fetch the corresponding ciphertext from the server using the decrypted ID. The recipient can then decrypt the ciphertext, by using their long-term APKE and PKE key if they are a source, or by trial-decrypting with their set of ephemeral key bundles if they are a journalist.

If the recipient has multiple new messages, they restart the message retrieval process for each message ID. This prevents a server from trivially learning how many messages are retrieved by a given user. Adversarial servers can try to infer that a user is fetching a set of messages from traffic analysis (see Section 6.2.2 for mitigations).

4.2 APKE and PKE Constructions

Our APKE and PKE constructions are built using the *hybrid public key encryption* (HPKE) specification [9]. We build upon the following primitives, further defined in Appendix A:

- A key encapsulation mechanism (KEM) KEM = (KGen, Encap, Decap). We write $(c, K) \xleftarrow{\$} \mathsf{Encap}(pk)$ for encapsulation and $K \leftarrow \mathsf{Decap}(sk, c)$ for decapsulation.
- An authenticated key encapsulation mechanism (AKEM) [3] AKEM = (KGen, AuthEncap, AuthDecap). We write $(c, K) \xleftarrow{\$} \mathsf{Encap}(sk_S, pk_R)$ for encapsulation with a sender private and recipient public key, and $K \leftarrow \mathsf{Decap}(sk_R, pk_S, c)$ for decapsulation with a sender public key, recipient private key, and ciphertext.

We restate the pre-shared key APKE scheme pskAPKE from [4] in Figure 4. The definition of pskAPKE[AKEM, KS, AEAD] corresponds exactly to the description of `mode_auth_psk` of HPKE [9], with the exception that in [9] an identifier for each pre-shared key is input into KS. Following [4], we abstract away this identifier and consider it to be encoded as part of *info*. Here, KS is a two-keyed function that will be instantiated with HKDF. We establish the split-key PRF security of KS in Appendix B, under appropriate assumptions of the underlying compression function.

pskAEnc($sk, pk, psk, m, ad, info$):
1 $(c_1, K_1) \xleftarrow{\$} \mathsf{AKEM.AuthEncap}(sk, pk)$
2 $(k, nonce) \leftarrow \mathsf{KS}(K_1, psk, info)$
3 $c' \leftarrow \mathsf{AEAD.Enc}(k, nonce, ad, m)$
4 return (c_1, c')

pskADec($pk, sk, psk, (c_1, c'), ad, info$):
5 $K_1 \leftarrow \mathsf{AKEM.AuthDecap}(sk, pk, c_1)$
6 $(k, nonce) \leftarrow \mathsf{KS}(K_1, psk, info)$
7 $m \leftarrow \mathsf{AEAD.Dec}(k, nonce, ad, c')$
8 return m

Figure 4: Encryption and decryption functions of pskAPKE[AKEM, KS, AEAD] **[4, 9], built from an AKEM** AKEM**, two-keyed function** KS**, and AEAD scheme** AEAD**.**

SD-APKE.KGen():
1 $(sk_1, pk_1) \xleftarrow{\$} \mathsf{AKEM.KGen}()$; $(sk_2, pk_2) \xleftarrow{\$} \mathsf{KEM_{PQ}.KGen}()$
2 $sk \leftarrow (sk_1, sk_2)$; $pk \leftarrow (pk_1, pk_2)$
3 return (sk, pk)

SD-APKE.AuthEnc($sk_S = (sk_1, sk_2), pk_R = (pk_1, pk_2), m, ad, info$):
4 $(c_2, K_2) \xleftarrow{\$} \mathsf{KEM_{PQ}.Encap}(pk_2)$
5 $(c_1, c') \xleftarrow{\$} \mathsf{pskAEnc}(sk_1, pk_1, K_2, m, ad, c_2 \| info)$
6 return $((c_1, c'), c_2)$

SD-APKE.AuthDec($sk_R = (sk_1, sk_2), pk_S = (pk_1, pk_2), ((c_1, c'), c_2), ad, info$):
7 $K_2 \leftarrow \mathsf{KEM_{PQ}.Decap}(sk_2, c_2)$
8 $m \leftarrow \mathsf{pskADec}(pk_1, sk_1, K_2, (c_1, c'), ad, c_2 \| info)$
9 return m

Figure 5: SecureDrop APKE, SD-APKE[AKEM, $\mathsf{KEM_{PQ}}$, AEAD, KS]**, built from an AKEM** AKEM**, post-quantum KEM** $\mathsf{KEM_{PQ}}$**, AEAD scheme** AEAD**, and two-keyed function** KS

SD-PKE.Enc(pk, m):
1 $(c, K_3) \xleftarrow{\$} \mathsf{KEM_H.Encap}(pk)$
2 $(k, nonce) \leftarrow \mathsf{KS}(K_3, -, -)$
3 $c' \xleftarrow{\$} \mathsf{AEAD.Enc}(k, nonce, -, m)$
4 return (c, c')

SD-PKE.Dec($sk, (c, c')$):
5 $K_3 \leftarrow \mathsf{KEM_H.Decap}(sk, c)$
6 $(k, nonce) \leftarrow \mathsf{KS}(K_3, -, -)$
7 $m \leftarrow \mathsf{AEAD.Dec}(k, nonce, -, c')$
8 return m

Figure 6: SecureDrop PKE, SD-PKE[$\mathsf{KEM_H}$, AEAD, KS]**, built from a hybrid KEM** $\mathsf{KEM_H}$**, AEAD scheme** AEAD**, and two-keyed function** KS**;defined according to HPKE Base mode [9].** SD-PKE.KGen() **is equivalent to** $\mathsf{KEM_H.KGen}()$**.**

We define an APKE scheme, SD-APKE, using `mode_auth_psk` of HPKE. Our scheme first uses a post-quantum KEM to generate a shared secret for the pre-shared key (PSK) input in an instance of HPKE in `mode_auth_psk`. This is a slight alteration of a construction from prior work [4]. In comparison to their construction, we use a KEM to generate the PSK instead of an AKEM, thus achieving only classical authentication guarantees. While there are recent proposals for hybrid or post-quantum AKEMs [56, 73], the constructions are either significantly more expensive in communication and computation costs, or make use of non-standard building blocks. Due to a desire for immediate deployability, we opt for an approach that is feasible to implement with existing HPKE APIs.

Our PKE construction, denoted SD-PKE, uses an unauthenticated instance of HPKE to encrypt the sender's (APKE) public key, as described in [9, §9.9]. The scheme makes no changes to the `Base` mode of HPKE and assumes that the underlying KEM is implicitly rejecting. The complete details of our APKE and PKE constructions are given in Figures 5 and 6, respectively.

Requirement	Against Adversary	PQ	Primitive-Level	Protocol-Level
Confidentiality	Active Server	✓	Thm. 5.1	Fig. 7
Authentication	Active Server	✗	Thm. 5.2	Fig. 8
Message Deniability	Active Server	✓	Sec. 5.1.3	–
Source-Existence Den.	Passive Server	✗	–	Sec. 5.3.2
Unlinkability	Passive Server	✗	–	Thm. 5.3
Unobservability	External	✗	–	Sec. 5.3.3

Table 1: System security properties by adversary, assuming that no cryptographic keys are revealed.

4.3 Instantiating the Protocol

The post-quantum KEM used to generate the pre-shared key for use in the APKE scheme is instantiated with ML-KEM-768 [102] and the AKEM with DH-AKEM [3]. The hybrid KEM used in SD-PKE is instantiated with X-Wing [7]. We use HKDF [79] in the key derivation function KS (Figure 16) and to generate source private keys from the master key. The AEAD scheme is instantiated with AES-GCM [100]. The nominal group used in the message retrieval mechanism is instantiated with Ristretto over Curve25519 [45].

5 Security Analysis

In this section, we provide security proofs of confidentiality, authentication, deniability, and unlinkability for the SecureDrop protocol. Table 1 provides an overview of our security analysis. With respect to the post-quantum (PQ) guarantees summarized in Table 1, confidentiality is ensured by our choice of hybrid KEM (Section 5.1.1) and message-level deniability is guaranteed against both quantum and classical adversaries by the deniability of the underlying PKE scheme (Section 5.1.3). We do not provide authentication against an active quantum attacker. We rely on a DH-based mechanism for message fetching; consequently, the unlinkability and unobservability guarantees only hold against classical adversaries.

We provide both pen-and-paper proofs in the computational model and machine-checked proofs in the symbolic model using the Tamarin prover [116], a state-of-the-art protocol verifier. For our computational analysis, we use a game-playing framework [12] and existing security notions where possible. We provide a concrete (rather than asymptotic) security treatment [10], giving reductions between adversaries and bounding their difference in advantage. The bound provides real numbers that may, in turn, be used to select appropriate parameters. In contrast to the *probabilistic* computational setting, the symbolic model makes some abstractions and is *possibilistic*, i.e., the adversary can either learn a given term or they cannot. The adversary cannot obtain partial knowledge of a term or succeed in an attack only with some probability. The upside of these abstractions is that Tamarin can consider larger or entire systems and that proofs are machine-checked.

In Section 5.1, we begin with game-based proofs of primitive-level confidentiality, authentication, and deniability of SD-APKE and SD-PKE. Then, in Section 5.2, we use those results to prove protocol-level confidentiality and authentication in the symbolic model using Tamarin. For this, we model the SecureDrop protocol and encode the security properties of SD-APKE and SD-PKE as axioms to verify that the protocol uses the primitives correctly. For protocol-level anonymity guarantees, we introduce a new game-based definition and proof of unlinkability in Section 5.3.

5.1 Computational Analysis of SD-APKE and SD-PKE

Our security analysis of the cryptographic primitives underlying the SecureDrop protocol follows standard computational analysis techniques. We rely on existing analyses of HPKE, including its pre-shared key variant, which were performed in prior work [3, 4]. SD-PKE is defined according to the Base mode of HPKE, which follows a standard KEM/DEM style construction [39], so following [3] we focus on our authenticated construction, SD-APKE.

We consider two existing security notions for APKE schemes: Insider-CCA captures that an adversary cannot distinguish two ciphertexts, and Outsider-Auth captures that an adversary cannot forge a ciphertext corresponding to an honest recipient's public key and an unrevealed sender's private key. We defer the definitions to Appendix A. Additionally, we consider deniability and anonymity properties. Cumulatively, these guarantees ensure that an adversarial server, whether passive or active, learns no information about the plaintext messages or the identities of the relevant users by inspecting the ciphertexts generated by SD-APKE.

5.1.1 Confidentiality. The AuthEnc algorithm in SD-APKE follows the specification of `mode_auth_psk` in [9]. Prior analysis of the pre-shared key mode of HPKE [4] demonstrates that one can build a hybrid APKE using the pre-shared key mode of HPKE, by first using an authenticated KEM and then the resulting KEM shared secret as the pre-shared key of the pskAPKE, similar to a KEM combiner. Our construction differs slightly, in that only the second KEM is authenticated; however, the proof follows similarly [4, Theorem 5].

We obtain a hybrid bound: one case reducing to the security of the (unauthenticated, post-quantum) KEM and the second reducing to the security of the (classical) AKEM. This reflects that the scheme is at least as hard to break as the harder of the two components, as is standard in approaches to post-quantum security [25]. We omit query numbers for simplicity and defer the proof to Appendix C.

Theorem 5.1 (Insider-CCA of SD-APKE). *Let* $\mathsf{KEM}_{\mathsf{PQ}}$ *be a KEM,* AKEM *an AKEM,* KS *a two-keyed function, and* AEAD *an AEAD scheme. Let* SD-APKE = SD-APKE[AKEM, $\mathsf{KEM}_{\mathsf{PQ}}$, AEAD, KS] *be as defined in Figure 5. For any* Insider-CCA *adversary* $\mathcal{A}$ *against* SD-APKE *we give algorithms* $\mathcal{B}_1$–$\mathcal{B}_3$ *and* C_1–C_2 *such that*

$$\mathsf{Adv}^{\mathsf{Insider\text{-}CCA}}_{\mathsf{SD\text{-}APKE}}(\mathcal{A}) \le \min\{2 \cdot \mathsf{Adv}^{\mathsf{IND\text{-}CCA}}_{\mathsf{KEM}_{\mathsf{PQ}}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{skPRF}}_{\mathsf{KS},2}(\mathcal{B}_2),$$
$$2 \cdot \mathsf{Adv}^{\mathsf{Insider\text{-}CCA}}_{\mathsf{AKEM}}(C_1) + \mathsf{Adv}^{\mathsf{skPRF}}_{\mathsf{KS},1}(C_2)\} + \mathsf{Adv}^{\mathsf{CCA}}_{\mathsf{AEAD}}(\mathcal{B}_3).$$

Concerning our choices of instantiations (Section 4.3), the split-key PRF security of KS is discussed in Appendix B. Various proofs of the IND-CCA security of ML-KEM are given in [2, 117], and the Insider-CCA security of DH-AKEM is proven in [3].

The server may attempt offline dictionary attacks on source passphrases and then attempt to decrypt messages. Also, due to the requirement that sources maintain no state, we do not obtain forward secrecy of messages sent from journalists to sources. In particular, if an adversarial server obtains a source passphrase, it can decrypt messages sent to that source. On the other hand, the use of one-time ephemeral keys for journalists guarantees forward secrecy

of messages from sources to journalists. Furthermore, under the assumption that the server deletes messages in a timely manner, we obtain a weaker notion of forward secrecy with respect to external adversaries (shown in Section 5.2.3).

5.1.2 Sender Authentication. The *outsider authenticity* notion, denoted Outsider-Auth, captures the inability of an adversary to forge ciphertexts that appear to come from an honest sender, to an honest receiver [3]. Outsider-Auth of SD-APKE follows directly from the proof of Outsider-Auth of the combined AKEM, proven in [4, Theorem 6]. Notably, our protocol differs from the protocol in [4, Listing 17] in that the KEM we use to generate the PSK is not authenticated. So, rather than obtaining a hybrid guarantee, the authentication relies solely on the (classical) assumptions underlying the AKEM. That is, only one of two bounds proven in [4, Theorem 6] carries over to our construction. The bound relies on Outsider-Auth and Outsider-CCA of the AKEM; these properties were proven for DH-AKEM in [3]. The proof is the same, so we omit the details. As before, we omit query numbers for simplicity.

Theorem 5.2 (Outsider-Auth of SD-APKE [3])**.** *Let* $\mathsf{KEM}_{\mathsf{PQ}}$ *be a KEM,* AKEM *an AKEM,* KS *a two-keyed function, and* AEAD *an AEAD scheme. Let* SD-APKE = SD-APKE[AKEM, $\mathsf{KEM}_{\mathsf{PQ}}$, AEAD, KS] *be as defined in Figure 5. For any* Outsider-Auth *adversary* $\mathcal{A}$ *against* SD-APKE*, we give algorithms* $\mathcal{B}_1$–$\mathcal{B}_4$ *such that*

$$\mathsf{Adv}^{\text{Outsider-Auth}}_{\text{SD-APKE}}(\mathcal{A}) \le \mathsf{Adv}^{\text{Outsider-CCA}}_{\text{AKEM}}(\mathcal{B}_1) + \mathsf{Adv}^{\text{Outsider-Auth}}_{\text{AKEM}}(\mathcal{B}_2) + \mathsf{Adv}^{\text{skPRF}}_{\text{KS},2}(\mathcal{B}_3) + \mathsf{Adv}^{\text{INT-CTXT}}_{\text{AEAD}}(\mathcal{B}_4).$$

5.1.3 Message-level Deniability. Cryptographic deniability captures the ability of a sender to plausibly deny having sent a message to a receiver; like most non-interactive protocols, we address offline deniability [28, 35]. *Dishonest receiver deniability* of a KEM requires that there exist a simulator that can generate a ciphertext and shared secret indistinguishable from those produced by Encap, given the receiver's private key. Deniability of AKEMs is discussed in [57], along with formal definitions. Deniability of SD-APKE follows directly from the fact that DH-AKEM has dishonest receiver deniability (as suggested in [57]). This property carries over directly to our use of DH-AKEM in PSK Mode of HPKE.

An immediate consequence of ensuring deniability is that the protocol is susceptible to key compromise impersonation (KCI) attacks [9]. The tension between deniability and resistance to KCI is well understood [26, 62]. It is for these reasons that we consider *outsider* authenticity, as opposed to *insider* authenticity notions (allowing for key compromises of honest parties) that cannot be achieved by HPKE's APKE construction [3]. In our design, we explicitly privilege deniability over KCI resistance, since protecting sources against retrospective attribution is our primary goal.

Furthermore, the system does not provide conversation-level deniability, since the source's advertisement of their key material to the journalist constitutes a declaration of contact. In contrast, systems like Signal achieve conversation-level deniability by making all key material advertisements public through the server.

5.1.4 Anonymity. Next, we consider anonymity properties of SD-APKE. We prove a multi-user notion of anonymity for the APKE scheme, mu-ANO-CPA (Figure 19). This property captures that an adversary observing an arbitrary number of APKE ciphertexts cannot distinguish which of two users' key material was used in generating the ciphertext(s). Since we consider only an honest-but-curious adversary for our later analysis of message unlinkability (Section 5.3), we need only a CPA-style notion. Our proof of mu-ANO-CPA relies on a weak anonymity property of the underlying AKEM, which holds in a classical (not post-quantum) setting. We defer the complete analysis to Appendix D.

5.2 Symbolic Analysis

Tamarin [116] is an automated protocol verifier that uses constraint solving and operates in the symbolic model. In this section, we describe how we model the SecureDrop protocol in Tamarin, and how we formalise protocol-level confidentiality and authentication. Modelling and verifying SecureDrop in its full complexity was far from trivial. For example, we were required to implement an optimisation for Tamarin's reasoning about Diffie–Hellman exponentiations [88]. As of writing, this optimisation has been approved and integrated into the Tamarin codebase by its maintainers.

5.2.1 Protocol and Adversary Model. In our Tamarin model, we comprehensively model the SecureDrop protocol and all its steps as described in Section 4. Our protocol model allows for unboundedly many participants (newsrooms, journalists, and sources) and unboundedly many (parallel) protocol runs. We allow the adversary to compromise the key material of any participant and to obtain read access to, or entirely control, a SecureDrop instance. We assume that sources can obtain the authentic public keys of newsrooms and that users connect to SecureDrop instances via a secure channel, modelling communication over Tor onion services. The secure channel authenticates the server but not clients.

We abstract message submissions as clients storing their submission directly on the server. The adversary may submit arbitrary messages, and can access submissions of compromised SecureDrop instances. We chose this model so that we can identify individual submissions both on the client and server side in auxiliary lemmas, e.g., the model allows us to express when a particular client submission was received from the SecureDrop instance.

We also model that SecureDrop instances delete submissions after they have been fetched. In reality, SecureDrop instances may serve submissions repeatedly, but only for a limited time.

5.2.2 Model of Cryptographic Primitives. Recent work suggests a new approach for more accurately capturing cryptographic primitives in the symbolic model [32, 40, 41, 72]. Instead of using equations to model cryptographic primitives, one formalises their properties, proven in the computational model, as axioms. We follow this approach and model that the APKE scheme provides correctness, Insider-CCA, and Outsider-Auth and that the PKE scheme provides IND-CCA and correctness (see Section 5.1). Our model additionally includes an oracle that models KCI attacks (see Section 5.1.3).

Honest protocol participants and the adversary can generate APKE and PKE ciphertexts using the `APKE_Enc` and `PKE_Enc` functions. `APKE_Enc` has five arguments: sender private key, recipient public key, plaintext, authenticated data, and info string. `PKE_Enc` has the same arguments but omits the sender key material as well as the unused authenticated data and info string.

Our model includes functions and associated equations that allow the adversary to decrypt ciphertexts provided they know the corresponding secret key material. Honest participants use *action facts* to decrypt messages. Action facts are events that label protocol model transitions, and formulas and axioms refer to these events. When honest participants decrypt messages, they choose the resulting plaintext from an APKE/PKE message space, and our axioms formalise how these plaintexts correspond to the decrypted ciphertext. For example, they formalise when decryption fails or when the resulting plaintext corresponds to the originally encrypted plaintext. Thus, our axioms capture the conditions in which the schemes are proven to provide their guarantees, and whenever these conditions are not met, we effectively allow the adversary to choose the results of a decryption.

5.2.3 Formalised Security Properties. We prove protocol-level confidentiality and authentication guarantees, which we formalise as message secrecy and non-injective agreement [91], respectively. We provide our formalisation of confidentiality in Figure 7 and authentication in Figure 8. Observe that our properties do not explicitly refer to the event that a source reveals their passphrase to the adversary. Our model captures that sources may reveal their passphrase, but we annotated the corresponding state transition rule with the source revealing their APKE, PKE, *and* fetching keys. For presentation purposes, we provide slightly abstracted properties. For example, arguments in square brackets only apply to journalist events. Complete formalisations are available in our artifact [15].

Message secrecy (Figure 7) formalises that the adversary cannot learn submitted messages. `K(x)` is a built-in Tamarin event, modelling that the adversary knows a term `x`. For journalist submissions, we assume that they are sent to an honest source (line 4). The property lists two cases in which the adversary can learn a submitted message. For source submissions, the adversary can learn the message contents by compromising the corresponding newsroom or journalist signing keys, then using them to sign adversary-controlled key material (lines 6-7). Otherwise, the adversary can only obtain a submission when they compromise both the recipient's APKE key (line 8) and the fetching mechanism (lines 9-15).

The fetching mechanism in turn can be compromised in three ways, plus two for journalist submissions. (A) The adversary compromises the recipient's fetching key before the submission is fetched and thus deleted (lines 12-13). This establishes a weak notion of forward secrecy as an external adversary never obtains the ciphertexts themselves. In reality, the server will delete the messages some time after they have first been fetched, but to the same effect. (B) and (C) the adversary compromises the SecureDrop instance, either with read-only access or with full compromise (lines 14-15). For journalist submissions, the adversary can also break the fetching mechanism by (D) compromising the journalist's ephemeral APKE key that the previous source's message was encrypted to (line 11) or (E) learning the source's long-term APKE key (which they must learn anyway, see above) *before* the journalist's message submission (line 9). In both cases, the adversary can forge a message to the journalist appearing to originate from the source, allowing them to include an adversary-controlled fetching key.

Non-injective agreement (Figure 8) formalises that for every message-receive event (line 2), there is a corresponding message

```
1  All id s sPk nr rPk rFetchPk m #t.
2      ( Submission(id,s,[sPk],nr,rPk,rFetchPk,m) @ #t)
3  ==>   (not Ex #x. K(m) @ #x)
4      | (s = 'journ' & not Ex #x. Source(rPk) @ #x)
5      | (s = 'source'
6        & ( (Ex #x. RevealNRKey(nr) @ & #x < #t)
7          | (Ex #x. RevealJournSIG(nr,_) @ #x & #x < #t)))
8      | ( Ex #x. RevealAPKEKey(rPk) @ #x
9          & ( (s = 'journ' & #x < #t)
10           | (s = 'journ'
11             & Ex #x. RevealEAPKEKey(sPk) @ #x & #x < #t)
12           | (Ex #x #y. #y < #x & Fetched(id) @ #x
13              & RevealFetchKey(rFetchPk) @ #y)
14           | (Ex #x. SDInstanceReadAccess(nr) @ #x)
15           | (Ex #x. SDInstanceCompromise(nr) @ #x)))
```

Figure 7: SecureDrop's protocol-level confidentiality property, modelled as message secrecy.

```
1  All r sPk rPk nr m #t.
2      Received(r,sPk,rPk,nr,m) @ #t
3  ==>   (Ex #x. Submit(sPk,rPk,nr,m) @ #x & #x < #t)
4      | (r = 'journ'
5        & ( (not Ex #x. Source(sPk) @ #x)
6          | (Ex #x. RevealEAPKEKey(rPk) @ #x & #x < #t)
7          | (Ex #x. RevealAPKEKey(sPk) @ #x & #x < #t)))
8      | (r = 'source'
9        & ( (Ex #x. RevealAPKEKey(rPk) @ #x & #x < #t)
10         | (Ex #x. RevealAPKEKey(sPk) @ #x & #x < #t)
11         | (Ex #x. RevealNRKey(nr) @ #x & #x < #t)
12         | (Ex #x. RevealJournSIG(nr,_) @ #x & #x < #t)))
```

Figure 8: SecureDrop's protocol-level authentication property, modelled as non-injective agreement [91].

submission and that the participants agree on the newsroom and keys used (line 3), assuming that neither the sender nor receiver APKE key was compromised (lines 6, 7, 9, 10). We assume that messages received by journalists appear to originate from an honest source (line 5) as submissions by the adversary and using their own key material trivially violate the property. For messages received by sources, authentication can again be violated when the newsroom's or a journalist's signing key is compromised (lines 11-12).

5.3 Analysis of Protocol-Level Anonymity

In this section, we consider anonymity properties of the SecureDrop protocol. For an honest-but-curious server, we provide a game-based indistinguishability-style notion capturing that messages viewed by the server are *unlinkable*. This means that a server which follows the protocol honestly, yet can control incoming messages, learns nothing about the identities of the senders or recipients. Furthermore, we argue that an external adversary can observe no information about the state of messages held by the server. This means that without knowing any honest participants' information, they learn nothing about the message patterns or the contents of the messages sent, capturing *unobservability*.

5.3.1 Unlinkability. We provide a game-based proof that messages viewed by an honest-but-curious server are *unlinkable* with respect to the game $G_{\Pi}^{\text{unl-hbc}}$ (Figure 20). At a high level, our definition allows an adversary to observe messages sent between a set of users of their choice and to observe the honest transcript generated

by a user of their choice fetching their messages. The adversary has access to three challenge oracles: ChallSend, ChallRecv, and ChallFetch. A challenge bit chosen by the game determines which of two provided (pairs of) identities are used in message sending (as sender or receiver) and message retrieval. To win the game, an adversary observing an honest execution must distinguish which of its chosen identities are being used.

The proof relies on the security of the password-based key derivation function and the underlying passphrase distribution, the multi-user anonymity property of SD-APKE (Section 5.1.4), anonymity of the KEM used in SD-PKE, IND-CPA of the AEAD scheme, and a decisional Diffie–Hellman assumption on the group used for the message retrieval mechanism. We therefore do not obtain post-quantum unlinkability. We defer the details to Appendix E.

Theorem 5.3. *Let* SD *be the protocol defined in Figures 3c and 3d, over nominal group* $\mathcal{N}$ *and passphrase distribution* $\mathcal{P}$*, and using* SD-APKE[AKEM, $\mathsf{KEM}_{\mathsf{PQ}}$, AEAD, KS] *and* SD-PKE[$\mathsf{KEM}_{\mathsf{H}}$, AEAD, KS]*, as defined in Figures 5 and 6. Then, for any adversary* $\mathcal{A}$ *against the* unl-hbc *security of* SD *using at most* q_{NU} *queries to* NewUser*,* q_{CS} *queries to* ChallSend*,* q_{CR} *queries to* ChallRecv *and* q_{CF} *queries to* ChallFetch*, we give algorithms* $\mathcal{B}_1$–$\mathcal{B}_7$ *such that*

$$\begin{aligned}\mathsf{Adv}^{\mathsf{unl\text{-}hbc}}_{\mathsf{SD}}(\mathcal{A}) \le\ & \mathsf{Adv}^{\mathsf{pwKDF}}_{\mathsf{PBKDF},\mathcal{P},q_{\mathrm{NU}}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{KDF}}(\mathcal{B}_2) \\ & + \mathsf{Adv}^{\mathsf{mu\text{-}ANO\text{-}CPA}}_{\mathsf{SD\text{-}APKE}}(\mathcal{B}_3) + 2\cdot(q_{\mathrm{CS}}+q_{\mathrm{CR}})\cdot\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{KEM_H}}(\mathcal{B}_4) \\ & + q_{\mathrm{CS}}\cdot\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathsf{AEAD}}(\mathcal{B}_5) + q_{\mathrm{CR}}\cdot\mathsf{Adv}^{\mathsf{wANO\text{-}CPA}}_{\mathsf{KEM_H}}(\mathcal{B}_6) \\ & + 2\cdot q_{\mathrm{CR}}\cdot\mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{N}}(\mathcal{B}_7).\end{aligned}$$

An active adversarial server can violate unlinkability of messages. During the message retrieval protocol, the server can derive a key from some recipient's public fetching key and a known ephemeral value, i.e., $pk_A^{fetch^r}$ for some r known to the server. Then, the server can generate a random message ID, rid, and encrypt this ID using the previously derived key. In the set of encrypted IDs sent to the user fetching messages, the server includes the encrypted rid and the public key g^r. If the server obtains rid as a response from the user then they learn that the user fetching messages corresponds to the user with public fetching key pk_A^{fetch}. By scaling this attack to include a random encrypted ID for every known public key and correlating this behaviour with retrievals of valid ciphertexts, the server can learn the recipient of a given message. We discuss possible mitigations in Section 6.2.2.

Beyond active attacks, an adversarial server can attempt to brute force private fetching keys in an offline manner (using the values X, Z included in the message sending protocol) in an attempt to identify recipients of messages. Lastly, we note that the message retrieval mechanism is not quantum-safe. Transitioning this to a post-quantum solution requires an analogous KEM-based solution that mirrors the commutativity of Diffie–Hellman. At the same time, this triple Diffie–Hellman approach is simple to implement and efficient to compute. We consider this a pragmatic trade-off, as a future break of this component of the protocol would leak to an adversary the (public key) identities of recipients of ciphertexts, but not the contents of messages themselves.

5.3.2 Source-Existence Deniability. The protocol provides source-existence deniability as a corollary of unlinkability while the adversary does not have the source's private key material. Source passphrases are designed to be memorizable or disguisable; further, a set of keys bootstrapped from an ersatz passphrase would execute the protocol identically to that of a previously-generated key with no pending messages, meaning application-level duress features are readily implementable. If, however, the adversary learns source private key material, source-existence deniability depends on whether the adversary can obtain any of that source's messages or challenges. An adversary active while the source is operating can collect such evidence, and subsequently prove the source's existence. Source-existence deniability therefore protects a whistleblower from after-the-fact investigations, but offers no protection if the source's keys are compromised and the adversary has been monitoring the instance or can retrieve historical data.

5.3.3 Unobservability. We obtain unobservability guarantees under external adversaries, capturing that an external adversary cannot detect the presence of message exchange. This is because a user with no access to honest users' private key material can only view encrypted ciphertext IDs sent from the server for message retrieval.

We provide an informal argument that unobservability under external adversaries holds. An external adversary may query the server for messages by sending `RequestMessages`. In response, they obtain a list of encrypted IDs for which they cannot know the key (by the security of the KDF and a Diffie–Hellman assumption relying on the fact that they know neither the server's ephemeral value r_k nor the private fetching key sk_R^{fetch}). Repeated queries for messages will return a completely new set of encrypted IDs because the server generates new values r_k upon each request. Therefore, provided that the number of message IDs is padded to a constant number, the external adversary learns no information about the message patterns (e.g., number of messages held by the server, when a new message is sent or retrieved) nor about message contents.

6 Implementation

We implemented the protocol primitives in Rust, using the formally verified `libcrux` [23, 76] cryptographic library. The implementation exposes three core client operations: *encrypt*, *decrypt* (for a single message), and *solve* (for a single challenge). In practice, journalist decryption will call *decrypt* for all active journalist key bundles until one succeeds, and source and journalist retrieval will call *solve* for all challenges returned by the server.

6.1 Benchmarks

Figure 9 shows the time required for a single iteration of each operation (n = 1000), measured on a MacBook Air M4. While each operation is benchmarked for a single iteration, the final cost depends on the system setup, such as the maximum number of supported messages or the number of journalist key bundles. The results show a performance gap between solve on one side and encrypt/decrypt on the other. Solve consists of a single Diffie–Hellman agreement followed by one AEAD decryption, while encrypt and decrypt employ post-quantum algorithms whose overhead is visibly higher.

We benchmarked both a native build, representative of the journalist's environment, and a WebAssembly build, which reflects the

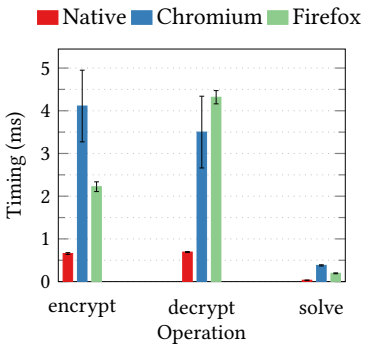


Figure 9: Execution time of the cryptographic operations.

intended runtime of sources' clients. To guide parameter choices such as the number of active key bundles and the maximum number of messages in the system, we refer to Appendix G, which charts iterative operations as parameters increase and shows the expected linear scaling with these configurable limits. We consider an upper bound of 500 ms per operation acceptable for interactive use on commodity hardware. To improve reproducibility, each iteration was executed in a fresh browser profile and process.

The results illustrate the upper configuration limits of the protocol in its current design. Default parameters such as 5000 maximum messages and 500 active key bundles appear reasonable while still providing an acceptable user experience ($\leq$ 500 ms per operation). Note that retrieve performance also depends on bandwidth: the challenge set grows linearly with the configured maximum amount of messages. In Tor Browser, this could become a relevant factor. In our current implementation, and depending on the serialization format and the length of message identifiers, the server-provided challenge set for 5000 maximum messages is on the order of 300 KB.

6.2 Deployment Considerations

6.2.1 Anonymous Transport. The protocol is designed for deployment on top of an anonymity network. There are other anonymity networks with stronger theoretical properties than the Tor network, but in keeping with the existing SecureDrop architecture and the Tor network's widespread use [75] and relative usability [47], we assume that the protocol runs over the Tor network: servers host a Tor onion service [48, 125], and Tor Browser is the browser client for sources, discussed in Section 6.2.3. Our protocol's hybrid post-quantum/traditional security is independent of Tor's quantum-readiness efforts [55, 98].

6.2.2 Server Side-Channels and Verifiability. The protocol does not provide guarantees about the integrity of server-side execution nor protect against side-channel leakage of request patterns and timing, which we treat as out of scope (see Section 2.2). These leakages can leave deployments vulnerable to both active and passive server adversaries aiming to break unlinkability. As active server adversaries need not follow the protocol, they can perform attacks as previously discussed (see Section 5.3.1), while passive server adversaries may correlate access patterns and timing over time. Side-channel leakage to external adversaries can be reduced by implementing endpoints in constant time, which we strongly recommend.

During our research, we considered server-side mitigations using cryptographic approaches such as zero-knowledge proofs and private information retrieval (PIR). However, integration would require a PIR scheme that supports frequent database updates; avoids disk persistence for browser-based clients [69, 101, 105]; resists ciphertext enumeration and avoids leaking database metadata [29]; is usable under the bandwidth limitations of Tor [84]; and can run in a single-server model [27, 46]. We find that no current PIR scheme satisfies these constraints, as we detail in Appendix F.

Future research into hardware-based isolation, such as trusted execution environments (TEE) with ORAM, could offer a deployable middle ground between no protection and currently impractical but more solid cryptographic solutions. These approaches have known limitations: attacks have targeted hardware and microarchitectural elements [31, 43, 58, 115, 138, 139], vendor firmware [86], trust anchors [61], and application code alike [85], and it remains challenging to implement such designs securely [134]. However, for deployments in cloud-based environments, they could raise the cost and improve the detectability of compromise. Their purpose is to elevate the attacker's required capabilities and slow the scale of data collection: an attacker attempting to break unlinkability would need to chain an attack against the TEE with a protocol side-channel attack. Similar trust assumptions have been widely studied [114, 123], and systems built on them have been deployed [97].

On-premises deployments operating in physically controlled spaces may not require such measures, as the threat of remote compromise is substantially reduced. Thus, isolation technologies should be viewed as an option rather than a universal requirement. FPF will publish and maintain deployment guidelines that document these risks and recommend countermeasures for common deployment scenarios. These guidelines will also identify cases for which the protocol is not currently recommended, and will be updated as research advances.

6.2.3 Client-Side Code Integrity. Using browser-based cryptography poses a fundamental trust problem [109]. Because JavaScript is fetched and executed dynamically, its integrity depends on its delivery infrastructure, and a compromised server could serve malicious or faulty JavaScript to sources, disabling encryption or exfiltrating information. JavaScript is the primary vector for most browser exploits, and enabling it significantly increases the attack surface, both for code execution and deanonymisation attacks [37, 124, 140].

Addressing this issue requires solving the problem of browser-based code integrity for JavaScript. For SecureDrop's threat model, such a code-integrity mechanism must inform the user of successful verification, but halt execution and alert the user if verification fails. For open-source, self-hosted software like SecureDrop, it must further be able to verify the authenticity of software deployed on third-party infrastructure. In addition, it must be auditable and remain backwards-compatible with applications without such requirements [20]. A proposed solution that meets these requirements and improves the threat model is WEBCAT [13], which authenticates and attests browser-side code execution using signatures and transparency logs. FPF is currently collaborating with the Tor

Project and other relevant stakeholders to implement a general-purpose solution. FPF will advocate for a production deployment of the new SecureDrop protocol only once the browser-side code integrity problem has been addressed [19, 21].

7 Conclusions and Future Work

We analysed and improved a new protocol for SecureDrop allowing it to be deployed in the cloud, thereby enabling SecureDrop to be used by a wider set of organisations. The protocol respects the real-world requirements FPF has gathered from SecureDrop's users. These include source and recipient anonymity, message-level deniability, stateless sources, protection against "harvest now, decrypt later" (HNDL) attacks by quantum adversaries, ease of deployment, and long-term maintainability. We have analysed the protocol using Tamarin's symbolic analysis approach in conjunction with game-based computational security proofs. This provides state-of-the-art coverage of SecureDrop's core security properties. We have also benchmarked the new protocol, showing that it achieves suitable levels of performance for the intended application.

This work leaves several open problems to be addressed, both practical and theoretical. First, the design and its implementation would benefit from formal verification of correctness and (more challenging still) security at the code level—currently, correctness is guaranteed by standard software testing practices, and our security analysis applies to a pseudocode or symbolic model of SecureDrop's cryptography. Second, the protocol does not currently support attachments, a feature expected by users of the system. Both of these open problems are on the FPF roadmap. Third, we found that several existing tools from the literature, such as oblivious message retrieval [90], PIR (Appendix F), and post-quantum deniable AKEMs [56, 57] were not sufficiently mature to meet our requirement of long-term maintainability. As these more advanced cryptographic tools mature and become standardised, we would like to explore how to apply them to the SecureDrop setting. A related challenge is to ensure the message retrieval protocol maintains its unlinkability properties in the face of quantum adversaries. Secure messaging in general will also benefit from such progress.

Acknowledgments

We thank Davide Silvetti and Abdel Adim Oisfi of Shielder for their help in developing the first proof of concept. We thank Jonathan Hoyland and Stephen Farrell, who chair the Usable Formal Methods Research Group (UFMRG) at the Internet Research Task Force (IRTF), for facilitating this collaboration. The Werner Siemens-Stiftung (WSS) provided funding for Felix Linker's research as part of the Centre for Cyber Trust (CECYT). We thank the WSS for their support of this project. We thank Alberto Sonnino for comments on the proof of unlinkability.

We generated a first draft of the code for Figure 1 and edited this paper for grammar using ChatGPT and GPT Codex.

References

[1] 2024. Justice Department 'won't force' journalist James Risen to reveal source. *The Guardian* (2024). https://www.theguardian.com/media/2014/dec/13/justice-department-wont-force-journalist-james-risen-to-reveal-source

[2] José Bacelar Almeida, Santiago Arranz Olmos, Manuel Barbosa, Gilles Barthe, François Dupressoir, Benjamin Grégoire, Vincent Laporte, Jean-Christophe Léchenet, Cameron Low, Tiago Oliveira, Hugo Pacheco, Miguel Quaresma, Peter Schwabe, and Pierre-Yves Strub. 2024. Formally Verifying Kyber - Episode V: Machine-Checked IND-CCA Security and Correctness of ML-KEM in EasyCrypt. In *CRYPTO 2024, Part II (LNCS, Vol. 14921)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, Cham, 384–421. doi:10.1007/978-3-031-68379-4_12

[3] Joël Alwen, Bruno Blanchet, Eduard Hauck, Eike Kiltz, Benjamin Lipp, and Doreen Riepel. 2021. Analysing the HPKE Standard. In *EUROCRYPT 2021, Part I (LNCS, Vol. 12696)*, Anne Canteaut and François-Xavier Standaert (Eds.). Springer, Cham, 87–116. doi:10.1007/978-3-030-77870-5_4

[4] Joël Alwen, Jonas Janneck, Eike Kiltz, and Benjamin Lipp. 2023. The Pre-Shared Key Modes of HPKE. In *ASIACRYPT 2023, Part VI (LNCS, Vol. 14443)*, Jian Guo and Ron Steinfeld (Eds.). Springer, Singapore, 329–360. doi:10.1007/978-981-99-8736-8_11

[5] Matilda Backendal, Mihir Bellare, Felix Günther, and Matteo Scarlata. 2023. When Messages Are Keys: Is HMAC a Dual-PRF?. In *CRYPTO 2023, Part III (LNCS, Vol. 14083)*, Helena Handschuh and Anna Lysyanskaya (Eds.). Springer, Cham, 661–693. doi:10.1007/978-3-031-38548-3_22

[6] Matilda Backendal, Hannah Davis, Felix Günther, Miro Haller, and Kenneth G. Paterson. 2024. A Formal Treatment of End-to-End Encrypted Cloud Storage. In *CRYPTO 2024, Part II (LNCS, Vol. 14921)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, Cham, 40–74. doi:10.1007/978-3-031-68379-4_2

[7] Manuel Barbosa, Deirdre Connolly, João Diogo Duarte, Aaron Kaiser, Peter Schwabe, Karoline Varner, and Bas Westerbaan. 2024. X-Wing. *CiC* 1, 1 (2024), 21. doi:10.62056/a3qj89n4e

[8] Richard Barnes, Benjamin Beurdouche, Raphael Robert, Jon Millican, Emad Omara, and Katriel Cohn-Gordon. 2023. The Messaging Layer Security (MLS) Protocol. RFC 9420. https://datatracker.ietf.org/doc/rfc9420/

[9] R. Barnes, K. Bhargavan, B. Lipp, and C. Wood. 2022. Hybrid Public Key Encryption. RFC 9180 (Informational). doi:10.17487/RFC9180

[10] Mihir Bellare, Joe Kilian, and Phillip Rogaway. 1994. The Security of Cipher Block Chaining. In *CRYPTO'94 (LNCS, Vol. 839)*, Yvo Desmedt (Ed.). Springer, Berlin, Heidelberg, 341–358. doi:10.1007/3-540-48658-5_32

[11] Mihir Bellare, Thomas Ristenpart, and Stefano Tessaro. 2012. Multi-instance Security and Its Application to Password-Based Cryptography. In *CRYPTO 2012 (LNCS, Vol. 7417)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer, Berlin, Heidelberg, 312–329. doi:10.1007/978-3-642-32009-5_19

[12] Mihir Bellare and Phillip Rogaway. 2006. The Security of Triple Encryption and a Framework for Code-Based Game-Playing Proofs. In *EUROCRYPT 2006 (LNCS, Vol. 4004)*, Serge Vaudenay (Ed.). Springer, Berlin, Heidelberg, 409–426. doi:10.1007/11761679_25

[13] Giulio Berra. 2025. WEBCAT: Web-based Code Assurance and Transparency. Cryptology ePrint Archive, Report 2025/797. https://eprint.iacr.org/2025/797

[14] Giulio Berra, Felix Linker, Luca Maier, Cory Francis Myers, Kenneth Paterson, Rowen Shane, and Shannon Veitch. 2026. The SecureDrop Protocol: End-to-End Encrypted Whistleblowing for All - Proof of Concept and Benchmarks. Zenodo. doi:10.5281/zenodo.20718663

[15] Giulio Berra, Felix Linker, Luca Maier, Cory Francis Myers, Kenneth Paterson, Rowen Shane, and Shannon Veitch. 2026. The SecureDrop Protocol: End-to-End Encrypted Whistleblowing for All - Tamarin Models. Zenodo. doi:10.5281/zenodo.20597107

[16] Giulio Berra, Cory Myers, Rowen Shane, and SecureDrop Team. 2024. Anatomy of a whistleblowing system. https://securedrop.org/news/anatomy-of-a-whistleblowing-system/

[17] Giulio Berra, Cory Myers, Rowen Shane, and SecureDrop Team. 2024. How to research your own cryptography and survive. https://securedrop.org/news/how-to-research-your-own-cryptography-and-survive/

[18] Giulio Berra, Cory Myers, Rowen Shane, and SecureDrop Team. 2024. Introducing SecureDrop Protocol. https://securedrop.org/news/introducing-securedrop-protocol/

[19] Giulio Berra, Cory Myers, Rowen Shane, and SecureDrop Team. 2024. The long and winding road to safe browser-based cryptography. https://securedrop.org/news/browser-based-cryptography/

[20] Giulio Berra and SecureDrop Team. 2025. WEBCAT: Towards auditable web application runtimes. https://securedrop.org/news/webcat-towards-auditable-web-application-runtimes/

[21] Giulio Berra, Rowen Shane, and SecureDrop Team. 2025. Introducing WEBCAT: Web-based Code Assurance and Transparency. https://freedom.press/tech/news/introducing-webcat-web-based-code-assurance-and-transparency/

[22] Charles Berret. 2016. Guide to SecureDrop. doi:10.7916/D84178B2

[23] Karthikeyan Bhargavan, Lasse Letager Hansen, Franziskus Kiefer, Jonas Schneider-Bensch, and Bas Spitters. 2025. Formal Security and Functional Verification of Cryptographic Protocol Implementations in Rust. In *ACM CCS 2025*, Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier (Eds.). ACM Press, 2729–2743. doi:10.1145/3719027.3765213

[24] Karthikeyan Bhargavan, Charlie Jacomme, Franziskus Kiefer, and Rolfe Schmidt. 2024. Formal verification of the PQXDH Post-Quantum key agreement protocol for end-to-end secure messaging. In *USENIX Security 2024*, Davide Balzarotti and Wenyuan Xu (Eds.). USENIX Association. https://www.usenix.org/conference/usenixsecurity24/presentation/bhargavan

[25] Nina Bindel, Jacqueline Brendel, Marc Fischlin, Brian Goncalves, and Douglas Stebila. 2019. Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange. In *PQCrypto 2019*, Jintai Ding and Rainer Steinwandt (Eds.). Springer, Cham, 206–226. doi:10.1007/978-3-030-25510-7_12

[26] Simon Blake-Wilson and Alfred Menezes. 1999. Authenticated Diffie-Hellman Key Agreement Protocols (Invited Talk). In *SAC 1998 (LNCS, Vol. 1556)*, Stafford E. Tavares and Henk Meijer (Eds.). Springer, Berlin, Heidelberg, 339–361. doi:10.1007/3-540-48892-8_26

[27] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2016. Function Secret Sharing: Improvements and Extensions. In *ACM CCS 2016*, Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi (Eds.). ACM Press, 1292–1303. doi:10.1145/2976749.2978429

[28] Jacqueline Brendel, Rune Fiedler, Felix Günther, Christian Janson, and Douglas Stebila. 2022. Post-quantum Asynchronous Deniable Key Exchange and the Signal Handshake. In *PKC 2022, Part II (LNCS, Vol. 13178)*, Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe (Eds.). Springer, Cham, 3–34. doi:10.1007/978-3-030-97131-1_1

[29] Sofia Celi and Alex Davidson. 2024. Call Me By My Name: Simple, Practical Private Information Retrieval for Keyword Queries. In *ACM CCS 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM Press, 4107–4121. doi:10.1145/3658644.3670271

[30] John Chan and Phillip Rogaway. 2019. Anonymous AE. In *ASIACRYPT 2019, Part II (LNCS, Vol. 11922)*, Steven D. Galbraith and Shiho Moriai (Eds.). Springer, Cham, 183–208. doi:10.1007/978-3-030-34621-8_7

[31] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten-Hwang Lai. 2019. SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution. In *2019 IEEE European Symposium on Security and Privacy*. IEEE Computer Society Press, 142–157. doi:10.1109/EuroSP.2019.00020

[32] Vincent Cheval, Cas Cremers, Alexander Dax, Lucca Hirschi, Charlie Jacomme, and Steve Kremer. 2023. Hash Gone Bad: Automated discovery of protocol attacks that exploit hash function weaknesses. In *USENIX Security 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 5899–5916. https://www.usenix.org/conference/usenixsecurity23/presentation/cheval

[33] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. 2020. A Formal Security Analysis of the Signal Messaging Protocol. *Journal of Cryptology* 33, 4 (Oct. 2020), 1914–1983. doi:10.1007/s00145-020-09360-1

[34] Christoph Coijanovic, Laura Hetz, Kenneth G. Paterson, and Thorsten Strufe. 2025. Sabot: Efficient and Strongly Anonymous Bootstrapping of Communication Channels. In *ACM CCS 2025*, Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier (Eds.). ACM Press, 558–572. doi:10.1145/3719027.3744803

[35] Daniel Collins, Simone Colombo, and Loïs Huguenin-Dumittan. 2025. Real-World Deniability in Messaging. *PoPETs* 2025, 1 (Jan. 2025), 320–340. doi:10.56553/popets-2025-0018

[36] Simone Colombo, Kirill Nikitin, Henry Corrigan-Gibbs, David J. Wu, and Bryan Ford. 2023. Authenticated private information retrieval. In *USENIX Security 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 3835–3851. https://www.usenix.org/conference/usenixsecurity23/presentation/colombo

[37] Mauro Conti, Stephen Crane, Tommaso Frassetto, Andrei Homescu, Georg Koppen, Per Larsen, Christopher Liebchen, Mike Perry, and Ahmad-Reza Sadeghi. 2016. Selfrando: Securing the Tor Browser against De-anonymization Exploits. *PoPETs* 2016, 4 (Oct. 2016), 454–469. doi:10.1515/popets-2016-0050

[38] Henry Corrigan-Gibbs, Dan Boneh, and David Mazières. 2015. Riposte: An Anonymous Messaging System Handling Millions of Users. In *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 321–338. doi:10.1109/SP.2015.27

[39] Ronald Cramer and Victor Shoup. 2003. Design and Analysis of Practical Public-Key Encryption Schemes Secure against Adaptive Chosen Ciphertext Attack. *SIAM J. Comput.* 33, 1 (2003), 167–226.

[40] Cas Cremers, Alexander Dax, Charlie Jacomme, and Mang Zhao. 2023. Automated Analysis of Protocols that use Authenticated Encryption: How Subtle AEAD Differences can impact Protocol Security. In *USENIX Security 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 5935–5952. https://www.usenix.org/conference/usenixsecurity23/presentation/cremers-protocols

[41] Cas Cremers, Alexander Dax, and Niklas Medinger. 2024. Keeping Up with the KEMs: Stronger Security Notions for KEMs and Automated Analysis of KEM-based Protocols. In *ACM CCS 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM Press, 1046–1060. doi:10.1145/3658644.3670283

[42] Cas Cremers, Charlie Jacomme, and Aurora Naska. 2023. Formal Analysis of Session-Handling in Secure Messaging: Lifting Security from Sessions to Conversations. In *USENIX Security 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 1235–1252. https://www.usenix.org/conference/usenixsecurity23/presentation/cremers-session-handling

[43] Jinhua Cui, Jason Zhijingcheng Yu, Shweta Shinde, Prateek Saxena, and Zhiping Cai. 2021. SmashEx: Smashing SGX Enclaves Using Exceptions. In *ACM CCS 2021*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, 779–793. doi:10.1145/3460120.3484821

[44] Emma Dauterman, Vivian Fang, Natacha Crooks, and Raluca Ada Popa. 2022. Reflections on trusting distributed trust. In *Proceedings of the 21st ACM Workshop on Hot Topics in Networks* (Austin, Texas) *(HotNets '22)*. Association for Computing Machinery, New York, NY, USA, 38–45. doi:10.1145/3563766.3564089

[45] H. de Valence, J. Grigg, M. Hamburg, I. Lovecruft, G. Tankersley, and F. Valsorda. 2023. The ristretto255 and decaf448 Groups. RFC 9496 (Informational). doi:10.17487/RFC9496

[46] Daniel Demmler, Amir Herzberg, and Thomas Schneider. 2014. RAID-PIR: Practical Multi-Server PIR. In *Proceedings of the 6th Edition of the ACM Workshop on Cloud Computing Security* (Scottsdale, Arizona, USA) *(CCSW '14)*. Association for Computing Machinery, New York, NY, USA, 45–56. doi:10.1145/2664168.2664181

[47] Roger Dingledine and Nick Mathewson. 2006. Anonymity Loves Company: Usability and the Network Effect. In *Proceedings of the Fifth Workshop on the Economics of Information Security (WEIS 2006)* (Cambridge, UK), Ross Anderson (Ed.).

[48] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. 2004. Tor: The Second-Generation Onion Router. In *USENIX Security 2004*, Matt Blaze (Ed.). USENIX Association, 303–320. doi:10.21236/ada465464

[49] Josiah Dykstra and Alan T. Sherman. 2012. Acquiring forensic evidence from infrastructure-as-a-service cloud computing: Exploring and evaluating tools, trust, and techniques. *Digital Investigation* 9 (2012), S90–S98. doi:10.1016/j.diin.2012.05.001 The Proceedings of the Twelfth Annual DFRWS Conference.

[50] Rune Fiedler and Christian Janson. 2024. A Deniability Analysis of Signal's Initial Handshake PQXDH. *PoPETs* 2024, 4 (Oct. 2024), 907–928. doi:10.56553/popets-2024-0148

[51] Kathryn Flynn. 2006. Covert Disclosures: Unauthorized Leaking, Public Officials and the Public Sphere. *Journalism Studies* 7, 2 (2006), 256–273. doi:10.1080/14616700500533577

[52] Freedom of the Press Foundation. 2018. SecureDrop License. https://github.com/freedomofpress/securedrop/blob/develop/LICENSE

[53] Freedom of the Press Foundation. 2025. Freedom of the Press Foundation's 2025-2026 Strategic Plan. https://freedom.press/about/announcements/freedom-of-the-press-foundations-2025-2026-strategic-plan/

[54] Freedom of the Press Foundation. n.d. SecureDrop. https://securedrop.org/

[55] Alexander Hansen Færøy. n.d. Post-Quantum Cryptography in Tor's TLS Layer: Help needed! https://forum.torproject.org/t/tor-relays-post-quantum-cryptography-in-tors-tls-layer-help-needed/21288 Accessed 2026-05-01.

[56] Phillip Gajland, Vincent Hwang, and Jonas Janneck. 2025. Shadowfax: A Deniability-Preserving AKEM Combiner. Cryptology ePrint Archive, Report 2025/154. https://eprint.iacr.org/2025/154

[57] Phillip Gajland, Jonas Janneck, and Eike Kiltz. 2024. Ring Signatures for Deniable AKEM: Gandalf's Fellowship. In *CRYPTO 2024, Part I (LNCS, Vol. 14920)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, Cham, 305–338. doi:10.1007/978-3-031-68376-3_10

[58] Stefan Gast, Hannes Weissteiner, Robin Leander Schröder, and Daniel Gruss. 2025. CounterSEVeillance: Performance-Counter Attacks on AMD SEV-SNP. In *NDSS 2025*. The Internet Society.

[59] Federico Giacon, Felix Heuer, and Bertram Poettering. 2018. KEM Combiners. In *PKC 2018, Part I (LNCS, Vol. 10769)*, Michel Abdalla and Ricardo Dahab (Eds.). Springer, Cham, 190–218. doi:10.1007/978-3-319-76578-5_7

[60] GlobaLeaks. n.d. GlobaLeaks - Free and Open-Source Whistleblowing Software. https://www.globaleaks.org/ Accessed 2025-10-30.

[61] Google Security Research. 2025. AMD: Microcode Signature Verification Vulnerability. GitHub Security Advisory. https://github.com/google/security-research/security/advisories/GHSA-4xq7-4mgh-gp6w GHSA-4xq7-4mgh-gp6w; CVE-2024-56161.

[62] M. Choudary Gorantla, Colin Boyd, and Juan Manuel González Nieto. 2009. Modeling Key Compromise Impersonation Attacks on Group Key Exchange Protocols. In *PKC 2009 (LNCS, Vol. 5443)*, Stanislaw Jarecki and Gene Tsudik (Eds.). Springer, Berlin, Heidelberg, 105–123. doi:10.1007/978-3-642-00468-1_7

[63] Paul Grubbs, Varun Maram, and Kenneth G. Paterson. 2022. Anonymous, Robust Post-quantum Public Key Encryption. In *EUROCRYPT 2022, Part III (LNCS, Vol. 13277)*, Orr Dunkelman and Stefan Dziembowski (Eds.). Springer, Cham, 402–432. doi:10.1007/978-3-031-07082-2_15

[64] Mario Heiderich, Nikolai K., and Fabian Fäßler. 2013. Pentest-Report SecureDrop 12.2013. https://media.securedrop.org/media/documents/pentest-report_securedrop.pdf

[65] Jennifer Helsby. 2020. Next-Generation SecureDrop: Protecting Journalists from Malware. USENIX Association. https://www.usenix.org/conference/enigma2020/presentation/helsby

[66] Jennifer Helsby and SecureDrop Team. 2020. securedrop-signal-poc. https://github.com/freedomofpress/securedrop-signal-poc

[67] Jennifer R. Henrichsen. 2019. Breaking Through the Ambivalence: Journalistic Responses to Information Security Technologies. *Digital Journalism* (2019). doi:10.1080/21670811.2019.1653207

[68] Ryan Henry, Yizhou Huang, and Ian Goldberg. 2013. One (Block) Size Fits All: PIR and SPIR with Variable-Length Records via Multi-Block Queries. In *NDSS 2013*. The Internet Society.

[69] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. 2023. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In *USENIX Security 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 3889–3905. https://www.usenix.org/conference/usenixsecurity23/presentation/henzinger

[70] Daniel Hugenroth, Sam Cutler, Dominic Kendrick, Mario Savarese, Zeke Hunter-Green, Philip McMahon, Marjan Kalanaki, Diana A. Vasile, Sabina Bejasa-Dimmock, Luke Hoyland, and Alastair R. Beresford. 2025. CoverDrop White Paper. https://www.coverdrop.org/coverdrop_guardian_implementation_june_2025.pdf

[71] Internet Engineering Task Force. 2023. Support for MLS. https://www.ietf.org/blog/support-for-mls-2023/ Accessed 2025-12-28.

[72] Dennis Jackson, Cas Cremers, Katriel Cohn-Gordon, and Ralf Sasse. 2019. Seems Legit: Automated Analysis of Subtle Attacks on Protocols that Use Signatures. In *ACM CCS 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM Press, 2165–2180. doi:10.1145/3319535.3339813

[73] Jonas Janneck, Jonas Meers, Massimo Ostuzzi, and Doreen Riepel. 2025. Snake Mackerel: An Isogeny-Based AKEM Leveraging Randomness Reuse. Cryptology ePrint Archive, Report 2025/1474. https://eprint.iacr.org/2025/1474

[74] Yanxue Jia, Varun Madathil, and Aniket Kate. 2024. HomeRun: High-efficiency Oblivious Message Retrieval, Unrestricted. In *ACM CCS 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM Press, 2012–2026. doi:10.1145/3658644.3670381

[75] Ishan Karunanayake, Nadeem Ahmed, Robert Malaney, Rafiqul Islam, and Sanjay K Jha. 2021. De-anonymisation attacks on Tor: A survey. *IEEE Communications Surveys & Tutorials* 23, 4 (2021), 2324–2350.

[76] Franziskus Kiefer, Karthikeyan Bhargavan, Lucas Franceschino, Denis Merigoux, Lasse Letager Hansen, Bas Spitters, Manuel Barbosa, Antoine Séré, and Pierre-Yves Strub. 2023. HACSPEC: a gateway to high-assurance cryptography. In *Real World Crypto 2023*. Tokyo, Japan. https://www.youtube.com/watch?v=lahO3de3k_0 Talk presented at Real World Crypto 2023.

[77] Ceren Kocaogullar, Daniel Hugenroth, Martin Kleppmann, and Alastair R. Beresford. 2024. Pudding: Private User Discovery in Anonymity Networks. In *2024 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 3203–3220. doi:10.1109/SP54263.2024.00167

[78] Tadayoshi Kohno, Yasemin Acar, and Wulf Loh. 2023. Ethical Frameworks and Computer Security Trolley Problems: Foundations for Conversations. In *USENIX Security 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 5145–5162. https://www.usenix.org/conference/usenixsecurity23/presentation/kohno

[79] Hugo Krawczyk. 2010. Cryptographic Extraction and Key Derivation: The HKDF Scheme. In *CRYPTO 2010 (LNCS, Vol. 6223)*, Tal Rabin (Ed.). Springer, Berlin, Heidelberg, 631–648. doi:10.1007/978-3-642-14623-7_34

[80] Christine Lam, Murat Gulcelik, Olga Rios, Martin Shelton, Jennifer R. Henrichsen, and Susan E. McGregor. 2024. Exposing Local Sources: The (Non)Use of Secure Tip Communication Methods by Local US News Organizations. In *USENIX Symposium on Usable Privacy and Security (SOUPS) Posters*. https://www.usenix.org/system/files/soups2024_poster42_abstract-lam_final.pdf

[81] David Lazar, Yossi Gilad, and Nickolai Zeldovich. 2018. Karaoke: distributed private messaging immune to passive traffic analysis. In *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation* (Carlsbad, CA, USA) *(OSDI'18)*. USENIX Association, USA, 711–725.

[82] David Lazar and Nickolai Zeldovich. 2016. Alpenhorn: Bootstrapping secure communication without leaking metadata. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 571–586.

[83] Sarah Jamie Lewis. 2018. Cwtch: Privacy Preserving Infrastructure for Asynchronous, Decentralized, Multi-Party and Metadata Resistant Applications. https://cwtch.im/cwtch.pdf

[84] Baiyu Li, Daniele Micciancio, Mariana Raykova, and Mark Schultz. 2024. Hintless Single-Server Private Information Retrieval. In *CRYPTO 2024, Part IX (LNCS, Vol. 14928)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, Cham, 183–217. doi:10.1007/978-3-031-68400-5_6

[85] Mengyuan Li, Luca Wilke, Jan Wichelmann, Thomas Eisenbarth, Radu Teodorescu, and Yinqian Zhang. 2022. A Systematic Look at Ciphertext Side Channels on AMD SEV-SNP. In *2022 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 337–351. doi:10.1109/SP46214.2022.9833768

[86] Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. 2021. CrossLine: Breaking "Security-by-Crash" based Memory Isolation in AMD SEV. In *ACM CCS 2021*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, 2937–2950. doi:10.1145/3460120.3485253

[87] Benoît Libert. 2024. Vector Commitments with Proofs of Smallness: Short Range Proofs and More. In *PKC 2024, Part II (LNCS, Vol. 14602)*, Qiang Tang and Vanessa Teague (Eds.). Springer, Cham, 36–67. doi:10.1007/978-3-031-57722-2_2

[88] Felix Linker. 2026. Rule Variant Restrictions for the Tamarin Prover. Cryptology ePrint Archive, Report 2026/230. https://eprint.iacr.org/2026/230

[89] Felix Linker, Ralf Sasse, and David Basin. 2024. A Formal Analysis of Apple's iMessage PQ3 Protocol. Cryptology ePrint Archive, Report 2024/1395. https://eprint.iacr.org/2024/1395

[90] Zeyu Liu and Eran Tromer. 2022. Oblivious Message Retrieval. In *CRYPTO 2022, Part I (LNCS, Vol. 13507)*, Yevgeniy Dodis and Thomas Shrimpton (Eds.). Springer, Cham, 753–783. doi:10.1007/978-3-031-15802-5_26

[91] Gavin Lowe. 1997. A Hierarchy of Authentication Specifications. In *Proceeding of the 10th Computer Security Foundations Workshop*. doi:10.1109/CSFW.1997.596782

[92] Kim Loyens, Jeroen Maesschalck, David Lewis, Richard Moberly, and Wim Vandekerckhove. 2014-01-01. Whistleblowing and power.

[93] Rasoul Akhavan Mahdavi and Florian Kerschbaum. 2022. Constant-weight PIR: Single-round Keyword PIR via Constant-weight Equality Operators. In *USENIX Security 2022*, Kevin R. B. Butler and Kurt Thomas (Eds.). USENIX Association, 1723–1740. https://www.usenix.org/conference/usenixsecurity22/presentation/mahdavi

[94] Varun Maram and Keita Xagawa. 2023. Post-quantum Anonymity of Kyber. In *PKC 2023, Part I (LNCS, Vol. 13940)*, Alexandra Boldyreva and Vladimir Kolesnikov (Eds.). Springer, Cham, 3–35. doi:10.1007/978-3-031-31368-4_1

[95] Tanya M. Marcum, Jacob Young, and Ethan T. Kirner. 2020. Blowing the Whistle in the Digital Age: Are You Really Anonymous? The Perils and Pitfalls of Anonymity in Whistleblowing Law. *DePaul Business and Commercial Law Journal* 17 (2020). Issue 1. https://via.library.depaul.edu/bclj/vol17/iss1/1

[96] Moxie Marlin and Trevor Perrin. 2016. The X3DH Key Agreement Protocol. https://signal.org/docs/specifications/x3dh/ Accessed 2025-10-27.

[97] Moxie Marlinspike. 2017. Technology preview: Private contact discovery for Signal. https://signal.org/blog/private-contact-discovery/ Accessed 2025-12-17.

[98] Nick Mathewson. 2025. Options for Postquantum Circuit Extension Handshakes. https://gitlab.torproject.org/tpo/core/torspec/-/blob/main/proposals/355-revisiting-pq.md Accessed 2026-05-01.

[99] Susan E. McGregor. 2021. *Information Security Essentials: A Guide for Reporters, Editors, and Newsroom Leaders*. Columbia University Press, New York, NY. doi:10.7312/mcgr19232

[100] David A. McGrew and John Viega. 2004. The Security and Performance of the Galois/Counter Mode (GCM) of Operation. In *INDOCRYPT 2004 (LNCS, Vol. 3348)*, Anne Canteaut and Kapalee Viswanathan (Eds.). Springer, Berlin, Heidelberg, 343–355. doi:10.1007/978-3-540-30556-9_27

[101] Muhammad Haris Mughees, Hao Chen, and Ling Ren. 2021. OnionPIR: Response Efficient Single-Server PIR. In *ACM CCS 2021*, Giovanni Vigna and Elaine Shi (Eds.). ACM Press, 2292–2306. doi:10.1145/3460120.3485381

[102] NIST. 2024. Module-Lattice-Based Key-Encapsulation Mechanism Standard. https://doi.org/10.6028/NIST.FIPS.203 FIPS 203..

[103] OTR. 2019. OTRv4. https://github.com/otrv4/otrv4/blob/master/otrv4.md

[104] Marek Palatinus, Pavol Rusnak, Aaron Voisine, and Sean Bowe. 2013. Mnemonic code for generating deterministic keys. https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki BIP39.

[105] Sarvar Patel, Giuseppe Persiano, and Kevin Yeo. 2018. Private Stateful Information Retrieval. In *ACM CCS 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM Press, 1002–1019. doi:10.1145/3243734.3243821

[106] Giovanni Pellerano. 2023. Replace Cross-Origin-Resource-Policy: 'same-site' with 'same-origin'. https://github.com/freedomofpress/securedrop/pull/6768

[107] Trevor Perrin, Moxie Marlinspike, and Rolfe Schmidt. 2025. The Double Ratchet Algorithm (revision 3). https://signal.org/docs/specifications/doubleratchet/ Accessed 2025-10-27.

[108] Tor Project. n.d. Onion Names for SecureDrop. https://onionservices.torproject.org/research/proposals/usability/discovery/onion-names/#onion-names-for-securedrop Accessed 2026-01-12.

[109] Thomas Ptacek. 2011. Javascript Cryptography Considered Harmful. https://web.archive.org/web/20200731144044/https://www.nccgroup.com/us/about-us/newsroom-and-events/blog/2011/august/javascript-cryptography-considered-harmful/ Accessed 2025-01-09.

[110] Reporters Sans Frontières. 2025. 2025 Security Indicator. https://rsf.org/en/index/score-saf?year=2025

[111] Pavol Rusnak, Andrew Kozlik, Ondrej Vejpustek, Tomas Susanka, Marek Palatinus, and Jochen Hoenicke. 2017. Shamir's Secret-Sharing for Mnemonic Codes. https://github.com/satoshilabs/slips/blob/master/slip-0039.md SLIP-0039.

[112] Philip Di Salvo. 2021. Securing Whistleblowing in the Digital Age: SecureDrop and the Changing Journalistic Practices for Source Protection. *Digital Journalism* 9, 4 (2021), 443–460. doi:10.1080/21670811.2021.1889384

[113] Sajin Sasy and Ian Goldberg. 2024. SoK: Metadata-Protecting Communication Systems. *PoPETs* 2024, 1 (Jan. 2024), 509–524. doi:10.56553/popets-2024-0030

[114] Sajin Sasy, Aaron Johnson, and Ian Goldberg. 2025. TEEMS: A Trusted Execution Environment based Metadata-protected Messaging System. *PoPETs* 2025, 4 (Oct. 2025), 56–75. doi:10.56553/popets-2025-0119

[115] Benedict Schlüter, Supraja Sridhara, Andrin Bertschi, and Shweta Shinde. 2024. WeSee: Using Malicious #VC Interrupts to Break AMD SEV-SNP. In *2024 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 4220–4238. doi:10.1109/SP54263.2024.00262

[116] Benedikt Schmidt, Simon Meier, Cas J. F. Cremers, and David A. Basin. 2012. Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties. In *CSF 2012 Computer Security Foundations Symposium*, Steve Zdancewic and Véronique Cortier (Eds.). IEEE Computer Society Press, 78–94. doi:10.1109/CSF.2012.25

[117] Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. 2022. *CRYSTALS-KYBER*. Technical Report. National Institute of Standards and Technology. available at https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[118] SecureDrop Team. 2023. SecureDrop is evolving: We want to hear from you. https://securedrop.org/news/securedrop-is-evolving-we-want-to-hear-from-you/

[119] SecureDrop Team. n.d. No third parties that can secretly be subpoenaed. https://securedrop.org/overview/no-third-parties-can-secretly-be-subpoenaed/ Accessed 2026-01-13.

[120] Martin Shelton. 2015. *The Role of Corporate and Government Surveillance in Shifting Journalistic Information Security Practices*. Ph. D. Dissertation. University of California, Irvine. https://mshelt.onl/p/shelton_2015.pdf Accessed 2025-12-10.

[121] Douglas Stebila. 2024. Security analysis of the iMessage PQ3 protocol. Cryptology ePrint Archive, Report 2024/357. https://eprint.iacr.org/2024/357

[122] Thea Riebe and Christian Reuter. 2019. *Dual-Use and Dilemmas for Cybersecurity, Peace and Technology Assessment*. Springer Fachmedien Wiesbaden, Wiesbaden, 169–188. doi:10.1007/978-3-658-25652-4_8

[123] Afonso Tinoco, Sixiang Gao, and Elaine Shi. 2023. EnigMap: External-Memory Oblivious Map for Secure Enclaves. In *USENIX Security 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 4033–4050. https://www.usenix.org/conference/usenixsecurity23/presentation/tinoco

[124] Pascal Tippe and Adrian Tippe. 2024. Onion Services in the Wild: A Study of Deanonymization Attacks. *PoPETs* 2024, 4 (Oct. 2024), 291–310. doi:10.56553/popets-2024-0117

[125] Tor Project. 2017. Tor Rendezvous Specification - Version 3. https://spec.torproject.org/rend-spec/index.html

[126] Eric Tucker and Alanna Durkin Richer. 2024. Justice Department ignored some policies when seizing reporters' phone records, watchdog finds. *AP News* (2024). https://apnews.com/article/justice-department-inspector-general-reporters-congress-records-d337eb87f286b9c14779adb8b4b14365 Accessed 2025-12-10.

[127] Umut Turksen, Adam Abukari, and Reinhard Kreissl. 2023. Public Interest Disclosure and Whistleblower Protection. In *Tax Crimes and Enforcement in the European Union: Solutions for law, policy and practice*. Oxford University Press. doi:10.1093/oso/9780192862341.003.0006

[128] Nirvan Tyagi, Yossi Gilad, Matei Zaharia, and Nickolai Zeldovich. 2017. Stadium: A Distributed Metadata-Private Messaging System. In *SOSP '17: Proceedings of the 26th Symposium on Operating Systems Principles*, Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier (Eds.). ACM Press, Shanghai, China, 423–440. doi:10.1145/3132747.3132783

[129] Nik Unger and Ian Goldberg. 2015. Deniable Key Exchanges for Secure Messaging. In *ACM CCS 2015*, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM Press, 1211–1223. doi:10.1145/2810103.2813616

[130] US Press Freedom Tracker. 2020. Trump Justice Department secretly obtained CNN Pentagon reporter's email, phone records. https://pressfreedomtracker.us/all-incidents/trump-justice-department-secretly-obtained-cnn-pentagon-reporters-email-phone-records/

[131] Adithya Vadapalli, Kyle Storrier, and Ryan Henry. 2022. Sabre: Sender-Anonymous Messaging with Fast Audits. In *2022 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1953–1970. doi:10.1109/SP46214.2022.9833601

[132] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. 2015. Vuvuzela: Scalable Private Messaging Resistant to Traffic Analysis. In *Proceedings of the 25th Symposium on Operating Systems Principles* (Monterey California). ACM, New York, NY, USA.

[133] Peter G. van der Velden, Mauro Pecoraro, Mijke S. Houwerzijl, and Erik van der Meulen. 2019. Mental Health Problems Among Whistleblowers: A Comparative Study. *Psychological Reports* 2, 122 (2019). doi:10.1177/0033294118757681

[134] Stephan van Schaik, Alexander Seto, Thomas Yurek, Adam Batori, Bader AlBassam, Daniel Genkin, Andrew Miller, Eyal Ronen, Yuval Yarom, and Christina Garman. 2024. SoK: SGX.Fail: How Stuff Gets eXposed. In *2024 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 4143–4162. doi:10.1109/SP54263.2024.00260

[135] Nihal Vatandas, Rosario Gennaro, Bertrand Ithurburn, and Hugo Krawczyk. 2020. On the Cryptographic Deniability of the Signal Protocol. In *ACNS 2020, Part II (LNCS, Vol. 12147)*, Mauro Conti, Jianying Zhou, Emiliano Casalicchio, and Angelo Spognardi (Eds.). Springer, Cham, 188–209. doi:10.1007/978-3-030-57878-7_10

[136] Thomas Voegtlin. 2023. Electrum Seed Version System. https://electrum.readthedocs.io/en/latest/seedphrase.html

[137] Maximilian Weisenseel, Christoph Döpmann, and Florian Tschorsch. 2025. The Last Hop Attack: The Last Hop Attack: Why Loop Cover Traffic over Fixed Cascades Threatens Anonymity. *PoPETs* 2025, 2 (April 2025), 382–397. doi:10.56553/popets-2025-0067

[138] Luca Wilke, Jan Wichelmann, Mathias Morbitzer, and Thomas Eisenbarth. 2020. SEVurity: No Security Without Integrity : Breaking Integrity-Free Memory Encryption with Minimal Assumptions. In *2020 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1483–1496. doi:10.1109/SP40000.2020.00080

[139] Luca Wilke, Jan Wichelmann, Anja Rabich, and Thomas Eisenbarth. 2024. SEV-Step A Single-Stepping Framework for AMD-SEV. *IACR TCHES* 2024, 1 (2024), 180–206. doi:10.46586/tches.v2024.i1.180-206

[140] Mojtaba Zaheri, Yossi Oren, and Reza Curtmola. 2022. Targeted Deanonymization via the Cache Side Channel: Attacks and Defenses. In *USENIX Security 2022*, Kevin R. B. Butler and Kurt Thomas (Eds.). USENIX Association, 1505–1523. https://www.usenix.org/conference/usenixsecurity22/presentation/zaheri

[141] Yuliang Zheng. 1997. Digital Signcryption or How to Achieve Cost(Signature & Encryption) ≪ Cost(Signature) + Cost(Encryption). In *CRYPTO'97 (LNCS, Vol. 1294)*, Burton S. Kaliski, Jr. (Ed.). Springer, Berlin, Heidelberg, 165–179. doi:10.1007/BFb0052234

Open Science

The proof-of-concept Rust implementation, along with the JavaScript benchmarking harness intended to run on Mac, is available at [14]. The Tamarin proofs, along with setup instructions for Ubuntu 24.04, are available at [15].

Ethical Considerations

Ethical Principles. Public-interest whistleblowing seeks to uphold the societal principles of justice and accountability by bringing harms such as corruption or government malfeasance to light [51, 92]. Our research considers the same overarching principles of justice and respect for the public interest. We additionally consider respect for persons and groups in regard to our wide range of stakeholders, discussed below.

Stakeholder Impact. We rely on FPF's prior stakeholder research [17, 65, 118], as well as on academic research [22, 51, 112, 120]. We note that, while we use the terms "source" and "whistleblower" for clarity, people who leak information to the media often do so without an affinity for these terms.

- Direct stakeholders, including sources, journalists, newsroom IT staff, news organisations as a whole, and organisations that have been unable to use SecureDrop for reasons discussed in Section 1 will be affected by deployability, security, and usability trade-offs discussed in Section 4.
- The design of the new protocol will require journalists in particular to adopt a new mental model with respect to message availability on different devices, similar to the differences between using email and Signal.
- Newsrooms will need to understand the regulations around deploying end-to-end encrypted anonymous communications tools in their region, as well as conduct a context- and region-specific risk assessment, as discussed in Section 6.2.
- SecureDrop's many prior and current contributors are also stakeholders, as are projects in the open-source ecosystem with which SecureDrop has collaborated [106, 108]. Redesign proposals have time and effort implications for the project's maintainers.

- There are other indirect stakeholder groups who may be interested in other applications of our research (for example, helpline operators).

Harms and Mitigations. Whistleblowers and newsrooms face diverse and highly variable risks, and could be harmed if our research applied one reductive threat model [92]. For this reason, we examine diverse organisational profiles in our analysis of risk-mitigating deployment strategies, discussed in Section 6.2. Journalists risk exposure to unwanted or abusive content. This risk, which is also present in the existing SecureDrop design and on other communications platforms used by journalists, is unmitigated at the protocol level, but can be partially mitigated at the application level, for example via content-blurring features.

Due to the anonymous, message-queue-style architecture, there is a risk that a deployed system could be used for unintended purposes, including covert communication between two "sources." This risk is unmitigated at the protocol level, and is a consequence of the system's privacy-preserving, metadata-minimising design.

SecureDrop is an open-source, AGPLv3-licensed project [52]. Open-source anonymous communication systems have the potential to be deployed for purposes other than those intended. This is true of our contributions as well as FPF's implementation.

Analysis. We apply a primarily pluralistic utilitarian lens [78] to weigh potential outcomes for our direct stakeholders and for society at large, with emphasis being on the human safety of sources. However, people don't become whistleblowers in order to minimise personal risk. In the style of [122], which applies a hybrid approach of weighing both potential outcomes and desired intent in their evaluative framework for dual-use research, we strive to include representation of our stakeholders' intentions, in addition to a purely outcome-driven analysis. We pay particular attention to risks that are newly introduced or remain unmitigated in our research.

The issue of covert communication is newly introduced and unmitigated. We conduct a cost-benefit analysis between these potential harms versus the reduction in server-side metadata that the redesigned SecureDrop protocol offers.

The issue of misusing open-source tooling is similar to the debate around providing backdoors into encryption technology. In our view, there is no responsible way to address this fundamental concern without introducing a far greater harm. In high-stakes contexts such as whistleblowing where trust, transparency, and security are paramount, closed-source products are non-viable. Further, without the ability for organisations to autonomously deploy and operate SecureDrop, practical and ethical concerns arise around who controls access to secure communications tools.

Decision. Our analysis of the potential covert communication channel led us to decide on balance that it is an acceptable risk. We suggest API-level mitigations during implementation to diminish the practicality and impact of this risk.

Considering other unintended uses, we remark that the open-source nature of the project is an essential part of its trust model, and favour the implications of this decision over any alternatives.

Against increasing surveillance and dwindling accountability mechanisms in society, we opted to proceed with our research. We hope our work allows more organisations to deploy confidential, anonymous communication channels, and more people to use them.

A Additional Definitions

Definition A.1 (Pseudorandom function)**.** *We define the* pseudorandom function *(*PRF*) advantage of an adversary $\mathcal{A}$ against a function* $\mathsf{F}\colon \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ *as*

$$\mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{F}}(\mathcal{A}) := \Pr\left[\mathcal{A}^{\mathsf{F}(k,\cdot)}() \Rightarrow 1 \mid k \xleftarrow{\$} \{0,1\}^{\kappa}\right] - \Pr\left[\mathcal{A}^{R(\cdot)}() \Rightarrow 1 \mid R \xleftarrow{\$} \{\textit{all functions}\colon \mathcal{Y} \to \mathcal{Z}\}\right].$$

We adopt the definition of a *split-key pseudorandom function* from [59], which is a generalization of a pseudorandom function to multiple key inputs that behaves like a random function if at least one of its key inputs is picked uniformly at random. We generalize the security definition to allow for multiple instances.

Definition A.2 (Split-key pseudorandom function)**.** *A* split-key pseudorandom function $\mathsf{F}\colon \mathcal{K}_1 \times \cdots \times \mathcal{K}_n \times \mathcal{X} \to \mathcal{Y}$ *takes as input a finite number of keys in $\mathcal{K}_1 \times \cdots \times \mathcal{K}_n$ and a label in $\mathcal{X}$ and produces an output in $\mathcal{Y}$. We define the* multi-instance split-key pseudorandom function (skPRF) *advantage of an adversary $\mathcal{A}$ against a function* F *as*

$$\mathsf{Adv}^{\mathsf{skPRF}}_{\mathsf{F},i}(\mathcal{A}) := \Pr\left[\mathcal{A}^{\mathsf{F}(\cdots,k_{i,1},\cdots),\ldots,\mathsf{F}(\cdots,k_{i,\ell},\cdots)}() \Rightarrow 1 \mid k_{i,1},\ldots,k_{i,\ell} \xleftarrow{\$} \mathcal{K}_i\right] - \Pr\left[\mathcal{A}^{R_1(\cdots),\ldots,R_\ell(\cdots)}() \Rightarrow 1\right],$$

where $R_1,\ldots,R_\ell$ are chosen uniformly at random from the set of all functions mapping $\mathcal{K}_{n\setminus i} \times \mathcal{X} \to \mathcal{Y}$, and $\mathcal{K}_{n\setminus i}$ denotes $\mathcal{K}_1 \times \cdots \times \mathcal{K}_{i-1} \times \mathcal{K}_{i+1} \times \cdots \times \mathcal{K}_n$.

Definition A.3 (Password-based key derivation function, PBKDF)**.** *A* password-based key derivation function $\mathsf{F}\colon \mathcal{X} \to \mathcal{Y}$ *takes as input a password $pw \in \mathcal{X}$ and produces an output in $\mathcal{Y}$. We define the* password-based key derivation function (pwKDF) *advantage of an adversary $\mathcal{A}$ against a function* F *and password distribution $\mathcal{P}$, for a number r of inputs as*

$$\mathsf{Adv}^{\mathsf{pwKDF}}_{\mathsf{F},\mathcal{P},r}(\mathcal{A}) := \Pr\left[\mathcal{A}^{\mathbf{K}}() \Rightarrow 1 \mid \textit{For } i = 1 \textit{ to } r \textit{ do } pw_i \xleftarrow{\$} \mathcal{P}, \mathbf{K}[i] \leftarrow \mathsf{F}(pw_i)\right] - \Pr\left[\mathcal{A}^{\mathbf{K}}() \Rightarrow 1 \mid \textit{For } i = 1 \textit{ to } r \textit{ do } \mathbf{K}[i] \xleftarrow{\$} \mathcal{Y}\right],$$

Definition A.4 (Signature scheme)**.** *A signature scheme* SIG = (KGen, Sign, Vfy) *consists of three algorithms:*

- $\underline{\mathsf{KGen}}() \xrightarrow{\$} (sk, vk)$ *is a probabilistic* key generation *algorithm that generates a signing key sk and corresponding verification key vk.*
- $\underline{\mathsf{Sign}}(sk, m) \xrightarrow{\$} \sigma$ *is a probabilistic* signing *algorithm that takes as input a signing key sk, and a message m and outputs a signature σ.*
- $\underline{\mathsf{Vfy}}(vk, m, \sigma) \to b$ *is a deterministic* verification *algorithm that takes as input a verification key vk, message m, and signature σ, and outputs a boolean value $b \in \{\mathsf{true}, \mathsf{false}\}$ denoting whether the signature verified or not.*

We recall standard syntax and security definitions for authenticated encryption with associated data (AEAD) schemes.

$\underline{\mathrm{G}^{(n,q_D)\text{-INT-CTXT}}_{\mathrm{AEAD}}(\mathcal{A})}$:
1 for $i \in [n]$
2 $k_i \xleftarrow{\$} \mathcal{K}$
3 $nonce_i \xleftarrow{\$} \mathcal{N}$
4 $\mathcal{E} \leftarrow \emptyset$
5 $b \xleftarrow{\$} \{0,1\}$
6 $b' \xleftarrow{\$} \mathcal{A}^{O_{\mathrm{Enc}}, O_{\mathrm{Dec}}}()$
7 return $[\![b = b']\!]$

$\underline{O_{\mathrm{Enc}}(i \in [n], m, ad)}$:
8 $c \xleftarrow{\$} \mathsf{AEAD.Enc}(k_i, nonce_i, ad, m)$
9 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(i, m, c, ad)\}$
10 return $(c, nonce_i)$

$\underline{O_{\mathrm{Dec}}(i \in [n], ad)}$:
11 if $b = 0$:
12 $m \leftarrow \mathsf{AEAD.Dec}(k_i, nonce_i, ad, c)$
13 else if $\exists m', (i, m', c, ad) \in \mathcal{E}$: $m \leftarrow m'$
14 else: $m \leftarrow \perp$
15 return m

$\underline{\mathrm{G}^{(n,q_D)\text{-CCA}}_{\mathrm{AEAD}}(\mathcal{A})}$:
1 for $i \in [n]$
2 $k_i \xleftarrow{\$} \mathcal{K}$
3 $nonce_i \xleftarrow{\$} \mathcal{N}$
4 $\mathcal{E} \leftarrow \emptyset$
5 $b \xleftarrow{\$} \{0,1\}$
6 $b' \xleftarrow{\$} \mathcal{A}^{O_{\mathrm{Enc}}, O_{\mathrm{Dec}}}()$
7 return $[\![b = b']\!]$

$\underline{O_{\mathrm{Enc}}(i \in [n], m_0, m_1, ad)}$:
8 $c \xleftarrow{\$} \mathsf{AEAD.Enc}(k_i, nonce_i, ad, m_b)$
9 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(i, c, ad)\}$
10 return $(c, nonce_i)$

$\underline{O_{\mathrm{Dec}}(i \in [n], c, ad)}$:
11 if $(i, c, ad) \in \mathcal{E}$:
12 return $\perp$
13 $m \leftarrow \mathsf{AEAD.Dec}(k_i, nonce_i, ad, c)$
14 return m

Figure 10: Security games for INT-CTXT **and** CCA **security of** AEAD = (KGen, Enc, Dec)**. Adversary $\mathcal{A}$ makes at most one query per index i to O_{Enc} and at most q_D queries to O_{Dec}.**

Definition A.5 (AEAD)**.** *A nonce-based authenticated encryption scheme with associated data, keyspace $\mathcal{K}$, and nonce space $\mathcal{N}$, consists of the following three algorithms:*

- KGen *is a key generation algorithm that outputs $k \in \mathcal{K}$.*
- AEAD.Enc *is a deterministic algorithm that takes as input a key $k \in \mathcal{K}$, a nonce $nonce \in \mathcal{N}$, associated data ad, a plaintext m and produces as output a ciphertext c. We write $c \leftarrow \mathsf{AEAD.Enc}(k, nonce, ad, m)$.*
- AEAD.Dec *is a deterministic algorithm that takes as input a key $k \in \mathcal{K}$, a nonce $nonce \in \mathcal{N}$, associated data ad, a ciphertext c, and produces as output a plaintext m or an error $m = \perp$. We write $m \leftarrow \mathsf{AEAD.Dec}(k, nonce, ad, c)$.*

For correctness, we require that for all $k \in \mathcal{K}$, $ad \in \{0,1\}^*$, $m \in \{0,1\}^*$, and $nonce \in \mathcal{N}$, the following holds:

$$\mathsf{AEAD.Dec}(k, nonce, ad, \mathsf{AEAD.Enc}(k, nonce, ad, m)) = m.$$

We adopt the multi-instance security games (n, q_D)-INT-CTXT and (n, q_D)-CCA of AEAD schemes from [4], shown in Figure 10. Note that this definition of CCA security can be reduced to a single-key definition at the penalty of introducing a small collision term (from key and nonce collisions).

Definition A.6 (Indistinguishability from random strings (IND$-CPA))**.** *Let* SE *be a symmetric encryption scheme with key space $\mathcal{K}$ such that the length $|c| = \ell(|m|)$ of the ciphertext output by encryption only depends on the input message's length $|m|$. We define the* IND$-CPA *security game as follows.*

(1) The challenger samples $b \xleftarrow{\$} \{0,1\}$ and $K \xleftarrow{\$} \mathcal{K}$.
(2) $\mathcal{A}$ provides chosen message m queries to the challenger, who returns either $\mathsf{SE.Enc}(K, m)$ if $b = 1$ or a random string $c \xleftarrow{\$} \{0,1\}^{\ell(|m|)}$ if $b = 0$.
(3) $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$.

$\underline{\mathrm{G}^{\mathrm{IND\text{-}CCA}}_{\mathrm{KEM}}(\mathcal{A})}$:
1 $b \xleftarrow{\$} \{0,1\}$
2 $\mathcal{E} \leftarrow \emptyset$
3 $\ell \leftarrow 0$
4 $b' \xleftarrow{\$} \mathcal{A}^{O_{\mathrm{Decap}}, O_{\mathrm{Gen}}, O_{\mathrm{Chall}}}()$
5 return $[\![b = b']\!]$

$\underline{O_{\mathrm{Gen}}()}$:
6 $\ell \leftarrow \ell + 1$
7 $(pk_\ell, sk_\ell) \xleftarrow{\$} \mathsf{KGen}()$
8 return (pk_ℓ, ℓ)

$\underline{O_{\mathrm{Decap}}(j \in [\ell], c)}$:
9 if $\exists K : (j, c, K) \in \mathcal{E}$:
10 return K
11 $K \leftarrow \mathsf{Decap}(sk, c)$
12 return K

$\underline{O_{\mathrm{Chall}}(j \in [\ell])}$:
13 $(c, K) \xleftarrow{\$} \mathsf{Encap}(pk_j)$
14 if $b = 1$:
15 $K \xleftarrow{\$} \mathcal{K}$
16 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(j, c, K)\}$
17 return (c, K)

Figure 11: Security game for multi-challenge IND-CCA **security of a KEM.**

We define the IND$-CPA *advantage of an adversary $\mathcal{A}$ as*

$$\mathsf{Adv}^{\mathrm{IND\$\text{-}CPA}}_{\mathsf{SE}}(\mathcal{A}) := 2 \cdot \Pr[b' = b] - 1.$$

We use Diffie–Hellman assumptions over *nominal groups*, denoted $\mathcal{N} = (\mathcal{G}, g, p, \mathcal{E}_H, \mathcal{E}_U, \exp)$, as in the prior analysis of HPKE [3]. For the complete definition of a nominal group, we refer the reader to [3].

Definition A.7 (Decisional Diffie–Hellman (DDH) problem)**.** *We define the advantage of an adversary $\mathcal{A}$ against the Decisional Diffie–Hellman problem over nominal group $\mathcal{N}$, as*

$$\mathsf{Adv}^{\mathrm{DDH}}_{\mathcal{N}}(\mathcal{A}) := 2 \cdot \Pr\left[\mathcal{A}(X, Y, Z_b) = b \;\middle|\; \begin{matrix} b \xleftarrow{\$} \{0,1\}, x \xleftarrow{\$} \mathcal{E}_U, y \xleftarrow{\$} \mathcal{E}_U \\ z \xleftarrow{\$} \mathcal{E}_U, Z_0 \leftarrow g^{xy}, Z_1 \leftarrow g^z \end{matrix}\right] - 1.$$

Definition A.8 (Key encapsulation mechanism (KEM))**.** *A key encapsulation mechanism* KEM = (KGen, Encap, Decap) *consists of three algorithms:*

- $\underline{\mathsf{KGen}}() \xrightarrow{\$} (sk, pk)$ *is a probabilistic* key generation *algorithm that generates a private key sk and corresponding public key pk.*
- $\underline{\mathsf{Encap}}(pk) \xrightarrow{\$} (c, K)$ *is a probabilistic* encapsulation *algorithm that takes as input a KEM public key pk, and outputs a ciphertext c and shared secret K.*
- $\underline{\mathsf{Decap}}(sk, c) \to K$ *is a deterministic* decapsulation *algorithm that takes as input a private key sk and ciphertext c, and outputs a shared secret K.*

We recall the definition of IND-CCA security of a KEM in Figure 11. Note that we use a multi-challenge definition of IND-CCA security. We define the IND-CCA advantage of an adversary $\mathcal{A}$ as

$$\mathsf{Adv}^{\mathrm{IND\text{-}CCA}}_{\mathrm{KEM}}(\mathcal{A}) := |\Pr[\mathrm{G}^{\mathrm{IND\text{-}CCA}}_{\mathrm{KEM}}(\mathcal{A}) \Rightarrow 1] - \frac{1}{2}|.$$

An *authenticated KEM* [3] is an extension of a KEM, where encapsulation is linked to the sender's authenticity (and takes as input their private key). The recipient uses both their private key and the sender's public key to verify the origin of the shared secret.

Definition A.9 (Authenticated KEM)**.** *An* authenticated key encapsulation mechanism (AKEM) AKEM = (KGen, AuthEncap, AuthDecap) *consists of three algorithms:*

- $\underline{\mathsf{KGen}}() \xrightarrow{\$} (sk, pk)$ *is a probabilistic key generation algorithm that generates a private key sk, and a public key pk.*

$\underline{\mathrm{G}^{(q_G,q_E,q_D)\text{-Outsider-CCA}}_{\mathrm{AKEM}}(\mathcal{A})}$:
1 $\ell \leftarrow 0$
2 $\mathcal{E} \leftarrow \emptyset$
3 $b \xleftarrow{\$} \{0,1\}$
4 $b' \xleftarrow{\$} \mathcal{A}^{O_{\mathsf{Gen}},O_{\mathsf{AEncap}},O_{\mathsf{ADecap}}}()$
5 return $[\![b = b']\!]$

$\underline{O_{\mathsf{Gen}}()}$:
6 $\ell \leftarrow \ell + 1$
7 $(sk_\ell, pk_\ell) \xleftarrow{\$} \mathsf{KGen}()$
8 return (pk_ℓ, ℓ)

$\underline{O_{\mathsf{ADecap}}(j \in [\ell], pk, c)}$:
9 if $\exists K : (pk, pk_j, c, K) \in \mathcal{E}$
10 return K
11 $K \leftarrow \mathsf{AuthDecap}(sk_j, pk, c)$
12 return K

$\underline{O_{\mathsf{AEncap}}(i \in [\ell], pk)}$:
13 $(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk)$
14 if $b = 1$ and $pk \in \{pk_1, \ldots, pk_\ell\}$:
15 $K \xleftarrow{\$} \mathcal{K}$
16 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk, c, K)\}$
17 return (c, K)

Figure 12: Security game for Outsider-CCA **security of an AKEM.** AKEM = (KGen, AuthEncap, AuthDecap)**. Adversary** $\mathcal{A}$ **makes at most** q_G **queries to** O_{Gen}**, at most** q_E **queries to** O_{AEncap}**, and at most** q_D **queries to** O_{ADecap}**.**

$\underline{\mathrm{G}^{(q_G,q_E,q_D,q_C)\text{-Insider-CCA}}_{\mathrm{AKEM}}(\mathcal{A})}$:
1 $\ell \leftarrow 0$
2 $\mathcal{E} \leftarrow \emptyset$
3 $b \xleftarrow{\$} \{0,1\}$
4 $b' \xleftarrow{\$} \mathcal{A}^{O_{\mathsf{Gen}},O_{\mathsf{AEncap}},O_{\mathsf{ADecap}},O_{\mathsf{Chall}}}()$
5 return $[\![b = b']\!]$

$\underline{O_{\mathsf{Gen}}()}$:
6 $\ell \leftarrow \ell + 1$
7 $(sk_\ell, pk_\ell) \xleftarrow{\$} \mathsf{KGen}()$
8 return (pk_ℓ, ℓ)

$\underline{O_{\mathsf{AEncap}}(i \in [\ell], pk)}$:
9 $(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_i, pk)$
10 return (c, K)

$\underline{O_{\mathsf{ADecap}}(j \in [\ell], pk, c)}$:
11 if $\exists K : (pk, pk_j, c, K) \in \mathcal{E}$
12 return K
13 $K \leftarrow \mathsf{AuthDecap}(sk_j, pk, c)$
14 return K

$\underline{O_{\mathsf{Chall}}(j \in [\ell], sk)}$:
15 $(c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk, pk_j)$
16 if $b = 1$:
17 $K \xleftarrow{\$} \mathcal{K}$
18 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(\mu(sk), pk_j, c, K)\}$
19 return (c, K)

Figure 13: Security game for Insider-CCA **security of an AKEM.** AKEM = (KGen, AuthEncap, AuthDecap)**. Adversary** $\mathcal{A}$ **makes at most** q_G **queries to** O_{Gen}**, at most** q_E **queries to** O_{AEncap}**, at most** q_D **queries to** O_{ADecap}**, and at most** q_C **queries to** O_{Chall}**. We assume there exists a function** μ **which returns the corresponding public key of a private key,** $\mu(sk) = pk$**.**

- $\underline{\mathsf{AuthEncap}}(sk_S, pk_R) \xrightarrow{\$} (c, K)$ *is a probabilistic encapsulation algorithm that takes as input a (sender) KEM private key* sk_S *and a (receiver) KEM public key* pk_R*, and outputs a ciphertext* c *and shared secret* K*.*
- $\underline{\mathsf{AuthDecap}}(sk_R, pk_S, c) \rightarrow K$ *is a deterministic decapsulation algorithm that takes as input a (receiver) private key* sk_R*, a (sender) public key* pk_S*, and a ciphertext* c*, and outputs a shared secret* K*.*

Definition A.10 (AKEM encapsulation/decapsulation correctness)**.** *We say that an AKEM* AKEM = (KGen, AuthEncap, AuthDecap) *is* δ_{AKEM}-correct *if* $\Pr[\mathsf{AuthDecap}(sk_R, pk_S, c) \neq K \mid (sk_S, pk_S) \xleftarrow{\$} \mathsf{KGen}(), (sk_R, pk_R) \xleftarrow{\$} \mathsf{KGen}(), (c, K) \xleftarrow{\$} \mathsf{AuthEncap}(sk_S, pk_R)] \leq \delta_{\mathsf{AKEM}}$.

Definition A.11 (Public key encryption (PKE))**.** *A public key encryption scheme* PKE *consists of the following three algorithms:*

- $\underline{\mathsf{KGen}}() \xrightarrow{\$} (sk, pk)$ *is a probabilistic key generation algorithm that outputs a private key* sk *and a public key* pk*.*

$\underline{\mathrm{G}^{(q_G,q_E,q_D,q_C)\text{-Insider-CCA}}_{\mathrm{APKE}}(\mathcal{A})}$:
1 $\ell \leftarrow 0$
2 $\mathcal{E} \leftarrow \emptyset$
3 $b \xleftarrow{\$} \{0,1\}$
4 $b' \xleftarrow{\$} \mathcal{A}^{O_{\mathsf{Gen}},O_{\mathsf{AEnc}},O_{\mathsf{ADec}},O_{\mathsf{Chall}}}()$
5 return $[\![b = b']\!]$

$\underline{O_{\mathsf{Gen}}()}$:
6 $\ell \leftarrow \ell + 1$
7 $(sk_\ell, pk_\ell) \xleftarrow{\$} \mathsf{KGen}()$
8 return (pk_ℓ, ℓ)

$\underline{O_{\mathsf{ADec}}(j \in [\ell], pk, c, ad, info)}$:
9 if $(pk, pk_j, c, ad, info) \in \mathcal{E}$
10 return $\perp$
11 $m \leftarrow \mathsf{AuthDec}(sk_j, pk, c, ad, info)$
12 return m

$\underline{O_{\mathsf{AEnc}}(i \in [\ell], pk, m, ad, info)}$:
13 $c \xleftarrow{\$} \mathsf{AuthEnc}(sk_i, pk, m, ad, info)$
14 return c

$\underline{O_{\mathsf{Chall}}(j \in [\ell], sk, m_0, m_1, ad, info)}$:
15 if $|m_0| \neq |m_1|$ return $\perp$
16 $c \xleftarrow{\$} \mathsf{AuthEnc}(sk, pk_j, m_b, ad, info)$
17 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(\mu(sk), pk_j, c, ad, info)\}$
18 return c

Figure 14: Security game for Insider-CCA **security of an APKE** APKE = (KGen, AuthEnc, AuthDec)**. Adversary** $\mathcal{A}$ **makes at most** q_G **queries to** O_{Gen}**, at most** q_E **queries to** O_{AEnc}**, at most** q_D **queries to** O_{ADec} **and at most** q_C **queries to** O_{Chall}**. We assume there exists a function** μ **which returns the corresponding public key of a private key,** $\mu(sk) = pk$**.**

- $\underline{\mathsf{Enc}}(pk, m) \xrightarrow{\$} c$ *is a probabilistic encryption algorithm that takes as input a receiver public key* pk *and a message* m*, and outputs a ciphertext* c*.*
- $\underline{\mathsf{Dec}}(sk, c) \rightarrow m$ *is a deterministic decryption algorithm that takes as input a receiver private key* sk *and a ciphertext* c*, and outputs a message* m*.*

Definition A.12 (Authenticated PKE (APKE))**.** *An authenticated public key encryption scheme* APKE *consists of the following three algorithms:*

- $\underline{\mathsf{KGen}}() \xrightarrow{\$} (sk, pk)$ *is a probabilistic key generation algorithm that outputs a private key* sk *and a public key* pk*.*
- $\underline{\mathsf{AuthEnc}}(sk_S, pk_R, m, ad) \xrightarrow{\$} c$ *is a probabilistic encryption algorithm that takes as input a (sender) private key* sk_S*, a (receiver) public key* pk_R*, a message* m*, and associated data* ad*, and outputs a ciphertext* c*.*
- $\underline{\mathsf{AuthDec}}(sk_R, pk_S, c, ad) \rightarrow m$ *is a deterministic decryption algorithm that takes as input a (receiver) private key* sk_R*, a (sender) public key* pk_S*, a ciphertext* c*, and associated data* ad*, and outputs a message* m*.*

For correctness of an APKE scheme, we require that for all messages $m \in \{0,1\}^*$, associated data $ad \in \{0,1\}^*$ and info strings $info \in \{0,1\}^*$, the following holds:

$$\Pr\left[\mathsf{AuthDec}(sk_R, pk_S, c, ad, info) = m \;\middle|\; \begin{array}{l}(sk_S, pk_S) \xleftarrow{\$} \mathsf{KGen}(),\\ (sk_R, pk_R) \xleftarrow{\$} \mathsf{KGen}(),\\ c \xleftarrow{\$} \mathsf{AuthEnc}(sk_S, pk_R, m, ad, info)\end{array}\right] = 1.$$

We repeat the definitions of security for APKE schemes from [3] in Figure 14 and 15. The definition of Outsider-Auth could be strengthened by allowing the adversary $\mathcal{A}$ to corrupt keys of users other than the ones used in its challenge output. Security in that case follows from a standard hybrid argument.

$\underline{\mathrm{G}^{(q_\mathrm{G},q_\mathrm{E},q_\mathrm{D})\text{-Outsider-Auth}}_{\mathrm{APKE}}(\mathcal{A}):}$

$\ell \leftarrow 0$
$\mathcal{E} \leftarrow \emptyset$
$(i^*, j^*, c^*, ad^*, info^*) \xleftarrow{\$} \mathcal{A}^{O_{\mathrm{Gen}}, O_{\mathrm{AEnc}}, O_{\mathrm{ADec}}}()$
return $[\![(pk_{i^*}, pk_{j^*}, c^*, ad^*, info^*) \notin \mathcal{E}$ and $\mathsf{AuthDec}(sk_{j^*}, pk_{i^*}, c^*, ad^*, info^*) \neq \perp]\!]$

$\underline{O_{\mathrm{Gen}}():}$

$\ell \leftarrow \ell + 1$
$(sk_\ell, pk_\ell) \xleftarrow{\$} \mathsf{KGen}()$
return (pk_ℓ, ℓ)

$\underline{O_{\mathrm{AEnc}}(i \in [\ell], pk, m, ad, info):}$

$c \xleftarrow{\$} \mathsf{AuthEnc}(sk_i, pk, m, ad, info)$
$\mathcal{E} \leftarrow \mathcal{E} \cup \{(pk_i, pk, c, ad, info)\}$
return c

$\underline{O_{\mathrm{ADec}}(j \in [\ell], pk, c, ad, info):}$

$m \leftarrow \mathsf{AuthDec}(sk_j, pk, c, ad, info)$
return m

Figure 15: Security game for Outsider-Auth **security of an APKE** APKE = (KGen, AuthEnc, AuthDec)**. Adversary $\mathcal{A}$ makes at most q_G queries to O_{Gen}, at most q_E queries to O_{AEnc}, and at most q_D queries to O_{ADec}.**

In $\mathrm{G}^{(q_\mathrm{G},q_\mathrm{E},q_\mathrm{D},q_\mathrm{C})\text{-Insider-CCA}}_{\mathrm{APKE}}(\mathcal{A})$, the adversary makes at most q_G queries to O_{Gen}, at most q_E queries to O_{AEnc}, at most q_D queries to O_{ADec} and at most q_C queries to O_{Chall}. We define the advantage of an adversary $\mathcal{A}$ as

$$\mathsf{Adv}^{(q_\mathrm{G},q_\mathrm{E},q_\mathrm{D},q_\mathrm{C})\text{-Insider-CCA}}_{\mathcal{A},\mathrm{APKE}} := \left| \Pr[\mathrm{G}^{(q_\mathrm{G},q_\mathrm{E},q_\mathrm{D},q_\mathrm{C})\text{-Insider-CCA}}_{\mathrm{APKE}}(\mathcal{A}) \Rightarrow 1] - \frac{1}{2} \right|.$$

In $\mathrm{G}^{(q_\mathrm{G},q_\mathrm{E},q_\mathrm{D})\text{-Outsider-Auth}}_{\mathrm{APKE}}(\mathcal{A})$, the adversary makes at most q_G queries to O_{Gen}, at most q_E queries to O_{AEnc}, and at most q_D queries to O_{ADec}. We define the advantage of an adversary $\mathcal{A}$ as

$$\mathsf{Adv}^{(q_\mathrm{G},q_\mathrm{E},q_\mathrm{D})\text{-Outsider-Auth}}_{\mathcal{A},\mathrm{APKE}} := |\Pr[\mathrm{G}^{(q_\mathrm{G},q_\mathrm{E},q_\mathrm{D})\text{-Outsider-Auth}}_{\mathrm{APKE}}(\mathcal{A}) \Rightarrow 1]|.$$

B PRF Security of KS

As a building block, we establish the split-key PRF security of KS as used in HPKE [9] and in our application. In Figure 4, we wrote that KS is a two-keyed function that takes as input a key K, a pre-shared key *psk*, and info *info*. In practice (and in the specification), the function also takes as input the HPKE mode and a psk identifier; as in [4], we abstract these away since they are merely hashed in as part of the label in the call to LabeledExpand. For our purposes, the KS function can be written as in Figure 16.

$\underline{\mathsf{KS}(K, psk, info):}$

$secret = \mathsf{LabeledExtract}(K, \text{“}\texttt{secret}\text{”}, psk)$
$key = \mathsf{LabeledExpand}(secret, \text{“key”}, info, keylength)$
$nonce = \mathsf{LabeledExpand}(secret, \text{“base_nonce”}, info, noncelength)$
return $(key, nonce)$

Figure 16: HPKE's key derivation function, KS **[9], with psk identifier and mode abstracted away.** *keylength* **and** *noncelength* **denote the desired key and nonce lengths, respectively.**

In our application, K will be a DH-AKEM key and *psk* will be an ML-KEM key, each 32 bytes and with high entropy. LabeledExpand and LabeledExtract are instantiated with HKDF. Specifically, $\mathsf{LabeledExtract}(K, \text{“}\texttt{secret}\text{”}, psk)$ runs $\mathsf{HKDF.Extract}(K, \text{“}\texttt{secret}\text{”} \| psk)$. Since all inputs here are fixed-length, unambiguously encoded, and fit into one HMAC block, the proof that HMAC is a dual-PRF [5] applies, under the assumptions that the compression function underlying the hash function used in HMAC has sufficiently large message-input size and appropriate PRF and collision resistance properties (see [5] for more details). It immediately follows that KS is a PRF in both key inputs, i.e., a split-key PRF.

C Proof of Theorem 5.1

Theorem 5.1. *Let* $\mathsf{KEM}_{\mathrm{PQ}}$ *be a KEM,* AKEM *an AKEM,* KS *a two-keyed function, and* AEAD *an AEAD scheme. Let* SD-APKE = SD-APKE[AKEM, $\mathsf{KEM}_{\mathrm{PQ}}$, AEAD, KS] *be as defined in Figure 5. For any* Insider-CCA *adversary* $\mathcal{A}$ *against* SD-APKE *we give algorithms* $\mathcal{B}_1$–$\mathcal{B}_3$ *and* $\mathcal{C}_1$–$\mathcal{C}_2$ *such that*

$$\begin{aligned}\mathsf{Adv}^{\text{Insider-CCA}}_{\text{SD-APKE}}(\mathcal{A}) \leq \min\{&2 \cdot \mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{KEM}_{\mathrm{PQ}}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{skPRF}}_{\mathsf{KS},2}(\mathcal{B}_2),\\ &2 \cdot \mathsf{Adv}^{\text{Insider-CCA}}_{\mathrm{AKEM}}(\mathcal{C}_1) + \mathsf{Adv}^{\mathsf{skPRF}}_{\mathsf{KS},1}(\mathcal{C}_2)\}\\ &+ \mathsf{Adv}^{\mathrm{CCA}}_{\mathrm{AEAD}}(\mathcal{B}_3).\end{aligned}$$

Proof. We separately consider the two advantage bounds via two proof cases.

Case I. In the first case, we reduce to IND-CCA security of the KEM (as opposed to the AKEM), and proceed via a series of game hops.

Game I.0. We start with the security game for Insider-CCA, $\mathrm{G}^{\text{Insider-CCA}}_{\text{SD-APKE}[\mathsf{KEM}_{\mathrm{PQ}},\mathrm{AKEM},\mathsf{KS},\mathrm{AEAD}]}(\mathcal{A})$.

Game I.1. In $\mathrm{G}_{\mathrm{I}.1}$, in queries to the challenge encryption oracle, O_{Chall}, we compute AuthEnc using the $b' = 0$ (real) output from the (multi-challenge) IND-CCA security game of the underlying KEM. For queries to O_{ADec}, we use the KEM decapsulation oracle, O_{Decap} to obtain the value K to be used as psk input to pskADec. By correctness of the KEM, this is primarily a syntactic change, so it follows that $\Pr[\mathrm{G}_{\mathrm{I}.0}] = \Pr[\mathrm{G}_{\mathrm{I}.1}]$.

Game I.2. In $\mathrm{G}_{\mathrm{I}.2}$, we replace the KEM shared secret of $\mathsf{KEM}_{\mathrm{PQ}}$ in the challenge oracle with a uniformly random value. That is, we use the $b' = 1$ (random) output of the (multi-challenge) IND-CCA game of the underlying KEM. For queries to O_{ADec}, if there exists a triple $(j, c, K) \in \mathcal{E}$ (of the IND-CCA game of the KEM) where j is the recipient identity and c the ciphertext in the query, then we decrypt using the (random) KEM shared secret K stored in the triple in $\mathcal{E}$ as the psk input into pskADec. Otherwise, we use the KEM shared secret output by the decapsulation oracle. We can bound the adversary's difference in advantage by a reduction $\mathcal{B}_1$ to the (multi-challenge) IND-CCA security of $\mathsf{KEM}_{\mathrm{PQ}}$. That is,

$$\mathsf{Adv}^{\mathrm{G}_{\mathrm{I}.2}}_{\text{SD-APKE}} \leq |\Pr[\mathrm{G}^{\text{IND-CCA},b=0}_{\mathsf{KEM}_{\mathrm{PQ}}}(\mathcal{B}_1)] - \Pr[\mathrm{G}^{\text{IND-CCA},b=1}_{\mathsf{KEM}_{\mathrm{PQ}}}(\mathcal{B}_1)]|.$$

By advantage rewriting, we have that,

$$\Pr[\mathrm{G}_{\mathrm{I}.1}] - \Pr[\mathrm{G}_{\mathrm{I}.2}] \leq 2 \cdot \mathsf{Adv}^{\text{IND-CCA}}_{\mathsf{KEM}_{\mathrm{PQ}}}(\mathcal{B}_1).$$

Game I.3. Next, in $G_{I.3}$, we replace the output of KS with uniformly random values. The decryption oracle is computed as in the previous game. By the previous game, the second key input to KS is uniformly random, so we can bound the adversary's difference in advantage by a reduction $\mathcal{B}_2$ to the multi-instance split-key PRF security, skPRF, of KS in its second input:

$$\Pr[G_{I.2}] - \Pr[G_{I.3}] \leq \mathsf{Adv}^{\mathsf{skPRF}}_{\mathsf{KS},2}(\mathcal{B}_2).$$

Game I.4. Now, the game for SD-APKE now uses random keys when computing AEAD.Enc in queries to O_{Chall}. Thus, the adversary's advantage in this game can be bounded by a reduction $\mathcal{B}_3$ to the CCA security of AEAD.

$$\Pr[G_{I.3}] - \Pr[G_{I.4}] \leq \mathsf{Adv}^{\mathsf{CCA}}_{\mathsf{AEAD}}(\mathcal{B}_3).$$

Case II. The second case reduces to Insider-CCA of the AKEM. It proceeds similarly to the first.

Game II.0. We start with the security game for Insider-CCA, $G^{\text{Insider-CCA}}_{\text{SD-APKE}[\text{KEM}_{\text{PQ}},\text{AKEM},\text{KS},\text{AEAD}]}(\mathcal{A})$.

Game II.1. In $G_{II.1}$, in queries to the challenge encryption oracle, O_{Chall}, we compute AuthEnc using the $b' = 0$ (real) output from an O_{AEncap} query in the Insider-CCA security game of the underlying AKEM. We use the AKEM decapsulation oracle, O_{ADecap}, to obtain the AKEM shared secret K to be used as input to pskADec on queries to O_{ADec} in the Insider-CCA game. This is primarily a syntactic change, so it follows that $\Pr[G_{II.0}] = \Pr[G_{II.1}]$.

Game II.2. In $G_{II.2}$, we replace the AKEM shared secret of AKEM in the challenge oracle with a uniformly random value. That is, we use the $b' = 1$ (random) output of the Insider-CCA game of the underlying AKEM. For queries to O_{ADec}, if there exists a quadruple $(pk, pk_j, c, K) \in \mathcal{E}$ (of the Insider-CCA game of the AKEM) where j is the recipient identity, pk the sender public key, and c the ciphertext in the query, then we decrypt using the (random) KEM shared secret K stored in $\mathcal{E}$ as the AKEM shared secret used as input to KS. Otherwise, we use the usual AKEM shared secret output by the decapsulation oracle. We can bound the adversary's difference in advantage by a reduction C_1 to the Insider-CCA security of AKEM. That is,

$$\mathsf{Adv}^{G_{II.2}}_{\text{SD-APKE}} \leq |\Pr[G^{\text{Insider-CCA},b=0}_{\text{AKEM}}(C_1)] - \Pr[G^{\text{Insider-CCA},b=1}_{\text{AKEM}}(C_1)]|.$$

By advantage rewriting, we have that,

$$\Pr[G_{II.1}] - \Pr[G_{II.2}] \leq 2 \cdot \mathsf{Adv}^{\text{Insider-CCA}}_{\text{AKEM}}(C_1).$$

Game II.3. Next, in $G_{II.3}$, we replace the output of KS with uniformly random values. The decryption oracle is computed as in the previous game. By the previous game, the first key input to KS is uniformly random, so we can bound the adversary's difference in advantage by a reduction C_2 to the multi-instance split-key PRF security, skPRF, of KS in its first input:

$$\Pr[G_{II.2}] - \Pr[G_{II.3}] \leq \mathsf{Adv}^{\mathsf{skPRF}}_{\mathsf{KS},1}(C_2).$$

KGen():

1 $sk \xleftarrow{\$} \mathcal{E}_H$
2 $pk \leftarrow g^{sk}$
3 return (sk, pk)

ExtractAndExpand(dh, $kem_context$):

1 $ikm \leftarrow$ "HPKE − v1"$\|suite_{id}\|$"eae − prk"$\|dh$
2 $info \leftarrow \mathsf{Encode}(N)\|$"HPKE − v1"$\|suite_{id}\|$ "shared_secret"$\|kem_context$
3 return KDF("", ikm, $info$)

AuthDecap($sk_R \in \mathcal{E}_H, pk_S \in \mathcal{G}, c \in \mathcal{G}$):

1 $kem_context \leftarrow c\|g^{sk_R}\|pk_S$
2 $dh \leftarrow {pk_E}^{sk_R}\|{pk_S}^{sk_R}$
3 $K \leftarrow$ ExtractAndExpand(dh, $kem_context$)
4 return K

AuthEncap($sk_S \in \mathcal{E}_H, pk_R \in \mathcal{G}$):

1 $(sk_E, pk_E) \xleftarrow{\$}$ KGen()
2 $dh \leftarrow {pk_R}^{sk_E}\|{pk_R}^{sk_S}$
3 $c \leftarrow pk_E$
4 $kem_context \leftarrow c\|pk_R\|g^{sk_S}$
5 $K \leftarrow$ ExtractAndExpand(dh, $kem_context$)
6 return (c, K)

Figure 17: DH-AKEM$[\mathcal{N}, \mathsf{KDF}]$ = (KGen, AuthEncap, AuthDecap)**, as defined in the HPKE specification [9], constructed from a nominal group $\mathcal{N}$ and key derivation function** KDF. $suite_{id}$ **identifies the KEM in use.** Encode(N) **is the two-byte encoding of the length N.**

Game II.4. Now, the game for SD-APKE now uses random keys when computing AEAD.Enc in queries to O_{Chall}. Thus, the adversary's advantage in this game can be bounded by a reduction to the CCA security of AEAD. This is the same as $G_{I.4}$, so we have that:

$$\Pr[G_{II.3}] - \Pr[G_{II.4}] \leq \mathsf{Adv}^{\mathsf{CCA}}_{\mathsf{AEAD}}(\mathcal{B}_3).$$

Collecting the bounds yields the theorem statement.

□

D Anonymity of SD-APKE

To enable our proof of message unlinkability in the protocol, we first prove anonymity properties of the underlying AKEM scheme. Then we prove that these imply a stronger multi-user multi-challenge property of the resulting APKE scheme. We recall the definition of DH-AKEM in Figure 17.

We require a weak notion of anonymity for the AKEM, where the adversary receives only the ciphertext output by the AKEM encapsulation, and does not have access to a decapsulation oracle. The formal definition, for both sender and receiver anonymity, is given in Figure 18. The definition is analogous to existing (weak) anonymity notions for KEMs [63]. We define the respective advantages of an adversary $\mathcal{A}$ against the $X \in$ {Send-wANO-CPA, Recv-wANO-CPA} security of an AKEM AKEM as

$$\mathsf{Adv}^{\text{Send-wANO-CPA}}_{\text{AKEM}}(\mathcal{A}) := 2 \cdot \Pr\left[G^{\text{Send-wANO-CPA}}_{\text{AKEM}}(\mathcal{A}) \Rightarrow 1\right] - 1,$$

and

$$\mathsf{Adv}^{\text{Recv-wANO-CPA}}_{\text{AKEM}}(\mathcal{A}) := 2 \cdot \Pr\left[G^{\text{Recv-wANO-CPA}}_{\text{AKEM}}(\mathcal{A}) \Rightarrow 1\right] - 1.$$

Considering DH-AKEM, it is clear that the Send-wANO-CPA and Recv-wANO-CPA properties trivially hold, since that KEM

$\underline{\mathrm{G}^{\text{Send-wANO-CPA}}_{\text{AKEM}}(\mathcal{A})}$:
1 $b \xleftarrow{\$} \{0,1\}$
2 For $i = 0..n$:
3 $\quad (sk_i, pk_i) \xleftarrow{\$} \mathsf{KGen}()$
4 $r \leftarrow \mathcal{A}(\{pk_i\}_{i\in[n]})$
5 $(c^*, K^*) \xleftarrow{\$} \mathsf{AuthEncap}(sk_b, pk_r)$
6 $b' \xleftarrow{\$} \mathcal{A}(c^*)$
7 return $[\![b = b']\!]$

$\underline{\mathrm{G}^{\text{Recv-wANO-CPA}}_{\text{AKEM}}(\mathcal{A})}$:
1 $b \xleftarrow{\$} \{0,1\}$
2 For $i = 0..n$:
3 $\quad (sk_i, pk_i) \xleftarrow{\$} \mathsf{KGen}()$
4 $s \leftarrow \mathcal{A}(\{pk_i\}_{i\in[n]})$
5 $(c^*, K^*) \xleftarrow{\$} \mathsf{AuthEncap}(sk_s, pk_b)$
6 $b' \xleftarrow{\$} \mathcal{A}(c^*)$
7 return $[\![b = b']\!]$

Figure 18: Security games for Send-wANO-CPA **and** Recv-wANO-CPA **of an AKEM** AKEM = (KGen, AuthEncap, AuthDecap) **capturing weak sender and receiver anonymity.**

$\underline{\mathrm{G}^{\text{mu-ANO-CPA}}_{\text{APKE}}(\mathcal{A})}$:
1 $b \xleftarrow{\$} \{0,1\}$; $\ell \leftarrow 0$
2 $b' \xleftarrow{\$} \mathcal{A}^{O_{\mathsf{Gen}}, O_{\mathsf{AEncSend}}, O_{\mathsf{AEncRecv}}}()$
3 return $[\![b = b']\!]$

$\underline{O_{\mathsf{Gen}}}$:
4 $(sk_\ell, pk_\ell) \xleftarrow{\$} \mathsf{KGen}()$
5 $\ell \leftarrow \ell + 1$
6 return (pk_ℓ, ℓ)

$\underline{O_{\mathsf{AEncSend}}((i_0, i_1), r, m)}$:
7 Require: $r, i_0, i_1 \le \ell$
8 $c \xleftarrow{\$} \mathsf{AuthEnc}(sk_{i_b}, pk_r, m)$
9 return c

$\underline{O_{\mathsf{AEncRecv}}((j_0, j_1), s, m)}$:
10 Require: $s, j_0, j_1 \le \ell$
11 $c \xleftarrow{\$} \mathsf{AuthEnc}(sk_s, pk_{j_b}, m)$
12 return c

Figure 19: Security game for mu-ANO-CPA **of an APKE** APKE = (KGen, AuthEnc, AuthDec) **capturing both sender and receiver anonymity. We omit associated data and info inputs and assume these are fixed for simplicity.**

ciphertext returned by AuthEncap is completely independent of the sender and receiver keys. In particular, the KEM ciphertext is an independent, ephemeral Diffie–Hellman public key. It follows that $\mathsf{Adv}^{\text{Send-wANO-CPA}}_{\text{AKEM}}(\mathcal{A}) = \mathsf{Adv}^{\text{Recv-wANO-CPA}}_{\text{AKEM}}(\mathcal{A}) = 0$.

Now that we have established the required anonymity properties of DH-AKEM, we show that these imply a stronger multi-user notion of anonymity of SD-APKE. The formal definition of multi-user anonymity is given in Figure 19. We note that we do not opt for a stronger notion such as that of anonymous AE [30] which accounts also for nonces in their anonymity definition. This is because for our use case, nonces are never sent on the wire and the associated data is constant across a given newsroom's SecureDrop instance, so we need only consider the APKE ciphertexts themselves.

The multi-user anonymity property relies on several other properties of the building blocks of SD-APKE. Anonymity of ML-KEM (ANO-CCA) is proven in [94], and Outsider-CCA of DH-AKEM is proven in [3].

Theorem D.1. *Let* SD-APKE = SD-APKE[AKEM, $\mathsf{KEM}_{\mathrm{PQ}}$, AEAD, KS] *be as defined in Figure 5, where* AKEM *is an AKEM,* $\mathsf{KEM}_{\mathrm{PQ}}$ *a KEM,* AEAD *an AEAD scheme, and* KS *a two-keyed function. For any adversary* $\mathcal{A}$ *against the* mu-ANO-CPA *security of* SD-APKE*, using at most* q_{R} *queries to* O_{AEncRecv} *and* q_{S} *queries to* O_{AEncSend}*, we give algorithms* $\mathcal{B}_1$–$\mathcal{B}_4$ *such that*

$$\begin{aligned}\mathsf{Adv}^{\text{mu-ANO-CPA}}_{\text{SD-APKE}}(\mathcal{A}) &\le q_{\mathrm{R}} \cdot \mathsf{Adv}^{\text{ANO-CPA}}_{\mathsf{KEM}_{\mathrm{PQ}}}(\mathcal{B}_1) \\ &+ 2 \cdot (q_{\mathrm{S}} + q_{\mathrm{R}}) \cdot \mathsf{Adv}^{\text{Outsider-CPA}}_{\text{AKEM}}(\mathcal{B}_2) \\ &+ q_{\mathrm{S}} \cdot \mathsf{Adv}^{\text{Send-wANO-CPA}}_{\text{AKEM}}(\mathcal{B}_3) + q_{\mathrm{R}} \cdot \mathsf{Adv}^{\text{Recv-wANO-CPA}}_{\text{AKEM}}(\mathcal{B}_4).\end{aligned}$$

Proof. **Game 0.** We start with the security game for mu-ANO-CPA, $\mathrm{G}^{\text{mu-ANO-CPA}}_{\text{SD-APKE}}(\mathcal{A})$ with $b = 0$, where SD-APKE = (KGen, AuthEncap, AuthDecap) is defined in Figure 5. We proceed via a series of game hops to end at the same game with $b = 1$.

Game 1. In G_1, we apply a series of hybrids over all queries to O_{AEncRecv} where we replace each (c_2, K_2) with the output of evaluating $\mathsf{KEM}_{\mathrm{PQ}}.\mathsf{Encap}(pk_{j_1})$ (Figure 6, line 4) rather than $\mathsf{Encap}(pk_{j_0})$. In each replacement, we can bound the difference in the adversary $\mathcal{A}$'s advantage by a reduction $\mathcal{B}_1$ to the ANO-CPA security of $\mathsf{KEM}_{\mathrm{PQ}}$. It follows that

$$\Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1] \le q_{\mathrm{R}} \cdot \mathsf{Adv}^{\text{ANO-CPA}}_{\mathsf{KEM}_{\mathrm{PQ}}}(\mathcal{B}_1).$$

Game 2. Next, apply a series of hybrids for each query to O_{AEncRecv}, where in each call to $\mathsf{AuthEnc}(sk_s, pk_{j_0}, m)$, we replace the KEM key that is output by the underlying AKEM call to $\mathsf{AuthEncap}(sk_s, pk_{j_0})$ (Figure 4, line 1) with a uniformly random KEM key. Similarly, we apply a series of hybrids for each query to O_{AEncSend}, where in each call to $\mathsf{AuthEnc}(sk_{i_0}, pk_r, m)$, we replace the KEM key that is output by the underlying AKEM call to $\mathsf{AuthEncap}(sk_{i_0}, pk_r)$ with a uniformly random KEM key. All values derived from this KEM key use the uniformly random value. We bound this game hop by a reduction $\mathcal{B}_2$ to the Outsider-CPA security of AKEM. Note that we need not simulate any decapsulation/decryption queries, and so CCA security is not required. In particular,

$$\Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_2] \le (q_{\mathrm{S}} + q_{\mathrm{R}}) \cdot \mathsf{Adv}^{\text{Outsider-CPA}}_{\text{AKEM}}(\mathcal{B}_2).$$

Game 3. We now use a series of hybrids over all queries to O_{AEncSend}, wherein we replace the KEM ciphertext that is output by $\mathsf{AKEM.AuthEncap}(sk_{i_0}, pk_r)$ with the KEM ciphertext resulting from the evaluation of $\mathsf{AKEM.AuthEncap}(sk_{i_1}, pk_r)$. We bound this by a reduction $\mathcal{B}_3$ to Send-wANO-CPA security of AKEM. Therefore,

$$\Pr[\mathrm{G}_2] - \Pr[\mathrm{G}_3] \le q_{\mathrm{S}} \cdot \mathsf{Adv}^{\text{Send-wANO-CPA}}_{\text{AKEM}}(\mathcal{B}_3).$$

Game 4. Similarly to the previous step, we use a series of hybrids, this time over all queries to O_{AEncRecv} to replace the KEM ciphertext output by $\mathsf{AKEM.AuthEncap}(sk_s, pk_{j_0})$ with the KEM ciphertext resulting from the evaluation of $\mathsf{AKEM.AuthEncap}(sk_s, pk_{j_1})$. Again, this can be bounded by a reduction to Recv-wANO-CPA of AKEM.

$$\Pr[\mathrm{G}_3] - \Pr[\mathrm{G}_4] \le q_{\mathrm{R}} \cdot \mathsf{Adv}^{\text{Recv-wANO-CPA}}_{\text{AKEM}}(\mathcal{B}_4).$$

Game 5. Finally, we recompute the KEM key output by calls to $\mathsf{AKEM.AuthEncap}(\cdot, \cdot)$ as the key generated by the evaluation of $\mathsf{AKEM.AuthEncap}(sk_{i_1}, pk_r)$ (in calls to $O_{\mathsf{AEncapSend}}$) or as the evaluation of $\mathsf{AKEM.AuthEncap}(sk_s, pk_{j_1})$ (in calls to $O_{\mathsf{AEncapRecv}}$). This is the inverse of G_2, so it follows that,

$$\Pr[\mathrm{G}_4] - \Pr[\mathrm{G}_5] \le (q_{\mathrm{S}} + q_{\mathrm{R}}) \cdot \mathsf{Adv}^{\text{Outsider-CPA}}_{\text{AKEM}}(\mathcal{B}_2).$$

Now this is exactly the game with $b = 1$. Collecting the bounds yields the theorem statement.

□

E Proof of Unlinkability of Messages

We begin with some general syntax to enable our analysis. We can describe our protocol SD that captures the message sending and retrieval protocols from Figures 3c and 3d. Formally, we write that the protocol SD specifies two randomized algorithms SD.X: $\{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ where $\mathsf{X} \in \{\mathsf{C},\mathsf{S}\}$ captures the client or server, respectively. Upon inputting its current state $st_\mathsf{X} \in \{0,1\}^*$, and a received message $m_{\text{in}} \in \{0,1\}^*$, a party computes an outgoing message m_{out} as well as an updated state st_X, written $(st_\mathsf{X}, m_{\text{out}}) \leftarrow \mathsf{SD.X}(st_\mathsf{X}, m_{\text{in}})$. A client's state, st_C, includes their long-term keys. Clients correspond to senders or receivers who may be sources or journalists interacting with the server. The server's state, st_S, includes the database of ciphertexts containing the ciphertext triples and their corresponding ids that is currently stored by the server. We exclude the process of key fetching (Figure 3b) from our analysis because every user, source or journalist, will execute this protocol upon each visit to the SecureDrop instance, so it provides no additional information for the server to distinguish users.

We define a transcript-generating function for the two components of the protocol: message sending and message retrieval. The transcript-generating function for message sending, denoted SD.Send, takes as input a client state st_C, a server state st_S, a message m_{in} corresponding to the plaintext message the client wants to send, and a recipient identifier *recv* corresponding to the user in the system to which they want to send the message. The transcript-generating function returns the ciphertext triple (C_S, X, Z) generated by honestly running the message sending protocol (Figure 3c) for client C sending the plaintext message m_{in} to the recipient *recv*. The function also returns the updated server state st_S, which will now include the database with the additional ciphertext triple (C_S, X, Z) and a newly generated ciphertext id.

$\underline{\mathsf{SD.Send}(st_\mathsf{C}, st_\mathsf{S}, m_{\text{in}}, recv)}$
$(st_i, m_{\text{out}}) \xleftarrow{\$} \mathsf{SD.C}(st_\mathsf{C}, (m_{\text{in}}, recv))$
// m_{out} contains the ciphertext triple (C_S, X, Z)
$(st_\mathsf{S}, \varepsilon) \xleftarrow{\$} \mathsf{SD.S}(st_\mathsf{S}, m_{\text{out}})$
Return $(st_\mathsf{S}, m_{\text{out}})$

The transcript-generating function for the message retrieval protocol, denoted SD.Retr, takes as input a client state st_C and a server state st_S and returns the transcript generated by (honestly) running the message retrieval protocol (Figure 3d), as well as the resulting server state st_S (which would be unchanged).

$\underline{\mathsf{SD.Retr}(st_\mathsf{C}, st_\mathsf{S})}$
$\tau \leftarrow ()$
repeat:
  $(st_\mathsf{S}, m_1) \xleftarrow{\$} \mathsf{SD.S}(st_\mathsf{S}, \mathtt{RequestMessages})$
  // m_1 contains the set of encrypted ids
  $(st_\mathsf{C}, m_2) \xleftarrow{\$} \mathsf{SD.C}(st_\mathsf{C}, m_1)$
  // m_2 contains a ciphertext id *cid* to fetch or the empty string ε
  if $m_2 = \varepsilon$: return (st_S, τ)
  $(st_\mathsf{S}, m_3) \xleftarrow{\$} \mathsf{SD.S}(st_\mathsf{S}, m_2)$
  // m_3 is the server response to *cid*
  $(st_\mathsf{C}, m_4) \xleftarrow{\$} \mathsf{SD.C}(st_\mathsf{C}, m_3)$
  // m_4 is RequestMessages or ε if there are no more messages to fetch
  τ.append(m_1, m_2, m_3)
until $m_4 = \varepsilon$
Return (st_S, τ)

We define a setup function, SD.Setup, to capture (honest) key generation for users, including all keys for message sending and message retrieval. The Setup function can be called on one of two roles, src or journ, and will generate keys according to how this is done in the protocol. For sources, a passphrase is sampled from the predefined passphrase distribution $\mathcal{P}$. We provide the server adversary with knowledge of all public keys even though in practice they only learn the public keys of journalists. In particular, Setup is defined as follows, where μ returns the corresponding public key of a private key, $\mu(sk) = pk$.

$\underline{\mathsf{SD.Setup}(\mathsf{role})}$
if role = src:
  passphrase $\xleftarrow{\$} \mathcal{P}$
  $mk \leftarrow \mathsf{PBKDF}(\mathsf{passphrase})$
  $sk_S^{fetch} \leftarrow \mathsf{KDF}(mk, \mathtt{sourcefetchkey})$
  $sk_S^{APKE} \leftarrow \mathsf{KDF}(mk, \mathtt{sourceAPKEkey})$
  $sk_S^{PKE} \leftarrow \mathsf{KDF}(mk, \mathtt{sourcePKEkey})$
  $pk_S^{fetch} \leftarrow \mu(sk_S^{fetch})$, $pk_S^{APKE} \leftarrow \mu(sk_S^{APKE})$, $pk_S^{PKE} \leftarrow \mu(sk_S^{PKE})$
  $pk_u \leftarrow (pk_S^{fetch}, pk_S^{APKE}, pk_S^{PKE})$
  $sk_u \leftarrow (sk_S^{fetch}, sk_S^{APKE}, sk_S^{PKE})$
else if role = journ:
  $sk_\mathcal{J}^{fetch} \xleftarrow{\$} \mathcal{E}_H$, $pk_\mathcal{J}^{fetch} \leftarrow g^{sk_\mathcal{J}^{fetch}}$
  $(sk_\mathcal{J}^{APKE}, pk_\mathcal{J}^{APKE}) \xleftarrow{\$} \mathsf{APKE.KGen}()$
  $(sk_\mathcal{J}^{PKE}, pk_\mathcal{J}^{PKE}) \xleftarrow{\$} \mathsf{PKE.KGen}()$
  $pk_u \leftarrow (pk_\mathcal{J}^{fetch}, pk_\mathcal{J}^{APKE}, pk_\mathcal{J}^{PKE})$
  $sk_u \leftarrow (sk_\mathcal{J}^{fetch}, sk_\mathcal{J}^{APKE}, sk_\mathcal{J}^{PKE})$
else return $\perp$
return (pk_u, sk_u)

To capture an adversary's ability to perform offline dictionary attacks on passphrases, prior work on password-based encryption in the game-based setting uses a password guessing game [6, 11]. In our case, however, the passphrase is chosen randomly from a distribution and provided to a user, thereby ensuring unique, higher entropy passphrases. Thus, we can rely on a modified notion of KDF security for the password-based key derivation function PBKDF (Definition A.3).

We can now give the formal definition of the message unlinkability game for an honest-but-curious server (Figure 20). The game is defined with respect to any protocol that implements compatible Send, Retr, and Setup functions (i.e., following the same API). The game generates a challenge bit b, and the goal of the adversary is to guess this challenge bit. The game provides to the adversary access to several oracles allowing it to perform actions such as generating new users (NewUser) and sending arbitrary messages between users (NewMsg) using the message sending protocol. The latter oracle returns to the adversary the corresponding message triple generated by the sending client. The adversary can also obtain transcripts generated by a user fetching messages via the Fetch oracle. Next, there are several *challenge* oracles. The ChallSend oracle provides to the adversary the transcript generated by having one of two clients send an arbitrary message to a recipient, where the designated sender is determined by the challenge bit. The ChallRecv oracle provides to the adversary the transcript

$\mathrm{G}^{\text{unl-hbc}}_{\Pi}(\mathcal{A})$

INITIALIZE:
1 users $\leftarrow 0$
2 $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3 \leftarrow \emptyset$
3 $b \xleftarrow{\$} \{0,1\}$

NEWUSER(role):
4 $u \leftarrow$ ++users
5 $(pk_u, sk_u) \xleftarrow{\$} \Pi.\mathsf{Setup}(\text{role})$
6 return (pk_u, u)

NEWMSG($s, r \in [\text{users}], m_{\text{in}} \in \{0,1\}^*$):
7 $\mathcal{E}_1 \leftarrow \mathcal{E}_1 \cup \{r\}$
8 $(st_S, m_{\text{out}}) \leftarrow \Pi.\mathsf{Send}(st_s, st_S, m_{\text{in}}, r)$
9 return (st_S, m_{out})

FETCH($r \in [\text{users}]$):
10 if $r \in \mathcal{E}_2$ or $r \in \bigcup \mathcal{E}_3$ return $\perp$
11 $(st_S, \tau) \leftarrow \Pi.\mathsf{Retr}(st_r, st_S)$
12 return (st_S, τ)

CHALLSEND($(i_0, i_1), r \in [\text{users}], m_{\text{in}} \in \{0,1\}^*$):
13 $\mathcal{E}_2 \leftarrow \mathcal{E}_2 \cup \{r\}$
14 $(st_S, m_{\text{out}}) \leftarrow \Pi.\mathsf{Send}(st_{i_b}, st_S, m_{\text{in}}, r)$
15 return (st_S, m_{out})

CHALLRECV($(j_0, j_1), s \in [\text{users}], m_{\text{in}} \in \{0,1\}^*$):
16 if $\{j_0, j_1\} \notin \mathcal{E}_3$ and $j_i \in \bigcup \mathcal{E}_3$ for $i \in \{0,1\}$ return $\perp$
17 $\mathcal{E}_3 \leftarrow \mathcal{E}_3 \cup \{\{j_0, j_1\}\}$
18 $(st_S, m_{\text{out}}) \leftarrow \Pi.\mathsf{Send}(st_s, st_S, m_{\text{in}}, j_b)$
19 return (st_S, m_{out})

CHALLFETCH(j_0, j_1):
20 if $j_0 \in \mathcal{E}_1 \cup \mathcal{E}_2$ or $j_1 \in \mathcal{E}_1 \cup \mathcal{E}_2$ return $\perp$
21 if $\{j_0, j_1\} \notin \mathcal{E}_3$ return $\perp$
22 $(st_S, \tau) \leftarrow \Pi.\mathsf{Retr}(st_{j_b}, st_S)$
23 return (st_S, τ)

FINALIZE(b'):
24 return $[\![b = b']\!]$

Figure 20: Honest-but-curious server message unlinkability security game for a protocol Π.

generated by having a sender (chosen by the adversary) send an arbitrary message to one of two recipients, where the recipient is determined by the challenge bit. Lastly, the CHALLFETCH oracle returns the transcript corresponding to one of two clients fetching their messages from the server. The goal of the adversary is to distinguish in any of the three challenge cases which user is sending, receiving, or fetching messages, respectively. To rule out trivial wins, the adversary can only call CHALLFETCH on pairs of users on which it had previously called CHALLRECV. To win the game, an adversarial server must be able to distinguish which user is interacting with the system/running the protocol. In other words, given control over all other users within the system, an adversarial server learns nothing about the users (senders or receivers) via the message sending or message retrieval protocol.

We note that the game trivially reveals to the adversary when multiple messages are retrieved together by the same recipient (i.e., in the game, the adversary does not learn the recipient, but learns the number of messages retrieved and that they share a recipient). In practice, this requires linking message retrievals over time, because only one message is fetched at a time in the message retrieval protocol. This leakage could be mitigated by introducing timing delays, dummy fetches, or PIR, which we discuss in Section F.

Definition E.1 (Unlinkability under honest-but-curious server)**.** *Let Π be a protocol with corresponding transcript-generating functions for message sending and retrieval,* (Send, Retr)*, and user setup function* Setup*, and $\mathrm{G}^{\text{unl-hbc}}_{\Pi}(\mathcal{A})$ be the honest-but-curious server message unlinkability security game defined in Figure 20 for an adversary $\mathcal{A}$. We define the advantage of $\mathcal{A}$ in breaking the* unl-hbc *security of Π as*

$$\mathsf{Adv}^{\text{unl-hbc}}_{\Pi}(\mathcal{A}) := 2 \cdot \Pr\left[\mathrm{G}^{\text{unl-hbc}}_{\Pi}(\mathcal{A}) \Rightarrow 1\right] - 1.$$

The proof of message unlinkability follows directly from the multi-user anonymity of SD-APKE ciphertexts established in Theorem D.1, the anonymity of the KEM $\mathsf{KEM}_{\mathsf{H}}$ used for HPKE metadata protection, and the DDH security of the group used for generating hints for the message retrieval protocol.

Theorem 5.3. *Let* SD *be the protocol defined in Figures 3c and 3d, over nominal group $\mathcal{N}$ and passphrase distribution $\mathcal{P}$, and using* SD-APKE[AKEM, $\mathsf{KEM}_{\mathsf{PQ}}$, AEAD, KS] *and* SD-PKE[$\mathsf{KEM}_{\mathsf{H}}$, AEAD, KS]*, as defined in Figures 5 and 6. Then, for any adversary $\mathcal{A}$ against the* unl-hbc *security of* SD *using at most q_{NU} queries to* NEWUSER*, q_{CS} queries to* CHALLSEND*, q_{CR} queries to* CHALLRECV *and q_{CF} queries to* CHALLFETCH*, we give algorithms $\mathcal{B}_1$–$\mathcal{B}_7$ such that*

$$\begin{aligned}\mathsf{Adv}^{\text{unl-hbc}}_{\mathsf{SD}}(\mathcal{A}) \leq\ & \mathsf{Adv}^{\mathsf{pwKDF}}_{\mathsf{PBKDF},\mathcal{P},q_{\text{NU}}}(\mathcal{B}_1) + \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{KDF}}(\mathcal{B}_2) \\ &+ \mathsf{Adv}^{\text{mu-ANO-CPA}}_{\text{SD-APKE}}(\mathcal{B}_3) + 2 \cdot (q_{\text{CS}} + q_{\text{CR}}) \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{KEM}_{\mathsf{H}}}(\mathcal{B}_4) \\ &+ q_{\text{CS}} \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{AEAD}}(\mathcal{B}_5) + q_{\text{CR}} \cdot \mathsf{Adv}^{\text{wANO-CPA}}_{\mathsf{KEM}_{\mathsf{H}}}(\mathcal{B}_6) \\ &+ 2 \cdot q_{\text{CR}} \cdot \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{N}}(\mathcal{B}_7).\end{aligned}$$

PROOF. We proceed via a series of game hops.

Game 0. We begin with the unlinkability game, $\mathrm{G}^{\text{unl-hbc}}_{\mathsf{SD}}(\mathcal{A})$ when $b = 0$.

Game 1. In G_1, we replace any instances of *mk* generated by a call to PBKDF by a random string in $\{0,1\}^{pbkl}$, where *pbkl* is the length of *mk*. We can bound the adversary $\mathcal{A}$'s difference in advantage exactly by a reduction $\mathcal{B}_1$ to the pwKDF security of PBKDF, the passphrase distribution $\mathcal{P}$, and parameterized by the number of sources generated by $\mathcal{A}$ (which is bounded by the number of calls to NEWUSER, which we denote q_{NU}). That is,

$$\Pr[\mathrm{G}_0] - \Pr[\mathrm{G}_1] \leq \mathsf{Adv}^{\mathsf{pwKDF}}_{\mathsf{PBKDF},\mathcal{P},q_{\text{NU}}}(\mathcal{B}_1).$$

Game 2. Next, we replace any keys generated from *mk* via a call to KDF with keys generated by the appropriate KGen function. In other words, Setup(src) now runs exactly as Setup(journ). Since all instances of *mk* are random by G_1, this reduces to the PRF security of KDF. In particular,

$$\Pr[\mathrm{G}_1] - \Pr[\mathrm{G}_2] \leq \mathsf{Adv}^{\mathsf{PRF}}_{\mathsf{KDF}}(\mathcal{B}_2).$$

Game 3. In G_3, we replace in challenge queries (CHALLSEND and CHALLRECV) the ciphertexts output by SD-APKE.AuthEnc($\cdot$) in the message sending protocol. In CHALLSEND queries, we replace each ciphertext from user i_0 with a ciphertext to user i_1. In

ChallRecv queries, we replace each ciphertext to user j_0 with a ciphertext to user j_1. We can bound the adversary $\mathcal{A}$'s difference in advantage exactly by a reduction $\mathcal{B}_3$ to the mu-ANO-CPA security of SD-APKE. Note that the mu-ANO-CPA game captures multiple challenge queries of sender and receiver identity with arbitrary messages. So, we have that

$$\Pr[G_2] - \Pr[G_3] \leq \mathsf{Adv}^{\text{mu-ANO-CPA}}_{\text{SD-APKE}}(\mathcal{B}_3).$$

Game 4. Next, in G_4, we replace in a series of hybrids corresponding to all ChallSend and ChallRecv queries the KEM key output by KEM_H with a uniformly random KEM key. In each query, we can bound the adversary $\mathcal{A}$'s advantage by a reduction $\mathcal{B}_4$ to the IND-CPA security of KEM_H. The reduction $\mathcal{B}_4$ obtains the IND-CPA challenge (pk, c^*, K^*) and simulates the game for $\mathcal{A}$ by using pk, c^* and K^* in place of pk, c, and K_3, respectively. If K^* is the real KEM key then $\mathcal{B}_4$ exactly simulates G_3 to $\mathcal{A}$; else, if K^* is random then $\mathcal{B}_4$ exactly simulates G_4 to $\mathcal{A}$. Therefore:

$$\Pr[G_3] - \Pr[G_4] \leq (q_{\text{CS}} + q_{\text{CR}}) \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{KEM}_\mathsf{H}}(\mathcal{B}_4).$$

Note that at this point, the values derived from K_3 use the random key.

Game 5. In G_5, we make replacements in a series of hybrids over all ChallSend queries the HPKE metadata protection ciphertexts (c_3, c_4), with the evaluation of $\mathsf{SD\text{-}PKE.Enc}(\cdot)$ on input sender public keys corresponding to i_1. The difference in the adversary $\mathcal{A}$'s advantage can be bounded by a reduction $\mathcal{B}_5$ to the IND-CPA security of the underlying AEAD scheme. This is clear because the KEM key is random by G_4, the KEM ciphertext c_3 is independent of the sender's identity, while the AE ciphertext c_4 is merely an encryption of the sender's public keys. Since each ciphertext is computed using a different key corresponding to the values from G_4, we have an additional factor in the number of queries to ChallSend, q_{CS}. Thus, it follows that

$$\Pr[G_4] - \Pr[G_5] \leq q_{\text{CS}} \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\text{AEAD}}(\mathcal{B}_5).$$

Game 6. Now for G_6, we again apply a series of hybrids, this time over all ChallRecv queries, in which we replace KEM ciphertexts c, output by KEM_H with a c generated by calling $\mathsf{KEM}_\mathsf{H}.\mathsf{Encap}(\cdot)$ on the public key corresponding to user j_1 input to ChallRecv (whereas previously this was for user j_0). We can bound the adversary $\mathcal{A}$'s difference in advantage by a reduction $\mathcal{B}_6$ to the weak anonymity wANO-CPA of the KEM KEM_H. Note that the adversary $\mathcal{A}$ at this point only sees a random KEM key, by G_4, so we need not apply a stronger notion of anonymity. Hence, we have that

$$\Pr[G_5] - \Pr[G_6] \leq q_{\text{CR}} \cdot \mathsf{Adv}^{\text{wANO-CPA}}_{\mathsf{KEM}_\mathsf{H}}(\mathcal{B}_6).$$

Game 7. This game similarly captures a series of hybrids over all ChallSend and ChallRecv queries. In G_7, we recompute the KEM key KEM_H, which was made to be random in G_4, as the KEM key output by calling $\mathsf{KEM}_\mathsf{H}.\mathsf{Encap}(\cdot)$ on the public key corresponding to user j_1 input to ChallRecv. As this is the natural inverse of G_4, we capture the difference in advantage by a reduction $\mathcal{B}_4$ to the IND-CPA security of KEM_H.

$$\Pr[G_6] - \Pr[G_7] \leq (q_{\text{CS}} + q_{\text{CR}}) \cdot \mathsf{Adv}^{\text{IND-CPA}}_{\mathsf{KEM}_\mathsf{H}}(\mathcal{B}_4).$$

We note that now, the ciphertext quadruple (c_1, c_2, c_3, c_4) generated in each call to ChallSend uses the sender i_1 and in each call to ChallRecv uses the receiver j_1. In what follows, we address the remainder of the values in the protocol, relying on the fact that the additional information submitted in a message triple, i.e., X and Z, is independent of the ciphertext quadruple generated via SD-PKE.

Game 8. In G_8, we rewrite the previous game such that on calls to ChallFetch, the private keys of *both* users j_0, j_1 are used in the retrieval protocol to fetch messages. That is, when running the client-side message retrieval protocol and capturing the transcript with $\mathsf{SD.Retr}(\cdot)$, the game fetches any *cid* that decrypts under one of $sk^{fetch}_{j_0}$, $sk^{fetch}_{j_1}$. We note that the restrictions on trivial wins ensure that ChallFetch may only be called on pairs of users that were previously used in ChallRecv. This implies that at this point, there will only be message triples using the fetching key of user j_0 and none using the fetching key of j_1. Hence, using both keys does not alter the view of the adversary and there is no additional advantage. Hence,

$$\Pr[G_7] = \Pr[G_8].$$

Game 9. Next, in G_9, we apply a series of hybrids over all queries to ChallRecv to replace Z with $Z^* = pk^x_{j_1}$, where x is the value sampled (by the game) when running the sender-side of the message sending protocol, and j_1 is the second user input to ChallRecv. Z must be replaced in every ChallRecv query, regardless of whether a ChallFetch follows, since ChallRecv returns the corresponding transcript to the adversary. By the previous game G_8, the number (and identifiers) of messages retrieved do not change, since both the keys corresponding to j_0 and j_1 are used for message retrieval. Thus, we can simply bound the difference in the adversary $\mathcal{A}$'s advantage by a reduction $\mathcal{B}_7$ (for each call to ChallRecv) to DDH over the relevant nominal group. We note that there is an additional factor of 2 because we first apply DDH to make the value Z random, and then again to recompute it as described above. Therefore, we have that,

$$\Pr[G_8] - \Pr[G_9] \leq 2 \cdot q_{\text{CR}} \cdot \mathsf{Adv}^{\mathsf{DDH}}_{\mathcal{N}}(\mathcal{B}_7).$$

Game 10. Finally, in G_{10}, we undo the change from G_8 and instead have the game use only j_1 when retrieving ciphertexts. By G_9, we are assured that now there will be no ciphertexts using the fetching key of j_0. It follows that

$$\Pr[G_9] = \Pr[G_{10}].$$

Now this is exactly the game with challenge bit $b = 1$. Collecting the bounds yields the theorem statement.

□

F PIR-based Approaches

One could replace the final step of the message retrieval protocol, where the recipient sends a ciphertext identifier to the server and obtains the corresponding ciphertext in response, with an instance of a private information retrieval (PIR) protocol. The introduction of PIR would reduce the amount of information learned by the server. In particular, it would learn only the number of queries instead of

exactly which ciphertexts are received, reducing the server's ability to perform traffic analysis on the ciphertexts. However, standard approaches to PIR assume that the database is public, and therefore any client may query for any index in the database.

Currently, the message retrieval protocol ensures that an external adversary cannot obtain a ciphertext from the server unless they know the corresponding identifier, thus authenticating them in some way. Even more, preventing external adversaries from obtaining real ciphertexts further protects the messages in case the sender's or one of the recipients' keys is later compromised. This is in line with the unobservability requirements discussed in Section 2. Thus, we do not want to naively introduce PIR as it would compromise these forward secrecy-like properties against external adversaries that are currently provided for messages on the SecureDrop instance.

A natural question is then: could one implement a PIR protocol that does not leak additional ciphertexts to external adversaries who are not authorized to retrieve them? The intuitive answer would be that one could apply keyword PIR, where a client privately retrieves a record from the database using a corresponding *keyword* rather than its index in the database. Ideally, an external adversary not knowing the keyword (in this case, the ciphertext identifier) would not be able to retrieve these records. However, most state-of-the-art keyword PIR protocols rely on a keyword-to-index mapping and then run an instance of an index PIR protocol, therefore adopting the property that arbitrary records in the database may be retrieved via their index in the database [29]. Alternatively, one could apply a keyword PIR scheme which computes homomorphic *equality operators* over the queried keyword and the database keywords, such as in [93] (being careful to ensure that potentially malformed queries do not leak information about other rows in the database); however, this is currently prohibitively expensive in latency.

It is possible to extend existing keyword PIR schemes with a non-interactive zero-knowledge proof of the validity of the PIR query (i.e., an RLWE ciphertext). This can be done generically via SNARKs or more efficiently from recent vector commitment schemes [87]. However, increased communication overhead must be considered, particularly since the protocol is intended to run over an anonymity network such as Tor. Further, most PIR schemes are designed for a relatively unchanging database with a large number of small records, and consider "large" records to be tens of kilobytes [36, 101]. By contrast, a messaging system such as the SecureDrop protocol anticipates regular database updates, adding complexity to many optimisation measures for existing PIR schemes, as well as ciphertexts exceeding 100 KB, which might require splitting into multiple blocks [68]. With concerns about engineering complexity plus communication overhead, we leave further exploration of PIR schemes for future work.

G Benchmarks

We inform the parameter choices for the protocol and evaluate its performance both natively and cross-compiled to WebAssembly. Figure 21 charts the linear increase in execution time as the number of challenges increases. As shown, the coefficient varies depending on the environment, incurring proportional performance penalties in browser environments. While there are per-browser

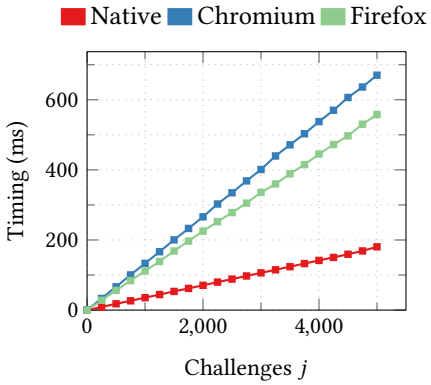


Figure 21: Execution time of *retrieve* as a function of the number of challenges.

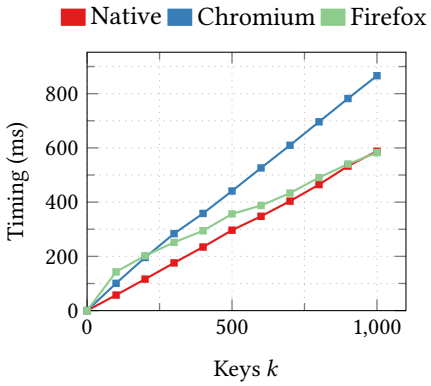


Figure 22: Execution time of *decrypt* as a function of the number of one-time keys.

optimizations of the WebAssembly runtime, performance also depends on the underlying platform and on whether, for instance, vector instructions are available and used.

Algorithm-specific hardware acceleration is not available via WebAssembly, while it is available through the Web Crypto API. However, the Web Crypto APIs have a significant per-invocation cost and are therefore efficient for large data, but not for bulk operations. In practice, the Web Crypto API performs better for a single call hashing or encrypting large chunks of data, while WebAssembly performs better for repeated or bulk operations over smaller chunks.

Figure 22 shows that, when WebAssembly optimizations take effect, effects emerge that impact the precise linear relationship. For benchmarking purposes, this is particularly relevant, as it could taint the results: running bulk WebAssembly benchmarks without cleaning up the environment might trigger heavy prediction and speculation effects that do not reflect real performance in fresh environments.