Jäger: Automated Telephone Call Traceback

David Adei
North Carolina State University
Raleigh, USA
dahmed@ncsu.edu

Varun Madathil
North Carolina State University
Raleigh, USA
vrmadath@ncsu.edu

Sathvik Prasad
North Carolina State University
Raleigh, USA
snprasad@ncsu.edu

Bradley Reaves
North Carolina State University
Raleigh, USA
bgreaves@ncsu.edu

Alessandra Scafuro
North Carolina State University
Raleigh, USA
ascafur@ncsu.edu

Abstract

Unsolicited telephone calls that facilitate fraud or unlawful telemarketing continue to overwhelm network users and the regulators who prosecute them. The first step in prosecuting phone abuse is traceback — identifying the call originator. This fundamental investigative task currently requires hours of manual effort per call. In this paper, we introduce Jäger, a distributed secure call traceback system. Jäger can trace a call in a few seconds, even with partial deployment, while cryptographically preserving the privacy of call parties, carrier trade secrets like peers and call volume, and limiting the threat of bulk analysis. We establish definitions and requirements of secure traceback, then develop a suite of protocols that meet these requirements using witness encryption, oblivious pseudorandom functions, and group signatures. We prove these protocols secure in the universal composibility framework. We then demonstrate that Jäger has low compute and bandwidth costs per call, and these costs scale linearly with call volume. Jäger provides an efficient, secure, privacy-preserving system to revolutionize telephone abuse investigation with minimal costs to operators.

CCS Concepts

• **Networks** → **Mobile networks**; • **Security and privacy** → **Mobile and wireless security**; Distributed systems security; **Privacy-preserving protocols**.

Keywords

Telephone Networks; STIR/SHAKEN; Traceback; Distributed System; Privacy-Preserving; Network Abuse

ACM Reference Format:

David Adei, Varun Madathil, Sathvik Prasad, Bradley Reaves, and Alessandra Scafuro. 2024. Jäger: Automated Telephone Call Traceback. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 24 pages. https://doi.org/10.1145/3658644.3690290

CCS '24, October 14–18, 2024, Salt Lake City, UT, USA

ACM ISBN 979-8-4007-0636-3/24/10
https://doi.org/10.1145/3658644.3690290

1 Introduction

Telephone networks are inundated with unsolicited "robocalls" for telemarketing or outright fraud. While individuals are bothered by the seemingly constant ringing of the phone, government agencies, enterprises, and non-profits now have greater difficulties reaching stakeholders for legitimate, desirable purposes. Public outcry has motivated policy makers, public officials, and phone providers to all take actions [29, 42] meant to address the problem.

Prior to the recent spate of incessant robocalls, the Federal Trade Commission (FTC) established the Telemarketing Sales Rule [32] to set the bounds on what forms of automated calls are considered permissible. A general summary is that callers must have affirmative, opt-in consent from the called party to dial them for commercial purposes. The FTC also maintains a "do-not-call" list that should prevent unsolicited calls to registered individuals. These measures are legal, not technical, so violations are pursued through regulatory action. These measures have clearly not been effective.

In late 2019, a sudden outbreak of bipartisanship struck the United States Congress, who passed the TRACED Act to combat illegal calling. Among other measures, the law further expanded penalties for illegal calling and empowered regulators to make substantial changes to network policy to reduce robocalls. These changes to date have included requiring providers to register and submit Robocall Mitigation Plans to the FCC, the mandatory blocking of calls claiming to originate from invalid numbers, encouraging the labeling of suspect calls by providers, setting deadlines on participation in robocall investigations, and mandating that all providers implement a call authentication mechanism known as STIR/SHAKEN (S/S). S/S requires originating providers to sign outbound call requests to indicate their actual source, similar to DKIM, expecting that it would allow regulators to identify the source of the call, prevent caller ID spoofing, and give robocallers "no place to hide."

In practice, all of these efforts have failed to significantly change the state of affairs. Call labelling is unreliable, robocallers have moved to using legitimate numbers for a very short period, and the majority of calls in the network arrive without a signature [69] because pre-VoIP networks cannot be modified to support S/S.

Robocallers continue to operate for a simple reason: it is profitable and low risk. While regulators and law enforcement have been successful in winning judgements against accused robocall operators, with fines into many millions of dollars, the defendants

often return to their schemes under assumed identities or are replaced by other "entrepreneuers" using similar techniques. Because the robocalling problem is so vast, it is reasonable to assume that they will not face penalties given how painstaking it is to bring a case against a single robocaller and how small the relevant agency staffs are.

One of the biggest hurdles is identifying the source of a call. The telephone network is a network-of-networks, like the Internet, and a given call often passes through many networks before reaching its destination. Routes change rapidly and unpredictably, and providers routing the call only know the previous "hop" and the next "hop." Fortunately, providers keep meticulous records on calls they route for billing or paying peers. To identify the source of a call, an investigator must start at the destination with knowledge of the call time, call destination, and claimed call source, and go hop-by-hop until they reach the originator. This process is termed **traceback**.

Prior to 2019, a traceback, could take multiple subpoenas and months to complete *for a single call* [11]. The 2019 TRACED Act mandated the creation of a clearinghouse to handle tracebacks, and it also mandated timely responses to traceback requests by providers. The result was the creation of the Industry Traceback Group (ITG) [42], and the FCC reported to Congress that currently a traceback can be completed in under 24 hours with the help of ITG.

Unfortunately, tracebacks are still largely manual, and the ITG has a skeleton staff of a few employees. They are (rightfully) proud that they manage to complete around 300 tracebacks per month, though this is far from sufficient to deal with millions of robocalls each month. It will certainly become a bottleneck if we want law enforcement to target perpetrators of high-touch fraud schemes like digital kidnapping, where a fraudster provides a convincing story and fabricated voice of the loved one of a target to extort a ransom.

Automation is needed to scale traceback to a significant fraction of the current abuse, but simple approaches will be unacceptable to some portion of current stakeholders. If each carrier were required to implement an API for traceback, it runs the risk of a malicious carrier fabricating incorrect responses. Note that some smaller carriers are known to skirt or outright violate laws for profit, including facilitating illegal robocalls. If each provider were required to submit their call routing records to a central source, subscribers would justifiably worry that their social networks and telephone activity could be leaked. Providers would balk at revealing peering arrangements and call volumes, and the central database would be a magnet for curious intelligence agencies and law enforcement dragnets.

In this paper, we present Jäger[1], a distributed system and protocol suite to provide rapid, automated traceback for phone calls. To trace an illegal call, an investigator will obtain the caller's and recipient's telephone numbers along with precise timing details about the call. This information is typically sourced from public complaints, industry honeypots, the ITG's data collection, consumer voicemails, or other commercial channels. Jäger enables the investigator with the call details to identify its originating network. The originating network can then be held liable or identify the customer responsible for the illegal call.

The key insight is to allow traceback over encrypted call records, with the caveat that only a party with detailed knowledge of the call can identify or decrypt the routing records for a given call. Our solution requires no modifications to the existing telephone network. Instead, we assume access to the billing systems that already maintain the records we need. We do not require interaction between providers at any point. The compute, storage, and bandwidth costs for providers are modest and scale linearly with call volume, so small carriers need few resources. We provide for cryptographic mechanisms to ensure that authorized traceback users can be appropriately rate-limited to prevent bulk abuse. Our system is robust against a single provider on a call who fabricates or does not submit a routing record for a call. Encrypted call records do not reveal the provider who submitted them, but a provider who submits an invalid or incorrect record can be identified. Because the purpose of traceback is to identify the source of a call, we do not actually need every routing record for a call to be present in Jäger. Ideally, at least the first, originating record, will be present, but if it isn't, any other record will still improve traceback performance. Our scheme will provide benefits even if some providers do not participate, so it can be deployed incrementally.

We make the following contributions:

- We specify and define the key properties and requirements for secure telephone call traceback.
- We design protocols that meet the requirements for secure traceback, and implement them in a prototype distributed system dubbed Jäger.
- We provide formal guaranties by proving the security of these protocols in the Universal Composability (UC) framework.
- We demonstrate that Jäger has low compute and bandwidth costs per call, and these costs scale linearly with call volume. In the process, we also develop a performant witness encryption library in C++. Code for that library and our full Jäger implementation is available [1, 2].

Robocall enforcement is ultimately a legal problem, albeit with technical challenges. Jäger fills a technical need for effective investigative tools, though Jäger, cannot independently determine whether a call was illegal. Nevertheless, because the telephone ecosystem is heavily regulated, honest participation can be incentivized through the risk of civil or criminal prosecution.

2 Background

This section discusses background information on the state of the telephone network abuse and prosecuting violators. In doing so, we described challenges with locating abuse actors.

2.1 Telephone Network Abuse

Phone network abuse is a global problem, with the United States being one of the most severely affected countries. One of the most common forms of abuse is pre-recorded automated bulk phone calls, or robocalls. Many robocalls violate US law, including sales calls made without affirmative opt-in. Fraudulent calls often impersonate government officials and steal millions of dollars from victims.

[1]Jäger, pronounced "YAYger," is German for "hunter." It can also refer to Jägermeister, a popular liqueur, making it an appropriate name for a protocol to supplement S/S.

Federal statues prescribe eyewatering financial penalties for each and every illegal call, but penalties require enforcement to deter abuse. Currently, phone abusers avoid prosecution by using spoofed or short-term telephone numbers, regularly changing service providers, and altogether vastly exceeding available enforcement resources.

Routing Phone Calls through the Network: Determining the source of a single call currently requires significant investigative effort, and part of the reason is that the phone network is a network-of-networks with no global vantage point and no single end-to-end authentication of identity.

A given telephone provider will connect with one or more other providers to send and receive call traffic. When a subscriber places a call, her provider "originates" the call and then uses a signalling protocol to communicate with its peer networks to find a route to the called party's network. When the call is finally set-up, over potentially many intermediate providers, the call is considered "terminated" and the call audio will begin.

Call routes are selected considering carrier[2] charges, network maintenance, and agreements with other carriers, and these factors change moment-to-moment. Each provider that carries the call will bill the provider who sent it, and Call Detail Records (CDRs) are kept to support this. However, only the originating provider knows any details about the call originator beyond the phone number the subscriber claimed when they set up the call.[3] No provider ever knows more about the call than the previous and next provider in the route. To identify a party responsible for an illegal call, an investigator must first find the originating provider, which is unknown to the terminating provider who delivered the call to its recipient.

Locating Abuse Actors: Authorities must find and prosecute perpetrators responsible for generating illegal calls. In the United States, the TRACED Act of 2019 [29] requires the FCC to mandate the S/S caller authentication framework. Although in principle S/S can be used to traceback illegal robocalls from their destination to their origin, there are several limitations. Industry reports from October 2023 estimate [69] more than half of all voice traffic in the US is still not signed using S/S, making it impossible to track the origin of such calls using S/S information alone. Industry insiders attribute this large portion of unauthenticated voice traffic to legacy infrastructure that does not support S/S. Furthermore, phone calls originating outside the US often do not contain S/S information since the framework is not mandated in other countries. Therefore, regulators, enforcement agencies, and other entities rely exclusively on manual traceback processes to identify the source of illegal robocalls [26].

Manual Traceback Process: As the TRACED Act requires, the FCC has designated the ITG [42] to serve as the central entity to coordinate the traceback process. The ITG manages the labor-intensive and time-consuming tasks of identifying the source of suspected illegal robocalls. The ITG constantly monitors active robocall campaigns using data from honeypots [57], consumer reports, and other sources. After assessing the legality of the robocall, the ITG initiates a traceback request and manually coordinates across numerous carriers to pinpoint the source of suspected illegal robocalls. The traceback process involves tracing the call path from the terminating carrier through transit carriers and ultimately to the originating carrier to identify the source of the call.

Traceback has proven to be a crucial tool for regulators and enforcement agencies to combat illegal robocalls. It has been used in almost every enforcement action filed by regulators against robocalling operations. However, successfully completing a traceback often takes several hours or days, with substantial effort from the ITG and the participating carriers. The manual and time-consuming nature of the traceback process significantly limits its effectiveness. Although the volume of illegal robocalls targeting US subscribers is estimated to be in the hundreds of millions per year, less than 3,000 tracebacks were completed over eleven months in 2022 [30]. By developing an automated, secure, and scalable traceback system, we can swiftly uncover the source of such calls, deter bad actors, and empower stakeholders to protect phone users from illegal robocalls.

2.2 Cryptographic Primitives

This section introduces the cryptographic primitives that serve as the building blocks for our protocol.

Witness Encryption Based on Signatures: A Witness Encryption scheme based on Signatures (WES) was recently proposed in [47] and [24]. These are encryption schemes where the encryption key is a tuple of a signature verification key (denoted vk) and a string chosen by the encryptor (denoted ℓ). The decryption key is a valid signature (denoted σ) on the string, such that the signature can be verified by the verification key. More specifically, a witness encryption based on signatures has two algorithms - WE.Enc, WE.Dec, where $\mathsf{WE.Enc}((\mathsf{vk}, \ell), m) \rightarrow \mathsf{ct}$, and $\mathsf{WE.Dec}(\sigma, \mathsf{ct}) \rightarrow m$. m denotes the plaintext, (vk, ℓ) corresponds to the encryption key, and $\sigma = \mathsf{Sign}(\mathsf{sk}, \ell)$ corresponds to the decryption key if $\mathsf{Sign.Vf}(\mathsf{vk}, \ell, \sigma) = 1$. Here $\mathsf{Sign}(\cdots)$ and $\mathsf{Sign.Vf}(\cdots)$ are the sign and verify procedures for the signature scheme. We acknowledge that using signature verification keys to encrypt messages and using signatures to decrypt ciphertexts is not intuitive, and the notation can be confusing. Observe that we denote witness encryption and decryption keys as *tuples* containing signing and verification keys, while the signature keys are written as single variables. [47] and [24] show that it is possible to construct such WES schemes efficiently based on BLS signatures [15].

Group Signatures: Group signatures [4] are a cryptographic primitive that allows group members to anonymously sign messages on behalf of the group. A designated authority, the group manager, generates a common public key gpk and issues a unique group member signing key gsk_i for each group member i. Any signature signed by any gsk_j in the group will verify with gpk. The group manager can also deanonymize signatures and identify the signer. Group signatures allow for anonymity while maintaining accountability.

Oblivious PRF: A pseudorandom function (PRF) F_k is a keyed function whose outputs look random to anyone without the secret key k. An oblivious pseudorandom function (OPRF) [20] is a two-party protocol where a server holds a secret key k for the PRF, and a client holds a secret input x to be evaluated. At the end of the

[2]In this work, we will use carrier and provider interchangeably.

[3]Most businesses expect to be able to specify the "from" field shown for caller ID. A common case is to allow a desk line to appear to be coming from the corporate switchboard number, but this feature is abused by illegal callers.

protocol, the client learns $F_k(x)$ while the server learns nothing.

3 Problem Statement

We begin this section by identifying the major stakeholders and adversaries impacting Jäger. We also specify the functional and security requirements which we aim to achieve.

3.1 Stakeholders and Adversaries

The ecosystem involves various stakeholders with distinct roles and interests, including service providers, the ITG, subscribers, and law enforcement agencies. Each group's goals and actions follow.

Providers: They route phone calls through the telephone network. By law [53], they are required to maintain the CDRs of each call and actively participate in the traceback process. They prioritize efficient record insertion and complete and correct traceback responses. Providers also desire the confidentiality of their customers, peering partners, and traffic volumes.

Subscribers: Subscribers initiate and receive phone calls. They also report fraudulent calls to authorities or their service provider. Subscribers seek to minimize receiving illegal robocalls and expect confidentiality for their call records.

Industry Traceback Group: The ITG oversees the tracing of illegal calls to their source by working with providers. They prioritize swift and accurate responses to traceback requests.

Regulatory and Law Enforcement Agencies (LEAs): LEAs work in conjunction with industry stakeholders to maintain secure and lawful communication networks. Their responsibilities include investigating suspected illegal calls, enforcing compliance, public education, and policy development. LEAs often submit traceback requests to the ITG, emphasizing the need for a timely response.

Adversaries: Adversaries may seek partial or full call records of one, many, or all subscribers. They may also seek privileged information about providers or the network structure. They may also aim to violate the integrity or availability of Jäger to prevent detection or investigation of illegal calls. Adversaries can include outside parties like private investigators[38], identity thieves, or even foreign intelligence agencies[31]. Insiders, including subscribers, providers, regulators and LEAs, and operators of Jäger entities may also behave dishonestly at any point. We design Jäger such that no single compromised entity alone can violate its security properties, and in many cases Jäger is resilient against collusion by more than one malicious entity.

3.2 Requirements

The main objective of Jäger is to enable secure and efficient traceback given a valid request containing source and destination telephone numbers along with the call timestamp.

Functional Requirements: To achieve this objective, the system is designed with the following key requirements:

(1) *Resilience*: A valid traceback request returns all available records for a call.
(2) *Precision*: A valid traceback request only returns relevant records. Malicious or incorrect records are still "relevant" if they match the traceback request.
(3) *Scalability*: Jäger must handle effectively arbitrary call volumes. For all entities, cost should scale linearly in the number of calls and/or participants (as appropriate).
(4) *Efficiency*: All operations should perform comparably to similar non-secure approaches. The financial costs should be a minor fraction of the total network revenue.
(5) *Information Gain*: Every traceback request should provide information to an investigator. A traceback request will result in one or more of the following:
   (a) Identify the originating provider for the call.
   (b) Reveal at least one claimed non-originating provider and shorten manual traceback.
   (c) Provide direct evidence that one or more providers act in bad faith (e.g., submitting false or contradictory records or no records for a call).

   In settings where Jäger is mandatory, all of these properties are obvious. Either one obtains a complete and consistent traceback, or at least one provider is violating the mandate. In partial deployment, these properties will still hold if at least one on-path provider participates.

Security Requirements: The guiding principle of secure traceback should be that no entity gains information about subscribers or providers in the absence of an authorized traceback request, even in the presence of a compromised Jäger entity. Additionally, no entity can provide false information without risk of detection and accountability. More formally, this mandates the following principles:

(1) *Trace authorization*: An entity can only trace a call they have definite knowledge of, and they must also have explicit authorization from a third party.
(2) *Call confidentiality*: No entity should determine source, destination, time, or route details about a call they do not already have without authorization for a traceback.
(3) *Trade secret protection*: No party should learn aggregate information about a provider's call volumes and peering relations except those revealed by an authorized, valid traceback or an authorized accountability request.
(4) *Record integrity*: Only authorized parties may contribute records.
(5) *Record accountability*: It must be possible to identify the contributor of a traceback record.

4 Our Approach

In the previous section, we specified requirements for secure traceback. To show how Jäger satisfies those requirements, in this section we will describe a functional but insecure strawman solution and iteratively improve it until it meets all of the security requirements.

4.1 Jäger Overview

An Insecure Strawman Approach: To enable traceback, we first introduce a central Record Store(RS) that collects and stores Call Detail Records (CDRs) from providers in a database $\mathcal{D}$. Any provider P_i in a call path, for instance, $P_1 \rightarrow P_2 \rightarrow P_3 \rightarrow P_4$, already keeps a CDR for each call they originate, transmit, or terminate. We model a CDR as a tuple $(src, dst, ts_i, P_{i-1}, P_i, P_{i+1})$ and further

divide it into two parts: call-details = $(src\|dst\|ts_i)$ and hop = $(P_{i-1}\|P_i\|P_{i+1})$, where in call-details, src and dst are source and destination telephone numbers common to all providers in the call path, ts_i(unique to P_i) is the time at which P_i receives the call and hop = $(P_{i-1}\|P_i\|P_{i+1})$ are the previous hop, current hop and next hop respectively. Phone call setup takes time to traverse through the network, and we assume an upper-bound setup time $t_{\max}$. Any CDR pertinent to the same call will have a ts^* in the range of $[ts_i - t_{\max}, ts_i + t_{\max}]$.

To enable traceback, each provider P_i contributes by sending command (CONTRIBUTE, P_i, call-details, hop) to the RS. The RS will then add the record to their database $\mathcal{D}$. Later, if a party wishes to trace a certain call with call-details = $(src\|dst\|ts_i)$, they can send the command RETRIEVE-REQ to RS, who will fetch all hops that have call-details = $(src\|dst\|ts^*), \forall ts^* \in [ts_i - t_{\max}, ts_i + t_{\max}]$.

Modeling a hop_i for a provider P_i as $(P_{i-1}\|P_i\|P_{i+1})$ allows P_i attest to their upstream and downstream provider's involvement in the call. This means that given a hop_2 from only P_2, we know the path $P_1 \rightarrow P_2 \rightarrow P_3$, so a traceback does not necessarily require records from P_1 and P_3. This design decision helps in partial deployment.

In the event of conflicting hops, for e.g., say P_2 submits $\text{hop}_2 = (P_1\|P_2\|P_3)$ indicating P_1 and P_3 as its previous and next hops. P_1 submits $\text{hop}_1 = (P_4\|P_1\|P_3)$. P_3 submits $\text{hop}_3 = (P_1\|P_3\|P_6)$. In this case, an investigator cannot tell if P_2 is misbehaving or P_1 and P_3 are misbehaving. Therefore the investigator will go to each of these providers and have them show their corresponding call records to identify and punish the misbehaving provider(s).

This strawman solution trivially meets the functional requirements of the system, but none of the security requirements we described in Sec. 3.2. Indeed, since records are stored in the clear for RS, no confidentiality is guaranteed to subscribers or providers. Furthermore, traceback could be done by any party with access to the records.

Toward Record Confidentiality: To achieve record confidentiality, the first natural step is to encrypt the call-details and the hop. Assume all providers use a shared public key (pk) to encrypt the hop and call-details using an IND-CPA secure encryption scheme. This means that RS will store a set of ciphertexts, and hence cannot learn anything about the CDR content. This guarantees the confidentiality of the records but unfortunately prevents the tracing process. Suppose an authorized party P_j wants to trace call call-details = $(src\|dst\|ts_j)$, they must send an encryption of call-details under pk to RS. However, the RS cannot find a matching record in the database, since the encryption scheme is not deterministic. Alternatively, the RS sends all the ciphertexts in the database to P_j, and the latter decrypts each ciphertext until it finds the call records pertinent to their call. This approach is inefficient and loses call confidentiality for other calls. To solve this problem, we introduce a deterministic index to identify ciphertexts related to a given call-details. Now upon receiving this index, the RS can return exactly one ciphertext to the provider. We elaborate on this below.

Adding Pseudorandom Labels to the Database: To identify the correct ciphertexts, we index each entry with a label, call-label, that can be computed only with the knowledge of call-details. When a provider P_i sends their contribution, they will send a pair (call-label$_i$, ctx_i) to the RS where ctx_i is an encryption of hop_i by P_i. Later, when a party P_j wants to trace a call call-details = $(src\|dst\|ts_j)$, they can use this information to compute call-label, and RS will be able to identify the ctx that is indexed with call-label. Note that P_j will compute all call-label* for call-details = $(src\|dst\|ts^*)$ $\forall ts^* \in [ts - t_{\max}, ts + t_{\max}]$ to retrieve all possible ciphertexts that belong to the call as specified earlier.

What function should we use to compute call-label? Perhaps the most natural candidate would be a hash function, i.e., call-label = H(call-details). However, this approach jeopardizes the confidentiality of the records once again. Indeed, anyone who gets access to the database $\mathcal{D}$ maintained by RS can "check" if a certain call $(src\|dst\|ts)$ took place by simply computing the hash of the call details and checking for that label in $\mathcal{D}$. Adding a large nonce as input to the hash function i.e. call-label = H(call-details$\|nonce$) is not helpful since during trace the provider will have to guess the nonce and this is infeasible in polynomial time.

This attack suggests that the label should not be computed using a public function that anyone can compute. Pseudorandom Functions (PRF) are the perfect candidate. They are deterministic, just like hash functions, but they can be evaluated only with the knowledge of a key. A label can be computed as call-label = F_k(call-details), where k is a key known only by the providers and F is a PRF. Hence, no one else, except carriers, can compute labels. However, this solution is not robust in our threat model, where RS could collude with providers . Indeed, it would be sufficient for only one provider to leak the PRF secret key k to expose records.

We solve this problem using a cryptographic tool called an Oblivious PRF [20] (OPRF). In an OPRF, the PRF is evaluated through a protocol between two parties: a server, who knows the key k, and a client, who knows the input x. At the end of the protocol, the client learns only the output of the PRF, while the server learns nothing. In our system, we introduce a new party called the Traceback Authority (denoted TA), which holds the secret key of the PRF and allows the providers to evaluate the PRF to compute labels. The TA, however, does not learn anything about the call-details.

To contribute a record, P_i will interact with the TA to compute call-label_i from call-details, and then compute $\text{idx}_i = H(\text{call-label}_i)$. Next, P_i will encrypt the hop_i under pk into ctx_i (as specified earlier) and submit (idx_i, ctx_i) to RS. The lookup index idx_i is a hash of the call-label_i so that if a record and/or the OPRF key is ever compromised, the call-label_i is not directly exposed.

Traceback would work as follows: An authorized party P_j who wants to trace call-details first obtains the label call-label_j from TA, then sends $\text{idx}_j = H(\text{call-label}_j)$ to RS. RS will use idx_j to identify and return the corresponding ciphertext.

However, there is still a problem: Recall that all ciphertexts are encrypted under the same key. Thus any provider (potentially unauthorized) colluding with the RS can potentially decrypt all ciphertexts trivially. Conversely, if each provider encrypts its record using a unique key, the party attempting to perform a traceback will obtain ciphertexts under different keys, requiring *all* providers to help with decryption. This defeats the purpose of the system.

Towards Encrypting Records: One potential solution is to have providers encrypt using the TA's public key. Thereafter, during the traceback, the provider retrieves the ciphertexts from the RS

and interacts with the TA to decrypt the ciphertexts. While this is a viable solution, we want to formally enforce the following properties:

(1) *Knowledge of call:* A party can trace a call only if they were part of the call i.e they *already know* the call-details. The party must be a provider in the call path.
(2) *Trace Authorization:* A party can trace a call only if they were authorized by the TA to trace that particular call.

To this end we use an asymetric encryption scheme called "witness encryption" that allows carriers to encrypt the hop such that only with the knowledge of the call-details and an authorization from the TA can they decrypt the ciphertext. In witness encryption, the encryption key is a verification key vk for a signature scheme and an arbitrary string ℓ (chosen by the encryptor). A ciphertext can be decrypted only with the knowledge of the string ℓ and a signature on ℓ that verifies under vk. In our case, we replace ℓ with call-label to enforce property (1). We enforce property (2) by requiring a signature on call-label signed by the TA.

Now to contribute records, P_i will compute call-label_i, $\text{idx}_i = H(\text{call-label}_i)$ and encrypt hop_i using $(\text{vk}, \text{call-label}_i)$ as the encryption key. Then P_i will send (idx_i, ctx_i) to RS. Using hash digests as indices instead of call-labels further enforces property (1) above. Since the RS is not part of the call path, it should not know the call-label.

Once the ciphertexts are retrieved from the RS, P_i must request authorization from the TA, in our case, a signature on call-label. This construction ensures that a ciphertext related to a call can be decrypted only by someone who knows a valid signature on the corresponding call-label.

Adding Carrier Anonymity and Accountability: Recall that to contribute, each provider sends authenticated encrypted records signed under their unique public key to RS. The RS can map their contributions to their identity, potentially learning trends about their activities. On the other hand, we cannot simply have the provider submit their records anonymously since we still need to hold them accountable for malformed or falsified contributions.

To protect carriers' business privacy but hold them accountable, we replace the regular signature scheme (used for authentication) with an anonymous group signature scheme.

Group signatures are anonymous signatures that can be validated on behalf of a group – in our case, the group of all carriers. More importantly, we choose group signatures instead of primitives like ring signatures because group signatures are efficient and allow us to trace traitors. Here, we use group signatures only for authenticating contribution requests and not for the witness encryption scheme.

In our system, the TA plays the role of the group manager and adds carriers to the group by assigning them group secret keys. Moreover, the TA is also responsible for the deanonymization of the group signatures in case any of the carriers submit bad requests.

Network Layer Anonymity: The group signature scheme for authenticating contribution requests guarantees anonymity at the application layer. Unfortunately, network features like IP addresses may still identify providers. There are a number of solutions to this problem that are orthogonal to Jäger, including proxy services like commercial VPNs. A provider may still be concerned that IP traffic volume might leak information about call volumes to the proxy. Providers can address this issue by splitting their traffic across multiple proxy services and/or transmit redundant or invalid records as cover traffic.

4.2 Threat Model and Resiliency

Jäger mandates the security requirements outlined in Sec 3.2. The Jäger system includes two entities besides providers: the RS and the TA. We assume that the RS and the TA do not collude and only one of the two entities may be malicious. We also allow collusion between providers and the corrupt entity. We show that even when the RS is malicious and is colluding with providers, none of the security requirements are violated. On the other hand, when the TA is corrupt, Jäger cannot guarantee record accountability.

We can improve the trustworthiness of the TA with several orthogonal techniques, described below. All of these options are feasible, but they add complexity and cost.

Splitting responsibilities: In the architecture of Jäger, the TA is responsible for group management, label generation, and authorizing traceback. Assigning these jobs to different entities will limit the damage should one be compromised, and our prototype actually already implements them independently.

Distributing Trust: Each of the operations can also be split among multiple entities using existing multiparty computation schemes, including threshold constructions of OPRFs[9, 43], group signatures[18, 33], and BLS signatures for use in Witness Encryption.

4.3 Frequently Asked Questions

In this section, we address some of the frequently asked questions that we have encountered.

If S/S automates traceback, why go through all this trouble?: The Public Switched Telephone Network (PSTN) is heterogeneous. Legacy infrastructure drops S/S signatures along the call path, making it ineffective for tracing call origins [11]. Call requests are signed by providers using a JSON Web Token in the SIP INVITE message with an x5u field pointing to the signing certificate. Malicious providers can exploit this by setting x5u to a timing-out link, increasing latency and forcing call transmission, posing a challenge for providers. Traceback attempts using such signatures reach dead ends, so manual processes are still needed. Additionally, the deployment of S/S remains limited. According to the Robocall Mitigation Database in the US (Feb 7, 2024) [25], among 7,109 providers, only 39.94% have fully implemented it, 23.96% are partially implemented, and 36.11% have no or unknown implementation status. We clarify that Jäger does not authenticate caller ID or block robocalls in real-time. Instead, it is a central repository of encrypted CDRs for traceback purposes.

Why can't we just put traceback info in headers: Implementing traceback information in headers faces the same hurdles as S/S.

Why develop a new protocol instead of automating the manual tasks done by groups in ITG?: The ITG currently maintains a semi-automated traceback system that sends notifications

to providers who are mandated to respond within 24 hours. Automating the current traceback tasks will require all providers to implement a traceback API that integrates with the ITG systems. While this automates the process, the gain on "traceback throughput", the number of computable traceback requests per month, remains low as the process is still serial and involves providers active participation for every traceback request. Tracebacks would run into dead-ends if a single provider does not cooperate, their portal goes down, responds with misleading information, or partial deployment.

For providers, this alternative adds an additional cost of maintaining an inbound traceback system. An adversary could exploit vulnerable API implementations of this mandate to access affected providers' sensitive data. Note that compromising a carrier's API server exposes its peers and network trends as well as subscriber call history, and well-funded companies who use industry best-practices are regularly breached. We designed Jäger as a centralized distributed system to alleviate serial traceback lookups, enabling providers to be passive entities rather than active for traceback computation. This design choice not only enhances the traceback throughput but also centralizes security management from several thousand providers to two organizations responsible for overseeing Jäger. Moreover, if any single Jäger entity is compromised, no plaintext data is leaked.

Why require participation from all providers if only the originating provider's records are needed?: If only originating providers submit records, traceback fails because malicious carriers likely won't comply. Limiting submissions to the originator and second hop is infeasible because providers cannot determine their sequence in the call path. Thus, requiring all providers to submit records becomes essential to trace back and identify people facilitating bad calls. Having the ability to construct the full call path has added advantages such as trace-forwards to debug call routing or blocking errors.

Who will operate the TA and RS?: The telephone network is the *ideal* environment for Jäger because regulators like the FCC already designate trusted third parties that provide singleton functions. Examples include toll-free numbering, local number portability (LNP) databases, S/S certificate authority governance, and the current traceback clearinghouse, ITG. The FCC periodically solicits applications to serve as these entities and a fair and competitive process among several for-profit enterprises follows. The selected entities can then charge reasonable fees for the service they provide.

For Jäger, there are two entities to recruit. The RS roughly corresponds to an entity like the LNP databases, and many vendors have the technical ability to serve in this role. The TA performs functions similar to the current ITG, like registering providers to their system and determining if a traceback query is appropriate, so modifying that existing role would provide a straightforward on-ramp to deployment.

What if providers refuse to participate?: Jäger must be mandated to be effective. Regulation can compel providers to participate or have their network access revoked. Unlike S/S, Jäger is compatible with all network technologies in use.

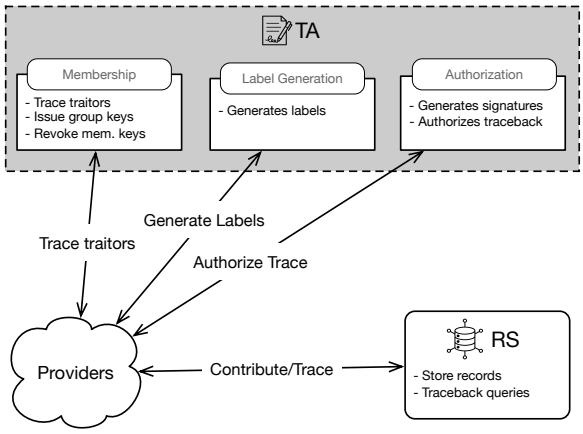

Figure 1: Jäger facilitates efficient and rapid traceback through collaborative efforts. Providers engage with the TA to secure membership, create labels, and acquire trace authorizations. Additionally, carriers send(receive) encrypted records to(from) the RS

Table 1: Notation used in the protocol

Notation	Description
RS	Record Store
TA	Traceback Authority
P_i	A carrier
src	Telephone number of call initiator
dst	Telephone number of call recipient
ts_i	Time call reached provider P_i
call-details	Defined as $src \| dst \| ts$
hop_i	Hop submitted by P_i defined as $P_{i-1} \| P_i \| P_{i+1}$
call-label	A label for a call

5 System and Protocol Design

In this section, we discuss Jäger's system architecture and detailed protocol.

5.1 Jäger Architecture

Figure 1 shows a high-level diagram of Jäger's system architecture. There are three kinds of entities in our system:

Carriers $\mathbf{P_i}$: A carrier P_i receives calls from either the source (src) or from a previous carrier (P_{i-1}) and forwards the call to either the destination (dst) or the next carrier P_{i+1}. A call is identified by its call-details = $(src \| dst \| ts)$ where ts is the time at which the call reaches the carrier.

Record Store (RS): The RS maintains a database that stores *encryptions* of the hops associated with a call. Recall that a hop is a tuple hop := $(P_{i-1} \| P_i \| P_{i+1})$.

Traceback Authority (TA): The TA has the following functions:

- *Authorizing Trace Requests*: The TA provides the signatures that enable a carrier to decrypt records retrieved from the RS. These signatures are computed on the call labels.
- *Managing Providers' Anonymous Authentication*: The TA manages the group signatures for the carriers. This consists of adding legitimate providers to the group and providing them

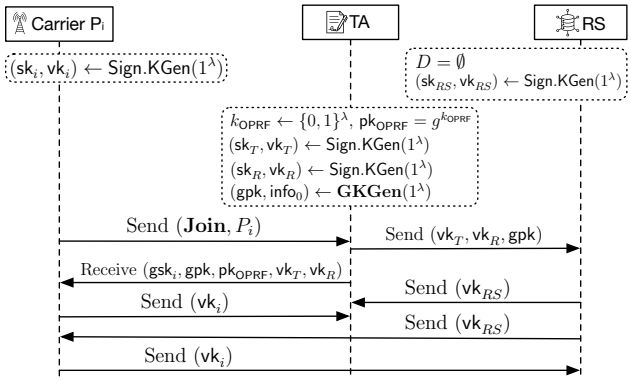

Figure 2: During setup protocol, TA generates and announces public parameters. Providers request group membership and are assigned member secret key upon acceptance by the TA

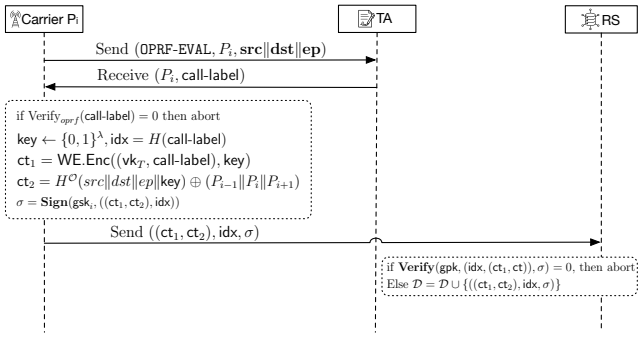

Figure 3: In contribution protocol, provider submits ciphertexts, compliant with the protocol, to the Record Store

with credential to sign on behalf of the group. The TA is responsible for accountability, it can deanonymize signatures in case of misbehavior and hold the corresponding entity responsible.

- *Generating Pseudorandom Labels*: The TA interacts with carriers to compute call-labels.

5.2 Protocol Overview

Jäger consists of four protocols: *Setup*, *Contribution*, *Trace*, and *Provider Accountability*, which we describe in detail below.

Cryptographic Primitives: As discussed in Sec. 2.2, Jäger uses a witness encryption scheme for signatures (WES)[24, 47], a group signature scheme[16], an oblivious PRF protocol[43], signature schemes, and a hash function. The notations used in our protocols are described in Table 1. We present the protocol in detail in Appendix B. Below we present an overview of the protocol.

Setup Protocol: The Setup protocol is described in Fig. 2. This protocol is run by the TA and carriers to set up their keys.

TA Setup: The TA sets up (1) the group with a group master key and secret key. (2) the PRF key for the oblivious PRF and announces a public key (denoted $\mathsf{pk}_{\mathsf{OPRF}}$) corresponding to the PRF key. (3) two signature key pairs $(\mathsf{sk}_T, \mathsf{vk}_T)$ and $(\mathsf{sk}_R, \mathsf{vk}_R)$ and announces vk_T and vk_R to all entities. Here, signatures using sk_T will be used to decrypt the witness encryption ciphertexts, and sk_R will be used to authorize trace requests.

Carrier $\mathbf{P_i}$ Setup: Each provider (denoted P_i) joins the system by first interacting with the TA to get a distinct group signing key gsk_i, which they can use to sign anonymously on behalf of the group. They generate a regular signing key pair $(\mathsf{sk}_i, \mathsf{vk}_i)$ for authenticated communication with the TA and RS (during trace).

Finally the RS initializes a database $\mathcal{D}$ and sets up a signature key pair $(\mathsf{sk}_{RS}, \mathsf{vk}_{RS})$ and announces vk_{RS}

Contribution Protocol: Providers record the hops with the RS using the Contribution protocol. We consider the case of submitting a single hop to the RS in Fig. 3. Each provider in the call path parses the CDR into call-details = $(src\|dst\|ts)$ and hop = $P_{i-1}\|P_i\|P_{i+1}$ as defined in Sec 4. To contribute call records, the provider P_i anonymously submits a witness encryption of the message hop = $(P_{i-1}\|P_i\|P_{i+1})$ to RS, with a pseudorandom label associated with it and a group signature for authentication. The ciphertext and the label leak no information about the call thus providing confidentiality of the call, and the group signature leaks no information about the provider sending this information, thus providing anonymity to the carrier. We elaborate on how the label, the encryption, and the signature are computed below.

The Contribution protocol consists of two phases: the label generation phase and the submission phase. In the label generation phase, the label is computed with the help of the TA, using an oblivious PRF protocol. The TA acts as a server and holds a PRF key, and the provider acts as a client with the input. We assume that each ts is associated with an epoch, ts truncated to nearest centisecond, denoted ep. The provider uses call-details = $(src\|dst\|ep)$ as input and the output of the protocol (denoted call-label) is learned only by the provider. We note that with the help of $\mathsf{pk}_{\mathsf{OPRF}}$, the provider can compute an efficient pairing check to verify that the output received from the TA is indeed correct, and that the PRF key k was used to compute the call-label as detailed in Fig. 9. The provider then computes $\mathsf{idx} = H(\text{call-label})$. Recall that idx is used instead of call-label to index the ciphertexts to prevent the TA from trivially decrypting all ciphertexts in the case that the database of ciphertexts and call-labels are leaked.

In the submission phase, the provider prepares the encryption as follows. P_i samples a λ-bit uniform key from $\{0,1\}^\lambda$ and encrypts key with the WES scheme using $(\mathsf{vk}_T, \text{call-label})$ as the encryption key to get a ciphertext ct_1. The WES scheme ensures that the ciphertext can only be decrypted using a signature on call-label signed using sk_T by the TA. P_i further computes $\mathsf{ct}_2 = \text{hop} \oplus H(\text{call-details}\|\text{key})$, where H is modeled as a random oracle. Note that we require call-details as input to the hash function to extract the call-details in the proof of security. Finally, the provider signs the message $((ct_1, ct_2), \mathsf{idx})$ using the group signature scheme, obtains σ, and sends the resulting tuple $((ct_1, ct_2), \mathsf{idx}, \sigma)$ to the RS. Upon receiving a submission request, RS validates the group signature σ and stores the tuple in the database; if verification fails, the request is dropped.

Trace Protocol: To trace a call, the provider must retrieve all the hops corresponding to a call from the RS. Fig. 4 illustrates the

sequence of events in the trace protocol. Initially, the provider needs to obtain the labels corresponding to these calls. For this purpose, the provider computes the labels with the assistance of the TA using the OPRF protocol described above. We assume an upper limit $t_{\max}$ on the duration of a call setup from the source to the destination. The provider computes epochs ep corresponding to the timestamps $ts* \in [ts - t_{\max}, ts + t_{\max}]$ and computes the labels associated with the call-details for each of these epochs.

Before we describe how the provider gets the full trace, we note that a malicious provider colluding with the RS could potentially compute labels on arbitrary call-details and check with RS if such labels exist in the store, revealing information about the existence of a call between a certain initiator and recipient. If the provider attempts to perform this attack for a specific initiator and recipient, we call it a "targeted attack". If the adversary's goal is to map the network by trying to compute labels on arbitrary call-details, we refer to such attacks as "grinding attacks". To mitigate the grinding attack, we implement rate-limiting on the number of requests a provider can make. For this purpose, when the provider interacts with the TA to compute a call-label, the TA maintains a count of requests made by a provider and will not authorize further requests if a certain limit is exceeded. To give authorization on this request, the TA will sign $\mathsf{idx} = H(\text{call-label})$ using sk_R, and this signature σ_R serves as an authorization for traceback. Moreover, consider the case that the TA is malicious (and the RS is honest) and is colluding with a provider, then they can easily mount the grinding attack described above by computing arbitrary labels and requesting the corresponding idx from the RS. To mitigate this, we will also require the RS to implement rate-limiting on trace requests. This will ensure that a provider colluding with a TA cannot mount grinding attacks.

The provider requests the RS for the ciphertexts corresponding to the idx. The RS first checks that the signature σ_R is verifiable using vk_R. It rejects the request if this is not the case. The RS identifies the ciphertexts corresponding to the idx in $\mathcal{D}$ and sends $\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{idx}, \sigma_{RS}$, where $\sigma_{RS} = \mathsf{Sign}(\mathsf{sk}_{RS}, (\mathsf{ct}_1, \mathsf{ct}_2, \mathsf{idx}))$. The provider then asks the TA for a signature on the call-label and uses these signatures to decrypt the ciphertexts and compute the hops.

Provider Accountability Protocol: Some providers may provide wrong hops to frame others. Since the encrypted hops are anonymously submitted to the RS, we need a mechanism to catch the malicious providers. Our group signature protocol allows the TA to *open* any group signature and reveal the provider that signed a message. Thus, if a trace seems malformed, the provider submits all ciphertexts, hops, and signatures for the call retrieved from the RS and sends it to the TA. The TA runs a Validate function that outputs the set of faulty hops. The GM then identifies the signatures that correspond to these hops and deanonymizes them to return the set of providers that submitted malformed/wrong hops.

5.3 Traceback Validation

Once a provider has hops, they must assemble the hops into the complete path. We call this step "Traceback Validation" because it will identify the correct path or detect inconsistencies or missing records that should be manually investigated.

We validate a traceback by inserting decrypted records into a directed multi-graph. In a multi-graph, two nodes can be connected by multiple edges. Each record is a graph with three nodes: P_{i-1}, P_i and P_{i+1}; and two edges: $P_{i-1} \rightarrow P_i$, and $P_i \rightarrow P_{i+1}$. A traceback is the multi-graph union of such individual sub-graphs(hops).

Ideal Scenario: Each P_i contributes a valid record for a given call. Figure 6(Full Path) visualizes this ideal scenario. Let $\deg_{in}$ and $\deg_{out}$ denotes in and out degrees respectively. The originating provider P_1 has $\deg_{in} = 0$ and $\deg_{out} = 2$. Here, one edge of $\deg_{out}$ (denoted by solid line) indicates its assertion to P_2 while the other edge (denoted by dashed-line) indicates P_2's assertion to P_1. The terminating provider P_4 has $\deg_{in} = 2$ and $\deg_{out} = 0$ for reasons symmetrical to one provided for the originating provider. Finally, each transit provider (P_2 and P_3) has $\deg_{in} = 2$ and $\deg_{out} = 2$. Any deviation from the ideal case helps us determine "faulty hops"—hops that are conflicting.

Determining Faulty Hops: We detect faulty hops by checking four properties formulated from the ideal scenario:

Origin invariant: A call can have only one originator. This property holds if there is exactly one node in the directed multi-graph with $\deg_{in} = 0$ and $\deg_{out} \in \{1, 2\}$. Otherwise, the originating record is missing, or some P_i submitted malformed records. Note that $\deg_{out} = 1$ does not necessarily imply a malicious originator since true originators will still have $\deg_{out} = 1$ if there is a missing record from their downstream provider. If there is more than one node with $\deg_{out} = 2$, then there are conflicting originators; in this case, we construct different call paths for each originator.

Terminating invariant: A call can have only one terminating provider. This property holds if there is exactly one node with $\deg_{in} \in \{1, 2\}$ and $\deg_{out} = 0$. Otherwise, the terminating record is missing, or a P_i submitted malformed records. This is symmetric to the origin invariant except that values for $\deg_{in}$ and $\deg_{out}$ are swapped.

Transit invariant: This property identifies all transit providers and validates for nodes having $\deg_{in} \in \{1, 2\}$ and $\deg_{out} \in \{1, 2\}$.

Connectivity invariant: This property determines if the full call path can be recovered. It holds if there is a path between every pair of vertices in the traceback graph.

Figure 5 presents the detailed algorithm for determining the faulty hops from the decrypted records and the ability to reconstruct the call path.

Traceback Robustness and Partial Deployment: In a scenario where all parties are honest, we derive all the benefits of the scheme. Importantly, even in partial deployment scenarios where only certain providers submit their records, our scheme can identify the call originator under certain conditions. For example, if the originating provider is the sole contributor for a call, the system can still successfully identify the call origin without the contributions from any other provider in the call path. If the second hop provider participates, our system can identify the origin even if no other provider participates. If there are ever conflicting origin claims, we initiate a manual investigation to identify the source and punish the dishonest party. If any other intermediate party participates honestly, we can still reduce manual traceback time. With the call path recovery algorithm, there may be cases where we can precisely identify the bad actor. However, this is an "added bonus" and not the main goal of the scheme. The tracing algorithm relies on a best-effort strategy,

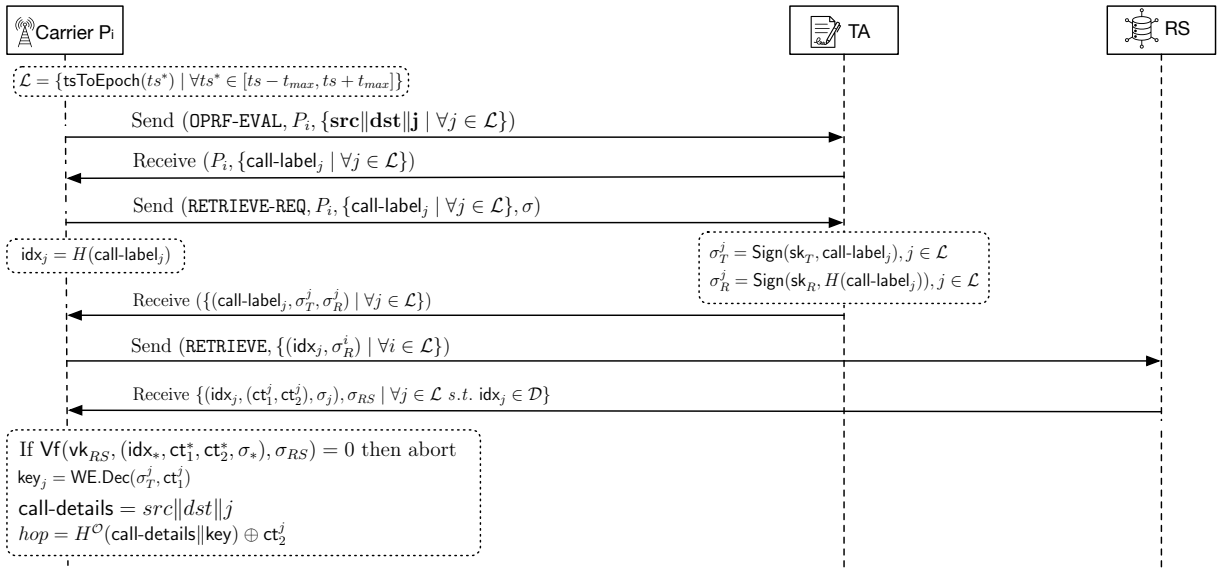

Figure 4: In the trace protocol, providers obtain labels, ciphertexts, and decryption authorization signatures for a call from TA. Here, the bold text shows $src\|dst\|j$ is not sent in plaintext but blinded as in OPRF

$\text{Validate}(\{\text{call-details}_j, \text{call-label}_j, (ct_0^j, ct_1^j), \sigma_j, \sigma_T, \text{hop}_j\}_{\text{hop}_j \in C_{\text{trace}}})$: The validation algorithm proceeds as follows:

(1) Verify that call-label_j corresponds to the call that is to be validated using call-details.
(2) Verify that hop_j can be computed from (ct_0^j, ct_1^j) using call-label_j.
(3) Each hop_j is defined as $P_{i-1} \,||\, P_i \,||\, P_{i+1}$. Create a directed multigraph $\mathcal{G}$ such that: $\mathcal{G}.V = \{P_k \mid P_k \in \text{hop}_j, \forall \text{hop}_j \in C_{\text{trace}}\}$ and $\mathcal{G}.E = \{\{P_{i-1} \rightarrow P_i, P_i \rightarrow P_{i+1} \mid \forall \text{hop}_j \in C_{\text{trace}}\}\}$. We use $\{\{\cdots\}\}$ to denote that $\mathcal{G}.E$ is a multi-set.
(4) For each $v \in \mathcal{G}.V$:
  (a) If $\mathcal{G}.\text{deg}_{\text{in}}(v) = 0$, add v to $\mathcal{O}$: *list of providers that claim to be originators.*
  (b) If $\mathcal{G}.\text{deg}_{\text{in}}(v) > 0$ and $\mathcal{G}.\text{deg}_{\text{out}}(v) = 0$, add v to $\mathcal{D}$: *list of providers that claim to be terminating.*
  (c) Else add v to $\mathcal{T}$: *list of providers that claim to be transit.*
(5) If $|\mathcal{O}| = 1$ and $\text{deg}_{\text{out}}(v) > 0$, set $\mathcal{P}_O = v \mid v \in \mathcal{O}$ else do:
  (a) For $v \in \mathcal{O}$: if $\mathcal{G}.\text{deg}_{\text{out}}(v) = 2$ add v to $\mathcal{O}_T$ (*possibly true origins*) else add v to $\mathcal{O}_F$ (*possibly faulty or missing attestation*)
  (b) If $|\mathcal{O}_T| = 1$ set $\mathcal{P}_O = u \mid u \in \mathcal{O}_T$: *(true origin)*
  (c) $\mathcal{F}_O = \mathcal{F}_O \cup \mathcal{O}_F$: *may require investigation from output graph*
(6) If $|\mathcal{D}| = 1$ and $\text{deg}_{\text{in}}(v) > 0$, set $\mathcal{P}_T = v \mid v \in \mathcal{D}$ else do:
  (a) For $v \in \mathcal{D}$: if $\mathcal{G}.\text{deg}_{\text{in}}(v) = 2$ add v to $\mathcal{D}_T$ else add v to $\mathcal{D}_F$
  (b) If $|\mathcal{D}_T| = 1$ set $\mathcal{P}_T = u \mid u \in \mathcal{D}_T$ *(true terminator)*
  (c) $\mathcal{F}_D = \mathcal{F}_D \cup \mathcal{D}_F$
(7) For $v \in \mathcal{T}$: if $\mathcal{G}.\text{deg}_{\text{in}}(v) \notin \{1, 2\}$ and $G.\text{deg}_{\text{out}}(v) \notin \{1, 2\}$ add v to $\mathcal{F}_T$.
(8) If $\mathcal{G}$ is weakly connected, then output $\mathcal{G}.\text{shortestPath}(\mathcal{P}_O, \mathcal{P}_T)$ else $\mathcal{G}.\text{subgraphs}()$
(9) Output $(\mathcal{F}_O, \mathcal{F}_T, \mathcal{F}_D), (\mathcal{P}_O, \mathcal{O}), (\mathcal{P}_T, \mathcal{D}), (\mathcal{T})$

Figure 5: The Validate algorithm analyzes deviations from the ideal scenario to determine the call path and faulty sets

and we believe that societal incentives, including civil or criminal liability, will motivate the entities to behave honestly. In Sec. 7.2, we present an evaluation of Jäger in partial deployment.

5.4 Security of Jäger

In this section we provide informal arguments that Jäger achieves the security properties outlined in Sec 3.2. The formal proof in the UC framework is in Appendix C .

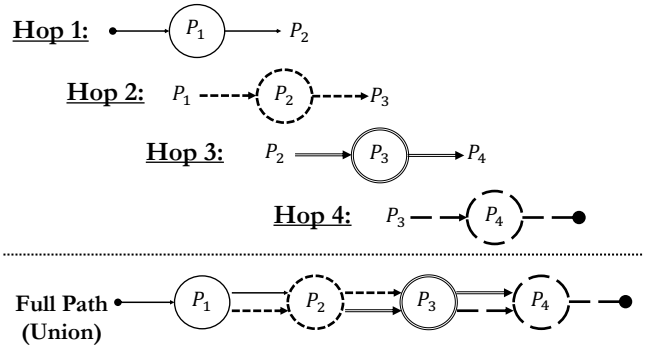

Figure 6: The full call path is a union of subgraphs from hop**s**

Theorem 1. [Informal] *Assuming the CPA security of the witness-encryption scheme, the unforgeability of the signature scheme, the security of the group signature scheme, the security of the OPRF protocol, and secure hash functions, Jäger achieves record confidentiality, the privacy of individual caller, blinds network trends and provider associations. Moreover if the TA is honest, Jäger additionally achieves provider accountability.*

Trace Authorization: Since a provider requires a signature from the TA to decrypt the ciphertexts, and a provider needs to know the call-details to request this signature, only authorized parties can perform a trace successfully.

Call Confidentiality: Recall that the *src*, *dst* information of the call is used only to compute the label. Since the computation of the label is through an OPRF, we guarantee that the TA does not learn the *src* and *dst* of the call. Since the OPRF output is pseudorandom, the label by itself will also not reveal any information about the caller and the callee of the call.

Trade Secret Protection: Recall that each hop is encrypted and stored at the *RS*. Decrypting the encrypted hop requires knowledge of the labels and a signature on the call-label, information accessible only to entities involved in the call or those accurately guessing the source, destination, and call time. The unforgeability property of the signature scheme ensures that an entity cannot forge a signature on behalf of the TA, preventing unauthorized decryption of the hops. Additionally, the CPA security of the WES scheme safeguards the contents of these ciphertexts. As previously described, a malicious provider might attempt to guess arbitrary call details, create corresponding labels, and decrypt records stored at the *RS*, i.e. they try to mount a grinding attack. To mitigate this risk, we implement rate-limiting on such requests by having the TA restrict the number of authorizations granted to each provider.

Record integrity: Since all contributed records are signed using a group signature, and the RS verifies this signature before adding the record to the database, we ensure that only authorized users can contribute records.

Record Accountability: We achieve provider anonymity since each submission to the *RS* does not include any identifier of the provider. They are instead signed using the group signature scheme, ensuring that the provider is anonymous within the group. When a provider is misbehaving (e.g. by sending a malformed hop) the group signature can be opened by TA thus revealing the provider that signed the submitted hop. We note that if the TA is malicious they may just not reveal any identity and accountability may not be guaranteed. But even if the TA is malicious they cannot frame an honest provider as the sender of the record.

6 Implementation

This section describes the prototype implementation of Jäger and how we obtained CDR data to evaluate it.

6.1 Prototype Implementation

We describe a prototype implementation for each component of Jäger, which enables us to evaluate its performance.

Traceback Authority: We implement the TA as an HTTP server that uses BLS Signatures [15] to compute authorization signatures. For this function, we exposed an endpoint for authorizing trace requests. We set up our group signature scheme using short group signatures [14] implemented by IBM's `libgroupsig` [41] and exposed an endpoint for opening signatures. Finally, we used the `ristretto255` elliptic curve (`oblivious` Python package) for our OPRF protocol [17]. The TA exposes an API endpoint for label generation.

Record Store: The RS is an HTTP server with a database for storing records. We use the `Click-House` columnar database. The RS exposes endpoints for contribution and traceback queries.

Carrier: We implement a carrier as a process that runs the protocol and interacts with other components in the system. For performance, we implemented the witness encryption scheme in C++ using the `BLS381` elliptic curve library[52] and wrote Python bindings using `Pybind11`.

6.2 Data Generation

Because CDRs are data protected by US laws, they are unobtainable. We are unaware of work that models contemporary PSTN call records so we develop a PSTN model to algorithmically generate data. We first generate a graph to represent the providers's peering relationships. We then model a social graph of telephone users and assign users to carriers. We then generate CDRs for calls between users. While we believe our model is reasonably accurate, in Section 7, we show that even if call volumes are significantly higher, Jäger will be practical.

Telephone Network: We use an iterative graph generation algorithm to construct a network consistent with real-world telephone topology. We have identified three properties that a reasonable model generator must consider:

Preferential Attachment: New carriers prioritize connecting with larger carriers that handle a significant traffic volume, so providers with wider coverage generally acquire more new customers. In our model, the number of carrier connections is proportional to its current degree.

Market fitness: Smaller providers can attract new customers but rarely surpass larger providers' market share. This feature enables us to mirror real-world scenarios, such as AT&T maintaining a

higher market share even as the network evolves.

Inter-carrier agreements: Represents financial agreements, such as mutual compensation for handling each other's traffic, rates for different types of traffic, and billing arrangements.

We use the Bianconi-Barabasi model [12] to achieve these properties. Our network consists of N nodes, each labeled P_i representing a unique carrier node. The weight of an edge between any two carrier nodes signifies the inter-carrier agreement amount, which we use in the shortest path computation. We assume each carrier node seeks to minimize the cost of transmitting call connections, a notion that aligns with real-world practices.

Subscribers Network: We model subscribers' social interactions using a scale-free network. We create a total of S subscribers, each represented by a phone number in the NPA-NXX-XXXX format. We allocate phone numbers to subscribers based on each carrier's market share. We constructed a Barabási-Albert [8] graph denoted as $G_s = (V_s, E_s)$ for the subscribers. Since Jäger is primarily interested in scenarios where the caller and the called party belong to different carrier networks, we minimized the probability that neighboring nodes of a given subscriber s_i are on the same network as s_i.

CDR generation: Each edge in the social network G_s represents a call between two subscribers s_i and s_j. For each call $s_i s_j$, we represent the call path as the shortest path between their respective providers in the topology. Each hop within the call path represents a CDR record.

7 Evaluation

For Jäger to succeed, runtime performance, queries, and insertions of records need to be fast to handle the volume of call traffic being processed daily by the network. Our prototype implementation allows us to test its performance for each protocol phase. We consider the following metrics: storage growth rate, minimum vCPUs required, time for each protocol, and the minimum bandwidth required as shown in Table 2.

Experiment Setup: Our Experiments were run on a Linux virtual machine with 32 vCPU and 64GB of memory. The host was a Super Micro Server with an Intel Xeon Gold 6130, ECC DDR RAM, and 12Gbps SAS drives. In experiment 1, we benchmark individual tasks such as label generation, record encryption and decryption, opening signatures, signing, and verifying signatures. We executed each process in a single thread 1,000 times to obtain the average, minimum, maximum, and standard deviation (SD) of the runtime.

In Experiment 2, we generated a network graph with 7,000[25] carriers and simulated $R_C = 10,000$ calls per second. No entity currently has visibility into the call volumes of all carriers in the United States. As a result, we did not find well-supported statistics on the overall call volume in the United States. We are especially concerned with the number of calls that transit multiple carriers, and, of course, this figure is even less attested. Of the statistics we found, many had no citations, did not describe their methodology, or were otherwise suspect in accuracy. As a result, we settled on the round number of 10,000 calls per second for the North American phone network. This corresponds to roughly 800,000,000 calls per day. We admit that this choice is arbitrary, but as we see later, our system has substantial headroom and is also horizontally scalable. Our evaluation in the following sections considers the TA and RS as singletons.

Table 2: Minimum system requirements for each component in Jäger suppose the network processes 10,000 calls per second.

Component	Storage	vCPU	Bandwidth
Traceback Authority			
Label Generation	O(# of Providers)	4	25 Mbps
Group Management	O(# of Providers)	1	20 Mbps
Trace Authorization	O(# of Traces)	1	10 Mbps
Record Store			
Process Submissions	O(# of Records)	257	800 Mbps
All Carriers			
Contribution	O(1)	208	800 Mbps

Table 3: Performance for protocols measured in milliseconds.

Task	Mean	Min	Max	Std
Label Generation	0.073	0.066	0.166	0.007
Contribution	4.143	3.708	5.980	0.173
Authorization	0.419	0.376	0.615	0.023
Decryption	0.847	0.780	1.064	0.039
Open	0.147	0.134	2.865	0.009
Verify Group Signature	2.310	2.118	2.865	0.098

7.1 Protocol Evaluation

Setup Protocol: The Setup is less than 10ms for all entities. Storing the identity of group members (providers) at the TA grows linearly in the number of providers.

Contribution Protocol: We measure the time and minimum system requirements to complete the protocol for the label generation and submission phases.

Label generation: Providers request a PRF evaluation from the TA. Table 3 shows that a single label generation takes 0.073ms on average (SD = 0.007ms). *Hence, the TA can evaluate* 13,698 *labels on average on a single vCPU per second.*

Bandwidth for label generation: We estimate the minimum bandwidth required to generate labels between providers and the TA over an HTTP connection from:

$$B_w = R_{rec} \cdot (S_{req} + S_{res}) \cdot (1 + O_{http}) \quad (1)$$

R_{rec} is the rate at which records are generated across the network, S_{req} and S_{res} are request size, response size, and O_{http} additional overhead (in percentage) introduced by HTTP, respectively. On average, calls generated by our network have 5 hops[4], thus $R_{rec} = 5 \cdot R_C$. Each request payload is 32 bytes; thus, $S_{req} = 256$ bits, likewise, $S_{res} = S_{req}$ since the PRF is length-preserving. We compute overhead using:

$$O_{http} = \text{Overhead per Request/Batch Size} \quad (2)$$

We measured the average overhead of about 652 bytes per request for the minimal headers when using HTTP/1.1. The group signature forms a significant fraction of this overhead. *The TA can handle label-generation requests even with substantial network overhead since 25 Mbps is vastly below nominal internet throughput.*

[4]ITG reports that tracebacks usually go through 4 or more hops[42]

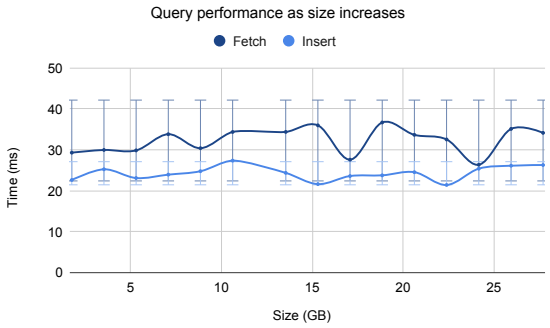

Figure 7: Performance measured in milliseconds for insertion and select queries as database size grows.

Record Submission: We measure the time it takes to contribute 1 CDR record. From Table 3, contributing 1 CDR takes 4.143ms on average with a SD of 0.173ms. A provider can process 241 records in a second on a single vCPU. *At a rate of 50,000 records per second*[5]*, a minimum of 208 vCPUs are required to encrypt all records.* This may seem high, but recall we estimated 7,000 providers, so the CPU requirements are small in practice for providers.

Bandwidth for record submission: The record submission request payload is 1,900 bytes in size (15,200 bits), comprising a label, ciphertext, and signature. Using Equation 1, we estimate the minimum bandwidth for submitting records as 800 Mbps. *Therefore, with 800 Mbps the RS can handle submission requests for the entire network.*

Once the RS receives the submission request, it verifies the signature and inserts it into the database. Table 3 shows that verifying a group signature takes 2.310ms on average with a SD of 0.098ms. *Hence, the RS can verify* 432 *submissions per core per second, thus requiring a minimum of 24 vCPUs to validate contribution requests.*

Record Store: Our second experiment evaluates the growth of storage and how that affects querying and inserting records. As shown in Figure 7, we observed that the time it takes to insert records into the database is independent of the database size, averaging about 24.28ms with a SD of 1.681ms. Note that 23.28ms represents the average time to process a single INSERT statement with 1 VALUE row. Insertion overhead could significantly improve if multiple values are appended to a single insert statement. *The RS can process 43 insert statements in a second on a single vCPU. To cover the rate at* 10,000 *calls per second requires a minimum of 233 vCPUs.* The database size grows at a rate of 1.5 TB per day, roughly $100 per day. Deployment consideration may require that records are only retained for a given period, after which they get expunged.

Trace Protocol: We consider the Trace protocol for a single traceback. Label generation is the same as before. Whenever authorization for a traceback is requested, the TA records this in the database along with the provider. Storage grows linearly with the number of traceback requests. To decrypt a record, we need a signature on the label. Signing a label takes a mean time of 0.419ms with an SD of 0.023ms. We measured that decrypting a single record takes 0.847ms on average with a SD of 0.039ms. *A single vCPU can decrypt 1,180 ciphertexts per second.*

Provider Accountability Protocol: In the Provider Accountability protocol, we determine the faulty hops from the decrypted records as described in section 5.3. We measured the runtime for analyzing the records and determining the faulty hops to be 0.052ms. Opening a signature takes 0.147ms on average with a SD of 0.009ms as shown in Table 3.

7.2 Traceback Evaluation

Throughput: We estimate the compute time for one traceback as the sum of the following components: 1) label generation, 2) trace authorization, 3) retrieving records from RS, and 4) decrypting records. In Section 5.2, we generate all labels in the range $\mathcal{L} = \{\text{tsToEpoch}(ts^*) \mid \forall ts* \in [ts{-}t_{\max}, ts{+}t_{\max}]\}$. Assuming $t_{\max} = 10$ seconds and ep is in seconds, we retrieve records for 21 labels ($2 \cdot t_{max} + 1$). The estimated average compute time for one traceback is 0.75 seconds. *Without communication latency, we can complete close to 3.5 million tracebacks in a month, increasing the current throughput by a multiplicative factor of 11,520.* Communication latency would reduce this number, but the result remains a significant fraction of the current abuse.

Partial Deployment: Jäger was explicitly designed to support partial deployment because of lessons learned from S/S and proposals like RPKI to secure Internet communications. Given that both of those systems failed to meet their goals at the current levels of deployment, it is worth considering how Jäger might perform.

First, though, we will need to establish definitions and deployment models. We say that a traceback is successful if the Jäger record store has at least one record that identifies the originating hop in the call path. For this analysis, we call providers that deploy Jäger "adopters," and define "adoption rate" as the fraction of all providers who are adopters. In S/S, the largest networks tended to be the earliest adopters, and only small networks continue operating without S/S (excluding, of course, legacy networks that are incompatible). This trend held because regulatory agencies gave smaller networks more time to comply with deployment mandates than larger ones. Published tracebacks and successful enforcement actions report that the vast majority of illegal calls come from small networks.

We used our network model from Section 6.2 to estimate traceback success in partial deployment. We simulate randomly dialed robocalls originating from the smallest r% of carriers and adopters as the largest a% of carriers. We find that if robocalls originate from the bottom 10% of networks, with Jäger adoption by only the largest 2% of carriers, Jäger can still successfully traceback *27%* of all robocalls. When adoption increases to 10%, which is still lower than the current rate of adoption of S/S, traceback success leaps to 55% of robocalls.

Given that robocallers place millions or billions of calls, even at low adoption Jäger would produce mountains of evidence that could lead to takedown of illegal robocalling operations. Of course, these results are likely sensitive to our model choices. Still, even if this estimate is wrong by an order of magnitude, tracing even a small fraction of the thousands or millions of robocalls is a vast

[5]For an average of 5 hops per call, 10,000 calls per second correspond to 50,000 CDRs per second

improvement compared to 300 per month we see today.

8 Deployment Considerations

In this section, we discuss practical deployment concerns relating to incentives, engineering concerns, and the trace authorization process.

Deployment Incentives: When integrating a new security mandate into an existing system, it is essential to consider incentives. How will a provider benefit from deploying Jäger? Eliminating bulk illegal calls aligns with providers' interests since these calls are often unanswered and do not generate revenue. However, this motivation alone may not be enough. Fortunately, the S/S framework has shown that regulatory mandates can drive widespread network changes.

Engineering Concerns: In our discussions with practitioners, we identified engineering concerns that, while challenging, are manageable. We anticipate that carriers will face hurdles integrating with billing systems and managing new cryptographic infrastructure. However, we anticipate that deployment will still be easier than S/S because it involves only latency-insensitive backends. Jäger itself will face typical site reliability engineering challenges, such as replication and public key infrastructure issues like governance. The industry has successfully navigated similar challenges with S/S, so we can have confidence that these engineering concerns are surmountable.

Rate Limiting and Authorization: Our performance evaluation assumed that traceback authorization would be instantaneous. In reality, there will be some processing time. Human intervention may be required to review requests before or after they are system-signed. In some cases, the trace authorizer could automatically approve requests, such as requests from honeypots for calls to their own numbers. Given that honeypots receive thousands of calls daily [57, 58], scalable traceback will significantly enhance investigative processes.

9 Related Work

Telecom fraud [67] is a long-standing problem [48, 49, 51] that continues to impact carriers [50, 62, 65, 66] and subscribers [3, 59]. Most fraud schemes stem from inadequate authentication mechanisms [21, 22, 60, 61, 73] and security flaws in legacy telecom signaling protocols [44, 63, 75, 76]. Attempts to address these problems through protocol enhancements [37, 67, 72] and defenses [54, 62] have had limited success. Illegal robocalling, a form of telecom fraud, frustrates phone users, carriers, and regulators. Over the past decade, widespread adoption of VoIP technology has led to a surge of scams [57, 58, 68, 72] perpetrated using robocalls. To study the operational characteristics of robocallers, researchers have employed a wide range of techniques such as CDR data mining [46, 64, 71], machine learning [36, 45, 77], audio processing [57, 58], carrier collaboration [6, 7], and reputation scoring [13, 56]. Mitigation techniques based on caller ID authentication [74], spam filtering [23], call-blocking apps [5, 39, 51, 55, 70], and increased penalties [27, 28] have been proposed. However, they have failed to significantly deter bad actors from originating illegal robocalls.

In Dec 2019, the US Congress passed the TRACED Act [27] to protect consumers from illegal robocalls. Consequently, the FCC designated the Industry Traceback Group (ITG) [42] to track down entities responsible for originating illegal robocall traffic using traceback. Tracebacks remain invaluable in uncovering and prosecuting numerous illegal robocalling operations [26]. However, its effectiveness is limited since it is a manual, iterative, and time-consuming process. Each traceback requires cooperation among carriers spanning multiple days to pinpoint the source of illegal robocall traffic. Network traceback methods like packet marking and router logging [10, 34, 35] are ineffective to traceback phone calls [40] due to general IP traceback limitations.

Our automated traceback technique addresses these challenges and encompasses all transit carriers. Notably, Jäger does not require modifications to existing infrastructure, making it compatible with other protocols.

10 Conclusion

In this paper, we described the design of Jäger, a distributed system to facilitate automatic call traceback. Jäger facilitates the anonymous-but-traceable submission of encrypted call records to a central source, which after vetting from an authorizer allows traceback only by parties with information about a call to be traced. We demonstrate that despite the expensive cryptographic primitives and coordination cost, the system is practical today with modest hardware and low latency. In so doing, we show that Jäger represents a powerful new tool to combat telephone abuse.

Acknowledgments

We thank our anonymous shepherd and reviewers for their support of the paper. This material is based upon work supported by the National Science Foundation under Award No. CNS-2142930. Funds from the 2020 Internet Defense Prize also supported portions of this work.

References

[1] D. Adei, V. Madathil, S. Prasad, B. Reaves, and A. Scafuro. GitHub: Jäger Source Code. https://github.com/wspr-ncsu/jaeger.git.
[2] D. Adei, V. Madathil, S. Prasad, B. Reaves, and A. Scafuro. Zenodo: Jäger Source Code. https://zenodo.org/doi/10.5281/zenodo.12733869.
[3] M. Arafat, A. Qusef, and G. Sammour. Detection of Wangiri Telecommunication Fraud Using Ensemble Learning. In *IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology (JEEIT)*, 2019.
[4] G. Ateniese, J. Camenisch, S. Hohenberger, and B. de Medeiros. Practical Group Signatures without Random Oracles. *IACR Cryptol. ePrint Arch.*, 2005.
[5] Aura. Intelligent Digital Safety for the Whole Family. https://www.aura.com.
[6] M. A. Azad, S. Bag, S. Tabassum, and F. Hao. Privy: Privacy preserving collaboration across multiple service providers to combat telecom spams. *IEEE Transactions on Emerging Topics in Computing*, 8(2):313–327, 2017.
[7] M. A. Azad and R. Morla. Rapid detection of spammers through collaborative information sharing across multiple service providers. *Future Generation Computer Systems*, 95:841–854, 2019.
[8] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.
[9] C. Baum, T. Frederiksen, J. Hesse, A. Lehmann, and A. Yanai. Pesto: proactively secure distributed single sign-on, or how to trust a hacked server. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2020.
[10] A. Belenky and N. Ansari. IP traceback with deterministic packet marking. *IEEE Communications Letters*, 7(4):162–164, 2003.
[11] J. Bercu. Industry traceback 2023 and beyond. In *SIPNOC*, 2023.
[12] G. Bianconi and A.-L. Barabási. Competition and multiscaling in evolving networks. *Europhysics Letters*, 54(4):436, 2001.
[13] H. K. Bokharaei, A. Sahraei, Y. Ganjali, R. Keralapura, and A. Nucci. You can SPIT, but you can't hide: Spammer identification in telephony networks. In *Proceedings IEEE INFOCOM*, 2011.

[14] D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In *Annual International Cryptology Conference*. Springer, 2004.

[15] D. Boneh, S. Gorbunov, H. Wee, and Z. Zhang. BLS signature scheme. Technical report, Technical Report draft-boneh-bls-signature-00, IETF, 2019.

[16] J. Bootle, A. Cerulli, P. Chaidos, E. Ghadafi, and J. Groth. Foundations of fully dynamic group signatures. In *International Conference on Applied Cryptography and Network Security*, pages 117–136. Springer, 2016.

[17] J. Burns, D. Moore, K. Ray, R. Speers, and B. Vohaska. Ec-oprf: Oblivious pseudorandom functions using elliptic curves. *Cryptology ePrint Archive*, 2017.

[18] J. Camenisch, M. Drijvers, A. Lehmann, G. Neven, and P. Towa. Short threshold dynamic group signatures. In *International Conference on Security and Cryptography for Networks*, pages 401–423. Springer, 2020.

[19] R. Canetti. A new paradigm for cryptographic protocols. *In Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 122–133, 2001.

[20] S. Casacuberta, J. Hesse, and A. Lehmann. Sok: Oblivious pseudorandom functions. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2022.

[21] Y. J. Choi and S. J. Kim. An improvement on privacy and authentication in GSM. In *Information Security Applications: 5th International Workshop, WISA 2004, Jeju Island, Korea, August 23-25, 2004, Revised Selected Papers 5*. Springer, 2005.

[22] I. Dacosta and P. Traynor. Proxychain: Developing a Robust and Efficient Authentication Infrastructure for Carrier-Scale VoIP Networks. In *USENIX Annual Technical Conference*, 2010.

[23] R. Dantu and P. Kolan. Detecting Spam in VoIP Networks. *SRUTI*, 5:5–5, 2005.

[24] N. Döttling, L. Hanzlik, B. Magri, and S. Wohnig. Mcfly: Verifiable encryption to the future made practical. *IACR Cryptol. ePrint Arch.*, page 433, 2022.

[25] FCC. Robocall Mitigation Database . https://fccprod.servicenowservices.com/rmd?id=rmd_welcome.

[26] FCC. FCC Fines Telemarketer $225 Million for Spoofed Robocalls.

[27] FCC. Telephone Robocall Abuse Criminal Enforcement and Deterrence Act. https://www.fcc.gov/TRACEDAct.

[28] FCC. Rules and Regulations Implementing the Telephone Consumer Protection Act (TCPA) of 1991, 2012.

[29] FCC. The Pallone-Thune Telephone Robocall Abuse Criminal Enforcement and Deterrence (TRACED) Act, 2021.

[30] FCC. Report to Congress on Robocalls And Transmission Of Misleading or Inaccurate Caller Identification Information, Dec 2022.

[31] E. F. Foundation. NSA Spying. https://www.eff.org/nsa-spying.

[32] FTC. Complying with the Telemarketing Sales Rule . https://www.ftc.gov/business-guidance/resources/complying-telemarketing-sales-rule.

[33] E. Ghadafi. Efficient distributed tag-based encryption and its application to group signatures with efficient distributed traceability. In *International Conference on Cryptology and Information Security in Latin America*. Springer, 2014.

[34] C. Gong and K. Sarac. IP traceback based on packet marking and logging. In *IEEE International Conference on Communications*, 2005.

[35] M. T. Goodrich. Probabilistic Packet Marking for Large-Scale IP Traceback. *IEEE/ACM Transactions on Networking*, 16(1):15–24, 2008.

[36] S. M. Gowri, G. S. Ramana, M. S. Ranjani, and T. Tharani. Detection of Telephony Spam and Scams using Recurrent Neural Network (RNN) Algorithm. In *International Conference on Advanced Computing and Communication Systems (ICACCS)*, volume 1. IEEE, 2021.

[37] P. Gupta, B. Srinivasan, V. Balasubramaniyan, and M. Ahamad. Phoneypot: Data-driven understanding of telephony threats. In *NDSS*, 2015.

[38] S. Heuser, B. Reaves, P. K. Pendyala, H. Carter, A. Dmitrienko, W. Enck, N. Kiyavash, A.-R. Sadeghi, and P. Traynor. Phonion: Practical Protection of Metadata in Telephony Networks. *Proc. Priv. Enhancing Technol.*, 2017(1):170–187.

[39] Hiya. Caller ID, Call Blocker & Protection. https://www.hiya.com.

[40] H.-M. Hsu, Y. S. Sun, and M. C. Chen. Collaborative scheme for VoIP traceback. *Digital Investigation*, 7(3-4), 2011.

[41] IBM. Libgroupsig. https://github.com/IBM/libgroupsig/wiki.

[42] ITG. Industry Traceback group. https://tracebacks.org/for-providers, 2015.

[43] S. Jarecki, A. Kiayias, H. Krawczyk, and J. Xu. Toppss: cost-minimal password-protected secret sharing based on threshold oprf. In *Applied Cryptography and Network Security: 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings 15*, pages 39–58. Springer, 2017.

[44] K. Jensen, T. Van Do, H. T. Nguyen, and A. Arnes. Better protection of SS7 networks with machine learning. In *International Conference on IT Convergence and Security (ICITCS)*. IEEE, 2016.

[45] H. Li, X. Xu, C. Liu, T. Ren, K. Wu, X. Cao, W. Zhang, Y. Yu, and D. Song. A machine learning approach to prevent malicious calls over telephony networks. In *IEEE Symposium on Security and Privacy*. IEEE, 2018.

[46] J. Liu, B. Rahbarinia, R. Perdisci, H. Du, and L. Su. Augmenting telephone spam blacklists by mining large CDR datasets. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, 2018.

[47] V. Madathil, S. A. K. Thyagarajan, D. Vasilopoulos, L. Fournier, G. Malavolta, and P. Moreno-Sanchez. Cryptographic Oracle-based Conditional Payments. In *Network and Distributed System Security Symposium*, 2023.

[48] S. Mavoungou, G. Kaddoum, M. Taha, and G. Matar. Survey on Threats and Attacks on Mobile Networks. *IEEE Access*, 4:4543–4572, 2016.

[49] N. McInnes, E. Zaluska, and G. Wills. Analysis of a PBX toll fraud honeypot. *Int. J. Inf. Secur. Res*, 9(1):821–830, 2019.

[50] I. Murynets, M. Zabarankin, R. P. Jover, and A. Panagia. Analysis and detection of SIMbox fraud in mobility networks. In *IEEE Conference on Computer Communications*, 2014.

[51] H. Mustafa, W. Xu, A. R. Sadeghi, and S. Schulz. You can call but you can't hide: detecting caller ID spoofing attacks. In *IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 2014.

[52] C. Network. Bls signatures library. https://github.com/Chia-Network/bls-signatures, 2024.

[53] C. of Federal Regulation. 42.6 Retention of telephone toll records. https://www.ecfr.gov/current/title-47/chapter-I/subchapter-B/part-42/subject-group-ECFR738054ac73e20e0/section-42.6.

[54] B. Oh, J. Ahn, S. Bae, M. Son, Y. Lee, M. Kang, and Y. Kim. Preventing SIM Box Fraud Using Device Model Fingerprinting. In *NDSS Symposium*, 2023.

[55] S. Pandit, K. Sarker, R. Perdisci, M. Ahamad, and D. Yang. Combating Robocalls with Phone Virtual Assistant Mediated Interaction. In *USENIX Security Symposium*. USENIX Association, 2023.

[56] P. Patankar, G. Nam, G. Kesidis, and C. R. Das. Exploring Anti-Spam Models in Large Scale VoIP Systems. In *International Conference on Distributed Computing Systems*, 2008.

[57] S. Prasad, E. Bouma-Sims, A. K. Mylappan, and B. Reaves. Who's Calling? Characterizing Robocalls through Audio and Metadata Analysis. In *USENIX Security Symposium*, 2020.

[58] S. Prasad, T. Dunlap, A. Ross, and B. Reaves. Diving into Robocall Content with SnorCall. In *USENIX Security Symposium*, 2023.

[59] A. Ravi, M. Msahli, H. Qiu, G. Memmi, A. Bifet, and M. Qiu. Wangiri Fraud: Pattern Analysis and Machine-Learning-Based Detection. *IEEE Internet of Things Journal*, 2023.

[60] B. Reaves, L. Blue, H. Abdullah, L. Vargas, P. Traynor, and T. Shrimpton. AuthentiCall: Efficient Identity and Content Authentication for Phone Calls. In *USENIX Security Symposium*, 2017.

[61] B. Reaves, L. Blue, and P. Traynor. AuthLoop: End-to-End Cryptographic Authentication for Telephony over Voice Channels. In *USENIX Security Symposium*, 2016.

[62] B. Reaves, E. Shernan, A. Bates, H. Carter, and P. Traynor. Boxed out: Blocking cellular interconnect bypass fraud at the network edge. In *USENIX Security Symposium*, 2015.

[63] U. U. Rehman and A. G. Abbasi. Security analysis of VoIP architecture for identifying SIP vulnerabilities. In *International Conference on Emerging Technologies*. IEEE, 2014.

[64] N. Ruan, Z. Wei, and J. Liu. Cooperative Fraud Detection Model With Privacy-Preserving in Real CDR Datasets. *IEEE Access*, 7:115261–115272, 2019.

[65] M. Sahin and A. Francillon. Over-the-top bypass: Study of a recent telephony fraud. In *ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[66] M. Sahin and A. Francillon. Understanding and Detecting International Revenue Share Fraud. In *NDSS*, 2021.

[67] M. Sahin, A. Francillon, P. Gupta, and M. Ahamad. SoK: Fraud in Telephony Networks. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2017.

[68] I. N. Sherman, J. D. Bowers, K. McNamara Jr, J. E. Gilbert, J. Ruiz, and P. Traynor. Are You Going to Answer That? Measuring User Responses to Anti-Robocall Application Indicators. In *NDSS*, 2020.

[69] TransNexus. STIR/SHAKEN statistics from October 2023. https://transnexus.com/blog/2023/shaken-statistics-october/, Nov 2023.

[70] Truecaller. Caller ID & Call Blocking App. https://www.truecaller.com.

[71] V. S. Tseng, J.-C. Ying, C.-W. Huang, Y. Kao, and K.-T. Chen. Fraudetector: A graph-mining-based framework for fraudulent phone call detection. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2015.

[72] H. Tu, A. Doupé, Z. Zhao, and G.-J. Ahn. SoK: Everyone hates robocalls: A survey of techniques against telephone spam. In *IEEE Symposium on Security and Privacy*, 2016.

[73] H. Tu, A. Doupé, Z. Zhao, and G.-J. Ahn. Toward authenticated caller ID transmission: The need for a standardized authentication scheme in Q. 731.3 calling line identification presentation. In *2016 ITU Kaleidoscope: ICTs for a Sustainable World*. IEEE, 2016.

[74] H. Tu, A. Doupe, Z. Zhao, and G.-J. Ahn. Toward Standardization of Authenticated Caller ID Transmission. *IEEE Communications Standards Magazine*, 1(3), 2017.

[75] K. Ullah, I. Rashid, H. Afzal, M. M. W. Iqbal, Y. A. Bangash, and H. Abbas. SS7 Vulnerabilities—A Survey and Implementation of Machine Learning vs Rule Based Filtering for Detection of SS7 Network Attacks. *IEEE Communications Surveys & Tutorials*, 22(2):1337–1371, 2020.

[76] B. Welch. Exploiting the weaknesses of SS7. *Network Security*, (1):17–19, 2017.

[77] Y.-S. Wu, S. Bagchi, N. Singh, and R. Wita. Spam detection in voice-over-IP calls through semi-supervised clustering. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 307–316, 2009.

Functionality $\mathcal{F}_{\text{TraceBack}}$

Entities: $\mathcal{P} = \emptyset$ is the set of carriers, $\mathcal{M}$ is the set of malicious carriers, TA is a traceback authority
This functionality is initialized with predicate Validate which determines if a list of records has conflicts. It takes as inputs a list of hop = $(P_{i-1} \| P_i \| P_{i+1})$ and outputs a set conflicting hop (denoted $C_{\text{conflicts}}$)
Data Structures: $\mathcal{D}$ is the table of records, that stores entries of the form (record-index : P_i, call-details, hop)

Interface:

- **Register**: Upon receiving (REGISTER, P_i) from a party P_i, do $\mathcal{P} = \mathcal{P} \cup \{P_i\}$ and send (REGISTER, P_i) to $\mathcal{A}$.
- **Contribute Call**: Upon receiving (CONTRIBUTE, call-details, hop) from some carrier P_i, check if $P_i \in \mathcal{P}$. If yes, store (record-index : P_i, call-details, hop) in $\mathcal{D}$. Send (CONTRIBUTE, record-index) to the $\mathcal{A}$. Upon receiving (CONTRIBUTE, OK) from $\mathcal{A}$, send (CONTRIBUTE, record-index) to P_i. Update record-index = record-index + 1.
- **Malicious Update to Database**: Upon receiving (MAL-UPDATE, (del, record-index)) from $\mathcal{A}$, update $\mathcal{D} \setminus \{$(record-index : $(\cdot)$)$\}$. Upon receiving (MAL-UPDATE, (add, $(\cdot)$: $(P_j$, call-details, hop))) from $\mathcal{A}$, update $\mathcal{D} \cup \{$(record-index : $(P_j$, call-details, hop))$\}$ and send record-index to $\mathcal{A}$.
- **Trace Call**: Upon receiving (TRACE, call-details, P_i) from P_i,
  (1) Retrieve all entries of the form (record-index : call-details*, $\cdot$, $\cdot$) in $\mathcal{D}$ where the ts^* of call-details* belongs to $[ts - max, ts + max]$, where max is the max size for a call and ts is the timestamp in call-details. Let $C_{\text{trace}} = \{$(record-index, hop)$\}$ for each retrieved hop
  (2) If there are no hop missing in C_{trace} send the set of retrieved ($\{$record-index$\}$) to $\mathcal{A}$. If there are hop missing, then send (call-details, $\{$record-index$\}$) to $\mathcal{A}$. Receive ($\{$MAL-UPDATE, (add, record-index$_{\mathcal{A}}$: P_j, call-details$_{\mathcal{A}}$, hop$_{\mathcal{A}}$)$\}$) or (MAL-UPDATE, (del, record-index)). If $P_j \in \mathcal{M}$, add these entries to $\mathcal{D}$, else ignore these messages. Let $C_{\text{trace}} = C_{\text{trace}} \cup C_{\text{malicious}}$, where $C_{\text{malicious}} = \{$record-index$_{\mathcal{A}}$, hop$_{\mathcal{A}}\}$.
  (3) Send (ALLOW-TRACE, P_i) to TA. Upon receiving (ALLOW-TRACE, P_i, OK) from TA, send C_{trace} to P_i else send $\emptyset$ to P_i
- **Open**: Upon receiving (OPEN, call-details, C_{trace}) from some party P_i, if TA is corrupt, send (OPEN, call-details, C_{trace}) to $\mathcal{A}$, and receive (OPEN, $\{P^*\}$). Else:
  (1) Compute $C_{\text{conflicts}} = \text{Validate}(C_{\text{trace}})$
  (2) For each hop$^* \in C_{\text{conflicts}}$ retrieve $P*$ such that $(P^*$, call-details, hop, $\cdot)$ in $\mathcal{D}$
  Return (OPEN, $\{P^*\}$) to the calling P_i

Figure 8: Private Traceback functionality

A Ideal functionality for Jäger

In this section we formalize the security properties of Jäger in the UC framework [19]. We define an ideal functionality $\mathcal{F}_{\text{TraceBack}}$ (Figure 8) that captures the correctness and the security properties of the Jäger system.

The $\mathcal{F}_{\text{TraceBack}}$ functionality maintains a database $\mathcal{D}$ and provides the following interface:

- REGISTER: Enables carriers to register with the system. Since this is public information, the identity of the carrier is leaked to the adversary.
- CONTRIBUTE: Allows carriers to submit records to the system. Recall that in the real world, an adversary can always learn when a record is submitted but does not learn the contents of the record, nor the identity of the carrier that submits the record. Therefore, the only information that is leaked to the adversary is the record-index which gives an indication of what time a call record was submitted. This captures the *anonymity* and the *confidentiality* guarantees.
- MAL-UPDATE: Allows the adversary to delete or add records to the database.
- TRACE: Enables carriers to retrieve the hops relevant to a specific call. All the hops that are currently in the database along with any hop that the adversary wants to append are returned to the carrier. Before sending these hops to the carrier, the functionality sends ALLOW-TRACE command to the TA, and only if the TA responds with (ALLOW-TRACE, OK) are these hops sent to the carrier. This captures the *trace authorization* requirement and ensures that a carrier cannot request a trace too many times and hence *rate-limits* their requests.
- OPEN: Allows a carrier to deanonymize the sender of hops that are malformed or conflicting. If the TA is honest, the functionality runs a Validate predicate to determine the malicious/conflicting hops and returns the identities of the corresponding carriers. On the other hand if the TA is malicious, the adversary is allowed to return the identities of any of the carriers. This captures the property that *accountability* is guaranteed as long as the TA is honest.

B The Jäger protocol

We need the following ingredients:

(1) Group signatures as defined in [16]
(2) An OPRF scheme
(3) Witness Encryption scheme for signatures.

In the presentation of Jäger below we will instantiate the OPRF

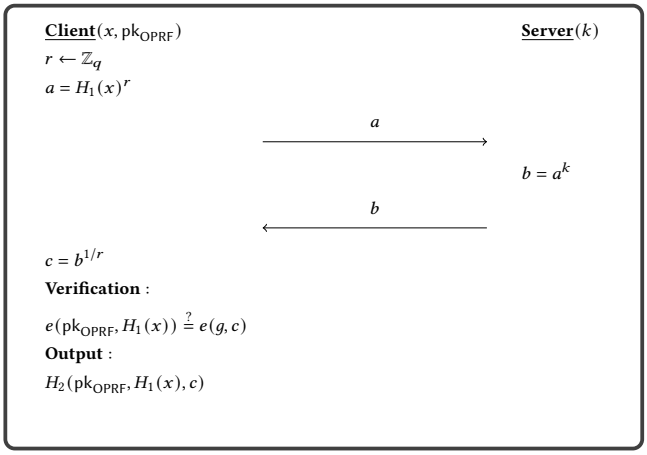

Figure 9: OPRF Scheme of [43] where $\mathsf{pk}_{\mathsf{OPRF}} = g^k$ and is previously announced by the server. All groups are pairing-friendly.

$\mathsf{Enc}((\mathsf{vk}_T, \text{call-label}), m)$: The encryption algorithm proceeds as follows:

- Sample $r_1 \leftarrow \mathbb{Z}_q$ and $r_2 \leftarrow \mathbb{G}_T$.
- Set $c_1 := g_0^{r_1}$
- Compute $h := H_\alpha(r_2)$.
- Compute $c_2 := (e(\mathsf{vk}_T, H_\beta(\text{call-label}))^{r_1} \cdot r_2)$ and $c_3 := (h + m)$
- Return $c := (c_1, c_2, c_3)$.

$\mathsf{Dec}(\sigma_T, c)$: The decryption algorithm proceeds as follows:

- Parse $c := (c_1, c_2, c_3)$.
- Compute $r := c_2 \cdot e(c_1, \sigma_T)^{-1}$.
- Compute $h := H_\alpha(r)$.
- Return $m := c_3 - h$.

Figure 10: Witness encryption based on BLS signatures from [47], here H_α and H_β are hash functions modeled as random oracles.

scheme using the 2HashDH OPRF scheme of Jarecki et. al. [43] and the witness encryption scheme presented in [47][24].

Jäger consists of four protocols: *Setup*, *Contribution*, *Trace*, and *Provider Accountability* protocols, which we describe in detail below.

Before we describe the protocol we present details of the OPRF scheme of [43] in Figure 9 and the Witness Encryption scheme of [47] in Figure 10.

B.1 Setup

The Traceback Authority TA does:

(1) Generate BLS signature keys $(\mathsf{sk}_T, \mathsf{vk}_T) \leftarrow \mathsf{Sign.KGen}(1^\lambda)$ and $(\mathsf{sk}_R, \mathsf{vk}_R) \leftarrow \mathsf{Sign.KGen}(1^\lambda)$
(2) Generate OPRF key $k \leftarrow \mathbb{Z}_q$ and announce the corresponding public key $\mathsf{pk}_{\mathsf{OPRF}} = g^k$
(3) Run the **GKGen** algorithm of the group signature scheme and announce $(\mathsf{gpk}, \mathsf{info}_0)$ which are the group manager public key and the initial group information.
(4) The TA initializes a counter rl-ctr_i corresponding to each P_i.

Each provider P_i does:

(1) Run the interactive joining protocol **Join** with TA and receive gsk_i
(2) Generate signing keys $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Sign.KGen}(1^\lambda)$ and announce vk_i.

The Record Store RS does:

(1) Initialize the database $\mathcal{D}$
(2) Generate signing keys $(\mathsf{sk}_{RS}, \mathsf{vk}_{RS}) \leftarrow \mathsf{Sign.KGen}(1^\lambda)$ and announce vk_{RS}

B.2 Contribution

Each provider P_i with input $(src \| dst \| ts)$ and $\mathsf{hop} = (P_{i-1} \| P_i \| P_{i+1})$ does:

(1) Compute $\mathsf{ep} = \mathsf{tsToEpoch}(ts)$ and run the OPRF protocol with TA using inputs $\text{call-details} = (src \| dst \| \mathsf{ep})$ as follows:
  (a) Pick $r \leftarrow \mathbb{Z}_q$ and compute $a = H_1(\text{call-details})^r$ and send a to TA
  (b) The TA computes $b = a^k$ and sends it back to P_i
  (c) P_i checks the following pairing equation to verify the OPRF was evaluated correctly: $e(\mathsf{pk}_{\mathsf{OPRF}}, H_1(\text{call-details})) = e(g, b^{1/r})$[6].
  (d) Output $\text{call-label} = H_2(\mathsf{pk}_{\mathsf{OPRF}}, \text{call-details}, b^{1/r})$
(2) The provider then encrypts the hop as follows:
  (a) Sample a random key $\mathsf{key} \leftarrow \{0,1\}^\lambda$
  (b) Encrypt key as as $\mathsf{ct}_1 = \mathsf{WE.Enc}((\mathsf{vk}_T, \text{call-label}), \mathsf{key})$
  (c) Compute $\mathsf{ct}_2 = H_3(\text{call-details} \| \mathsf{key}) \oplus \mathsf{hop}$
(3) The provider signs the ciphertexts and the $H(\text{call-label})$ using the group signature scheme: $\sigma = \mathbf{Sign}(\mathsf{gsk}_i, (\mathsf{ct}_1, \mathsf{ct}_2, H(\text{call-label})))$ and sends $(H(\text{call-label}), (\mathsf{ct}_1, \mathsf{ct}), \sigma)$ to RS

The Record Store RS upon receiving $(H(\text{call-label}), (\mathsf{ct}_1, \mathsf{ct}), \sigma)$ does the following:

(1) Check $\mathbf{Verify}(\mathsf{gpk}, (H(\text{call-label}), (\mathsf{ct}_1, \mathsf{ct})), \sigma) = 1$.
(2) If yes, write $(H(\text{call-label}), (\mathsf{ct}_1, \mathsf{ct}), \sigma)$ to the database. Else ignore the message.

B.3 Trace

Each provider P_i with input $(src \| dst \| ts)$ does:

(1) For $ts* \in [ts - t_{\max}, ts + t_{\max}]$ compute $\mathsf{ep}^* = \mathsf{tsToEpoch}(ts^*)$
(2) For each $\mathsf{ep}*$ compute call-label^* as above with inputs $\text{call-details}^* = (src \| dst \| \mathsf{ep}^*)$ by running the OPRF protocol with the TA
(3) Request a signature on $\mathsf{idx} = H(\text{call-label}^*)$ from TA.
(4) The TA checks if $\text{rl-ctr}_i > T$, if yes, reject the request, else the TA computes $\sigma_R = \mathsf{Sign}(\mathsf{sk}_R, \mathsf{idx})$ and sends it back to P_i.
(5) Send $(\mathsf{idx}^*, \sigma_R)$ to RS and receive the corresponding records back $((\mathsf{idx}^*, \mathsf{ct}_1^*, \mathsf{ct}_2^*, \sigma^*), \sigma_{RS})$ if they exist.
(6) Verify $\mathsf{Vf}(\mathsf{vk}_{RS}, ((\mathsf{idx}^*, \mathsf{ct}_1^*, \mathsf{ct}_2^*, \sigma^*), \sigma_{RS})) = 1$. If not, reject.

[6]The groups are pairing friendly, and hence we can verify the correctness of the OPRF via pairing equations [43]

(7) Request a signature on each call-label* from TA by sending $\sigma_i = \mathsf{Sign}(\mathsf{sk}_i, \text{call-label}^*)$ to the TA. The TA compute $\sigma_T = \mathsf{Sign}(\mathsf{sk}_T, \text{call-label}^*)$ and sends it to P_i.
(8) Decrypt the ciphertexts as follows:
  (a) Compute $\text{key}^* = \mathsf{WE.Dec}(\sigma_T, \mathsf{ct}_1^*)$
  (b) Compute $\text{hop}^* = H_3(\text{call-details}^* \| \text{key}^*) \oplus \mathsf{ct}_2$
(9) Append hop^* to C_{trace}

B.4 Provider Accountability

If a hop^* is malformed or conflicts with another record, the provider P_i can request the TA to open the corresponding signature to deanonymize that provider and hold them accountable.

(1) P_i sends call-details, call-label*, $\mathsf{ct}_1^*, \mathsf{ct}_2^*, \sigma^*$), the list of {hop} to the TA
(2) The TA runs Validate algorithm to determine if the provider is misbehaving and needs to be deanonymized.
(3) The TA computes $P_j^* = \textbf{Open}(\text{gsk}, (\text{call-label}^*, \mathsf{ct}_1^*, \mathsf{ct}_2^*, \sigma^*))$ and returns P_j^*

C Formal Proofs of Security of Jäger

In this section we will present the formal proofs of security. We will consider three cases of corruption: (1) Only a subset of the providers are corrupt (2) The record store RS is corrupt and can collude with any of the providers and (3) the traceback authority TA is corrupt and can collude with any of the providers.

For completeness we first show a simulator where no parties are corrupt.

Case 0: No entities are corrupt:

Setup: Simulate the Traceback Authority:

(1) Generate BLS signature keys $(\mathsf{sk}_T, \mathsf{vk}_T) \leftarrow \mathsf{Sign.KGen}(1^\lambda)$ and $(\mathsf{sk}_R, \mathsf{vk}_R) \leftarrow \mathsf{Sign.KGen}(1^\lambda)$
(2) Generate OPRF key $k \leftarrow \mathbb{Z}_q$ and announce the corresponding public key $\mathsf{pk}_{\mathsf{OPRF}} = g^k$
(3) Run the **GKGen** algorithm of the group signature scheme and announce $(\mathsf{gpk}, \mathsf{info}_0)$ which are the group manager public key and the initial group information.

Simulate honest providers: Upon receiving $(\texttt{REGISTER}, P_i)$ from $\mathcal{F}_{\mathsf{TraceBack}}$: Generate signing keys $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Sign.KGen}(1^\lambda)$ and announce vk_i.

Contribution:

Simulating honest contributions: Upon receiving (CONTRIBUTE, record-index) from $\mathcal{F}_{\mathsf{TraceBack}}$, just store (record-index, $(\cdot)$) in a database $\mathcal{D}$ and return (CONTRIBUTE, OK) to $\mathcal{F}_{\mathsf{TraceBack}}$.

Trace:

Honest trace request:

(1) Upon receiving (TRACE, record-index) from $\mathcal{F}_{\mathsf{TraceBack}}$, send $\emptyset$ back to $\mathcal{F}_{\mathsf{TraceBack}}$
(2) Upon receiving (ALLOW-TRACE, P_i) from $\mathcal{F}_{\mathsf{TraceBack}}$, send (ALLOW-TRACE, P_i, OK) back to $\mathcal{F}_{\mathsf{TraceBack}}$.

C.1 Case 1: Only a subset of the providers are corrupt and the TA and RS are honest

To prove UC security we need to show that there exists a simulator that produces a transcript in the ideal world that is indistinguishable from the real world. Below we present the simulator for the case when only a subset of the providers are malicious and all other entities are honest.

Setup: Simulate the Traceback Authority:

(1) Generate BLS signature keys $(\mathsf{sk}_T, \mathsf{vk}_T) \leftarrow \mathsf{Sign.KGen}(1^\lambda)$ and $(\mathsf{sk}_R, \mathsf{vk}_R) \leftarrow \mathsf{Sign.KGen}(1^\lambda)$
(2) Generate OPRF key $k \leftarrow \mathbb{Z}_q$ and announce the corresponding public key $\mathsf{pk}_{\mathsf{OPRF}} = g^k$
(3) Run the **GKGen** algorithm of the group signature scheme and announce $(\mathsf{gpk}, \mathsf{info}_0)$ which are the group manager public key and the initial group information.

Simulate honest providers: Upon receiving $(\texttt{REGISTER}, P_i)$ from $\mathcal{F}_{\mathsf{TraceBack}}$: Generate signing keys $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Sign.KGen}(1^\lambda)$ and announce vk_i.

Malicious provider P_j: Simulate the interactive joining protocol **Join** with P_j, and send $(\texttt{REGISTER}, P_i)$ to $\mathcal{F}_{\mathsf{TraceBack}}$.

Contribution:

Simulating honest contributions: The simulator does not need to simulate interactions between honest providers and RS and TA, since this is not in the view of the malicious providers. Therefore, upon receiving (CONTRIBUTE, record-index) from $\mathcal{F}_{\mathsf{TraceBack}}$, just store (record-index, $(\cdot)$) in a database $\mathcal{D}$ and return (CONTRIBUTE, OK) to $\mathcal{F}_{\mathsf{TraceBack}}$.

Simulating random oracle invocations:

(1) Upon receiving input x for random oracle H_1, check if (x, y) exists in Q_1. If yes return y, else sample a random y, store $(x, y) \in Q_1$ and return y.
(2) Upon receiving input x for random oracle H_2, check if (x, y) exists in Q_2. If yes return y, else sample a random y, store $(x, y) \in Q_2$ and return y.
(3) Upon receiving input x for random oracle H_3, check if (x, y) exists in Q_3. If yes return y, else sample a random y, store $(x, y) \in Q_3$ and return y.

Simulating malicious contributions:

(1) Upon receiving a on behalf of TA from the adversary, compute $b = a^k$ and send it back to the $\mathcal{A}$.
(2) Upon receiving $(\text{call-label}, \mathsf{ct}_1, \mathsf{ct}_2, \sigma)$ from $\mathcal{A}$:
  (a) Check $(x, \text{call-label})$ exists in Q_2. If not, abort with ROFail_2.
  (b) Else parse x as $(\mathsf{pk}_{\mathsf{OPRF}}, \text{call-details}, b^*)$
  (c) Check that $(\text{call-details}, y)$ exists in Q_1. If not, abort with ROFail_1.
  (d) Check that $e(\mathsf{pk}_{\mathsf{OPRF}}, y) = e(g, b^*)$. If not, abort with PRFFail
  (e) Check $((\text{call-details} \| \text{key}^*), z)$ exists in Q_3. If not, abort with ROFail_3.
  (f) Else compute $\text{hop}^* = z \oplus \mathsf{ct}_2$
(3) If all checks pass, compute $P_j^* = \textbf{Open}(\mathsf{gsk}_i, (\text{call-label}, \mathsf{ct}_1, \mathsf{ct}_2, \sigma))$. If P_j^* corresponds to that of an honest party, abort with GroupSigFail.
(4) Send (CONTRIBUTE, call-details, hop) on behalf of P_j^* to $\mathcal{F}_{\mathsf{TraceBack}}$.
(5) Receive (CONTRIBUTE, record-index) from $\mathcal{F}_{\mathsf{TraceBack}}$ and store (record-index, $(P_j$, call-details, hop)) in $\mathcal{D}$.

Trace: We consider two cases: 1) an honest trace request 2) a malicious trace request

(1) Honest trace request:
  (a) Upon receiving (TRACE, record-index) from $\mathcal{F}_{\text{TraceBack}}$, send $\emptyset$ back to $\mathcal{F}_{\text{TraceBack}}$
  (b) Upon receiving (ALLOW-TRACE, P_i) from $\mathcal{F}_{\text{TraceBack}}$, send (ALLOW-TRACE, P_i, OK) back to $\mathcal{F}_{\text{TraceBack}}$.

(2) Malicious trace request:
  (a) Upon receiving (call-label*, σ^*) from $\mathcal{A}$ (on behalf of P_j^*):
    (i) if σ^* verifies under a honest party P_i's vk abort with SigFail
    (ii) If call-label*, $(\cdot)$ exists in the list of ciphertexts, send (call-label*, $(\text{ct}_1, \text{ct}_2, \sigma)$) to $\mathcal{A}$. Else:
    (iii) Check $(x$, call-label) exists in Q_2. If not, abort with ROFail_2.
    (iv) Else parse x as $(\text{pk}_{\text{OPRF}}$, call-details, $b^*)$
    (v) Check that (call-details, y) exists in Q_1. If not, abort with ROFail_1.
    (vi) Check that $e(\text{pk}_{\text{OPRF}}, y) = e(g, b^*)$. If not, abort with PRFFail
  (b) Send (TRACE, call-details, P_j^*) on behalf of P_j^* to $\mathcal{F}_{\text{TraceBack}}$
  (c) Receive ({record-index}) from $\mathcal{F}_{\text{TraceBack}}$. If any of the record-index correspond to that of a malicious contribution, send {record-index, P_j, call-details, hop} to $\mathcal{F}_{\text{TraceBack}}$.
  (d) Upon receiving ALLOW-TRACE from $\mathcal{F}_{\text{TraceBack}}$, check that requesting provider has not requested too many records and send (ALLOW-TRACE, OK) to $\mathcal{F}_{\text{TraceBack}}$.
  (e) Receive C_{trace} from $\mathcal{F}_{\text{TraceBack}}$.
  (f) Encrypt each of the hop using the corresponding call-label and vk_T in the following way:
    (i) Sample a random key key $\leftarrow \{0,1\}^\lambda$
    (ii) Compute $\text{ct}_1 = \text{WE.Enc}((\text{call-label}, \text{vk}_T), \text{key})$
    (iii) Sample a random ct_2. Set $H_3(\text{call-details}\|\text{key}) = \text{ct}_2 \oplus \text{hop}$
  (g) Using the group signature scheme, compute σ on behalf of P_i, where P_i is the honest sender of the record.
  (h) Compute $\sigma_{RS} = \text{Sign}((\text{call-label}^*, \text{ct}_1^*, \text{ct}_2^*, \sigma^*))$
  (i) Send the $(\text{call-label}^*, \text{ct}_1^*, \text{ct}_2^*, \sigma^*), \sigma_{RS}$ that correspond to call-label* to the adversary.

Provider Accountability: Upon receiving **Open** request from $\mathcal{A}$ for a particular record, $(\text{call-label}^*, \text{ct}_1^*, \text{ct}_2^*, \sigma^*), \sigma_{RS}$, check that σ_{RS} is a signature that was computed by the simulator, if not abort with SigFail_2. Else send OPEN, call-details, C_{trace} to $\mathcal{F}_{\text{TraceBack}}$ and output whatever $\mathcal{F}_{\text{TraceBack}}$ returns.

Proof By Hybrids: Now to prove that the simulated world and the real world are indistinguishable we proceed via a sequence of hybrids, starting from the real world until we reach the ideal world. We show that each of these hybrids are indistinguishable and therefore the real world and the simulated world are indistinguishable.

Hybrid$_0$ This is the real world protocol

Hybrid$_1$ This hybrid is identical to the previous hybrid except that the simulator may abort with GroupSigFail. By the non-frameability property of the group signature scheme, the simulator aborts with negligible probability and therefore this hybrid is indistinguishable from the previous one.

Hybrid$_2$ This hybrid is identical to the previous hybrid except that the simulator may abort with ROFail_1. Since we use a random oracle and require the adversary to use the RO, the probability of this event occurring is negligible, and therefore this hybrid is indistinguishable from the previous one.

Hybrid$_3$ This hybrid is identical to the previous hybrid except that the simulator may abort with ROFail_2.Since we use a random oracle and require the adversary to use the RO, the probability of this event occurring is negligible, and therefore this hybrid is indistinguishable from the previous one.

Hybrid$_4$ This hybrid is identical to the previous hybrid except that the simulator may abort with ROFail_3. Since we use a random oracle and require the adversary to use the RO, the probability of this event occurring is negligible, and therefore this hybrid is indistinguishable from the previous one.

Hybrid$_5$ This hybrid is identical to the previous hybrid except that the simulator may abort with SigFail. Since we use unforgeable signatures, this event occurs with negligible probability and therefore the two hybrids are indistinguishable.

Hybrid$_6$ This hybrid is identical to the previous hybrid except that the simulator may abort with SigFail_2. Since we use unforgeable signatures, this event occurs with negligible probability and therefore the two hybrids are indistinguishable.

Since this hybrid is identical to the simulated world, we have shown that the real world and ideal world are indistinguishable, and that concludes the proof of security for the case when only a subset of carriers are corrupt.

C.2 Case 2: TA and subset of providers are corrupt, and RS is honest

We present the simulator for this case below:

Setup: The adversary $\mathcal{A}$ runs the algorithms of the TA. Receive $\text{vk}_T, \text{pk}_{\text{OPRF}}, (\text{gpk}, \text{info}_0)$ from $\mathcal{A}$

Simulate honest providers: Upon receiving (REGISTER, P_i) from $\mathcal{F}_{\text{TraceBack}}$:

(1) Generate signing keys $(\text{sk}_i, \text{vk}_i) \leftarrow \text{Sign.KGen}(1^\lambda)$ and announce vk_i.
(2) Interact with $\mathcal{A}$ to run the **Join** protocol and learn gsk_i.

Malicious provider P_j: Upon receiving (**UpdateGroup**, P_j^*) from $\mathcal{A}$ on behalf of TA, send (REGISTER, P_j^*) to $\mathcal{F}_{\text{TraceBack}}$.

Contribution:

Simulating honest contributions: Note that the simulator only simulates the computation of the label, since that is the only interaction with the adversary. Since the RS is honest the simulator does not need to compute the ciphertexts or interact with $\mathcal{A}$. Therefore, upon receiving (CONTRIBUTE, record-index) from $\mathcal{F}_{\text{TraceBack}}$:

(1) Compute a label as follows:
  (a) Sample random $r \leftarrow \mathbb{Z}_q$ and compute $a = H_1(0)^r$ and send a to $\mathcal{A}$ on behalf of TA.
  (b) Receive b from $\mathcal{A}$.
  (c) Check that $e(\text{pk}_{\text{OPRF}}, H_1(0)) = e(g, b^{1/r})$
  (d) Output call-label $= H_2(\text{pk}_{\text{OPRF}}, 0, b^{1/r})$ and store (record-index, call-labe

Simulating random oracle invocations:

(1) Upon receiving input x for random oracle H_1, check if (x, y) exists in Q_1. If yes return y, else sample a random y, store $(x, y) \in Q_1$ and return y.

(2) Upon receiving input x for random oracle H_2, check if (x, y) exists in Q_2. If yes return y, else sample a random y, store $(x, y) \in Q_2$ and return y.
(3) Upon receiving input x for random oracle H_3, check if (x, y) exists in Q_3. If yes return y, else sample a random y, store $(x, y) \in Q_3$ and return y.

Simulating malicious contributions:

(1) Upon receiving (call-label, $\mathsf{ct}_1, \mathsf{ct}_2, \sigma$) from $\mathcal{A}$:
  (a) Check (x, call-label) exists in Q_2. If not, abort with ROFail_2.
  (b) Else parse x as ($\mathsf{pk}_{\mathsf{OPRF}}$, call-details, b^*)
  (c) Check that (call-details, y) exists in Q_1. If not, abort with ROFail_1.
  (d) Check that $e(\mathsf{pk}_{\mathsf{OPRF}}, y) = e(g, b^*)$. If not, abort with PRFFail
  (e) Check $((\text{call-details}\|\mathsf{key}^*), z)$ exists in Q_3. If not, abort with ROFail_3.
  (f) Else compute $\mathsf{hop}^* = z \oplus \mathsf{ct}_2$
(2) Send (CONTRIBUTE, call-details, hop) on behalf of $\mathcal{A}$ to $\mathcal{F}_{\mathsf{TraceBack}}$.
(3) Receive (CONTRIBUTE, record-index) from $\mathcal{F}_{\mathsf{TraceBack}}$ and store (record-index, ($\mathcal{A}$, call-details, hop)) in $\mathcal{D}$.

Trace: We consider two cases: 1) an honest trace request 2) a malicious trace request

(1) Honest trace request: Upon receiving (TRACE, record-index) from $\mathcal{F}_{\mathsf{TraceBack}}$
  (a) If (record-index, ($\mathcal{A}$, call-details, hop)) $\in \mathcal{D}$
    (i) Sample random $r \leftarrow \mathbb{Z}_q$ and compute $a = H_1(\text{call-details})^r$ and send a to $\mathcal{A}$
    (ii) Receive b from $\mathcal{A}$.
    (iii) Check that $e(\mathsf{pk}_{\mathsf{OPRF}}, H_1(\text{call-details})) = e(g, b^{1/r})$.
    (iv) Output call-label$^* = H_2(\mathsf{pk}_{\mathsf{OPRF}}, \text{call-details}, b^{1/r})$
    (v) Request $\mathcal{A}$ for a signature on call-label*.
    (vi) If no signature received send $\emptyset$ to $\mathcal{F}_{\mathsf{TraceBack}}$. And upon receiving (ALLOW-TRACE, P_i) from $\mathcal{F}_{\mathsf{TraceBack}}$, return (ALLOW-TRACE, $\perp$)
    (vii) Else Decrypt $\mathsf{ct}_1, \mathsf{ct}_2$ corresponding to call-label* if it exists and send record-index, $\mathcal{A}$, call-details, hop to $\mathcal{F}_{\mathsf{TraceBack}}$. And upon receiving (ALLOW-TRACE, P_i) from $\mathcal{F}_{\mathsf{TraceBack}}$, return (ALLOW-TRACE, OK)
    (viii) If no ciphertexts exist corresponding to this call-label* abort with failure VerifyOPRFFail.
  (b) If (record-index, ($\mathcal{A}$, call-details, hop)) $\notin \mathcal{D}$,
    (i) simulate the computation of a label as in honest contribution
    (ii) Sample a random string call-label* .
    (iii) Request $\mathcal{A}$ for a signature on call-label*. If a signature is received, send (ALLOW-TRACE, OK) to $\mathcal{F}_{\mathsf{TraceBack}}$, else send (ALLOW-TRACE, $\perp$) upon receiving (ALLOW-TRACE, P_i) from $\mathcal{F}_{\mathsf{TraceBack}}$.
(2) Malicious trace request:
  (a) Upon receiving (call-label$^*, \sigma_i, \sigma_R$) from $\mathcal{A}$ (on behalf of P_j^*):
    (i) If σ_i corresponds to that of an honest party, abort with SigFail.
    (ii) If call-label$^*, (\cdot)$ exists in the list of ciphertexts, send (call-label*, ($\mathsf{ct}_1, \mathsf{ct}_2, \sigma$)) to $\mathcal{A}$. Else:
    (iii) Check (x, call-label) exists in Q_2. If not, abort with ROFail_2.
    (iv) Else parse x as ($\mathsf{pk}_{\mathsf{OPRF}}$, call-details, b^*)
    (v) Check that (call-details, y) exists in Q_1. If not, abort with ROFail_1.
    (vi) Check that $e(\mathsf{pk}_{\mathsf{OPRF}}, y) = e(g, b^*)$. If not, abort with PRFFail
  (b) Send (TRACE, call-details, P_j^*) on behalf of P_j^* to $\mathcal{F}_{\mathsf{TraceBack}}$
  (c) Receive ({record-index}) from $\mathcal{F}_{\mathsf{TraceBack}}$. If any of the record-index correspond to that of a malicious contribution, send {record-index, P_j, call-details, hop} to $\mathcal{F}_{\mathsf{TraceBack}}$.
  (d) Upon receiving ALLOW-TRACE from $\mathcal{F}_{\mathsf{TraceBack}}$, send (ALLOW-TRACE, OK) to $\mathcal{F}_{\mathsf{TraceBack}}$.
  (e) Receive C_{trace} from $\mathcal{F}_{\mathsf{TraceBack}}$.
  (f) Encrypt each of the hop using the corresponding call-label and vk_T in the following way:
    (i) Sample a random key key $\leftarrow \{0,1\}^\lambda$
    (ii) Compute $\mathsf{ct}_1 = \mathsf{WE.Enc}((\text{call-label}, \mathsf{vk}_T), \text{key})$
    (iii) Sample a random ct_2. Set $H_3(\text{call-details}\|\text{key}) = \mathsf{ct}_2 \oplus \mathsf{hop}$
  (g) Using the group signature scheme, compute σ on behalf of P_i, where P_i is the honest sender of the record.
  (h) Compute $\sigma_{RS} = \mathsf{Sign}((\text{call-label}^*, \mathsf{ct}_1^*, \mathsf{ct}_2^*, \sigma^*))$
  (i) Send the (call-label$^*, \mathsf{ct}_1^*, \mathsf{ct}_2^*, \sigma^*$), σ_{RS} that correspond to call-label* to the adversary.

Provider Accountability: Upon receiving (OPEN, call-details, C_{trace}) request from $\mathcal{F}_{\mathsf{TraceBack}}$ send **Open** request to $\mathcal{A}$ and output whatever $\mathcal{A}$ returns.

(1) If the adversary opens a signature submitted by a malicious party as an honest party's identity abort with FrameFail.
(2) If the adversary opens a record (call-label$^*, \mathsf{ct}_1^*, \mathsf{ct}_2^*, \sigma^*$), σ_{RS} where σ_{RS} was not computed by the simulator abort with $\mathsf{SigFail}_2$.

Proof By Hybrids: Now to prove that the simulated world and the real world are indistinguishable we proceed via a sequence of hybrids, starting from the real world until we reach the ideal world. We show that each of these hybrids are indistinguishable and therefore the real world and the simulated world are indistinguishable.

Hybrid$_0$ This is the real world protocol

Hybrid$_1$ This hybrid is identical to the previous hybrid except that the simulator may abort with GroupSigFail. By the non-frameability property of the group signature scheme, the simulator aborts with negligible probability and therefore this hybrid is indistinguishable from the previous one.

Hybrid$_2$ This hybrid is identical to the previous hybrid except that the simulator may abort with ROFail_1. Since we use a random oracle and require the adversary to use the RO, the probability of this event occurring is negligible, and therefore this hybrid is indistinguishable from the previous one.

Hybrid$_3$ This hybrid is identical to the previous hybrid except that the simulator may abort with ROFail_2.Since we use a random oracle and require the adversary to use the RO, the probability of this event occurring is negligible, and therefore this hybrid is indistinguishable from the previous one.

Hybrid$_4$ This hybrid is identical to the previous hybrid except that the simulator may abort with ROFail_3. Since we use a random

oracle and require the adversary to use the RO, the probability of this event occurring is negligible, and therefore this hybrid is indistinguishable from the previous one.

Hybrid$_5$ This hybrid is identical to the previous hybrid except that the simulator may abort with SigFail. Since we use unforgeable signatures, this event occurs with negligible probability and therefore the two hybrids are indistinguishable.

Hybrid$_6$ This hybrid is identical to the previous hybrid except that the simulator may abort with VerifyOPRFFail. Since we use a verifiable OPRF scheme this occurs with negligible probability.

Hybrid$_7$ This hybrid is identical to the previous hybrid except that the simulator simulates the OPRF calls using 0 as input instead of the call-details. By the obliviousness property of the underlying OPRF scheme, these two hybrids are indistinguishable.

Hybrid$_8$ This hybrid is identical to the previous hybrid except that the simulator may abort with $\mathsf{SigFail}_2$. Since we use unforgeable signatures, this event occurs with negligible probability and therefore the two hybrids are indistinguishable.

Since this hybrid is identical to the simulated world, we have shown that the real world and ideal world are indistinguishable, and that concludes the proof of security for the case when only a subset of carriers are corrupt.

C.3 Case 3: RS and a subset of the providers are corrupt, and TA is honest

Setup: Simulate the Traceback Authority:

(1) Generate BLS signature keys $(\mathsf{sk}_T, \mathsf{vk}_T) \leftarrow \mathsf{Sign.KGen}(1^\lambda)$ and $(\mathsf{sk}_R, \mathsf{vk}_R) \leftarrow \mathsf{Sign.KGen}(1^\lambda)$
(2) Generate OPRF key $k \leftarrow \mathbb{Z}_q$ and announce the corresponding public key $\mathsf{pk}_{\mathsf{OPRF}} = g^k$
(3) Run the **GKGen** algorithm of the group signature scheme and announce $(\mathsf{gpk}, \mathsf{info}_0)$ which are the group manager public key and the initial group information.

Simulate honest providers: Upon receiving $(\mathtt{REGISTER}, P_i)$ from $\mathcal{F}_{\mathsf{TraceBack}}$: Generate signing keys $(\mathsf{sk}_i, \mathsf{vk}_i) \leftarrow \mathsf{Sign.KGen}(1^\lambda)$ and announce vk_i.

Malicious provider P_j: Simulate the interactive joining protocol **Join** with P_j, and send $(\mathtt{REGISTER}, P_i)$ to $\mathcal{F}_{\mathsf{TraceBack}}$.

Contribution:

Simulating honest contributions: Upon receiving $(\mathtt{CONTRIBUTE}, \text{record-index})$ from $\mathcal{F}_{\mathsf{TraceBack}}$,

(1) Sample a random call-label
(2) Send a random key and compute $\mathsf{ct}_1 = \mathsf{WE.Enc}((\mathsf{vk}_T, \text{call-label}), \text{key})$
(3) Sample a random string ct_2
(4) Compute a group signature σ on behalf of some honest party P_i
(5) Send $(\mathsf{ct}_1, \mathsf{ct}_2, \text{call-label}, \sigma)$ to *RS* and send $(\mathtt{CONTRIBUTE}, \mathtt{OK})$ to $\mathcal{F}_{\mathsf{TraceBack}}$. Store $(\text{record-index}, \text{call-label}, \mathsf{ct}_1, \mathsf{ct}_2)$.

Simulating random oracle invocations:

(1) Upon receiving input x for random oracle H_1, check if (x, y) exists in Q_1. If yes return y, else sample a random y, store $(x, y) \in Q_1$ and return y.
(2) Upon receiving input x for random oracle H_2, check if (x, y) exists in Q_2. If yes return y, else sample a random y, store $(x, y) \in Q_2$ and return y.
(3) Upon receiving input x for random oracle H_3, check if (x, y) exists in Q_3. If yes return y, else sample a random y, store $(x, y) \in Q_3$ and return y.

Simulating malicious contributions: Note that since the *RS* is corrupt, the simulator only needs to simulate the label generation.

(1) Upon receiving a on behalf of TA from the adversary, compute $b = a^k$ and send it back to the $\mathcal{A}$.

Trace: We consider two cases: 1) an honest trace request 2) a malicious trace request

(1) Honest trace request:
  (a) Upon receiving $(\mathtt{TRACE}, \text{record-index}, \text{call-details})$ from $\mathcal{F}_{\mathsf{TraceBack}}$,
    (i) Retrieve call-label that corresponds to record-index if it exists.
    (ii) Send $(\text{call-label}, \sigma_R)$ to *RS* and receive $(\text{call-label}, \mathsf{ct}_1, \mathsf{ct}_2, \sigma)$. If no such record was received, send $(\mathtt{MAL\text{-}UPDATE}, (\mathtt{del}, \text{record-index})$ to $\mathcal{F}_{\mathsf{TraceBack}})$
    (iii) If no call-label exists, compute $\text{call-label} = H_2(\mathsf{pk}_{\mathsf{OPRF}}, \text{call-details}, H_1(\text{call-details})^k)$ and send call-label to $\mathcal{A}$. If any $\mathsf{ct}_1, \mathsf{ct}_2$, call-label, σ received, first check if σ corresponds to that of an honest party, if this is the case abort the simulation with GroupSigFail else decrypt the ciphertexts to retrieve the hop and send $(\mathtt{MAL\text{-}UPDATE}, (\mathtt{add}, (P_j^*, \text{call-details}, \text{hop})))$ to $\mathcal{F}_{\mathsf{TraceBack}}$. Upon receiving record-index, send record-index, P_j^*, call-details, hop to $\mathcal{F}_{\mathsf{TraceBack}}$.
  (b) Upon receiving $(\mathtt{ALLOW\text{-}TRACE}, P_i)$ from $\mathcal{F}_{\mathsf{TraceBack}}$, send $(\mathtt{ALLOW\text{-}TRACE}, P_i, \mathtt{OK})$ back to $\mathcal{F}_{\mathsf{TraceBack}}$.
(2) Malicious trace request:
  (a) Upon receiving a query for H_3 on call-details$\|$key, send $(\mathtt{TRACE}, \text{call-details}, P_i)$ to $\mathcal{F}_{\mathsf{TraceBack}}$ and receive back $C_{\text{trace}} = \{\text{record-index}, \text{hop}\}$.
  (b) Retrieve $(\mathsf{ct}_1, \mathsf{ct}_2, \text{call-label}, \sigma)$ that correspond to record-index and check that this is the same key that was encrypted in ct_1. If yes, compute $z = \mathsf{ct}_2 \oplus \text{hop}$ and send z in response and store $((\text{call-details}\|\text{key}), z)$ in Q_3. If not, sample a random $z \leftarrow \{0,1\}^\lambda$ and send z in response.

Provider Accountability: Upon receiving **Open** request from $\mathcal{A}$ for a particular record, send $\mathtt{OPEN}$, call-details, C_{trace} to $\mathcal{F}_{\mathsf{TraceBack}}$ and output whatever $\mathcal{F}_{\mathsf{TraceBack}}$ returns.

Proof By Hybrids: Now to prove that the simulated world and the real world are indistinguishable we proceed via a sequence of hybrids, starting from the real world until we reach the ideal world. We show that each of these hybrids are indistinguishable and therefore the real world and the simulated world are indistinguishable.

Hybrid$_0$ This is the real world protocol

Hybrid$_1$ This hybrid is identical to the previous hybrid except that the simulator may abort with GroupSigFail. By the non-frameability property of the group signature scheme, the simulator aborts with negligible probability and therefore this hybrid is indistinguishable from the previous one.

Hybrid$_2$ This hybrid is identical to the previous hybrid except that the simulator may abort with ROFail_1. Since we use a random

oracle and require the adversary to use the RO, the probability of this event occurring is negligible, and therefore this hybrid is indistinguishable from the previous one.

Hybrid$_3$ This hybrid is identical to the previous hybrid except that the simulator may abort with ROFail_2.Since we use a random oracle and require the adversary to use the RO, the probability of this event occurring is negligible, and therefore this hybrid is indistinguishable from the previous one.

Hybrid$_4$ This hybrid is identical to the previous hybrid except that the simulator may abort with ROFail_3. Since we use a random oracle and require the adversary to use the RO, the probability of this event occurring is negligible, and therefore this hybrid is indistinguishable from the previous one.

Hybrid$_5$ This hybrid is identical to the previous hybrid except that the simulator may abort with SigFail. Since we use unforgeable signatures, this event occurs with negligible probability and therefore the two hybrids are indistinguishable.

Hybrid$_6$ This hybrid is identical to the previous hybrid except that the simulator simulates the honest contributions by sampling a random call-label. By the pseudorandomness property of the underlying OPRF scheme, these two hybrids are indistinguishable.

Hybrid$_7$ This hybrid is identical to the previous hybrid except that the ciphertext ct_2 is a randomly sampled string. By the perfect security of the stream cipher (OTP), these two hybrids are indistinguishable.

Hybrid$_8$ This hybrid is identical to the previous hybrid except that the group signature corresponds to that of a random honest party. By the anonymity guarantees of the underlying group signature scheme, these two hybrids are indistinguishable.

Since this hybrid is identical to the simulated world, we have shown that the real world and ideal world are indistinguishable, and that concludes the proof of security for the case when only a subset of carriers are corrupt.

D Artifact Appendix

D.1 Abstract

We developed a prototype of the Jäger system and conducted a thorough performance evaluation. The Jäger artifact is composed of four key components: Group Membership Management, Label Generation, Trace Authorization, and Record Storage. Each of these components was containerized using Docker, and we orchestrated them together with Docker Compose. Additionally, we integrated auxiliary services, including a web-based GUI, to facilitate interaction with the database.

We generated a dataset of Call Detail Records and evaluated Jäger's performance. Our experimental results indicate that Jäger incurs minimal computational and bandwidth overhead per call, with these costs scaling linearly with the increase in call volume.

D.2 Description & Requirements

The Jäger prototype comprises four integral components:

Membership Management: The Group Manager (GM) oversees membership management. This component enables the GM to issue new group membership keys or revoke existing ones, and it also facilitates the tracing of traitors. Within our implementation, the TA assumes the role of the GM.

Label Generation: Label generation is controlled by the Label Manager (LM). The LM collaborates with providers to evaluate pseudorandom functions using the Oblivious Pseudorandom Function protocol. In our system, the TA also fulfills the role of the LM.

Trace Authorization: This component is responsible for generating authorization signatures required to decrypt ciphertexts. The TA acts as the Trace Authorizer in our implementation.

Record Storage: The Record Storage component stores ciphertexts and provides the results of match trace queries.

All the above components are implemented in Python. However, for performance optimization, we implemented the witness encryption in C++ and created Python bindings to integrate the library into our prototype.

D.2.1 Security, privacy, and ethical concerns. None

D.2.2 How to access. We have archived the witness encryption and the Jäger prototype into a zip file, which is publicly accessible on Zenodo at the following link: https://zenodo.org/doi/10.5281/zenodo.12733869. Additionally, we maintain an active version of the artifact in our GitHub repositories. The source code for Witness Encryption can be found at https://github.com/wspr-ncsu/BLS-Witness-Encryption, while the Jäger prototype is available at https://github.com/wspr-ncsu/jaeger.

D.2.3 Hardware dependencies. Running Jäger does not necessitate any specific hardware requirements. However, to achieve results comparable to those presented in the paper, our experiments were conducted on a Linux virtual machine equipped with 32 vCPUs and 64 GB of memory. The underlying host was a Super Micro Server featuring an Intel Xeon Gold 6130 processor, ECC DDR RAM, and 12 Gbps SAS drives.

D.2.4 Software dependencies. For ease of setup, we recommend configuring the project using Docker. If you prefer not to use Docker, our repositories provide detailed instructions on how to set up the project without it. For the remainder of this artifact appendix, we will focus exclusively on the setup and execution of experiments using Docker.

D.2.5 Benchmarks. None

D.3 Set Up

D.3.1 Installation. You need to install the appropriate version of Docker based on your operating system. If your Docker installation does not include the Docker Compose plugin, be sure to install Docker Compose separately. Additionally, download the Jäger prototype source code from either the GitHub repository or Zenodo.

We have published the Jäger Docker image on Docker Hub as `kofidahmed/jager`. If this image is not available, navigate to the root directory and build the Jäger Docker image using the following command:

```
docker build -t kofidahmed/jager .
```

Note that the `-t` option does not necessarily need to be `kofidahmed/jager`. We use `kofidahmed/jager` to align with the image on dockerhub.

D.3.2 Basic test. Generate secret keys for label generation, group master and public keys for group management, as well as private and public keys for BLS signatures and witness encryption by running the following command:

```
docker run \
  -v $(pwd):/app \
  --rm kofidahmed/jager \
  python keygen.py -a
```

The `-a` option instructs the script to generate all necessary keys. If you only need to generate keys for specific components, use the following options: `-lm` for label generation, `-gm` for group management, and `-ta` for trace authorization/witness encryption. After running the command, verify that the `.env` and `membership-keys.json` files have been created and that the variables within are populated with the appropriate keys.

D.4 Evaluation Workflow

D.4.1 Major claims.

(C1): The average runtime for the following operations are as follows: Label generation takes 0.073 ms, the contribution protocol takes 4.143 ms, trace authorization takes 0.419 ms, decryption takes 0.847 ms, opening a signature takes 0.147 ms, and verifying a group signature takes 2.310 ms. Experiment (E1) substantiates these performance metrics.

D.4.2 Experiments.

(E1): Benchmark Jäger operations in Table 3.

Preparation: Run the Docker Compose command to start the services, and then log in to the Docker container by executing:

```
docker compose up -d
docker exec -it jager-exp bash
```

Execution: To benchmark the operations, run the following command:

```
python benchmarks.py -a
```

This will display the results on the console and create a `results` folder inside the project root.

Results: The file `results/bench.csv` contains a summary of the benchmarks, while `results/index-timings.csv` records the individual runs. We used `results/index-timings.csv` to determine the mean, min, max, and standard deviations. To aggregate the benchmark results from `results/index-timings.csv`, as shown in Table 3 (in the paper), run `python aggregate-benchmark.py`. This script generates a CSV file `results/bench-summary.csv` with the aggregated results.

(E2): Determine Bandwidth, Storage Growth, and Query Performance as illustrated in Fig. 7.

Preparation: Data generation

(1) Run Docker Compose to start the services, and log in to the Docker container as demonstrated in E1.
(2) Generate telephone and social network data by running the command: `python datagen.py -n 100 -s 10000 -c -y`. The `-n` option specifies the number of carriers, `-s` specifies the number of subscribers, `-c` determines whether CDRs should be generated, and `-y` skips all prompts. Note that in the paper, `-n` is set to 7000 and `-s` is set to 300M.
(3) View the Generated Dataset: We will connect to the ClickHouse database using a web browser. Ensure that your browser has network access to both ports 5521 and 8123. We have added a UI service that allows you to connect to the database. Visit `http://localhost:5521` in your browser.
   – Enter `http://localhost:8123` as the ClickHouse URL, `default` as the Username, and `secret` as the Password, then click the `Submit` button. Once successful, click the `Go back to the home page` link.
   – On the home page, select `Jager` in the database field, which will load the tables. You can then click on any table to view its Details, Schema, or preview rows.
   – If you prefer to run your own SQL queries, click on the new file icon/button with the orange background. Type `select * from jager.raw_cdrs limit 10;` in the query field and click the `Run Query` button.

Execution: To run the contributions protocol, execute the command: `python run-contribution.py -b 3 -r 100`. The `-b` option specifies the number of batches, which is the number of times you wish to run the contribution experiment (it defaults to 1). After each batch, database stats are measured and stored in the `results` folder. The `-r` option is required and specifies the number of records per batch.

Results: The results are saved in `results/db_stats.csv` and `results/queries.csv`. To generate the fetch and insert query performance graph shown in Fig. 7 from the results in `results/queries.csv`, run `python plot-db-stats.py`. This creates a PNG file at `results/query_performance.png`.

(E3): Run a Traceback (Optional)

Preparation: Start the Docker services and log in to the Docker container as described in E1. Visit `http://localhost:8123` in your browser, and execute the SQL command `select src, dst, ts from jager.raw_cdrs where status = 1;`. Refer to the E2 preparation section for instructions on executing the SQL query. The results from this query are the calls whose ciphertexts have been submitted to the RS.

Execution: Run the following command: `python run-trace.py -s src -d dst -t ts`. Replace `src`, `dst`, and `ts` with the values from any row in the query results obtained in the preparation step above.

Results: This command will:

– Generate call labels within the $[ts - t_{\max}, ts + t_{\max}]$ range
– Request authorization signatures from the TA
– Retrieve ciphertexts from the RS
– Decrypt and analyze the records to determine the origin and call path.
– Display the results to your console.

D.5 Notes on Reusability

Our implementation includes Docker services defined in the `compose.yml` file. Once the compose-up command is running, the following services are exposed via `http://localhost:{PORT}`:

- The Group Management server runs on `9990`. The implementation is defined in `app-gm.py`.
- The Label Generation server runs on `9991`. The implementation is defined in `app-lm.py`.
- The Trace Authorization server runs on `9992`. The implementation is defined in `app-ta.py`.
- The Record Store server runs on `9993`. The implementation is defined in `app-rs.py`.

For more information on the commands available for customizing our implementation, please refer to our GitHub repository.

D.6 Version

Based on the LaTeX template for Artifact Evaluation V20220926.