Karaoke: Distributed Private Messaging Immune to Passive Traffic Analysis

David Lazar, Yossi Gilad, and Nickolai Zeldovich
MIT CSAIL

This introduction describes a knowledge gap with two specific challenges, expanding on the problem statement

Abstract

Karaoke is a system for low-latency metadata-private communication. Karaoke provides differential privacy guarantees, and scales better with the number of users than prior such systems (Vuvuzela and Stadium). Karaoke achieves high performance by addressing two challenges faced by prior systems. The first is that differential privacy requires continuously adding noise messages, which leads to high overheads. Karaoke avoids this using *optimistic indistinguishability*: in the common case, Karaoke reveals no information to the adversary, and Karaoke clients can detect precisely when information may be revealed (thus requiring less noise). The second challenge lies in generating sufficient noise in a distributed system where some nodes may be malicious. Prior work either required each server to generate enough noise on its own, or used expensive verifiable shuffles to prevent any message loss. Karaoke achieves high performance using *efficient noise verification*, generating noise across many servers and using Bloom filters to efficiently check if any noise messages have been discarded. These techniques allow our prototype of Karaoke to achieve a latency of 6.8 seconds for 2M users. Overall, Karaoke's latency is 5× to 10× better than Vuvuzela and Stadium.

1 Introduction

general introduction

Text messaging systems are often vulnerable to traffic analysis, which reveals communication patterns like who is communicating with whom. Hiding this information can be important for some users, such as journalists and whistleblowers. However, building a messaging system just for whistleblowers is not a good idea, because using this system would be a clear indication of who is a whistleblower [9]. Thus, it is important to build metadata-private messaging systems that can support a large number of users with acceptable performance, so as to provide "cover" for sensitive use cases.

problem statement

quantifies problem statement

A significant limitation of prior work, such as Vuvuzela [26], Pung [1], and Stadium [25], is that they incur high latency. For example, with 2 million connected users, Vuvuzela has an end-to-end latency of 55 seconds, and the latencies of Pung and Stadium are even higher. Such high latencies hinder the adoption of these designs.

Main contribution: Karaoke is 10x faster than previous work

This paper presents Karaoke, a metadata-private messaging system that reduces latency by an order of magnitude compared to prior work. For instance, Karaoke achieves an end-to-end latency of 6.8 seconds for 2 million connected users on 100 servers (on Amazon EC2 with simulated 100 msec round-trip latency between servers), 80% of which are assumed to be honest, and achieves differential privacy guarantees comparable to Vuvuzela and Stadium. Furthermore, Karaoke can maintain low latency even as the number of users grows, by scaling horizontally (i.e., having independent organizations contribute more servers). Karaoke supports 16 million users with 28 seconds of latency, a 10× improvement over Stadium.

quantify improvement for some specific parameters

Achieving high performance requires Karaoke to address two challenges. The first challenge is that differential privacy typically requires adding noise to limit data leakage. Prior work achieves differential privacy for private messaging by enumerating what metadata an adversary could observe (e.g., the number of messages exchanged in a round of communication), and adding fake messages ("noise") that are mixed with real messages to obscure this information. This translates into a large number of noise messages that have to be added every round, and handling these noise messages incurs a high performance cost.

why is high performance difficult?
A: generating noise

Karaoke addresses this challenge using *optimistic indistinguishability*. Karaoke's design avoids leaking information in the common case, when there are no active attacks. Karaoke further ensures that clients can precisely detect whether any information was leaked (e.g., due to an active attack), so that the clients can stop communicating to avoid leaking more data. This allows Karaoke to add fewer noise messages, because the noise messages need to mask fewer message exchanges (namely, just those where an active attack has occurred).

1st contribution: handle active and passive attacks differently

The second challenge lies in generating the noise in the presence of malicious servers. One approach is to require every server to generate all of the noise on its own, under the assumption that every other server is malicious [26]. This scheme leads to an overwhelming number of noise messages as the number of servers grows. Another approach is to distribute noise generation across many servers. However, a malicious server might drop the noise messages before they are mixed with messages from legitimate users. As a result, achieving privacy requires the use of expensive zero-knowledge proofs (e.g., verifiable shuffles) to ensure that an adversary cannot drop messages [25]. This approach reduces the number

why is generating noise difficult?
A: malicious servers may not add noise

This paper appeared in the Proceedings of the 13th USENIX Symposium on Operating System Design and Implementation (OSDI 2018).

of noise messages, but leads to significant CPU overheads due to cryptography.

2nd insight: use Bloom filters to check noise is observed, not all messages

Karaoke's insight is that verifiable shuffles are overkill: it is not necessary for all messages to be preserved, and it is not necessary to prove this fact to arbitrary servers. Instead, to achieve privacy, it suffices for each server to ensure that its noise is observed by all other servers. This can be done efficiently using Bloom filters, without having to reveal which messages are noise and which messages come from real users.

The contributions of this paper are as follows:

summarize contributions in order of importance

- The design of Karaoke, a metadata-private text messaging system that achieves an order of magnitude lower latency than prior work.
- Two techniques, optimistic indistinguishability and efficient noise verification, which allow Karaoke to achieve high performance.
- A privacy analysis of Karaoke's design that supports the use of these techniques.
- An experimental evaluation of a prototype of Karaoke.

One limitation of Karaoke is that it does not provide fault tolerance, since it requires all servers to be online. Handling server outages and denial-of-service attacks is an interesting direction for future work.

The Karaoke paper does a good job concisely motivating the work; the introduction early on says "...building a messaging system just for whistleblowers is not a good idea, because using this system would be a clear indication of who is a whistleblower. Thus, it is important to build metadata-private messaging systems that can support a large number of users with acceptable performance, so as to provide 'cover' for sensitive use cases." Not only does the introduction argue why metadata-private messaging is important, it makes a concise case why performance is important for achieving privacy. Next, the introduction makes the problem statement concrete by citing the performance achieved by the prior work.

It also gives just enough technical detail to clarify what the paper is about without going into so much depth as to lose any readers. For example, "Karaoke addresses this challenge using optimistic indistinguishability. Karaoke's design avoids leaking information in the common case, when there are no active attacks. Karaoke further ensures clients can precisely detect whether any information was leaked (e.g., due to an active attack)." This gives an overall flavor of what the technique accomplishes (use simpler techniques to hide information if there aren't active attacks, detect and address active attacks separately), without describing how it is implemented. Furthermore, the name "optimistic indistinguishability" alludes to a similar technique called optimistic concurrency control that readers are likely to be familiar with.