Panini — Anonymous Anycast and an Instantiation

Christoph Coijanovic[1], Christiane Weis[2], and Thorsten Strufe[1]

[1] Karlsruhe Institute of Technology firstname.lastname@kit.edu
[2] NEC Laboratories Europe firstname.lastname@neclab.eu

Abstract. Anycast messaging (i.e., sending a message to an unspecified receiver) has long been neglected by the anonymous communication community. An *anonymous* anycast prevents senders from learning who the receiver of their message is, allowing for greater privacy in areas such as political activism and whistleblowing. To design protocols with provable guarantees for anonymous anycast, a formal consideration of the problem is necessary, but missing in current work. We use a game-based approach to provide formal definitions of anycast functionality and privacy. Our work also introduces Panini, the first anonymous anycast protocol that requires only existing infrastructure.
We show that Panini allows the actual receiver of the anycast message to remain anonymous, even in the presence of an honest but curious sender. In an empirical evaluation, we find that Panini adds only minimal overhead over regular unicast: Sending a message anonymously to one of eight possible receivers results in an end-to-end latency of 0.76s.

1 Introduction

In an *anycast*, messages are received by any one of a group of eligible receivers. Anycast is widely used in domain name resolution and content delivery networks [8]. Because the actual receivers are not predetermined, anycast also lends itself naturally to anonymous communication: Consider a group of political activists who fear retribution from the opposing regime. The activists want to implement *dead man's switches* among themselves, i.e., if someone is caught, someone else will be notified and can leak sensitive documents or take over their duties. If an arrested activist played a prominent role in the opposition, the regime will be particularly interested in her replacement. We can derive two main requirements from sending the dead man's notification via anonymous anycast:

First, no one (including the anycast sender) should be able to identify the receiver. This way, the captured activist cannot be forced to reveal her successor. To hide this information, the receiver must be chosen non-deterministically.

Second, the set of possible receivers should be constrainable by the sender. This ensures that one of the activist's trusted allies becomes her successor.

A third non-functional requirement is that sending an anonymous anycast should be as easy to set up as possible. Any obstacles, such as the need to set up a server infrastructure, will limit adoption.

Of course, anonymous anycast is not limited to political activism. Anonymous anycast is preferable to the much more common anonymous *unicast* [20, 14, 12] in any setting where receiver information is not relevant to, or should be hidden from the sender.

While not receiving the same amount of attention as anonymous unicast and multicast, there is some literature that addresses related issues. Mislove et al. mention that their AP3 protocol can be extended to support anycasts [27]. A recent line of research [4, 6, 16] focuses on anonymously selecting committee members to receive messages. We see one major shortcoming in these contributions:

Related work considers anonymous anycast from a protocol perspective, rather than a formal viewpoint. Without a formal understanding of the properties of anonymous anycast, it is difficult to compare current and future protocols. Each may define its own 'flavor' and requirements. Thus, our goal is to provide a concrete definition of the anonymous anycast problem and to formally define the main privacy goals of an anonymous anycast system.

Based on the desired functionality, we identify Message Confidentiality, Fairness, and Receiver Anonymity as the main goals of anonymous anycast. We formalize these goals using a game-based approach as it is common in cryptography (e.g., IND-CPA [3]) and already well established in anonymous unicast communication [21]. Our game-based privacy goals are unambiguously defined and allow for rigorous analysis of anycast protocols.

In this paper, we also propose Panini, an anonymous anycast protocol, to show that our defined privacy goals are achievable by efficient protocols. Panini relies on an anonymous unicast channel (e.g., Nym[3]) over which randomness is sent from possible receivers to the anycast sender. From this randomness, the sender can derive the key of an unknown receiver. This key is then used to encrypt a broadcast message. In summary, we make the following contributions:

- The formalization of functionality and privacy goals in anonymous anycast
- The proposal of Panini, the first protocol that allows anonymous anycast over readily available infrastructure
- A security analysis of Panini, showing that it achieves our previously defined privacy goals.
- In-depth empirical evaluation of Panini including long-term latency measurements of Nym.

The rest of this paper is organized as follows: Sec. 2 introduces the related work in greater detail. Sec. 3 presents the necessary background on provable privacy, linkable ring signatures, and Nym. Sec. 4 contains our formal treatment of the anonymous anycast problem including definitions of the privacy goals. Sec. 5 describes the Panini protocol. Sec. 6 contains Panini's empirical evaluation. Finally, Sec. 7 concludes the paper.

[3] https://nymtech.net — Accessed 08/24/2023

2 Related Work

Recall our requirements for anonymous anycast: (1) no entity expect the receiver (including the anycast sender) should learn who is receiving the anycast message, (2) the set of possible receivers should be constrainable, and (3) the anycast should be as easy to set up as possible. In this section, we will present anonymous anycast-related work and discuss whether it meets these requirements.

Target-Anonymous Channels A recent line of work considers *target-anonymous channels* [4, 16]. Benhamouda et al. informally define a target-anonymous channel as one that allows "anyone to post a message to an unknown receiver" [4]. In both papers, one protocol participant is chosen to select the receiver of the channel. The participant then provides all other participants with a way to contact the receiver without revealing the receiver's identity to them. Since the selecting participant inherently learns who the receiver will be in future uses of this target-anonymous channel, our first requirement is not met.

AP3 AP3 [27] is a mix network that implements the publish/subscribe communication pattern. Publishers and subscribers are both connected through the mix network to a common root node. The root node receives messages from the publishers and forwards them to the subscribers. Mislove et al. do not discuss in detail how AP3 can be extended to provide anycast functionality. We assume, based on the available information, that the root node randomly selects a subset of subscribers as actual receivers, rather than forwarding to all.

In AP3, the anycast sender must trust the root node to perform the anycast correctly (e.g., not send the message to all users). AP3's authors do not mention that the ability to subscribe to a publisher is limited. Thus, there seems to be no way for the anycast sender to define the set of possible receivers. So our second requirement is not satisfied.

Encryption to the Future Encryption to the Future (EtF) [6, 13, 7] is a cryptographic primitive where messages can be encrypted for a given *role*, rather than for a specific receiver. Later, a lottery is held to determine who gets to hold the role and thus be able to decrypt the ciphertext. To the best of our knowledge, suitable lottery primitives are all based on proof-of-stake blockchains [15, 2].

Even assuming the general availability of a suitable blockchain, the requirement that users acquire a cryptocurrency stake in order to receive an anycast message is a significant barrier to adoption. Thus, our third requirement is not met.

3 Background

This section gives background on provable privacy (Sec. 3.1), linkable ring signatures (Sec. 3.2), and the Nym anonymous communication protocol (Sec. 3.3).

3.1 Provable Privacy

When designing protocols for sensitive information, privacy and security are critical. Concrete proof that a protocol protects sensitive information is desirable, but it requires unambiguous definitions of what information must be protected. In cryptography, formal definitions of security have been established since the 1980s using indistinguishability games such as IND-CPA [17].

In privacy, there is a much wider variety of possible goals than in classical security: Some protocols may focus on protecting the privacy of the sender, while others may consider the receiver, or both. Indeed, many provable privacy frameworks have been proposed [21, 29, 19, 1]. We base our formalized anycast privacy on the work of Kuhn et al. [21], as their framework can express all the previous goals and bases them on a common indistinguishability game.

Kuhn et al.'s game is played between a challenger and an adversary. The adversary gets to choose two sets of communications, denoted as scenarios. The challenger chooses a scenario at random and simulates the protocol execution of the enclosed communications. The adversary receives any protocol output it could observe in the real world and has to decide which of its sets was selected.

Different privacy goals are expressed by restrictions on how communication may differ between scenarios: Any information that the protocol does not aim to hide must be identical between the sets. These restrictions ensure that the adversary cannot gain an unfair advantage.

A common goal of anonymous communication protocols is to unlink senders from their messages [22, 32, 14, 10]. Kuhn et al. formalize this goal in the privacy notion of Sender-Message Pair Unlinkability $(SM)\overline{L}$. Intuitively, a protocol that achieves $(SM)\overline{L}$ can reveal which senders are active and even which messages are being sent, but not who is sending which message.

3.2 Linkable Ring Signatures

Linkable ring signature (LRS) schemes [24] allow verification against a set of multiple verification keys. Linkability allows a verifier to determine whether two signatures were created using the same key. LRS schemes provide unforgeability (i.e., verification succeeds only if the signature was created with one of the corresponding secret keys) and signer anonymity (i.e., it cannot be determined which secret key was used to sign). For a more formal definition of LRS, see Liu et al.'s model of a linkable ring signature system [23]. An LRS scheme consists of the following algorithms [5]:

- $\textsc{Sig.Setup}(1^\lambda) \to pp$: On input of security parameter 1^λ, return public parameters pp.
- $\textsc{Sig.KeyGen}(pp) \to (vk, sk)$: On input of public parameters pp, returns a pair of public and secret key (vk, sk).
- $\textsc{Sig.Sign}(sk, m, R) \to \sigma$: On input of a secret key sk, a message m, and a ring $R = \{vk_0, \ldots, vk_\ell\}$, output a signature σ.

- SIG.VERIFY$(\sigma, m, R) \rightarrow \{0,1\}$: On input of a signature σ, a message m, and a ring $R = \{vk_0, \ldots, vk_\ell\}$, output 1 (accept), iff σ was generated by executing SIGN(sk, m, R), where sk corresponds to some $vk \in R$ and 0 (reject) else.
- SIG.LINK$(\sigma, \sigma') \rightarrow \{0,1\}$: On input of two signatures σ and σ' output 1, iff σ and σ' were created using the same secret key and 0 otherwise.

3.3 Nym

Most proposed anonymous communication networks exist only on paper, and do not provide a public instance for people to use. A recent exception is Nym [11], which provides both client software to download and servers to connect to[4].

While Nym has not yet been subjected to much scientific scrutiny, it adopts the communication architecture of the well-established Loopix mix network design [30]. In Nym, messages are onion-encrypted and sent through a series of mix nodes, each of which removes an encryption layer and adds a random delay to the messages. Nym also uses cover traffic from both clients and servers to 1) hide communication patterns, 2) detect denial of service attacks, and 3) ensure that mix nodes have a sufficient amount of alternative traffic in which to hide messages. As a result, Nym unlinks senders from their messages, as long as at least one mix node is honest.

Nym differentiates itself from Loopix by introducing an optional blockchain-based reward system. The reward system is independent of the actual communication infrastructure, not relevant to our use of the system, and therefore not considered in the remainder of this paper.

4 Problem Definition

In this section, we consider the anonymous anycast problem from a formal perspective. Sec. 4.1 defines which functionality an anycast protocol has to provide to be considered *correct*. Sec. 4.2 introduces our assumed adversary model. Sec. 4.3 contains game-based formalizations of anycast privacy goals.

In the following, we use $X \subset_i Y$ to express that set X is a strict subset of set Y consisting of i elements. $X \subseteq_i Y$ is used analogously. Further, $|X|$ expresses the number of elements in X. We use U to denote the set of all protocol participants.

4.1 Functionality

An anonymous anycast is a protocol between $q = |U|$ users. The anycast's *sender* selects a set of $l \leq q$ *possible receivers* U_p, of which $n \leq l$ shall receive the anycast message. The anycast functionality then selects n *actual receivers* U_a out of the set of possible receivers at random and forwards the message to them. This functionality can be trivially provided by a trusted third party $\mathcal{F}$. Definition 1 describes $\mathcal{F}$'s behavior and bases anycast correctness on equivalence to it.

[4] https://nymtech.net — Accessed 08/24/2023

Definition 1 (Anonymous Anycast Correctness). *The anonymous anycast functionality $\mathcal{F}$ interacts with a set of q users $U = \{u_1, \ldots, u_q\}$. It behaves as follows:*

1. *$\mathcal{F}$ waits for input of the form (m, n, U_p) from sender $u_s \in U$. The symbol m denotes the message, $U_p \subseteq_l U$ the set of* possible *receivers and $n \in \{1, \ldots, l\}$ the requested number of actual receivers.*
2. *$\mathcal{F}$ locally selects $U_a \subset_n U_p$ uniformly at random.*
3. *$\forall u \in U_a$: $\mathcal{F}$ sends m to u.*

An anonymous anycast protocol is correct *if it provides $\mathcal{F}$'s functionality.*

Following Definition 1, we consider a multicast (i.e., $n = l$/all possible receivers are selected as actual ones) a special form of anycast.

4.2 Adversary Assumptions

We assume an adversary $\mathcal{A}$ who can globally observe any network link, as well as actively interfere (i.e., drop, delay, modify, insert, and replay) with arbitrary packets. We further assume that $\mathcal{A}$ can corrupt the sender as well as a fraction of possible receivers. We assume that corrupted users are honest but curious and share all their knowledge with $\mathcal{A}$. We exclude arbitrarily malicious users, since they can trivially bypass any protocol's protection mechanism and send the anycast message directly to a receiver of their choice.

4.3 Privacy and Security Goals

In general, we must assume that both senders and receivers of an anycast are of interest to an adversary. However, to express sender-related privacy goals, existing notions of privacy for unicast communication can be used [21]. Thus, when considering the privacy and security goals of anonymous anycast, we focus on the receiver side. Based on the anonymous anycast functionality, we propose the following three main goals for our anycast setting:

1. **Message Confidentiality (MC).** Outside of the sender and actual receivers, nobody shall learn information about the content of the anycast message.
2. **Receiver Anonymity (RA).** Any adversary shall only learn trivial information about the *actual* receivers. Trivial information includes, for example, that a user learns that she is an actual receiver.
3. **Fairness** (F)**.** Any possible receiver shall be chosen as the actual receiver with the same probability, except for negligible deviations. In a protocol without fairness, an adversary learns that some users are more likely to receive the anycast message, even without observation.

While these informal descriptions of our goals give a good intuition of the information that should be protected, stating them informally is not sufficient to prove that a protocol achieves them. Thus, we use a game-based approach to formalize our privacy goals next.

Our games have a common structure: They are played between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$. The challenger internally simulates the anycast protocol Π. $\mathcal{A}$ can provide input (a "challenge") to the protocol and receives its output from $\mathcal{C}$. Based on the output, $\mathcal{A}$ must determine some information about the protocol execution.

Formalizing Message Confidentiality To formalize message confidentiality, we build on Kuhn et al.'s privacy game (see Sec. 3.1). We need to make the following modifications to adapt to the anycast setting:

1. While unicast communications are defined by a sender, a message, and a single receiver, an anycast message is defined by the sender, the message, the number of intended receivers, and the set of possible receivers. Thus, communications are expressed as tuples (s, m, n, U_p).
2. We assume that $\mathcal{A}$ is able to corrupt both the sender and a fraction of the receivers. Note that the anycast sender trivially learns the content of the message. Possible receivers also learn the message content if they are selected as actual receivers. To give the protocol a fair chance of achieving message confidentiality, $\mathcal{C}$ only provides protocol output to $\mathcal{A}$ if the sender and the actual receivers are not corrupted.

The resulting game $\mathcal{G}_{MC}$ for anycast protocol Π proceeds as follows:

1. $\mathcal{C}$ selects a challenge bit $b \in \{0, 1\}$ uniformly at random
2. $\mathcal{A}$ submits a challenge $Ch = (s, \{m_0, m_1\}, n, U_p)$
3. $\mathcal{C}$ simulates the anycast protocol Π's execution of (s, m_b, n, U_p) and saves the set of actual receivers U_a as well as Π's output $\Pi(s, m_b, n, U_p)$
4. If $\forall u \in U_a \cup \{s\} : u$ is *not* corrupted, $\mathcal{C}$ forwards $\Pi(s, m_b, n, U_p)$ to $\mathcal{A}$
5. $\mathcal{A}$ submits her guess $b' \in \{0, 1\}$ to $\mathcal{C}$. $\mathcal{A}$ wins if $b = b'$ and looses otherwise.

Analogous to Kuhn et al.'s game, steps (2-4) can be repeated an arbitrary number of times (with different challenges) to allow $\mathcal{A}$ to adapt its strategy based on its observations.

We say that a protocol Π achieves message confidentiality if there is no probabilistic polynomial-time algorithm $\mathcal{A}$ that can win $\mathcal{G}_{MC}$ with a non-negligible advantage over random guessing. Since there are two possible values for b, random guessing has a probability of success of $1/2$.

Receiver Anonymity One could define a receiver anonymity game analogous to the message confidentiality game: $\mathcal{A}$ submits two possible sets of actual receivers and must decide which one was chosen by the challenger. However, since actual receivers are supposed to be chosen non-deterministically, they cannot be set by the challenger. We adapt the game model so that the adversary has to make a guess for an actual receiver instead of making a binary decision.

Note that, as with message confidentiality, user corruption may allow $\mathcal{A}$ to trivially determine an actual receiver. If one of the users corrupted by $\mathcal{A}$ receives the anycast message, $\mathcal{A}$ unambiguously learns that this user was chosen as the

actual receiver. If $\mathcal{A}$ has corrupted all but n users, and none of them receives the anycast message, then all remaining users must be actual receivers. Thus, the challenger must check if one of these conditions is true after the actual receivers have been selected, and stop the game accordingly.

The complete $\mathcal{G}_{RA}$ game proceeds as follows:

1. $\mathcal{A}$ submits a challenge $Ch = (s, m, n, U_p)$
2. $\mathcal{C}$ simulates the protocol Π's execution of Ch and saves the chosen actual receivers U_a. $\mathcal{C}$ checks if $\mathcal{A}$ can trivially win due to user corruption. This is the case if there is a corrupted $u \in U_a$, or if *all* $u \in U_p \setminus U_a$ are corrupted. In case of a trivial win, $\mathcal{C}$ discards Π's output $\Pi(s, m, n, U_p)$. Otherwise, $\Pi(s, m, n, U_p)$ is forwarded to $\mathcal{A}$.
3. $\mathcal{A}$ can either choose to (a) unveil the challenge or (b) submit her guess $u^* \in U$
   (a) If $\mathcal{A}$ requested to unveil the challenge, $\mathcal{C}$ forwards U_a to $\mathcal{A}$
   (b) If $\mathcal{A}$ submitted her guess, $\mathcal{C}$ checks if $u^* \in U_a$. If so, $\mathcal{A}$ wins.

To allow $\mathcal{A}$ to adapt her strategy, steps (1-3) can be repeated a polynomial number of times as long as $\mathcal{A}$ chooses to unveil the challenge. Once $\mathcal{A}$ has submitted her guess, the game ends.

Analogous to message confidentiality, Π achieves receiver anonymity if there is no probabilistic polynomial-time algorithm $\mathcal{A}$ that can win $\mathcal{G}_{RA}$ with a non-negligible advantage over random guessing. Note that random guessing gives a probability of success of $n/|U_p|$, not $1/2$.

Formalizing Fairness Fairness is closely related to receiver anonymity: If the protocol is not fair and favors some receivers over others, this information can be used by $\mathcal{A}$ to gain an advantage in $\mathcal{G}_{RA}$. However, a protocol could be perfectly fair but fail to achieve receiver anonymity; Consider a toy protocol that chooses the actual receivers uniformly at random, but then announces them publicly.

If the protocol is not fair, $\mathcal{A}$ should have an advantage in guessing the actual receivers without relying on the protocol output. Thus, $\mathcal{G}_F$ differs from $\mathcal{G}_{RA}$ only in that $\mathcal{A}$ must submit her guess with the challenge, *prior* to receiving the protocol output. The resulting game is similar to the EUF-CMA game used to test the unforgeability of digital signatures [18].

User Corruption Within Games As described in Sec. 4.2, the adversary has the ability to corrupt the anycast sender as well as a fraction of the possible receivers. This ability is implemented in the games via a special challenge: Instead of sending a challenge *Ch*, $\mathcal{A}$ can send a *corruption query* specifying which users to corrupt in future runs. In response, $\mathcal{C}$ returns the internal state of the specified users. The protocol output $\Pi(Ch)$ may also change in future runs.

5 Protocol

Next, we propose Panini, a possible instantiation of anonymous anycast, to demonstrate that our defined notions of privacy are readily achievable. Panini

relies on a unicast channel that unlinks senders from their messages (i.e., the receiver or any outside observer does not learn which message was sent by whom). This channel is used to provide the anycast sender with randomness, which is used to determine the actual receiver.

Prerequisites Sending an anonymous anycast via PANINI requires:

1. An authenticated and confidential bidirectional unicast communication channel between the anycast sender and each possible receiver. We denote s sending a message m to r over this channel as $Ch_{\text{sec}}(s, m, r)$. Candidates for Ch_{sec} include the Signal messaging application[5], or email with S/MIME [31].
2. A unidirectional unicast communication channel that achieves Sender-Message Pair Unlinkability $(SM)\overline{L}$ and end-to-end confidentiality from every possible receiver to the sender. We denote s sending message m to r over this channel as $Ch_{\text{anon}}(s, m, r)$. Candidates for Ch_{anon} include Nym (see Sec. 3.3).

We first present a base version of PANINI secure against passive adversaries in Sec. 5.1, then extend it to protect against active adversaries in Sec. 5.2. Appendix A presents a pseudocode description of the full PANINI protocol. Finally, we analyze PANINI's security against our privacy notions in Sec. 5.3.

5.1 Basic Panini

The PANINI protocol works in three distinct phases: In phase P_0 (Init), the sender uses Ch_{sec} to send a KEYREQ message to all possible receivers, notifying them of the pending anycast. The KEYREQ message contains instructions for the receivers to contact the sender via Ch_{anon}.

During the second phase P_1 (Key Submit), each possible receiver u generates a random symmetric key k_u and sends it to the sender using Ch_{anon}. We cannot expect all receivers to send their keys at exactly the same time, especially if the adversary has the ability to selectively delay packets. Therefore, we assume that Ch_{anon} has the ability to compensate for delays up to some threshold T[6]. If the keys are delayed within this threshold, Ch_{anon} will still unlink them from their sender. If the threshold is exceeded and the anycast sender has not received all the keys, it terminates the protocol run.

The third phase P_2 (distribution) begins once the sender has received a key from each possible receiver. First, the sender verifies that all keys are unique. Then, the sender chooses a random key from the received keys and uses it to encrypt the message m to be anycast along with a publicly known *tag* which is used to check for correctness after decryption. The resulting ciphertext is then distributed to all possible receivers using Ch_{sec}. Each receiver decrypts the ciphertext with their k_u and checks if the revealed *tag* matches the correct one. If so, the receiver knows that she has been selected as the actual receiver and saves the message. The *tag* does not serve any privacy or security purpose, it is

[5] https://signal.org — Accessed 08/24/2023
[6] The exact value for T depends on the protocol used to initialize Ch_{anon}.

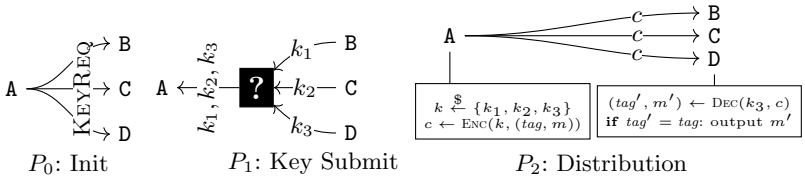

Fig. 1: Simplified Panini protocol run. ? unlinks senders from their messages. We use $x \xleftarrow{\$} X$ to denote that x is chosen uniformly at random from set X.

only used to determine if the message was correctly decrypted in cases where it is not obvious from the revealed plaintext. So it can be a fixed byte that is hard-coded into the protocol. Figure 1 visualizes a simplified run of Panini.

If the sender wants to anycast to more than one possible receiver, P_2 can be repeated n times, discarding previously selected keys. To send subsequent messages to the same actual receiver, the sender can use the same encryption key as for the initial message and multicast the resulting ciphertext again.

5.2 Defending against Active Adversaries

The basic Panini protocol described in the previous section protects against passive adversaries: Ch_{anon} ensures that the adversary cannot link keys to receivers, while the encrypted broadcast in phase P_2 ensures that the adversary cannot identify the actual receiver from the message sent to her. Basic Panini, however, cannot achieve confidentiality against *active* attacks:

During phase P_1, $\mathcal{A}$ discards some fraction or even all of the temporary keys submitted to the sender and replaces them with self-chosen ones. Then, during phase P_2, $\mathcal{A}$ intercepts the ciphertext. If the sender (unknowingly) chose one of $\mathcal{A}$'s keys, then she can decrypt it and break confidentiality.

If possible receivers add a digital signature to their temporary keys, then $\mathcal{A}$ is no longer able to exchange them for their own keys without the sender noticing. In our setting, the signature should only reveal that the communication partner is part of the set of possible receivers, not her concrete identity. To do this, we can use a linkable ring signature scheme [5, 28, 26]. See Sec. 3.2 for background on linkable ring signatures.

To protect against *external* active adversaries, phases P_0 and P_1 are updated as follows: In P_0, the anycast sender runs Sig.Setup and distributes the public parameters pp as part of the KeyReq message. After receiving pp, each possible receiver u executes Sig.KeyGen to generate its own signing key pair (vk_u, sk_u). sk_u is stored for future use and vk_u is sent to the anycast sender using Ch_{sec}. After receiving a verification key from each possible receiver, the anycast sender assembles $R \leftarrow \{vk_1, \ldots, vk_l\}$ and sends it to each possible receiver using Ch_{sec}. Each receiver checks if R contains its verification key and, if so, saves R for later

use. If it does not, the sender is assumed to be malicious and the receiver is dropped from the anycast.

In P_1, each possible receiver u generates a signature σ_u for its temporary key k_u by executing SIG.SIGN(sk_u, k_u, R). The receiver u then sends (k_u, σ_u) to the anycast sender using Ch_{anon}. Finally, the anycast sender executes SIG.VERIFY(σ_u, k_u, R) on each received key to ensure that all keys were generated by someone within the set of possible receivers (and not an external adversary).

While the steps described above prevent an external adversary from inserting keys, a corrupted possible receiver could expose their private signature key to the adversary. Using this key, the adversary can still generate (and validly sign) multiple keys on behalf of the malicious receiver without the sender noticing (assuming the same number of keys from other receivers are dropped by the adversary). To ensure that each possible receiver can only submit one key, we can use the *linkability* property of the signature: The anycast sender executes SIG.LINK($\sigma_u, \sigma_{u'}$) for each pair of signatures received. If SIG.LINK returns '1' for at least one pair of signatures, the sender detects that *some* malicious possible receiver has sent multiple keys to increase their chance of being selected. In response, the sender terminates the protocol run. If SIG.LINK returns '0' in all cases, the sender proceeds as described above.

Receiver Impersonation. We do not limit validity of the submitted keys to support (very) asynchronous communication. This comes with some security drawbacks: For example, an actively malicious possible receiver who was previously a sender within this group of receivers could replace all submitted keys with known keys from the previous protocol run to ensure that she can decrypt the anycast. As we assume that senders are honest-but-curious, such attacks are out of scope in this work. However, to handle active attacks, one can add a signed timestamp to each submitted key and let the sender discard received keys with too out-of-date timestamps.

5.3 Security Analysis

Finally, we want to show that PANINI achieves our privacy notions of message confidentiality, receiver anonymity, and fairness.

We start by proving that PANINI achieves message confidentiality. This is intuitively the case, as messages are encrypted such that only the actual receiver can unveil the plaintext. The use of linkable ring signatures further ensures that an adversary cannot insert their own keys.

Theorem 1. PANINI *achieves message confidentiality against the adversary* $\mathcal{A}$.

Refer to the extended version of this paper [9] for proof of Theorem 1.

Theorem 2. PANINI *achieves receiver anonymity against the adversary* $\mathcal{A}$.

Argument. Our goal is to show that there is no efficient $\mathcal{A}$ who has an advantage over random guessing in winning the $\mathcal{G}_{RA}$ game. To do so, we iterate through

all of $\mathcal{A}$'s abilities as listed in Sec. 4.2 and argue that none of them helps her to gain an advantage.

Passive Observation. Passive observation allows $\mathcal{A}$ to analyze incoming and outgoing packets anywhere in the network. During phase P_0, the sender sends an identical KeyReq package to every possible receiver. As all packages are identical, they cannot contain information about any actual receiver. We can analogously argue for phase P_2, where the sender sends an identical ciphertext to all possible receivers. In phase P_1, each possible receiver sends a unique key to the sender. If $\mathcal{A}$ were able to track who sends which key, she could identify the actual receiver based on their key and the ciphertext from phase P_2. However, if this were the case, $\mathcal{A}$ would also be able to break $(SM)\overline{L}$ for Ch_{anon}.

Timing. $\mathcal{A}$ can time sending behavior of any user. In phases P_0 and P_2, packets are sent simultaneously by the sender to the receivers. In P_2, the sender selects a random key from the received ones and encrypts using the selected key prior to sending. As all keys are randomly chosen and of equal length, we can assume that this selection and encryption do not vary in the time it takes based on which key is selected. If $\mathcal{A}$ were able to utilize timing in P_1 to identify actual receivers, she could also break $(SM)\overline{L}$ for Ch_{anon}.

Active Interference. $\mathcal{A}$ can actively interfere (i.e., drop, delay, modify, insert, and replay) with arbitrary packets. Active interference in phase P_1 would break $(SM)\bar{L}$ of Ch_{anon}.

Drop. If $\mathcal{A}$ drops KeyReq messages, the receiving clients are not informed about being possible receivers and will not participate further in the protocol. As the sender will only choose actual receivers if she has received the expected number of keys, the anycast will not be executed. The same behavior occurs if $\mathcal{A}$ drops keys in P_1. If $\mathcal{A}$ drops ciphertexts in P_2, the actual receivers may not receive the message. However, as we assume that receivers show no outward reaction to received (or not received) data, this does not reveal any information about the identity of possible receivers to $\mathcal{A}$.

Delay. Delays of packets other than the receivers' keys have no effect other than prolonging the protocol execution, as users wait for all expected packets to arrive before continuing with the execution. Delays of receivers' keys within the threshold T disclose no additional information as we assume that Ch_{anon} can compensate for these delays. Delays in excess of T cause the anycast sender to terminate the run prior to selecting the actual receiver and thus cannot reveal any information about the actual receiver to $\mathcal{A}$.

Modify. Due to the use of MACs, KeyReqs cannot be modified without detection. If keys in P_1 or ciphertexts in P_2 are modified, actual receivers might not be able to successfully decrypt the message. However, as we assume no reaction from receivers, this does not reveal any information to $\mathcal{A}$.

Insert. In phase P_0, $\mathcal{A}$ cannot insert further valid KeyReqs due to the use of MACs. In phase P_1, $\mathcal{A}$ cannot insert further keys, as the sender only proceeds with the anycast if the expected number of keys arrives. If $\mathcal{A}$ drops keys to insert its own ones, it is not able to do so without detection due to the use of linkable

ring signatures for the key messages. In phase P_2, $\mathcal{A}$ may insert new ciphertexts, but the receiving clients will not show any outward reaction.

Replay. Replaying KEYREQs in phase P_0 will only lead to the receiving clients discarding any extra ones. Replaying keys in P_1 will result in the sender receiving more keys than expected and not executing the anycast as a result. Replaying ciphertexts in P_2 elicits no reaction from receivers by assumption.

User Corruption. $\mathcal{A}$ can corrupt the sender as well as a fraction of possible receivers. Recall that $\mathcal{A}$ only receives protocol output (and therefore has a chance to not randomly guess) in $\mathcal{G}_{RA}$ if no actual receiver is corrupted and there exists at least one other possible receiver who is not corrupted.

By corrupting a receiver, $\mathcal{A}$ gains access to their internal state, including all key material. However, as the anycast message is only encrypted with the keys of actual receivers (who are not corrupted) and there remain honest possible receivers, $\mathcal{A}$ cannot use the gained information to determine which honest clients are actual receivers versus non-chosen possible receivers.

Ch_{anon}'s achievement of Sender-Message Pair unlinkability ensures that a corrupted anycast sender cannot link received keys to their owner. The ring signature scheme's signer anonymity property ensures that the corrupted anycast sender cannot link based on the key's ring signature. Thus, we have argued that none of $\mathcal{A}$'s abilities help her in winning $\mathcal{G}_{RA}$.

As fairness is implied by receiver anonymity (see extended version [9]), Theorem 2 also implies that PANINI achieves fairness.

6 Evaluation

We evaluate the performance of PANINI concerning two metrics: 1) Computational overhead for senders and receivers (Sec. 6.1) and 2) end-to-end latency between sender and actual receiver (Sec. 6.2). We suspect that PANINI's latency largely depends on that of the underlying anonymous channel and hence benefits from improvements in this active field of research. To get a more robust view of latency, the extended version [9] provides long-term latency measurements of Nym.

6.1 Computational Overhead

We want to determine how the computational overhead of PANINI for sender and receivers scales with the number of possible receivers. A PANINI protocol run can be divided into several distinct steps. Each step consists of different cryptographic operations and might scale differently with the number of receivers. Thus, we construct a series of microbenchmarks to measure each step separately.

We have split protocol execution between sender and receivers into six distinct steps to gain insight into the contribution to the overhead of different components:

- KG is executed by each receiver in phase P_0 and entails the generation of one linkable ring signature key pair.

- SIG is executed by each receiver in phase P_1 and entails the generation of a 32-byte AES key and the signing of it.
- VER is executed by the anycast sender in phase P_1 and entails the ring signature verification.
- LINK is executed by the anycast sender in phase P_1 and entails the linkability test of each signature.
- S&E is executed by the anycast sender in phase P_2 and entails the selection of receiver key(s) as well as the encryption of the message.
- D&C is executed by each receiver in phase P_2 and entails the decryption of the received ciphertext as well as the check of the tag.

We have implemented a prototype in go, which can be found on GitHub[7]. For all our measurements, we use a virtual machine running Ubuntu 22.04.1 on a server with an AMD EPYC 7502 Processor, 2 assigned cores, and 4GB of RAM. We use `lirisi`[8] as our linkable ring signature, which implements a signature scheme proposed by Liu et al. [25]. For symmetric encryption, go's standard `crypto/aes` package is used. To determine the impact of the number of possible receivers on the computational overhead, we execute each step for 10, 20, and 40 possible receivers and one actual receiver. In all experiments, a 1024-byte message of random content is used.

We expect steps VER and LINK to account for most of the computational overhead and to scale quadratically in the number of possible receivers: For VER, one signature needs to be verified for every receiver and each verification requires computations dependent on every public key. For LINK, every signature has to be compared to every other signature. We expect steps SIG and S&E to scale linearly in the number of possible receivers as the required computations depend on every receiver. Finally, we expect steps KG and D&C to be independent of the number of possible receivers.

We provide mean values as well as standard deviation of 100 individual measurements for each step in Table 1. One can see that the ring signature verification has by far the biggest impact on the computational overhead for the sender. For receivers, the largest contributor to computational overhead is also the linkable ring signature scheme, which is used to sign the keys.

Results for steps KG, SIG, VER, and D&C are as expected. However, LINK shows superlinear, not quadratic growth. We suspect compiler optimizations to cause this discrepancy. Step S&E takes constant rather than the expected linear growth. Note the high standard deviation on our measurements. We suspect CPU scheduling to be the cause of the high deviation and to hide the linear growth.

As we suspected, signature generation and verification are responsible for the largest part of computational overhead by far. Verification further scales quadratically in the number of receivers, which limits scalability. On the bright side, we have seen that Panini's other computational steps are very lightweight. Future—more efficient—linkable ring signature schemes, therefore, have the potential to also make Panini equally more efficient.

[7] https://github.com/coijanovic/anycast-bench — Accessed 08/24/2023
[8] https://github.com/zbohm/lirisi — Accessed 08/24/2023

Note that we have evaluated anycast to a *single* actual receiver. For n actual receivers, the computational overhead for steps S&E and D&C increases by a factor of n, as it has to be repeated for every actual receiver. The overhead for the other steps is independent of the number of actual receivers.

# Pos. Rec. 10	20	40
KG $15.79\mu s \pm 0.95$	$15.96\mu s \pm 1.02$	$15.88\mu s \pm 1.17$
SIG $2.70ms \pm 0.12$	$5.45ms \pm 0.19$	$10.85ms \pm 0.22$
VER $28.07ms \pm 1.12$	$109.78ms \pm 2.84$	$432.18ms \pm 5.76$
LINK $0.80\mu s \pm 0.175$	$2.67\mu s \pm 0.45$	$8.62\mu s \pm 0.95$
S&E $1.82\mu s \pm 0.59$	$1.69\mu s \pm 0.39$	$1.63\mu s \pm 0.38$
D&C $0.85\mu s \pm 0.27$	$0.75\mu s \pm 0.14$	$0.74\mu s \pm 0.12$

Table 1: PANINI microbenchmarks. Results are mean of 100 runs ± std. dev.

6.2 End-To-End Latency

In Sec. 1, we suggested that political activists can use anonymous anycast to implement a dead man's switch. We expect the dead man's notification to be similar in size and expected latency to instant messaging (IM). ITU Recommendation G.1010[9] states that "delays of several seconds are acceptable" for IM. We thus want to determine how latency in PANINI scales with the number of receivers and size of the message and if it meets G.1010.

We define the end-to-end latency as the time difference between the start of the sending client and the plaintext output of the actual receiver. To determine the impact of the number of receivers on the end-to-end latency, we will run an experiment with a fixed message size of 512 Byte and vary the number of receivers between 4 and 16. To determine the impact of the message size on the end-to-end latency, we will run an experiment with a fixed number of receivers of 8 and vary the message size between 512 Byte and 2 KB. To enable these experiments, we implemented a prototype of PANINI in go[10]. Ch_{sec} was instantiated with aes-encrypted and ecdsa-signed messages sent over TCP. Ch_{anon} was instantiated with Nym (WebSocket client v1.1.1). All clients ran on an AMD Ryzen 5 5600G with 32 GB of RAM. We repeat each measurement 16 times and present the median of the observed latencies ± standard deviation.

We expect PANINI's latency to be largely independent of both the number of receivers as well as the message size. As we have shown in Sec. 6.1, computational times for senders and receivers are well below 0.5s. Message size only impacts phase P_2, where the message is sent via Ch_{sec}. However, for IM-size messages, we expect Ch_{sec} to not be a bottleneck. The more receivers participate, the more

[9] https://www.itu.int/rec/T-REC-G.1010-200111-I — Accessed 08/24/2023
[10] https://github.com/coijanovic/panini — Accessed 08/24/2023

connection over Ch_{anon} have to be made. While receivers can send their data in parallel, the sender has to wait for the slowest receiver before continuing the execution. If there is variance in the latency of Ch_{anon}, we can expect Panini's end-to-end latency to increase with the number of receivers.

We measured a median end-to-end latency of $0.71s \pm 0.64$ for 4 receivers, $0.76s \pm 0.34$ for 8 receivers, and $0.82s \pm 2.13$ for 16 receivers. For the message size experiments, we measured $0.65s \pm 0.62$ end-to-end latency for 256 B message, $0.76s \pm 0.34$ for 512 B, $0.84s \pm 0.83$ for 1024 B, and $0.75s \pm 0.54$ for 2048 B.

As we expected, latency increases with the number of possible receives. For the message size experiments, we suspect that Nym's latency variance is to blame for the unexpected results: As sending 2048 B messages leads to lower latency than sending 1024 B messages, it seems unlikely that the message size itself is to blame. Nym's latency fluctuates over time, depending e.g., on network utilization. If the measurements for 2048 B messages were made during a period of lower utilization than the measurements for 1024 B messages, our results can be explained. Finally, the high standard deviation we observed in our measurements can also be explained by Nym's latency variation.

In summary, we have shown that—for up to 16 receivers and 2 KB messages—Panini achieves sub-second end-to-end latency and is therefore suitable for IM according to the ITU's recommendation. For use cases with larger payloads, we expect the ciphertext distribution in phase P_2 to dominate latency, but leave concrete evaluation to future work. While we have only evaluated Panini with Nym, we want to note that the two protocols are not inherently linked to each other. If a future anonymous communication network that achieves sender-message pair unlinkability with lower latency is proposed, Panini can utilize it, lowering its end-to-end latency in turn.

7 Conclusion

In this paper, we considered anycast from the perspective of anonymous communication. As necessary privacy goals, we identified message confidentiality, receiver anonymity, and fairness, which we also formalized in the style of indistinguishability games. Based on our formal definitions, it is now possible to analyze anonymous anycast protocols and provide rigorous proofs of privacy. We have also introduced Panini, the first protocol that allows anonymous anycast over readily available infrastructure. We have provided proof that Panini satisfies all of our previously defined privacy goals. Our empirical evaluation shows that Panini introduces only minimal computational overhead for anycast senders and receivers, and achieves end-to-end latency suitable for instant messaging.

Acknowledgements This work has been funded by the Helmholtz Association through the KASTEL Security Research Labs (HGF Topic 46.23), and by funding of the German Research Foundation (DFG, Deutsche Forschungsgemeinschaft) as part of Germany's Excellence Strategy – EXC 2050/1 – Project ID 390696704 – Cluster of Excellence "Centre for Tactile Internet with Human-in-the-Loop" (CeTI) of Technische Universität Dresden.

References

1. Backes, M., et al.: Anoa: A framework for analyzing anonymous communication protocols. IEEE CSF (2013)
2. Baldimtsi, F., et al.: Anonymous lottery in the proof-of-stake setting. IEEE CSF (2020)
3. Bellare, M., Desai, A., Pointcheval, D., Rogaway, P.: Relations among notions of security for public-key encryption schemes. In: IACR Cryptol. ePrint Arch. (1998)
4. Benhamouda, F., et al.: Can a public blockchain keep a secret? In: TCC (2020)
5. Beullens, W., et al.: Calamari and falafl: Logarithmic (linkable) ring signatures from isogenies and lattices. In: IACR Cryptol. ePrint Arch. (2020)
6. Campanelli, M., et al.: Encryption to the future: A paradigm for sending secret messages to future (anonymous) committees. IACR Cryptol. ePrint Arch. (2021)
7. Cascudo, I., et al.: Yolo yoso: Fast and simple encryption and secret sharing in the yoso model. In: IACR Cryptology ePrint Archive (2022)
8. Cicalese, D., Rossi, D.: A longitudinal study of ip anycast. Comput. Commun. Rev. (2018)
9. Coijanovic, C., et al.: Panini: Anonymous anycast and an instantiation (extended version). ArXiv (2023)
10. Das, D., et al.: Organ: Organizational anonymity with low latency. PoPETs (2022)
11. Díaz, C., et al.: The nym network the next generation of privacy infrastructure (2021)
12. Dingledine, R., et al.: Tor: The second-generation onion router. In: USENIX Security (2004)
13. Döttling, N., et al.: Mcfly: Verifiable encryption to the future made practical. IACR Cryptol. ePrint Arch. (2022)
14. Eskandarian, S., et al.: Express: Lowering the cost of metadata-hiding communication with cryptographic privacy. In: USENIX Security (2021)
15. Ganesh, C., et al.: Proof-of-stake protocols for privacy-aware blockchains. IACR Cryptol. ePrint Arch. (2018)
16. Gentry, C., et al.: Random-index pir with applications to large-scale secure mpc. IACR Cryptol. ePrint Arch. (2020)
17. Goldwasser, S., Micali, S.: Probabilistic encryption. J. Comput. Syst. Sci. (1984)
18. Goldwasser, S., et al.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. (1988)
19. Hevia, A.G., Micciancio, D.: An indistinguishability-based characterization of anonymous channels. In: PETS (2008)
20. van den Hooff, J., et al.: Vuvuzela: scalable private messaging resistant to traffic analysis. SOSP (2015)
21. Kuhn, C., et al.: On privacy notions in anonymous communication. PoPets (2019)
22. Langowski, S., et al.: Trellis: Robust and scalable metadata-private anonymous broadcast (2022)
23. Liu, J.K., Wong, D.S.: Linkable ring signatures: Security models and new schemes. In: ICCSA (2005)
24. Liu, J.K., et al.: Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). IACR Cryptol. ePrint Arch. (2004)
25. Liu, J.K., et al.: Linkable spontaneous anonymous group signature for ad hoc groups (extended abstract). IACR Cryptol. ePrint Arch. (2004)
26. Lu, X., et al.: Raptor: A practical lattice-based (linkable) ring signature. IACR Cryptol. ePrint Arch. (2018)

27. Mislove, A., et al.: Ap3: cooperative, decentralized anonymous communication. In: EW 11 (2004)
28. Nassurdine, M., et al.: Identity based linkable ring signature with logarithmic size. In: Inscrypt (2021)
29. Pfitzmann, A., Hansen, M.: A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management (2010)
30. Piotrowska, A.M., et al.: The loopix anonymity system. ArXiv (2017)
31. Schaad, J., Cellars, A., et al.: Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 4.0. RFC 8551, RFC Editor (April 2019)
32. Vadapalli, A., et al.: Sabre: Sender-anonymous messaging with fast audits. IEEE SP (2022)

A Panini Pseudocode

Refer to Algorithm 1 for a pseudocode description of Panini.

Algorithm 1 Sender and receiver behavior for PANINI. $\textsc{Send}(s, m, n, U_p)$ is executed by user s who wants to send message m to n users out of the set of possible receivers U_p. $\textsc{Receive}(u)$ is executed by receiver u to receive possible anycast messages. λ and tag are fixed protocol parameters known to all users. During the execution of SEND, three lists are assembled: R contains the possible receivers' signature public keys, K contains the possible receivers' ephemeral keys, and Σ contains their signatures.

procedure $\textsc{Send}(s, m, n, U_p)$
  $pp \leftarrow \textsc{Sig.Setup}(1^\lambda)$
  $\textsc{KeyReq} \leftarrow (hello, pp)$
  for $u \in U_p$ **do**
    $Ch_{\text{sec}}\,(s, \textsc{KeyReq}, u)$
  $R \leftarrow \{\}$
  while $|R| < |U_p|$ **do**
    on vk **from** u **do**
      $R \leftarrow R \cup \{vk\}$
  for $u \in U_p$ **do**
    $Ch_{\text{sec}}\,(s, R, u)$
  $t \leftarrow \textsc{Timer.start}()$
  $K \leftarrow \{\}$
  $\Sigma \leftarrow \{\}$
  while $|K| < |U_p|$ **do**
    if $t \geq T$ **then**
      return
    on (k, σ) **from** $\perp$ **do** ▷ Ch_{anon} does not disclose sender.
      if $\textsc{Sig.Verify}(\sigma, k, R) \neq 1$ **or** $k \in K$ **then**
        return
      $K \leftarrow K \cup \{k\}$
      $\Sigma \leftarrow \Sigma \cup \{\sigma\}$
  for $\sigma \in \Sigma$ **do**
    for $\sigma' \in \Sigma$ **do**
      if $\sigma \neq \sigma' \wedge \textsc{Sig.Link}(\sigma, \sigma') \neq 0$ **then**
        return
  for $i \in \{1, \ldots, n\}$ **do**
    $k^* \xleftarrow{\$} K$
    $K \leftarrow K \setminus \{k^*\}$
    $c \leftarrow \textsc{Cipher.Enc}((tag, m), k^*)$
    for $u \in U_p$ **do**
      $Ch_{\text{sec}}\,(s, c, u)$
procedure $\textsc{Receive}(u)$
  on KeyReq **from** s **do**
    $(sk, vk) \leftarrow \textsc{Sig.KeyGen}(pp)$
    $Ch_{\text{sec}}\,(u, vk, s)$
  on R **from** s **do**
    if $vk \notin R$ **then**
      return
    $k \leftarrow \textsc{Cipher.KeyGen}(1^\lambda)$
    $\sigma \leftarrow \textsc{Sig.Sign}(sk, k, R)$
    $Ch_{\text{anon}}\,(u, (k, \sigma), s)$
  on c **from** s **do**
    $(tag', m') \leftarrow \textsc{Cipher.Dec}(c, k)$
    if $tag' = tag$ **then**
      return m'