Enhancing Privacy and Security on the Extensible Internet

William Lin

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2022-75
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-75.html

May 12, 2022

Acknowledgement

I would like to thank Professor Scott Shenker and the other members of the Extensible Internet group for their guidance and collaborative efforts throughout my research. I would also like to thank my friends and family for their unconditional support during my studies.

Enhancing Privacy and Security on the Extensible Internet

by William Lin

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:

Professor Scott Shenker
Research Advisor

May 12, 2022

(Date)

* * * * * * *

Professor Aurojit Panda
Second Reader

April 29, 2022

(Date)

Enhancing Privacy and Security on the Extensible Internet

William Lin
University of California, Berkeley
will.lin@berkeley.edu

Abstract

The Extensible Internet (EI) is an initiative to transform the Internet architecture by introducing a new Interposition Layer between L3 and L4 for services to be deployed on. Backed by verifiably secure infrastructure, EI provides new opportunities to improve the privacy and security of Internet clients. In this work, we address two challenges that the Internet faces today: maintaining privacy in DNS lookups and ensuring the integrity of third-party application servers. Specifically, we design and implement services for EI to facilitate oblivious DNS and remote attestation verification. We evaluate their execution and demonstrate how EI can help mimic and deploy these proposals without significantly compromising on performance.

1 Introduction

The architecture of the Internet provides a primary service model of best-effort packet delivery and has largely remained static. Unfortunately, this service model alone is insufficient for the extensive use cases of the modern Internet, which consist of content and service-oriented applications for mobile clients. Furthermore, the Internet lacks fundamental security features to protect client privacy and security.

Private network operators have deployed in-network services that provide additional functionalities, like caching and load balancing, to accommodate for the Internet's lack of features. However, these services are restricted to private networks and are proprietary, making them incompatible with the rest of the Internet. The explosion of in-network services threaten to relegate the public Internet to a second-class, last-mile delivery service.

The Extensible Internet (EI) [2] is a proposal that aims to bridge the functionality gap between the public Internet and private networks. EI introduces a new Interposition Layer between L3 and L4 for a common set of public services to be deployed on. This layer is supported by commodity machines, called *service nodes*, which are run by traditional network operators.

The verifiably secure infrastructure of EI presents new opportunities to deploy services to increase the privacy and security of Internet clients. In this work, we leverage EI's deployment model to tackle two privacy and security-related challenges that the Internet faces today: (1) performing private DNS lookups and (2) verifying the integrity of third-party application servers.

The first challenge is motivated by the lack of privacy in traditional DNS lookup systems. Clients must send their DNS requests directly to a DNS resolver, which exposes their communication partners and access patterns. A malicious resolver can use this data to create invasive profiles on its clients and potentially sell them to other parties. Oblivious DNS (oDNS) [9] aims to solve this issue by adding an additional forwarding node in between the client and the DNS resolver. However, deploying oDNS requires extra infrastructure to support the additional nodes.

The second challenge is motivated by the practice of publishers outsourcing the deployment of their applications to third-party platforms, like Amazon AWS. As a result, hosting platforms become prime targets of attack for both internal and external attackers. In these environments, remote attestation [3] can add a layer of protection by enabling clients to check the integrity of the application before communicating with it. Unfortunately, adopting remote attestation at a large scale introduces its own issues. For every application that needs to be verified, the client must somehow know the hash of its correct code. Developers must distribute this information to every client on every update through a separate channel (which could be compromised itself). Deploying verifiable applications thus run into significantly scalability and latency issues.

The challenges that oDNS and remote attestation face have limited their adoption on the public Internet. In the following sections, we describe how EI can help address these barriers through services deployed on its platform. We design two services for EI to help facilitate oDNS lookups and remote attestation verification. Finally, we evaluate them to demonstrate their viability in a practical setting.

The remainder of the paper is as follows:

- In Section 2, we provide additional background behind oDNS and remote attestation and the challenges limiting their adoption. We also introduce EI and highlight the opportunities for deploying privacy and security-enhancing services with its framework.
- In Sections 3 and 4, we introduce the high level overview of the oDNS service and the remote attestation service. We highlight their designs and the motivations behind them.
- In Sections 5 and 6, we describe lower-level details of our implementations and present benchmarks showing their performances in different setups.

2 Background and Motivation

2.1 Domain Name System

The Domain Name System (DNS) is used to map human-readable domain names to IP addresses. The issue with traditional DNS name resolution is that clients must compromise their privacy to use the system. They send DNS lookup requests **directly** to a DNS resolver, revealing their communication partners and access patterns. The client can only hope that the DNS resolvers do not abuse this information.

2.1.1 Oblivious DNS

Oblivious DNS (oDNS) [9] addresses the privacy issues of DNS lookups by adding an additional layer of indirection between the client and the DNS resolver. In an oDNS setup, there are three **independent** parties: a client, a forwarder, and a DNS resolver.

An oDNS lookup can be summarized as such:

1. The client encrypts their DNS request such that only the DNS resolver can decrypt it.
2. The client sends the encrypted request to the forwarder, who then forwards it to the DNS resolver.
3. The DNS resolver decrypts the encrypted request to obtain the request contents.
4. The DNS resolver performs a normal DNS name resolution to obtain the DNS response.
5. The DNS resolver encrypts the response such that only the client can decrypt it.
6. The DNS resolver sends the encrypted response to the forwarder, who then forwards it to the client.
7. The client decrypts the response to retrieve the response contents.

If the DNS resolver does not support encrypted DNS requests, then the setup requires an additional node (the oDNS resolver) to sit between the forwarder and the DNS resolver. The oDNS resolver performs cryptographic operations on behalf of the DNS resolver, decrypting requests and encrypting responses.

oDNS protects the privacy of its clients by disassociating the identity of the client from the contents of their requests. The forwarder knows who the client is but does not know what the client is requesting because the DNS request is encrypted for the resolver. The resolver knows what the client is requesting but does not know who the client is because the packet comes from the forwarder. Assuming that the forwarder and resolver do not collude, no single party can construct a profile mapping a client to their requests.

2.1.2 Challenges

Although oDNS addresses the privacy problems that traditional DNS resolution suffers from, there are still challenges slowing its deployment. First, while oDNS can reuse existing DNS infrastructure to perform name resolutions, oDNS still requires extra servers running oDNS code to support the intermediate nodes.

In addition, oDNS is still vulnerable to a sufficiently powerful network adversary, like a government entity. If an adversary can compromise both the forwarder and resolver, then they can easily link a client to their DNS request. Even network adversaries observing traffic going into and coming out of the forwarder and resolver can compromise privacy through timing attacks. For instance, suppose the adversary observes that a forwarder receives an oDNS request and observes a packet going from the same forwarder to a resolver. Any name resolution soon after from the resolver can be linked back (with some probability) to the original client. This attack defeats any privacy gains of oDNS.

2.2 Hardware Enclaves

Third-party applications running on remote servers risk being compromised by malicious attackers, like server operators. Hardware enclaves add an additional layer of protection against such attacks by providing a shielded execution environment for applications to run in. The shielded execution environment maintains the confidentiality and integrity of a program's state, even against a compromised operating system or hypervisor. These guarantees make hardware enclaves an attractive option for securely outsourcing application runtimes to untrusted third-party platforms. Examples of hardware enclaves include Intel's Software Guard Extensions (SGX) [3] and AMD's Secure Encrypted Virtualization (SEV) [4]. For this work, we primarily focus on Intel SGX enclaves due to its extensive literature and existing tooling.

2.2.1 Remote Attestation

Hardware enclaves would be useless if the client is unable to verify that the application is running on a legitimate enclave and that the application code has not been tampered with. As a result, hardware enclaves provide remote attestation capabilities that enable clients to verify these conditions hold. The client interacts with an application if and only if the application has been checked with remote attestation.

For Intel SGX enclaves, the remote attestation process requires the server to return a signed report, called a *quote*, that is generated by its hardware enclaves. The report contains a hash of the running code and is signed by a separate "quoting" enclave. Upon receiving the quote, the client compares the hash against a known, correct hash. The client also sends the quote to a third-party attestation service, called the Intel Attestation Service (IAS), to check that the report is signed by a legitimate Intel SGX enclave. If the hashes match and the signature is IAS-approved, then the client can establish a secure channel with the server.

2.2.2 RA-TLS

RA-TLS [5] merges the Intel SGX remote attestation flow with the Transport Layer Security (TLS) handshake process. During a setup phase, the enclaved application generates a new key pair and computes a remote attestation quote cryptographically bound to the new public key. The application then embeds the quote inside a new, self-signed TLS certificate. During secure channel establishment, a RA-TLS handshake is nearly identical to a TLS handshake. However, instead of checking the certificate with a Certificate Authority, the client instead extracts the embedded quote and performs remote attestation. After verification, the TLS handshake continues as normal.

2.2.3 Challenges

Despite the powerful guarantees of remote attestation on hardware enclaves, there are still challenges with supporting it on a large scale. First, the remote attestation process requires the publisher to distribute up-to-date hashes of the application code to every client that verifies the program. Depending on how often the code is updated and how many clients there are, the deployment process runs into clear scalability and latency issues. The channel that the publisher uses to distribute the hashes may also become a target for attack itself.

Second, for hardware enclaves like Intel SGX, remote attestation requires the client to communicate with a third-party attestation service to validate that the attestation reports are generated by legitimate enclaves. This third-party service may become a bottleneck if many Internet clients rely on it to verify **every** application they run.

2.3 The Extensible Internet

The Extensible Internet (EI) [2] is an initiative to add a new in-network layer, called the Interposition Layer, between L3 and L4. The Interposition Layer introduces new services to the public Internet and are backed by commodity servers, called *service nodes*, on the network edge. Unlike the overlay networks of private networks, the Interposition Layer is tightly knit into the rest of the Internet, with traditional network operators managing service nodes as a part of their infrastructure. All service nodes offer the same set of open source services with limited levels of computation to operate at reasonable speeds.

2.3.1 Threat Model

Because service nodes interact with potentially many clients, they become compelling targets for internal and external attackers. However, EI focuses on protecting against internal attackers, as they are strictly more powerful than external attackers; internal attackers have physical and login access to the service nodes. They consist of entities like malicious administrators and may try to steal sensitive client information through observing machine state or injecting malicious code.

Side channel attacks that gain information through observing I/O or memory access patterns are out of scope, as the noisiness of Internet traffic make these attacks significantly less effective. However, if a service node does not have enough clients, then it could theoretically employ defense mechanisms, like ORAM [7], to mask access patterns from observers.

2.3.2 Securing Service Nodes

Service nodes use hardware to protect against internal attackers. Specifically, they run on processors that support hardware enclaves, trusted platform modules (TPMs), and hardware security modules (HSMs). The hardware enclaves ensure that a malicious actor cannot observe or modify the state of the service node. They also allow clients to verify the service code is trustworthy through remote attestation. The TPMs provide attestation capabilities to authenticate the underlying platform. Finally, the HSMs store and protect secrets, like encryption keys, that are needed for secure communication. These hardware capabilities protect the service node from internal attackers by isolating the service code from the underlying platform. Hardware protections are already available in a number of cloud platforms, making them accessible for EI systems to use.

2.3.3 Opportunities

EI offers a set of secure servers that can provide valuable services to Internet clients without requiring traffic to be routed through private networks. Because services are open source

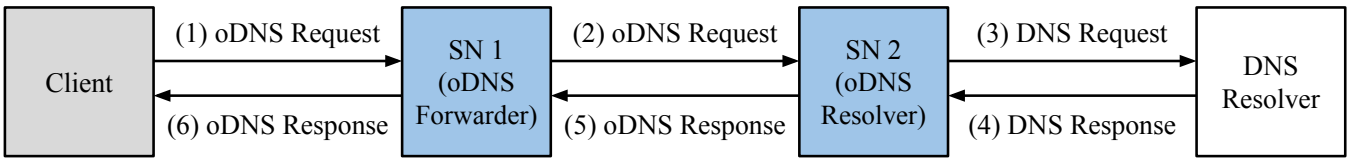

Figure 1: A high level overview of the types of packets transferred during an oblivious DNS lookup on EI.

and service nodes are verifiable through attestation, clients can trust service nodes to act as intermediaries on their behalf to enhance individual privacy and security. Additionally, service nodes can verify one another to form a secure network for propagating data through EI. Finally, service nodes provide services for multiple clients, so there may be opportunities to amortize service costs through methods like caching. In this work, we build on these benefits to implement services that mimic existing proposals and address the challenges limiting their deployment.

3 Oblivious DNS Service

Recall from Section 2.1.2 that a barrier for oDNS adoption is the lack of infrastructure required to support its deployment. EI can help alleviate this concern by providing a set of verifiable machines to deploy oDNS on top of. We construct an EI service similar to the scheme described in [9]. Our oDNS EI service is split into three components: the oDNS client, the oDNS forwarder, and the oDNS resolver. The client machine runs the oDNS client code locally, which helps translate DNS requests into oDNS requests. Service nodes can act as both an oDNS forwarder and an oDNS resolver. However, a service node cannot act as both for a single oDNS request to maintain privacy.

In our designs, we make the following assumptions. First, we assume that there exists a public-key infrastructure (PKI) in EI that allows clients to look up the public key of a specific service node. Second, we assume that the client knows a set of service node pairs, such that one may act as the oDNS forwarder while the other may act as the oDNS resolver. Finally, we assume that all service nodes involved have been verified through remote attestation.

Figure 1 demonstrates the high-level overview of our oDNS deployment on EI. We describe each step of the diagram in the following sections. For our descriptions, we refer to the locally-running oDNS client code as the *client*. We refer to the service node running the oDNS forwarder as the *forwarder*. Because the DNS resolver and the oDNS resolver are different parties, we continue to use their full descriptions to distinguish between them.

3.1 Initiating a Lookup

The client translates traditional DNS requests into oDNS requests supported by EI. To initiate an oDNS lookup, the client first generates a new session key and stores it locally. The client chooses a service node to act as the oDNS resolver and another service node to act as the forwarder. In the selection process, the client ensures that they choose a pair of independent but peered service nodes to prevent collusion between the oDNS resolver and the forwarder.

The client symmetrically encrypts their DNS request with the session key and encrypts the session key with the oDNS resolver's public key. The client concatenates the encrypted session key, the encrypted DNS request, and other metadata (like a request identifier and the target oDNS resolver) to form the oDNS request. The client sends the oDNS request to the forwarder (Step 1).

3.2 Forwarding the Request

Upon receiving an oDNS request, the forwarder records the identity of the sender and maps them to the specific request identifier. Doing so ensures that any oDNS responses with the same request identifier can be routed back to the correct client later on. Finally, the forwarder sends the oDNS request to the oDNS resolver indicated in the request (Step 2).

While the forwarder learns the client's identity, they do not learn the content of the client's actual request. The DNS request is encrypted and the metadata contains only enough information to forward the packet to the correct node.

3.3 Resolving the Request

When the oDNS resolver receives an oDNS request from the forwarder, it uses its private key to decrypt the encrypted session key and uses the session key to decrypt the encrypted DNS request. The oDNS resolver forwards the DNS request to a dedicated DNS resolver (Step 3) to obtain the DNS response (Step 4). While the oDNS resolver can perform the name resolution itself, leveraging existing DNS infrastructure is preferable to avoid unnecessary I/O burdens on the service node.

In this setup, the additional layers of indirection protects the client's identity. Both the oDNS and DNS resolvers learn the client's requests contents but do not learn the client's identity; the oDNS resolver sees the forwarder's IP address,

while the DNS resolver sees the oDNS resolver's IP address. Neither can link the request back to the actual requester.

3.4 Returning the Result

The oDNS resolver encrypts the DNS response with the original session key and sends it back to the forwarder with the same request identifier (Step 5). The forwarder looks up the original requester associated with the request identifier and forwards the packet back to the client (Step 6). Finally, the client decrypts the encrypted response with their locally-stored session key.

3.5 Defenses Against Global Adversaries

In the case of forwarder and oDNS resolver collusion, EI's verifiably secure infrastructure prevents adversaries from observing or modifying the state of the service nodes to trivially connect a client with their requests. However, they may still compromise client privacy by observing network traffic and using timing attacks. For example, if an adversary observes an oDNS request from a client and soon after sees a DNS request from a service node, then they can conclude with some probability that the client sent the request. These types of global attacks are a second challenge that oDNS faces.

While our design does not explicitly defend against such an attack, we can easily add protections to help mitigate their effectiveness in EI. The oDNS resolver can mask actual client requests by generating dummy requests that mimic DNS traffic. Because the oDNS resolver runs in a shielded execution environment, an adversary cannot simply observe the state of or modify the oDNS resolver to determine which requests are fake and are for real clients. The details of how an oDNS resolver can generate convincing traffic are out of scope for this work, as they require a more in-depth analysis into DNS request patterns.

4 Remote Attestation Service

As introduced in Section 2.2.3, the adoption of remote attestation faces two challenges. First, the client must know the most up-to-date hashes of the correct code they wish to verify. Application publishers must distribute the hashes of their applications to every client on every update, which introduces scalability and latency issues to deployment. Second, every client must communicate with a third-party attestation service, like the IAS, to verify the attestation reports. Dependencies on a single third-party service may cause bottlenecks, especially if the service artificially rate-limits the number of attestations supported.

We leverage the deployment model of EI to address these issues. We observe that service nodes can perform remote attestation verification on one another to form a secure network for information to flow through EI. With a secure network, EI makes the distribution of code hashes more scalable; instead of distributing hashes to each client individually, publishers communicate them with a small number of EI servers and let the network distribute them to client-facing nodes. The service nodes then help facilitate remote attestation for their clients through a remote attestation service.

However, clients still need to retrieve the correct hashes of the service node code to use the service. Fortunately, service nodes are not updated as frequently as other applications, so their hashes may be distributed through more expensive but scalable methods, like operating system or browser updates.

For this work, we are not focused on the details of how the hashes propagate through the network. Rather, we are more concerned with how the service nodes can effectively facilitate remote attestation in a service once the hashes are already known by client-facing service nodes. We explore two approaches of a remote attestation service: hash distribution and attestation-checking.

Because the end-goal of remote attestation is for clients to have a secure channel established with an application server, we build our approaches on top of RA-TLS because it integrates remote attestation verification into the TLS handshake process. The RA-TLS handshake abstracts away the lower level details of using remote attestation results in secure channel establishment. However, RA-TLS restricts us to remote attestation for Intel SGX enclaves. Nevertheless, we believe the high-level ideas of our designs translate to other hardware enclaves with similar remote attestation flows.

4.1 Application Setup

On startup, an instance of the application running in an Intel SGX enclave generates a new key pair and keeps it stored in enclave memory. The enclave's shielded execution environment prevents malicious actors from directly retrieving the private key. Using the newly generated key pair, the runtime generates a new RA-TLS certificate as detailed in Section 2.2.2 and in [5].

To ensure security, the private key should not be transferred to parties outside of the enclave. If such restrictions hold, the private key and the associated RA-TLS certificate are bound to the specific instance of the application runtime. If the application is rebooted, then the private key is lost and the new instance must generate another key pair and RA-TLS certificate. While it is possible to use other hardware mechanisms to securely store the enclave secret, we opt to use the simple case of instance-specific keys.

Note that a malicious third-party cannot use an RA-TLS certificate from another enclave to establish a secure channel with a client. Without the associated private key (which is isolated in enclave memory), the RA-TLS handshake cannot complete.

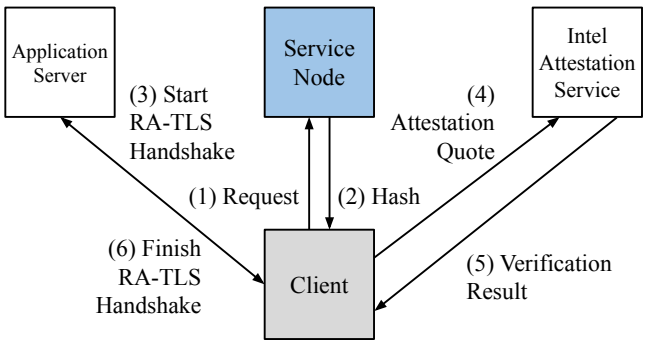

Figure 2: The hash distribution approach of the remote attestation service.

4.2 Hash Distribution Approach

A trivial approach for a remote attestation service is to simply treat each service node as a hash storage and distribution system. Figure 2 demonstrates the communication flow of this approach. When a client wants to perform remote attestation verification, they contact their local service node (Step 1) to retrieve the correct hash (Step 2). The client begins an RA-TLS handshake with the application server (Step 3) to retrieve the RA-TLS certificate containing the attestation quote. The client verifies the quote's legitimacy with the Intel Attestation Service (Steps 4-5) and compares the quote's hash against the trusted hash from the service node. If the hashes match and the quote is verified, then the RA-TLS handshake can complete (Step 6).

This approach addresses the scalability issue of clients needing to have up-to-date hashes for verification; they can now retrieve them from the EI network rather than directly from the developer. However, every client still needs to communicate with the Intel Attestation Service (IAS) every time they wish to perform remote attestation, potentially introducing a bottleneck in the system.

4.3 Attestation-checking Approach

Instead of requiring clients to verify applications themselves, the service nodes can perform remote attestation on behalf of the client. This approach has multiple benefits. First, only service nodes need to regularly communicate with the Intel Attestation Service. Clients infrequently perform remote attestation for their service node and leave the rest to the remote attestation service. Second, the attestation-checking approach can amortize the costs of remote attestation through caching. The idea is that if a specific application instance has been verified with remote attestation, then it does not need to be verified again until a restart. Together, these benefits reduce the reliance on the IAS to eliminate a possible bottleneck.

In this approach, when the client wants to verify an application instance, they communicate with their local service node.

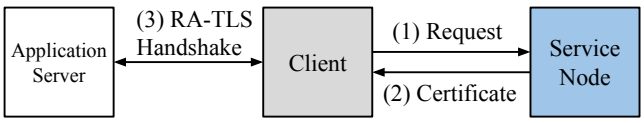

Figure 3: The attestation-checking approach of the remote attestation service, assuming that the service node has previously cached required remote attestation results.

If the service node has not previously verified the application instance, then it performs a new RA-TLS handshake with the untrusted server. If the handshake is successful, then the service caches and returns the RA-TLS certificate obtained. For subsequent verification requests for the same application instance, the service node can immediately return the certificate without repeating the handshake process. Figure 3 shows the communication flow for the cached case.

The client uses the RA-TLS certificate from the service node to establish a separate secure channel directly with the application instance. During the RA-TLS handshake, the client does not perform remote attestation process and can instead use the verified RA-TLS certificate from the service node.

5 Implementation

We implement the oDNS service detailed in Section 3 on top of the most up-to-date version of the EI framework. For our implementation, we use AES-GCM [8] for the symmetric key encryption algorithm for its ability to ensure both the confidentiality and the integrity of the underlying plaintext data. For our public key encryption scheme, we use RSA [6] with 2048 bit keys because of its availability in our cryptographic libraries.

We also implement both approaches (hash distribution and attestation-checking) of the remote attestation service detailed in Section 4. Unfortunately, some limitations with our testing infrastructure has limited us to building the remote attestation service on top of an outdated version of the EI framework. However, we expect that the general conclusions of the evaluation to remain the same. Our services use the Gramine project's [1] implementation of the RA-TLS library for the handshake process.

6 Evaluation

6.1 Oblivious DNS Service

We examine the latencies of name resolutions in both standard DNS and oblivious DNS. Each experiment performs name resolutions for the top 10,000 domains from the Alexa Top Sites list. We focus on learning the latency characteristics of our requests, so we perform name resolutions one after

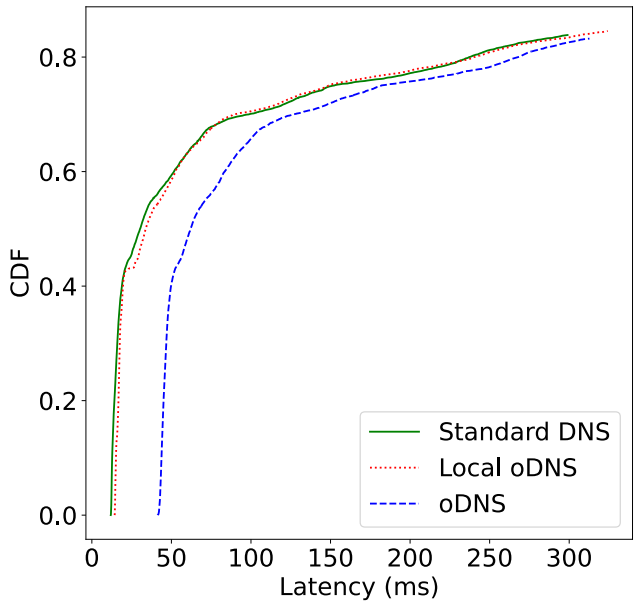

Figure 4: A graph demonstrating DNS lookup latencies with different setups.

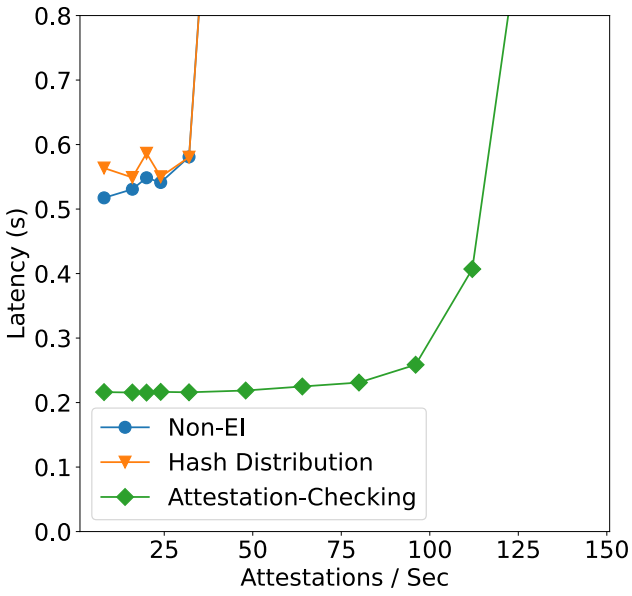

Figure 5: A graph demonstrating how latency changes as we increase the rate of attestation in different experimental setups.

another and record the time between having DNS request and having the DNS response. We compare three setups:

1. **Standard DNS.** The client performs traditional DNS lookups without obliviousness.
2. **Local oDNS.** The client makes oDNS requests as detailed in Section 3. However, the client machine runs the oDNS forwarder and oDNS resolver locally. This setup is not meant to be used in practice but rather serves to demonstrate the overheads of our oDNS service without network latencies.
3. **oDNS.** The client makes oDNS requests as detailed in Section 3. The oDNS forwarder and oDNS resolver are located on remote service node machines. This setup imitates a real-world deployment of the oDNS service.

The machines for the client and the service nodes are hosted by Google Cloud Platform. The client machine runs on a `n2-standard-4` machine with 16GB of RAM. The two service nodes runs on a `n2d-standard-2` machine with 8GB of RAM and confidential computing (AMD SEV) enabled. The oDNS client code, the oDNS forwarder, and the oDNS resolver each run on a single core of their respective machines. We use Cloudflare's recursive resolver (1.1.1.1) to execute the name resolution process for both DNS and oDNS experiments. The round-trip time (RTT) latencies between:

- The **client** and the **oDNS forwarder** is 12-13ms.
- The **oDNS forwarder** and the **oDNS resolver** is 13-14ms.
- The **oDNS resolver** and Cloudflare's **DNS resolver** is 11-12ms.
- The **client** and Cloudflare's **DNS resolver** is 11-12ms.

Figure 4 shows the results of our experiments. We see from our Local oDNS experiments that running oDNS on EI introduces relatively small overheads. The median overhead is approximately 2-3ms, which is mostly attributed to the cryptographic overheads of the public key decryption. The oDNS setup with remote servers demonstrate larger performance differences, with a median overhead of approximately 30-31ms. The primary overhead from this setup comes from the two additional network hops required to resolve oDNS queries. The overheads of running in EI is relatively insignificant compared to the network costs.

6.2 Remote Attestation Service

We evaluate the two approaches to our remote attestation service from Section 4. We vary the rate of attestation and record the latency time in between when the client wants to communicate with an application server and when the client has a TLS connection set up. For each rate of attestation, we perform 512 rounds of verifications following a Poisson arrival process to mimic the spiky behavior of Internet traffic. We compare three setups:

1. **Non-EI**. The client somehow already has the correct hash of the application they wish to verify. They do not use EI at all and perform remote attestation themselves.

2. **Hash Distribution**. The remote attestation service is treated as a hash storage system, as detailed in Section 4.2. At runtime, the client retrieves the correct hash from their service node and performs remote attestation themselves.

3. **Attestation-checking**. The remote attestation service performs remote attestation on behalf of the client, as detailed in Section 4.3. We assume that the clients access the same application instance and that the service node caches remote attestation results.

Our machine setup is as follows. The application servers run in a `Standard_DC2s_v2` virtual machine hosted by Microsoft Azure, using a single core with 8GB of RAM. The service node runs on a `n2d-standard-2` virtual machine hosted by Google Cloud Platform, using a single core with 8GB of RAM and confidential computing (AMD SEV) enabled. We emulate multiple clients on a single `e2-standard-8` virtual machine hosted by Google Cloud Platform, using eight CPU cores and 32GB of RAM. The round-trip time (RTT) latencies between:

- The **client** and the **service node** is 20-21ms.
- The **service node** and the **application server** is 40-41ms.
- The **client** and the **application server** is 30-31ms.

Figure 5 demonstrate the results of our experiments. At lower rates of attestation, the Hash Distribution setup has a latency overhead of approximately 30ms over the Non-EI setup. The slowdown can be attributed to the additional round of communication required for the client to retrieve the correct hash from the service node. The Attestation-Checking setup has the lowest latency and the highest rate of attestation out of the three setups. The Non-EI and Hash Distribution setups have significantly lower throughput because the Intel Attestation Service (IAS) artificially limits the rate of attestation during our experiments. The Attestation-Checking approach does not run into this issue because it is not as reliant on the IAS; remote attestation is performed only once and the cached results are reused for later verifications. Caching also enables lower latencies because the remote attestation process can be skipped altogether.

7 Discussion

Our experiments demonstrate that EI can facilitate the adoption of oDNS and remote attestation without significantly compromising performance. As shown by our oDNS service experiments, the primary latency overheads of oDNS on EI come from the extra network hops, which are not related to the EI framework. The extra hops are already part of the oDNS construction and can vary depending on the physical distances between the machines.

From our remote attestation service experiments, we demonstrate two key points. First, our Hash Distribution approach adds relatively small overheads compared to the cost of the overall remote attestation process. If the client wants to perform remote attestation themselves, then the Hash Distribution approach can be feasible. Second, our Attestation-Checking approach demonstrates the benefits of reducing the reliance on the IAS and successfully eliminates a key bottleneck from our system.

These results show that deploying security-enhancing services on EI is a practical endeavor. While these services can be deployed by themselves, EI helps scale them up without significantly compromising on performance. For future work, we may implement additional services to further improve client privacy and security. One example of a new service is one that has a Tor-like design for hiding the source of Internet traffic. We may also expand on our current services. For example, we can add a request generator for our oDNS service discussed in Section 3.5.

8 Conclusion

In this work, we demonstrate potential use cases for EI to enhance the privacy and security of clients on the public Internet. We leverage EI's unique position on the network and its infrastructure to tackle challenges of existing proposals and deploy services to help facilitate their operations on a large scale. Specifically, we design, implement, and evaluate two services built on top of the EI framework to assist with oblivious DNS lookups and remote attestation. Through these services and others like them, we believe that EI can help make the Internet more private and secure.

References

[1] gramineproject/gramine: A library OS for Linux multi-process applications, with Intel SGX support. https://github.com/gramineproject/gramine.

[2] Hari Balakrishnan, Sujata Banerjee, Israel Cidon, David Culler, Deborah Estrin, Ethan Katz-Bassett, Arvind Krishnamurthy, Murphy McCauley, Nick McKeown, Aurojit Panda, et al. Revitalizing the public internet by making it extensible. *ACM SIGCOMM Computer Communication Review*, 51(2):18–24, 2021.

[3] Victor Costan and Srinivas Devadas. Intel SGX Explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118, 2016.

[4] David Kaplan, Jeremy Powell, and Tom Woller. Amd Memory Encryption. *White paper*, 2016.

[5] Thomas Knauth, Michael Steiner, Somnath Chakrabarti, Li Lei, Cedric Xing, and Mona Vij. Integrating remote attestation with transport layer security. *arXiv preprint arXiv:1801.05863*, 2018.

[6] Kathleen Moriarty, Burt Kaliski, Jakob Jonsson, and Andreas Rusch. PKCS #1: RSA Cryptography Specifications Version 2.2. RFC 8017, November 2016.

[7] Benny Pinkas and Tzachy Reinman. Oblivious ram revisited. In *Annual cryptology conference*, pages 502–519. Springer, 2010.

[8] Joseph A. Salowey, David McGrew, and Abhijit Choudhury. AES Galois Counter Mode (GCM) Cipher Suites for TLS. RFC 5288, August 2008.

[9] Paul Schmitt, Anne Edmundson, Allison Mankin, and Nick Feamster. Oblivious DNS: Practical Privacy for DNS Queries. *Proceedings on Privacy Enhancing Technologies*, 2:228–244, 2019.