Mixclave Networks: Building Mixnets with Hardware Enclaves

Mark Theis

Acknowledgement

I foremost would like to thank the NetSys Lab faculty advisors, Scott and Professors R, for their guidance, support, and the opportunity for research within the lab. I would also like to thank my collaborators and co-authors, without whom this project would have not been possible, Chris Douglas and Vikranth Srivatsa. I would also like to recognize additional colleagues who provided essential insights based on their experience, which informed the design, implementation, and motivation for Mixclaves. Alphabetically: Emmanuel Amaro, Tiemo Bang, Natacha Crooks, Vivian Fang, John Kubiatowicz, Zhuohan Li, Mae Milano, Micah Murray, and Aurojit Panda.

Mixclave Networks: Building Mixnets with Hardware Enclaves

by Mark Theis

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

[illegible]

Professor Scott Shenker
Research Advisor

April 22, 2023

(Date)

* * * * * * *

[illegible]

Professor Sylvia Ratnasamy
Second Reader

May 4, 2023

(Date)

Abstract

Mixclave Networks: Building Mixnets with Hardware Enclaves

by

Mark Theis

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Scott Shenker, Research Advisor

All secure messaging systems protect the content and integrity of users' messages, but the oblivious routing of messages concealing who communicates with whom (metadata-private messaging) is increasingly crucial for privacy. Existing techniques conceal routing metadata using *mix networks* (mixnets) made up of multiple nodes that batch and forward traffic to confound traffic analysis. State-of-the-art mix networks remain resilient to a passive global adversary even as attackers compromise up to 20% of the mix nodes.

As infrastructure moves to the cloud, threat models for metadata-private messaging must assume an adversary that is both active and even present on machines routing user data. This paper proposes *Mixclaves*, a scalable, metadata-private messaging architecture that builds on hardware enclaves to provide a cost-efficient, low-latency mixnet implementation deployable in public clouds. Building on stronger guarantees provided by enclaves not only simplifies the implementation of mixnets, it also admits novel features and lower operating costs. Compared to Loopix and Groove, two popular mixnet implementations, *mixclaves are 54% cheaper on cost to achieve the same message throughput.*

Contents

List of Figures

List of Tables

Acknowledgments

I foremost would like to thank the NetSys Lab faculty advisors, Scott and Professors Ratnasamy, for their guidance, support, and the opportunity for research within the lab. I would also like to thank my collaborators and co-authors, without whom this project would have not been possible, Chris Douglas and Vikranth Srivatsa. I would also like to recognize additional colleagues who provided essential insights based on their experience, which informed the design, implementation, and motivation for Mixclaves. Alphabetically: Emmanuel Amaro, Tiemo Bang, Natacha Crooks, Vivian Fang, John Kubiatowicz, Zhuohan Li, Mae Milano, Micah Murray, and Aurojit Panda.

Chapter 1

Introduction

We kill people based on metadata.

Gen. Michael Hayden, former NSA Director, 2014 [26, 53]

The scope of state-sponsored and corporate surveillance is so widespread, it is no longer covert; it is assumed. While this awareness spurred broad adoption of authenticated encryption, *metadata* remains a key vector to track and target individuals through their networks. By analyzing how data are routed and accessed over time, an adversary can not only infer confidential information, but also implicate individuals by association. Metadata leak confidential and identifying information on journalists, whistleblowers, activists, business executives, and government officials. Leading consumer communication services Signal [45], WhatsApp [57], and iMessage [30] support end-to-end encryption to guard the content of users' communication, even from the providers of those services. However, the provider has sufficient data to reveal users' contacts if coerced or inclined. Since metadata are collected and sold openly, regulations restricting access are easily sidestepped or completely ignored [13].

To address this gap, metadata-private messaging systems from the research community conceal not only message content, but also routing from adversaries and curious providers. State-of-the-art designs such as Groove [9] and Loopix [46] refine the *mixing networks* (mixnets) introduced by David Chaum (Section 2.1). These messaging systems offer statistical proofs of privacy by confounding network analysis that could correlate the routes and timing of messages through the mixnet. Both require large networks of machines to offer statistical privacy guarantees for users, since an adversary may control a substantial fraction of the network. Revealing and uprooting an adversary in the mixnet improves privacy of the network [39], but it cannot retract leaked metadata.

Privacy-conscious users may route traffic through multiple independent providers to decrease the probability of a correlated compromise [31]. However, this not only increases latency and harms efficiency, it also burdens the user with a molten set of providers to manage. Long paths also create a complex capacity planning problem for providers. Billing is a uniquely awkward proposition for networks designed to conceal identity, and volunteer anonymizing networks can draw unwanted attention even in unregulated jurisdictions [27, 6]. Mixclave networks propose federated administrative domains that can operate commercially. Mixclaves do not seek to conceal the existence of the network, only the traffic it forwards.

The continuing expansion of edge computing is a testament to the predominance of centralized and commercially backed services. The Cloudlet model, introduced by M. Satyanarayanan in 2008 [59], was composed of a vision for decentralized VM based Cloudlets to enable nearby computation offload for mobile devices. Modern edge computing has materialized in a different light, where companies centrally manage a large majority of the edge in the forms of colocation and CDNs [32] rather than in the original decentralized Cloudlet vision. But the importance of moving compute to many nodes has shown virtue. There is a need for an architecture that offers completely anonymous, user verifiable, communication that may be managed by a central provider.

Hardware enclaves offer a confidential and attested compute environment ideal for mixnet nodes, but simply running a mixnet within enclaves is insufficient for confidential routing. Surprisingly, by addressing gaps in enclave confidentiality using oblivious data structures, we find mixnet invariants are satisfiable by a single node. Anchored in enclaves, Mixclave networks can operate confidentially, elastically, and cost-effectively compared to traditional mixing networks.

With the mixnet shrinking to as small as one node, we included functionality in Mixclaves that enables the network to auto-scale and respond to load. This is possible because we may trust the nodes in the network, so we may have assurance about the metrics they report and may assign a leader node in a domain to handle scaling.

We claim the following contributions. To the best of our knowledge, this is the first implementation of a mixnet that employs hardware enclaves not only to facilitate metadata-private routing under differential privacy, but also to enable dynamic scaling via a covert control channel.

Cost. Enclaves allow us to collapse the mix network to a single node, saving up to 82% in cost in cloud settings.

Elasticity. Metrics from enclave mix nodes are securely sampled and used to scale mixnet capacity within an administrative domain by adding nodes to the mixnet.

Practical deployment. Mixclaves can be deployed in a federation of administrative domains with attestation guarantees provided by hardware enclaves.

The paper is structured as follows. Section 2 provides context and assumptions for Mixclaves. Section 3 describes the Mixclaves architecture, particularly how its design satisfies our goals from Section 2. Section 4 describes our prototype instantiation of the architecture, including caveats and practical details. Section 5 measures our prototype against our goals from Section 2. Section 6 contrasts the Mixclaves architecture with existing work before discussing implications and future work in Section 7. Section 8 concludes.

Chapter 2

Background

2.1 Mixnets

David Chaum proposed *mixing networks* (mixnets) in 1981 [16]. This paper established the following two criteria for anonymizing network traffic in a mixnet. First, an adversary cannot infer a path between the source and destination on either side of the mixnet by comparing payloads. In a Chaum mixnet, layered encryption provides *bitwise unlinkability*, as removing each layer yields a new, uncorrelated payload. Subsequent work on mixnet packet encoding [20] ensures that other attributes like packet size leak no signal to an adversary. Nodes also deduplicate traffic to prevent an active attacker from replaying and linking datagrams; a single node is guaranteed to receive replay traffic, since the packet is encrypted with that node's public key.

Second, mixnets must obscure *correlations in time* between packet flows. In a Chaum mixnet, messages are batched at each node until a threshold, then released to a successor. Deduplication and batching create **anonymity sets** of packets; given a packet entering the mixnode, packets exiting the mixnode with a non-zero probability of being derived from that packet are in its anonymity set. While in Chaum mixnets that set is an explicitly signed batch, in modern mixnets the probability of a packet being in an anonymity set may be a non-symmetric distribution[1]

Modern mixnets add two additional criteria to the original model. The mixing network must be *resilient to compromised nodes*. In traditional mixnets, threat models assume that an attacker may observe or even control some fraction of nodes in the mixing network. To compensate, packets are routed through multiple nodes to increase the probability of encountering at least one *honest node* to confound traffic analysis. Finally, modern mixnets generate synthetic *cover traffic* to ensure that user datagrams flowing through the mix network cannot be distinguished from noise or overwhelmed by an active adversary. For example, an

[1]For example, if packet A arrives at t_A and B arrives at t_B, $t_A < t_B$, packets emitted between t_A and t_B are in A's anonymity set but not in B's. As $t_A \ll t_B$, the probability of B being in A's anonymity set diminishes.

attacker could fill batches in a classic Chaum mixnet with its own synthetic traffic, effectively shrinking the batch size. Mixclaves produce some cover traffic, but metadata-private applications must be responsible for concealing their own workload (Section 2.1).

In summary, we preserve four properties of a modern mixnet: **bitwise unlinkability**, **protection from correlations in time**, **resilience to compromised nodes**, and **cover traffic**.

Differential Privacy

Differential privacy describes the promise from a data holder to a user that "You will not be affected, adversely or otherwise, by allowing your data to be used in any study or analysis, no matter what other studies, data sets, or information sources, are available" [25]. Applied to a mixnet, this means that the network guarantees that for any pair of users A, B, the probabilities that the two are communicating or not communicating are close, as defined by the parameters ϵ, δ [37] (e^{ϵ} is a multiplicative factor, and δ is additive). Statistically, an adversary cannot prove within those bounds that A and B exchanged authentic (i.e., non-cover) traffic.

Given this framing, we show in Section 2.4 that a single Mixclave node is sufficient to provide privacy guarantees to the client as strong as those provided by a mixing network.

Application Architecture

Alone, a mixnet cannot prevent an application from leaking information that compromises user privacy. For example, consider a user B orders of magnitude more active than any other user in a messaging platform. A metadata-private messaging system may not know the source IP, but the destination IP at the other side of the mixnet can be observed and distinguished by a passive adversary with a full view of the network (Section 2.3). Since it lacks all knowledge of the workload, the mixnet can only passively forward datagrams it receives and generate generic cover traffic.

Systems like Loopix [46] and Groove [9] generate cover traffic not only from clients, but also from interstitial *provider* nodes. These application-specific nodes not only run a cover protocol with clients, they also generate synthetic cover traffic into the mixnet. In the metadata-private messaging example, these nodes would be responsible for concealing the prenominate, popular user B by throttling delivery rates and/or sharding its mailbox across provider nodes. Providers may be trusted (e.g., Loopix) or untrusted (e.g., Groove), depending on the application threat model.

Mixclaves focus exclusively on the cost and operability of the mixnet component of metadata-private applications. Consequently, we are most concerned with the performance of a Mixclave network at saturation.

2.2 Deployment Model

Relay networks like Tor [22] are deployed by volunteers, placing few or no restrictions on who may join the network. This *distributed deployment* model resists analysis by requiring an attacker to add malicious nodes until its targets route sufficient traffic through its network. The probability of clients choosing paths through not only corrupt nodes, but corrupt nodes controlled by a particular attacker is unknowable, but assumed to be sufficiently low that every path through the mixnet encounters at least one “honest” node, as in Figure 2.1.

Egalitarian architecture deployed on volunteer infrastructure succeeded spectacularly in the early evolution of the public internet. Today, the resources and expert knowledge necessary to operate a secure service— particularly one that draws the attention of state-sponsored surveillance— exceed the grasp of most enthusiasts. Mixnets in particular are challenging to deploy, since a provider either joins an existing, volunteer mixnet or bootstraps one with new users. In the former case, capacity planning is not a function of its paying users, but of the broader network. In the latter, user traffic is anonymized by a smaller population, providing weaker guarantees.

Fortunately, we have recent examples of privacy-preserving infrastructure deployed at scale. Aversion to tracking motivated development and widespread adoption of technology to confound data collection, rather than controlling its dissemination by fiat [49, 35]. iCloud Private Relay [31, 52] is an oblivious DNS over HTTP (ODoH) implementation that obscures clients’ identities by encrypting and routing requests through multiple proxies. Notably, this implementation also involves multiple vendors by design. The first proxy is owned and operated by Apple, but a second proxy tier is operated by third party CDN providers (e.g., Akamai, Fastly, Cloudflare [36]). A *federated deployment* allows each vendor to collect sufficient data for capacity planning and cross-billing without identifying individual users. Exposing any user’s DNS requests requires collusion among multiple vendors.

By assuming providers are stable in a federation, Mixclave networks are feasible to deploy at scale. Peering relationships could meter, manage, and price traffic through the federated mixnet. Endpoints could verify their right to access the mixnet in that domain without associating their identity [24, 41, 5], while traffic between domains could be shaped to do credible capacity planning. Aligning incentives for honest operation of administrative domains is outside our scope, but we note that an operator has more flexibility for its subset of endpoints than in the distributed deployment model.

2.3 Threat Model

We assume our adversary can passively observe the entire network and memory traces on mixnet nodes. We further assume that an attacker can inject traffic into the network, also dropping, replaying, and delaying traffic arbitrarily. All information shared with clients is also known by the adversary, particularly public keys and membership information. Mixclaves use the Sphinx packet format [20, 19] to provide bitwise unlinkability. All Sphinx

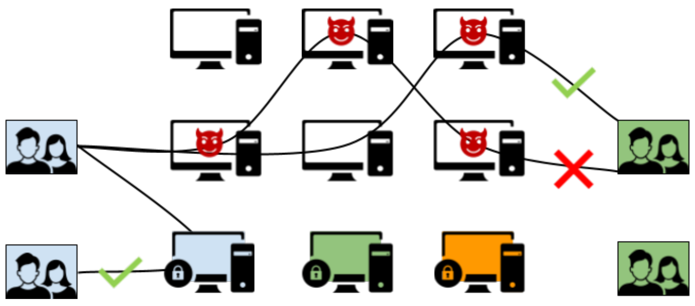

Figure 2.1: **Mixnet Topology.** To preserve privacy, clients must route packets through at least one honest node or privacy may be compromised. Within a domain, it is sufficient to route packets through a single Mixclave node.

messages are padded to the same length and allow for detection of tagging and replay attacks.

Mixclaves assume that an adversary may control a majority of devices, including mixnet nodes. Adversaries can control the hardware of the machine to monitor the systems's network and hardware. Any mixclave node can also become unvailable, due to tampering or denial of service (DoS) attacks, but provide the adversary no new information.

We assume a computationally restricted adversary, so the cryptography is sound and decrypting packets (particularly chosen by the adversary) inside the enclave cannot leak information about either decrypted packets or any other traffic. This includes the secure cryptographic assumptions of secure hashing, key encryption, and key exchange. As we discuss in section 2.4, delays chosen by an attacker must be obscured by Mixclaves when the message is queued.

As discussed in Section 2.1, we assume that any application over the mixnet generates sufficient cover traffic to conceal its workload between the mixnet and its endpoints. As in Groove [9] and Loopix [46], an application can deploy *provider* nodes that generate continuous, synthetic cover traffic if endpoints are only intermittently online. Similarly, any property of or over the data such as forward secrecy is maintained by the application. Mixclave nodes also generate cover traffic, following an exponential distribution chosen by the operator [46].

Finally, we assume that clients can learn the public keys not only for other clients, but also to at least one node in the active mixnet. Bootstrapping metadata-private communication is not necessarily out of band [38]. Section 3 describes how endpoints learn about the keys for previously unknown nodes in the mixnet.

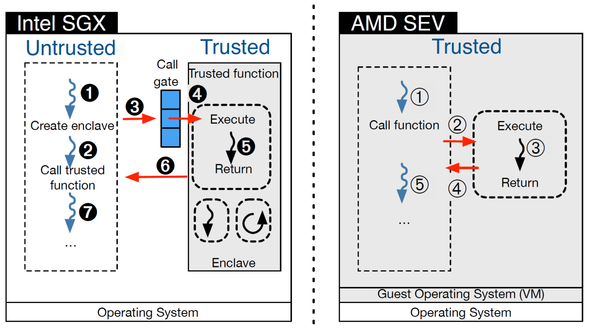

Figure 2.2: **Intel SGX enclaves versus AMD SEV-SNP [28].**

2.4 Hardware Enclaves

Enclave Overview

An ideal secure enclave enforces a hardware isolation boundary such that even privileged processes cannot tamper with or inspect the execution of a binary executing within the enclave. Enclaves support binary attestation, which enables external systems to verify the identity and code running in the enclave [4, 3].

Both AMD and Intel support enclaves in their modern server processors. Intel's SGX implementation runs individual binaries in enclaves, encouraging hybrid application architectures that avoid overheads for code that runs outside the enclave. Data are exchanged across this boundary as in Figure 2.2. In contrast, AMD's SEV approach runs an entire VM inside the enclave. Mixclaves are designed to operate almost entirely within the enclave; its architecture could apply in either setting.

We assume two properties of hardware enclaves. First, we assume the code executed inside the enclave is *attested* and both operators and clients can verify its integrity. Consequently, any code inside the enclave that should execute, will run; we assume a fail-stop model and exclude Byzantine mixnet nodes within an administrative domain (Section 2.2). Second, we assume code removing a layer of encryption from the packet and examining its metadata inside the enclave leaks no information to an adversary, conceding that in current enclave implementations this comes with caveats [44, 56, 40]; attested code executes confidentially, excluding side-channels discussed in Section 2.4.

Mixclave nodes buffer messages with a random delay. Removing a layer of encryption within the enclave requires no additional mitigation, but an adversary could infer timing information from the message buffer that could reduce the cardinality of its anonymity set.

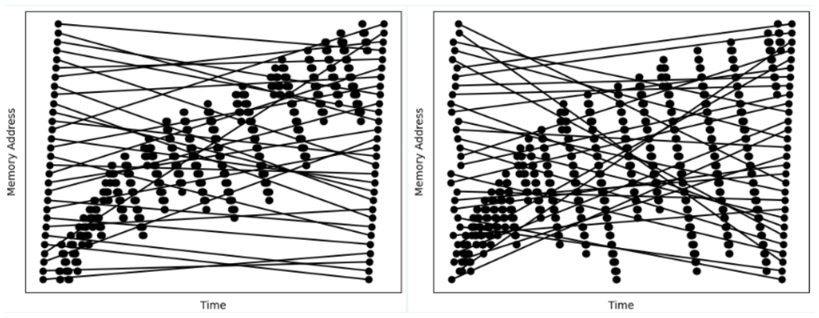

Figure 2.3: **Message Buffer Access Patterns.** Message buffering access patterns in Loopix (left) vs Mixclaves (right). Loopix has a a correlation coefficient of 0.738 from input to output address accesses, whereas Mixclaves is 0.03.

Oblivious Data Structures

Oblivious data structures conceal not only their referents, but also access patterns that could leak confidential information. For example, an oncologist checking health records could not only reveal that a patient has cancer, but access frequency may also reveal information about its type and severity. While metadata-private applications must conceal access frequency and timings of persisted data, Mixclaves have a comparably straightforward workload to obscure. Abstractly, each packet is inserted into a queue ordered by a deadline and dequeued some time after its deadline expires. Packets present in the queue, including packets inserted later, are in its anonymity set.

With access to the machine, an adversary could correlate packets entering the mix node with memory access patterns. The contents of enclave memory are encrypted, but the addresses are not and may be tracked. Incoming packets are buffered by Twisted, an event-driven network engine that powers Mixclaves and our reference mixnet, Loopix [46]. We examined the messages stored between processing and sending. Twisted appends new messages generated by a task to the end of an array. When the task completes, the array is sorted by its target execution time and the next task is dequeued from the front. Loopix and Mixclaves messages are delayed by a random value chosen from an exponential distribution (see Section 4.1). This process rearranges some of the messages in the buffer as it is periodically sorted, but there remains a strong, positive correlation between the input and output addresses. In Figure 2.3, we examine memory accesses to a message buffer with storage algorithms from Loopix and Mixclaves. We see the messages in Loopix are highly correlated (R=0.738) from the address they are stored at upon receipt and where they are accessed at for sending. This is especially problematic if two users exchange significant traffic; then it becomes possible to correlate that the two are communicating, revealing metadata about the communication.

We altered Twisted's packet buffer to obliviously store messages to prevent leaking memory access correlations. When inserting a new message, we randomly select an index from a uniform distribution across the length of the array and insert at that location. In the case of a conflict, the random index is preserved and all entries below the index are shifted to the right. The sort of the buffer is performed on batches of messages to meet these random deadlines.

By adding the oblivious buffer inside an ideal enclave, we can safely assume that memory access patterns and compute is hidden. The sort operation is hidden due to taking place inside the enclave, which provide a confidential compute environment. The oblivious buffer obscures the original memory address of a particular message so that once it is accessed for sending, the packets leaving the enclave cannot be correlated to the original storage address. The correlation coefficient of store vs read memory addresses became R=0.03, as illustrated in Figure 2.3.

This straightforward oblivious data structure is sufficient to evaluate the prototype's viability. One could improve its efficiency and scalability by adopting more sophisticated approaches from the literature [42, 51, 18, 21]. Message dispatch requires only oblivious sending of messages that reached their deadline, not sorting. We leave refinement of these oblivious data structures to future work, particularly for managing larger queues for messages with high delays [28].

Differential Privacy in Mixclaves

With differential privacy as described in Section 2.1, we can prove that we require only 1 mixclave node to provide strong privacy guarantees to the client. Based on previous work in mixnets, a message must route through at least one honest node in the network to be differentially private [9, 37]. Consider a message sent between two clients, A and B. The probability that this condition fails to hold is the probability of meeting 0 honest nodes on a path of length L in the mixnet, i.e. $BinomialCDF(0, L, p)$. And therefore the probability that the condition *does* hold is $1 - BinomialCDF(0, L, p)$. In previous work, the likelihood p of meeting an honest node has always been less than 1. But with Mixclaves, a client will always select an honest node, so $p = 1$. Therefore, Mixclaves always achieves differential privacy, even with a network as small as 1 node.

Chapter 3

Architecture

In this section we describe the Mixclave architecture in detail. Section 4 describes how our prototype implements this architecture to evaluate its practicality.

3.1 System Overview

A Mixclave node is composed of service modules, as shown in Figure 3.1. Nodes expose two public APIs, `accept(packet)` and `config()`. Sphinx packets routed to `accept` are inserted into an oblivious buffer inside the hardware enclave (Section 2.4). Once decrypted inside the enclave, DROP messages are discarded. Unsurprisingly, ROUTE messages are reinserted into the oblivious buffer to be forwarded after the user-configured delay. Cover traffic in Mixclaves are generated following an exponential distribution across all known mix nodes,

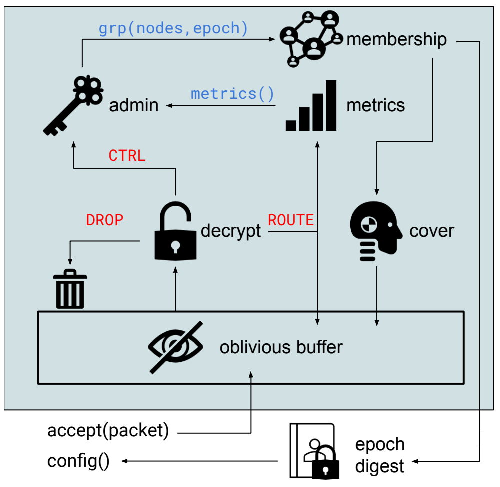

Figure 3.1: **Mixclave Mix Node.**

as in Loopix [46].

What distinguishes a Mixclave node from a traditional mix node are the control packets (CTRL). These packets are signed by an administrator key recorded in the enclave. Control packets can coordinate updates to the domain membership, gather statistics, and change internal configuration state, like parameters for cover traffic. We discuss these in greater detail in the following sections. Once committed, the epoch digest is published through the `config` API.

3.2 Deployment

Mixclaves accept control packets signed by an operator within the enclave. Control packets can collect metrics, change settings, or add and remove nodes from the domain (reconfiguration, see Section 3.3). Control packets appear as normal mixnet traffic outside the enclave, but are distinguished in three ways.

First, control packets are **signed** by an operator whose public key is associated with permissions to the mixnet node. Our prototype does not partition operator capabilities, but one could separate monitoring, scaling, and reconfiguration across keys, or even include a "poison pill" that destroys the mixnode. Second, control packets may contain a **return address**. The mix node has the public key for the operator to verify the control packet, but it needs to know where to route the response. Not all control traffic generates a response. For example, one could implement a "warrant canary" packet that affirms the operator has *not* been forced to disclose information by means that are illegal for it to acknowledge [58]. Third, control packets may **read metrics** from nodes. Normal packets can push updates, but cannot read metric data. Control packets querying metrics are written and routed through the broader mixnet back to the return address. In the following section, we use this channel to effect an automatic scaling policy from within the mixnet.

Control packets may be sent either by an operator or by other Mixclave nodes in the domain. Traffic loops could gather not only the round-trip time from a node, but also reliable observations from other nodes in the domain. Most importantly, all the operational traffic is concealed by the cover and real traffic to the enclave. Building a control plane in traditional mixnets would require one to adopt Byzantine protocols and the overheads of running at least $3f + 1$ control nodes, as in systems like OceanStore [50, 34]. By adopting hardware enclaves and adopting a federated trust model, we can operate a system with similar or stronger guarantees at a fraction of the cost (see Table 5.1).

3.3 Reconfiguration

To meet our goals for elasticity, an operator node must be able to add and remove other mix nodes. Since mixnets offer only best-effort delivery and persist no state, applications written for mixnets tolerate data loss; we are principally concerned with preserving anonymity during

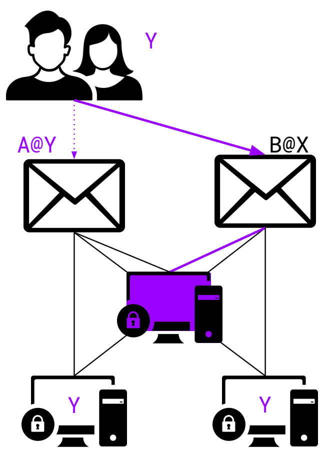

Figure 3.2: **Reconfiguration.** Messaging client C and provider A are at an epoch Y, provider B is at epoch X, $X \subset Y$. C changes providers from A to B, but since B sends no cover traffic to the new node, traffic to N must be from C.

and after a reconfiguration. When nodes are added to a mixnet, cover traffic must include the new nodes before user traffic can be routed to it. In our architecture, an application is responsible for ensuring its own cover traffic was added to the mix before informing its clients, if necessary.

To see why this may be necessary, recall the provider nodes in the Loopix messaging service generate cover traffic in the mixnet. Before informing users, a Loopix service could first require consensus among its providers on the new nodes to ensure cover traffic includes them uniformly; mix nodes also generate noise, but they cannot cover asymmetric traffic from application services. Consider the scenario in Figure 3.2 with no cover traffic. Assume the epoch changes from X to Y and Y adds a new node N. If a client changes providers from A at epoch Y to B at the earlier epoch X, then all traffic addressed to N from B will be client traffic[1]. Whether the cover traffic generated by the client and mixnet is sufficient is a choice for the application and domain operator and outside the purview of the mixnet.

Reconfiguration can be subtle [2], particularly when an operator supports concurrent reconfigurations from multiple operators or avoids pauses during reconfiguration (*online* reconfiguration). For example, virtual synchrony [11] maintains both group membership within a view and offers a reliable broadcast service, but typically pauses to install a new view. With an administrative domain composed of widely distributed, small clusters of machines [59] may make different tradeoffs than one running in well-connected datacenters [8].

While we ruled out Byzantine behavior in the threat model from Section 2.3, attacks on hardware enclaves continue to evolve and could include a full architectural break. Even if

[1]There are many fixes the application could apply, such as B dropping routes to unknown mix nodes or forcing refresh at B, but both of these policies are affected by the implementation of mixnet reconfiguration in the domain.

an operator excluded the node, if a node can be prevented from shutting down, even when the node is isolated from other nodes in its domain, a client with an outdated view of the network could validate a compromised, Byzantine mix node. One solution could include a signed, timestamped lease recording the last contact with the operator. The client could use the lease to detect parts of the network abandoned by the operator or unreachable during reconfiguration[2], though rendering the domain unusable when an operator misses a check-in or asking a client to set a threshold for staleness both harm usability.

Selecting a particular reconfiguration algorithm entails material tradeoffs for a mixnet operator. In a federated model, the operator of a domain can select an algorithm for reconfiguration independently without affecting or informing the rest of the federation.

We do constrain reconfiguration by requiring that each mix node provide an API exposing the list of nodes, public keys, and optional node metadata with which it is configured. This *epoch digest* must be timestamped, versioned, and signed by the operator or by an infrastructure key that committed the reconfiguration. The epoch digest should also be signed by the mix node, including a nonce chosen by the client. Versions do not need to be totally ordered, but they must form a join semilattice such that concurrent reconfigurations have a deterministic merge function [2]. As a consequence, reconfigurations can never be retracted once committed; if an operator wants mix nodes to rejoin the mixnet after a crash (rather than as a new node with a new identity), the mix node must persist the digest before installing a new configuration. Epoch digests are public and can be used both by clients and also by other domains.

[2]This could also serve as a "canary" for a compromised network.

Chapter 4

Implementation

In this section we describe the prototype implementation of the architecture in Section 3 to inform the evaluation in Section 5.

4.1 System Overview: Loopix

Our prototype for Mixclaves extends Loopix [46]. Loopix is a message-based mixnet that adds a random delay to every layer of encryption to confound traffic analysis. The random delay is drawn from an exponential distribution. Loopix uses a *stratified topology* for its traffic. The mixnet is separated into layers such that each node is connected only to the mix nodes in adjacent layers; traffic flows in one direction.

Internally, Loopix uses the Twisted network engine [17] not only to schedule delivery of UDP packets, but also to schedule periodic system functions. As discussed earlier in Section 2.3, Loopix includes a *provider* node responsible for generating cover traffic and hosting mailboxes for intermittently connected users of its messaging service. Clients poll the provider at regular intervals for messages, retrieving authentic messages for the user, drop packets generated by other clients, and synthetic messages generated by the provider. When clients exchange messages, the path includes their respective providers on either side of the mixnet.

Mixclaves are intended as a general-purpose mixnet, so we focus our evaluation on its peak throughput, overheads on latency, and operating cost. To measure these across applications, we modify the Loopix implementation by merging its client and provider into a load generator that measures round-trip latency of self-addressed paths through a mixnet. Measuring Mixclave packet latency rather than Loopix message latency eliminates the client's polling loop its mailbox at the provider. We also eliminate application-specific noise generated by the provider so we can measure the aggregate throughput of the network using only benchmark packets and intra-mixnet noise.

Our prototype does not enforce a topology among mix nodes, so clients may choose any path. A malicious client could encode up to 20 hops before the Sphinx format cannot encode it given the target message size, but this is a mild optimization of a denial-of-service attack when the Sphinx library can generate 360M three-hop messages per second on a GCP `n2d-standard-2` machine. Since clients will choose short routes uniformly distributed among nodes in a domain, a fully connected topology allows Mixclaves to expand elastically and uniformly receive load as clients learn of new resources.

We modify the message buffer for the Twisted network engine as described in Section 2.4 so messages in the send buffer cannot be correlated. We did not implement a signature scheme for control packets, as the overhead of validation is negligible in our evaluation.

To demonstrate both anonymous administration and scaling, we implement an *updater* by creating a variant of the Loopix client. The updater samples metrics using the covert control plane described in Section 3. The updater is also responsible for informing the mixnet of a new epoch when nodes are added to the domain. Benchmark clients are informed by the mixnet via a polling loop. Anonymously gathering statistics, resizing the mixnet, and informing clients that refresh their topology to include new nodes is sufficient as a proof of concept of the scalable Mixclave architecture.

An *epoch* is a monotonically increasing counter used to identify the set of mix nodes in the domain. Nodes are added to the cluster in a particular epoch. The set of public keys for generating synthetic traffic and authenticating messages is stored locally in a database, versioned by epoch. In our prototype, the node starts with a database populated for its epoch. If the update fails to converge, the updater will refresh and retry adding its nodes at the next-highest epoch. Since the updater for our cluster is outside the mixnet, if it crashes during reconfiguration then it may leak resources. If the updater receives acknowledgements from all nodes, then it updater can commit the result by sending a message to the mixnet nodes. Once committed, any mixnet can publish the new set of nodes in that epoch. Any mix nodes that do not learn the epoch from the updater can learn that it is active when client traffic in that epoch is routed to it.

4.2 Deployment

Our prototype uses Docker [23] and Terraform [29] to automate deployment and the initial setup. Terraform not only provisions VMs for each entity, but it also configures a private overlay network (VPC) among the nodes in the mixnet. We also use Terraform to allocate new nodes and include them in the mixnet VPC. Docker adds some overhead that adversely impacts overall performance [14], but relative comparisons of mix nodes operating in and out of enclaves should be comparable. As shown in Figure 5.1, while these tools greatly simplified our packaging and deployment, they also added significant provisioning overhead and consequently, hysteresis to scaling decisions. We will explore strategies for reducing allocation overheads in future work.

4.3 Reconfiguration

Our prototype uses control packets to update mixnode membership. Rather than implementing reliable broadcast or distributed consensus, we demonstrate the control path using a simple gossip protocol. We implement the updater client described in Section 3.1 by polling the mixnet node(s) with a control packet querying metrics, including a return address for the updater. The node generates a reply packet with its current throughput using the updater's public key. We use the throughput measured in Section 5 to populate a table for scaling thresholds. If the throughput exceeds a threshold, the updater will allocate a new node to the mix net. Once configured, the updater generates a new database for the epoch and copies it to the mix nodes.

The updater then repeatedly sends control packets to mix nodes, instructing them to install the database assigned to that epoch. If the mix node is at an epoch below the threshold, it will attempt to load the database for that epoch. If successful, it generates a response packet to the updater. Clients query the epoch digest, which in our prototype is implemented using the same, straightforward notification protocol for the current epoch.

While straightforward, this reconfiguration procedure demonstrates that Mixclaves can pass information in and out of the enclave, use those data to make scaling decisions, and conceal its reconfiguration traffic with cover and benchmark traffic. Populating updates to the database with control packets, implementing signing and certificates, and fault-tolerant protocols we leave to future work.

Chapter 5

Evaluation

In this section, we measure the performance of Mixclaves and compare against our reference implementation. We investigated the performance of Mixclaves running in Google Cloud Platform (GCP). Each mixnode and load generator ran in its own VM. Except where explicitly noted, we use N2D instances in the `n2d-standard-2` profile with 2 vCPUs, 8 GB of RAM, and 10 Gbps of bandwidth. These machines are configured with AMD SEV-SNP enclaves, so the entire VM runs within the enclave. The OS was Ubuntu 22.04, and the provisioning was automated with Terraform and Docker.

Provisioning time not only from the provider but for our packaging is significant, even after tuning. Since these overheads are not critical for our evaluation, and general techniques for reducing these overheads are well-known [47], we pre-allocate a pool of VMs where elasticity applies. Our measurements varied significantly, but we show the rough breakdown of allocation cost in Figure 5.1.

Recall from Section 4.1 our benchmark client that creates self-addressed loops through the mixnet. Benchmark packets include an experiment identifier, timestamp, and sequence

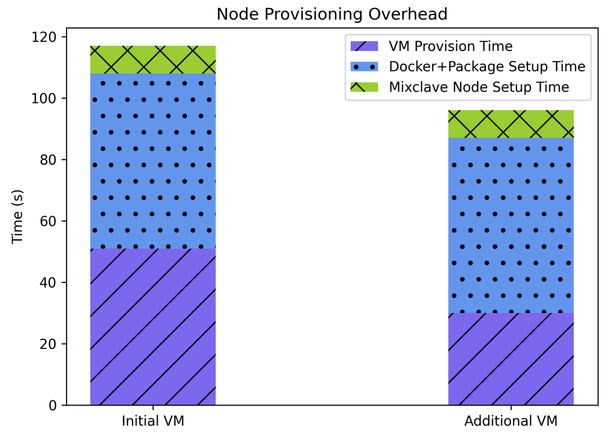

Figure 5.1: **Breakdown of salient platform and framework provisioning costs.**

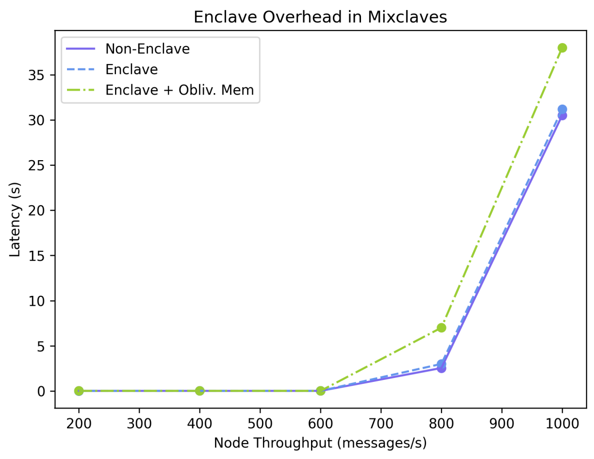

Figure 5.2: **Single Node Microbenchmark.** Throughput vs latency for a single node. Running in enclaves reduces peak throughput by around 200 messages per second.

number. We draw the delay per layer of encryption from an exponential distribution around 1ms in all experiments unless noted. The client records the round-trip time (RTT) of these loops as the latency of the mix net, which should match applications' experience. Since latency is recorded by the same process, we avoid any issues with clock drift across machines. If the latency exceeds a configurable threshold then the packet is recorded as dropped. Late-arriving packets also report latency using the embedded timestamp.

5.1 Microbenchmarks

Enclave overhead

To demonstrate that the enclave would not become a bottleneck, we ran a network benchmark to saturation in and outside of enclaves using `iperf3`. We omit the graph for space, but the latency at lower throughput was indistinguishable and peak throughput saturated above 9.73Gbps outside the enclave and 9.55Gbps within it. This tranquilized any anxiety around the enclave implementation creating a network bottleneck for the Mixclave prototype. Docker did add a significant overhead for network traffic running in an enclave, reducing measured throughput to 4.98Gpbs.

Single Node

Figure 5.2 shows the throughput/latency for a single mix node running outside the enclave and within the enclave, with and without oblivious mitigation enabled (Section 2.4). The

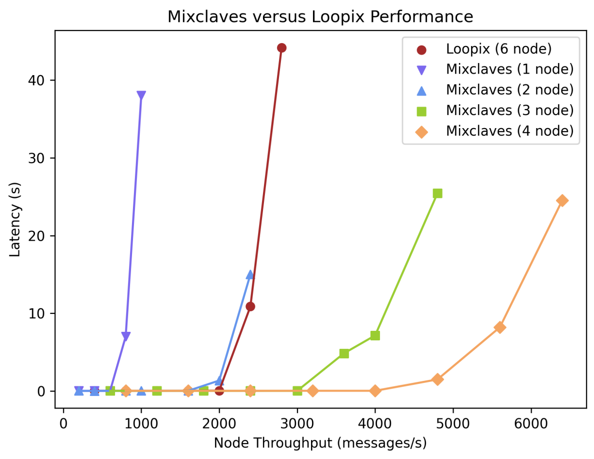

Figure 5.3: **Network Throughput.** A network of six Loopix nodes saturates below 2400 messages per second (3 hops), but Mixclave cluster of three nodes saturates above 3000 messages per second at 54% of the cost.

oblivious buffer adds overhead that lowers peak throughput per node. The enclave adds only a slight overhead for the low-latency workload, as expected.

5.2 Scalability

To evaluate the scalability of Mixclaves, we measure throughput of the mix network to saturation in multiple, fixed configurations, illustrated in Figure 5.3. The six-node mixnet (labeled "Loopix") records the round-trip latency through a three-hop network of non-enclave machines. We increase the load on one through four Mixclaves nodes until saturation, recording the round-trip latency and with drop logic disabled. The oblivious buffer is disabled for the "Loopix" experiments, which run with the Mixclaves topology rather than the stratified topology of the original system.

Despite slightly lower performance per node, a Mixclave network supports *higher throughput at lower cost* than a multi-hop network. Decreasing packet path length more than overcomes the enclave overhead.

5.3 Elasticity

To demonstrate support for reconfiguration (Sections 3.3 and 4.3), we gradually increase client traffic and show that Mixclaves throughput increases beyond saturation points mea-

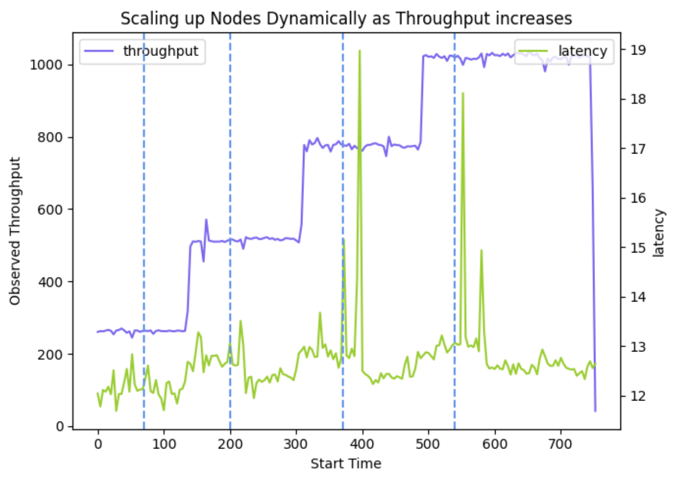

Figure 5.4: **Elasticity Experiment.** Adding nodes as throughput increases. A node is added to the mixnet at every dashed line, preserving latency.

Mixnet	# Nodes	Peak Throughput (msg/s)	Monthly Cost ($)
Loopix	6	< 2400 / s	$296.04
Mixclaves	1	< 800 / s	$53.34
Mixclaves	2	< 2000 / s	$106.68
Mixclaves	3	< 3600 / s	$160.02
Mixclaves	4	< 4800 / s	$213.36

Table 5.1: **Cost and throughput comparison of Loopix to two different Mixclave topologies.**

sured in Section 5.2. As shown in Figure 5.4, during reconfiguration throughput sometimes fluctuates for reasons as yet unexplained, but it recovers quickly.

5.4 Cost

As of this writing, running a cloud VM in an enclave adds an 8% cost premium per node. Table 5.1 records the costs of operating a mix network on a fixed set of nodes, per month in GCP [48]. As shown in Figure 5.3, at or above the peak throughput the mix network saturates and drops traffic to avoid collapse. The figure also suggests savings available to an elastic network that can adapt to load, rather than provision for peak traffic.

Enclaves set a higher minimum price for compute resources. Public clouds offer less expensive VMs, particularly if an operator is willing to accept preemption of those resources in spot instances and preemptable VMs [48]. This is attractive as mix nodes persist no state, but an "honest" mix node cannot be fully stateless without accepting caveats for mix network availability. If the node changes its identity on restart due to preemption, this is akin to failure and/or reconfiguration of the mix network (Section 3.3) with all its attendant complexity and impact on mix network throughput. The costs recorded in Table 5.1 reflect monthly costs for comparable service.

5.5 Reconfiguration

At our scale, reconfiguration of the mixnet is near-instantaneous after provisioning is completed. Dissemination of the updated epoch digest to clients will vary depending on the application, as discussed in Section 3.3. Our handful of benchmark clients usually noted and applied the new node configuration in less than a second, though recall from Section 4.3 that the updated epoch digest is already copied locally; the update packet only records which epoch the client should load for generating benchmark traffic.

Our experiments did reveal a challenge to concealing administrative traffic to *overloaded nodes*. Services often expose a high-priority channel reserved for administrative traffic— even running on a dedicated port— to ensure admin commands are processed promptly and in a coherent order. These strategies are unavailable in our setting, as admin traffic is designed to look identical to real traffic. Our updater resends the update message to all nodes until it receives an acknowledgment, but in heavily overloaded Mixnet networks with dropping enabled, some reconfiguration attempts still failed.

Chapter 6

Related Work

Circuit mixnets like Real-Time Mixes [33], Vuvuzela [55], and Groove [9] route traffic along a consistent path, rather than routing every message independently. Circuits can also be used to build continuous, low-latency channels through a mixnet [33] for telecommunications. Similar to Tor [22], these metadata-private messaging systems connect endpoints using bidirectional circuits through the mixnet. Pairs of endpoints rotate circuits periodically based on a shared secret. In Groove, messages are not delivered directly between endpoints but rather to an intermediate *provider* node. Provider nodes solve the problem of intermittently-connected clients. Any application using a mixnet for anonymity must ensure that signal is buried in noise; little is accomplished if an attacker can observe multiple rounds where Alice sends exactly k messages into the mixnet and Bob subsequently fetches k messages. Providers not only generate synthetic traffic into the mixnet that simulates the application workload when clients are disconnected, they also ensure traffic between clients confounds traffic analysis.

Metadata-private messaging services like Groove group messages in rounds, ensuring that every circuit in the network has at least one message in each round. The 30 second message latency in Groove and other systems in its pedigree is not inherent to circuit mixnets, but an application decision. To ensure at least one honest node even in highly corrupt networks (20% adversarial), Groove routes can exceed 11 hops. Trust in enclaves simplifies our architecture and significantly reduces costs.

The Tor [22] network consists of a network of relay nodes and a directory service. At regular intervals on the order of minutes, users select a sequence of nodes from the directory based on configurable weights for desired bandwidth, accepted ports, and other criteria. The user creates a circuit through which their traffic is routed, with consistent entry and exit nodes. Tor has been vulnerable to analysis based on non-uniform packet sizes and timing attacks [15, 7]. Mixnets are designed to resist exactly this traffic analysis.

ConsenSGX is a technique for the Tor network to leverage secure enclaves and oblivious RAM to hide which parts of the Tor network a particular client knows about. It addresses epistemic attacks, where an adversary observes a directory server that distributes the global view of Tor nodes and then learns which parts of the network a client does not know about.

This would allow the adversary to rule out those servers from carrying that client's traffic (and conversely know more about which nodes are routing it). If the directory servers use secure enclaves and oblivious RAM, they can provide the the network view to clients without leaking what the client has learned. In Mixclaves, we also prevent epistemic attacks with the signed epoch digest, which enables clients to see the current network configuration and to not send messages to nodes that are insufficiently covered in noise, are not longer part of the network, etc.

Loopix [46] is a low-latency mixnet that provides time independence using a per-layer delay chosen by the client. The delay is chosen from an exponential distribution such that any packet entering the node since it was last empty is a potential member of its anonymity set. In practice this is vanishingly unlikely and the client chooses delays that satisfy low-latency service-level objectives (SLO) from the mixnet. Binary unlinkability is ensured by the Sphinx [20] packet format, also used by Mixclaves.

All of these systems [33, 9, 55, 46, 16, 15, 22] rely on at least one honest node in the mixnet.

One work in particular, "Towards User Privacy for Subscription Based Services" [10] builds an implementation named "Mixnet" that runs in trusted execution environments (Intel SGX), but this name is something of a misnomer. The system is a proxy service that scrambles user identities before connecting to a third party subscription service. While an interesting use case of enclaves, this work is unrelated to mixnets.

Chapter 7

Discussion

Mixnet Reconfiguration. Our heuristics demonstrate that Mixclave networks can scale out based on confidential, intra-domain metrics, but these cover only a small corner of the workload and cost space. While a mixnet can safely add new nodes, we have not proven that shrinking the cluster is safe; an honest mixnet should generate plausible, diminishing cover traffic to ensure that not all packets arriving at the old address are real traffic. Given an algorithm to remove nodes, the workload could also scale vertically to different VM instances. We did not explore "scale up" regions of the cost space since Twisted makes limited use of multi-core processors.

Oblivious scaling. We did not explore scaling strategies that resist analysis. One would expect diurnal expansions and contractions of the mixnet, but expanding the size of the cluster out of phase could leak information about traffic if the scaling decision were not based solely on externally-visible data.

Health checks. Using intra-domain loops measuring round-trip time, nodes in a domain could gather information about the health of the broader mix network. This could not only inform clients of failed or slow nodes in the network, but also direct clients away from malicious or failed nodes while maintaining cover traffic [39]. Packet delivery through the mixnet is only best-effort, but the operator of a domain can improve reliability for its users with these metrics. Our analysis was limited to scaling based on traffic in a domain, but peering relationships between domains could even define SLAs for loop traffic that cannot be separated from real and cover traffic.

Multipath mixnets. All mixnet formats restrict packets to a single path through the network, as offering mixnodes a choice only helps corrupted nodes direct traffic to an adversary's advantage. Instead, if a symmetric key were encrypted with the public keys of multiple nodes, a trusted Mixclave node could filter out nodes it suspects have failed. The list of nodes (even in other federations) is not confidential, but requiring the client to discover and resynchronize with the network state may be more brittle than granting the forwarding nodes the option. Within a domain, this technique could also load balance among mix nodes based on queue lengths [43]. However, one would need to ensure that packets are only forwarded once, to avoid duplicate packets violating bitwise unlinkability.

Node labels. Vulnerabilities in hardware enclaves such as Plundervolt [44], Foreshadow [54], and ÆPIC Leak [12] undermine a core assumption of the Mixclave architecture. Since the Mixnode will faithfully and accurately report its configuration from the enclave, users may elect to route their traffic through a diverse set of architectures to avoid relying on a single vendor. We assume an attacker already has this information in our threat model, so node labels should grant an adversary no new advantages.

Utility noise. The cover traffic generated by modern mixnets [46, 9] is randomly generated. While Mixclaves only pass control traffic among trusted nodes amid synthetic and user traffic, the ability to examine metadata safely within the enclave also admits the possibility of self-addressed loops of *data* traffic, similar to Broadcast Disks [1]. Data loops could pass through non-enclave nodes and nodes in other domains, with random delays in minutes or hours. Re-purposing packets with high delays as the payload for cover traffic is also admissible; replicate a packet in the mixnet such that fragments arrive approximately when it should be forwarded. Given a time-to-live (TTL) and a forgiving retrieval SLO, one could provide an archive storage service to subsidize the cost and legitimize the purpose of an anonymizing mixnet.

Improved oblivious data structures. Metrics retained for mixnet monitoring could leak information about traffic patterns, if flushed to memory. Metadata derived from message flows within the enclave must resist analysis, though these techniques are beyond the scope of this paper. While our oblivious algorithm worked well for a low-latency workload with shallow queues, an adversary could fill the queue with high-delay packets. Since message dispatch does not require sorting but only oblivious sending of messages that reached their deadline, one could integrate more scalable oblivious data structures for the message queue [51].

Chapter 8

Conclusion

This paper proposes *Mixclaves*, an enclave-backed mixnet for metadataprivate communication. Mixclaves makes it easier to operate mixnets, requiring fewer nodes and enabling automatic scalability to be pushed into the network. It employs oblivious message buffering to mask memory access patterns at a node, enabling a mixnet to operate in a single enclave node while still maintaining differential privacy. This enables Mixclaves to support a centrally managed deployment model. Mixclaves also enables elasticity by allowing the trusted enclave nodes to measure performance characteristics of the network and send control packets to signal a network expansion.

We implemented Mixclaves as an extension of Loopix, which serves as our reference benchmark. We compared the performance of both in VMs running in Google Cloud Platform, demonstrating Mixclaves does see an overhead of the enclave, but that the cost is lower and and overall bandwidth is higher in Mixclaves due to messages routing through fewer nodes with our architecture. Mixclaves also proactively scaled the network at runtime to accommodate higher throughput as needed.

Bibliography

[1] Swarup Acharya et al. "Broadcast disks: Data management for asymmetric communication environments". In: *Mobile Computing.* Springer, 1995, pp. 331–361.

[2] Marcos K Aguilera et al. "Dynamic atomic storage without consensus". In: *Journal of the ACM (JACM)* 58.2 (2011), pp. 1–32.

[3] AMD. *https://developer.amd.com/sev/*. Accessed 2023-01-07. 2022.

[4] Ittai Anati et al. "Innovative technology for CPU based attestation and sealing". In: *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy.* ACM. 2013.

[5] Elli Androulaki et al. "PAR: Payment for anonymous routing". In: *International Symposium on Privacy Enhancing Technologies Symposium.* Springer. 2008, pp. 219–236.

[6] Jacob Appelbaum et al. *NSA targets the privacy-conscious.* https://daserste.ndr.de/panorama/aktuell/nsa230_page-1.html. 2014.

[7] Michael Backes et al. "(Nothing else) MATor (s) Monitoring the Anonymity of Tor's Path Selection". In: *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security.* 2014, pp. 513–524.

[8] Hari Balakrishnan et al. "Revitalizing the Public Internet by Making It Extensible". In: *SIGCOMM Comput. Commun. Rev.* 51.2 (May 2021), pp. 18–24. ISSN: 0146-4833. DOI: 10.1145/3464994.3464998. URL: https://doi.org/10.1145/3464994.3464998.

[9] Ludovic Barman et al. "Groove: Flexible Metadata-Private Messaging". In: *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22).* 2022, pp. 735–750.

[10] Allan Benelli. "Towards User Privacy for Subscription Based Services". MA thesis. ETH Zurich, 2022.

[11] Ken Birman and Thomas Joseph. "Exploiting virtual synchrony in distributed systems". In: *Proceedings of the eleventh ACM Symposium on Operating systems principles.* 1987, pp. 123–138.

[12] Pietro Borrello et al. "ÆPIC Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture". In: *31st USENIX Security Symposium (USENIX Security 22).* 2022, pp. 3917–3934.

[13] Albert Fox Cahn and Jake Laperruque. *Putting a price on privacy: Ending police data purchases.* `https://www.protocol.com/government-buying-location-data`. Accessed 2023-01-08. 2020.

[14] Emiliano Casalicchio and Vanessa Perciballi. "Measuring Docker Performance: What a Mess!!!" In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion.* ICPE '17 Companion. L'Aquila, Italy: Association for Computing Machinery, 2017, pp. 11–16. ISBN: 9781450348997. DOI: `10.1145/3053600.3053605`. URL: `https://doi.org/10.1145/3053600.3053605`.

[15] David Chaum et al. "cMix: Anonymization by high-performance scalable mixing". In: *USENIX Security.* 2016.

[16] David L Chaum. "Untraceable electronic mail, return addresses, and digital pseudonyms". In: *Communications of the ACM* 24.2 (1981), pp. 84–90.

[17] Twisted Community. *Twisted.* `https://www.twisted.org/`. 2022.

[18] Natacha Crooks et al. "Obladi: Oblivious serializable transactions in the cloud". In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18).* 2018, pp. 727–743.

[19] George Danezis. *The Sphinxmix python package.* https://github.com/UCL-InfoSec/sphinx. 2022.

[20] George Danezis and Ian Goldberg. "Sphinx: A compact and provably secure mix format". In: *2009 30th IEEE Symposium on Security and Privacy.* IEEE. 2009, pp. 269–282.

[21] Emma Dauterman et al. "Snoopy: Surpassing the scalability bottleneck of oblivious storage". In: *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles.* 2021, pp. 655–671.

[22] Roger Dingledine, Nick Mathewson, and Paul Syverson. *Tor: The second-generation onion router.* Tech. rep. Naval Research Lab Washington DC, 2004.

[23] Inc Docker. *Docker.* `https://docker.com`. 2022.

[24] Cynthia Dwork, Moni Naor, and Amit Sahai. "Concurrent zero-knowledge". In: *Journal of the ACM (JACM)* 51.6 (2004), pp. 851–898.

[25] Cynthia Dwork and Aaron Roth. "The Algorithmic Foundations of Differential Privacy". In: *Found. Trends Theor. Comput. Sci.* 9.3–4 (Aug. 2014), pp. 211–407. ISSN: 1551-305X. DOI: `10.1561/0400000042`. URL: `https://doi.org/10.1561/0400000042`.

[26] Lee Ferran. *Ex-NSA Chief: 'We Kill People Based on Metadata'.* `https://abcnews.go.com/blogs/headlines/2014/05/ex-nsa-chief-we-kill-people-based-on-metadata`. 2014.

[27] Eva Galperin, Kurt Opsahl, and Dia Kayyali. *Dear NSA, Privacy is a Fundamental Right, Not Reasonable Suspicion.* `https://www.eff.org/deeplinks/2014/07/dear-nsa-privacy-fundamental-right-not-reasonable-suspicion`. 2014.

[28] Christian Gottel et al. “Security, Performance and Energy Trade-Offs of Hardware-Assisted Memory Protection Mechanisms”. In: *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. IEEE. 2018.

[29] HashiCorp. *Terraform*. `https://www.terraform.io/`. 2022.

[30] Apple Inc. `https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf`. (Accessed 2022-12-13). 2022.

[31] Apple Inc. *iCloud Private Relay Overview*. `https://www.apple.com/privacy/docs/iCloud_Private_Relay_Overview_Dec2021.PDF`. Accessed 2023-01-08. 2021.

[32] Cloudflare Inc. *Edge computing*. `https://www.cloudflare.com/learning/serverless/glossary/what-is-edge-computing/`. Accessed 2023-01-11. 2023.

[33] Anja Jerichow et al. “Real-time mixes: A bandwidth-efficient anonymity protocol”. In: *IEEE Journal on Selected Areas in Communications* 16.4 (1998), pp. 495–509.

[34] John Kubiatowicz et al. “Oceanstore: An architecture for global-scale persistent storage”. In: *ACM SIGOPS Operating Systems Review* 34.5 (2000), pp. 190–201.

[35] United States. Department of Labor. Employee Benefits Security Administration. *Health Coverage Portability: Health Insurance Portability and Accountability Act of 1996 (HIPAA)*. US Department of Labor, Employee Benefits Security Administration, 2004.

[36] Will Law. *Performance-under-Privacy: Delivering Commercial Streaming Content in a Privacy-First World*. `https://youtu.be/MLBvQ9MJawE?t=3892`. 2022.

[37] David Lazar, Yossi Gilad, and Nickolai Zeldovich. “Karaoke: Distributed private messaging immune to passive traffic analysis”. In: *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 2018, pp. 711–725.

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

[39] Hemi Leibowitz et al. “No right to remain silent: isolating malicious mixes”. In: *28th USENIX security symposium (USENIX security 19)*. 2019, pp. 1841–1858.

[40] Mengyuan Li et al. “CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel”. In: *30th USENIX Security Symposium (USENIX Security 21)*. 2021, pp. 717–732.

[41] Li Lu et al. “Pseudo trust: Zero-knowledge authentication in anonymous P2Ps”. In: *IEEE Transactions on Parallel and Distributed Systems* 19.10 (2008), pp. 1325–1337.

[42] Pratyush Mishra et al. “Oblix: An efficient oblivious search index”. In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, pp. 279–296.

[43] Michael Mitzenmacher. “The power of two choices in randomized load balancing”. In: *IEEE Transactions on Parallel and Distributed Systems* 12.10 (2001), pp. 1094–1104.

[44] Kit Murdock et al. "Plundervolt: Software-based fault injection attacks against Intel SGX". In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 1466–1482.

[45] Trevor Perrin and Moxie Marlinspike. "The double ratchet algorithm". In: *GitHub wiki* (2016).

[46] Ania M Piotrowska et al. "The loopix anonymity system". In: *26th USENIX Security Symposium (USENIX Security 17)*. 2017, pp. 1199–1216.

[47] Google Cloud Platform. *Guides: Create custom images*. `https://cloud.google.com/compute/docs/images/create-custom`. 2022.

[48] Google Cloud Platform. *VM Instance Pricing*. `https://cloud.google.com/compute/vm-instance-pricing`. Accessed 2022-12-09. 2022.

[49] Protection Regulation. "Regulation (EU) 2016/679 of the European Parliament and of the Council". In: *Regulation (eu)* 679 (2016), p. 2016.

[50] Sean C Rhea et al. "Pond: The OceanStore Prototype." In: *FAST*. Vol. 3. 2003, pp. 1–14.

[51] Sajin Sasy, Aaron Johnson, and Ian Goldberg. "Fast Fully Oblivious Compaction and Shuffling". In: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2022, pp. 2565–2579.

[52] Sudheesh Singanamalla et al. "Oblivious dns over https (odoh): A practical privacy enhancement to dns". In: *arXiv preprint arXiv:2011.10121* (2020).

[53] Johns Hopkins Foreign Affairs Symposium. *The Price of Privacy: Re-Evaluating the NSA*. `https://youtu.be/kV2HDM86XgI?t=1073`. 2014.

[54] Jo Van Bulck et al. "Foreshadow: Extracting the Keys to the Intel {SGX} Kingdom with Transient {Out-of-Order} Execution". In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018, pp. 991–1008.

[55] Jelle Van Den Hooff et al. "Vuvuzela: Scalable private messaging resistant to traffic analysis". In: *Proceedings of the 25th Symposium on Operating Systems Principles*. 2015, pp. 137–152.

[56] Yingchen Wang et al. "Hertzbleed: Turning Power {Side-Channel} Attacks Into Remote Timing Attacks on x86". In: *31st USENIX Security Symposium (USENIX Security 22)*. 2022, pp. 679–697.

[57] WhatsApp. *WhatsApp Encryption Overview*. `https://www.whatsapp.com/security/WhatsApp-SecurityWhitepaper.pdf`. 2016.

[58] Wikipedia. *Warrant Canary*. `https://en.wikipedia.org/wiki/Warrant_canary`. 2022.

[59] Adam Wolbach et al. “Transient Customization of Mobile Computing Infrastructure”. In: *Proceedings of the First Workshop on Virtualization in Mobile Computing.* MobiVirt ’08. Breckenridge, Colorado: Association for Computing Machinery, 2008, pp. 37–41. ISBN: 9781605583280. DOI: `10.1145/1622103.1622108`. URL: `https://doi.org/10.1145/1622103.1622108`.