Secure Services for the Extensible Internet

William Lin Cathy Lu Zach Van Hyfte

Abstract

Connecting to applications on the Internet presents a range of different security and privacy issues for clients. Most connections begin with a Domain Name System (DNS) lookup performed by a recursive resolver, which learns the domains that a client connects to. Furthermore, the server that the client communicates with may be compromised by an unauthorized party, resulting in the potential leakage of private information. In-network services provide a method of addressing these issues, but such services are generally only deployed within private networks; there is currently no equivalent service infrastructure in place for the public Internet. The Extensible Internet (EI) is an initiative to introduce a new unified "service layer" that works across public and private networks. In this paper, we leverage the EI deployment model to address the issues of DNS lookup privacy and compromised servers. We design and evaluate security– and privacy-focused EI services that assist clients with oblivious DNS lookups and remote attestation, demonstrating how EI can ease the deployment of secure services and facilitate the widespread adoption of enhanced security practices. Our evaluations show that our services achieve reasonable performance in comparison to existing solutions.

1 Introduction

The Internet's architecture has remained roughly the same over the past few decades, despite its shortcomings. The Internet's primary service model of best-effort packet delivery may not be suitable for today's common Internet usage scenarios, which primarily consist of content– and service-oriented use cases on mobile clients. Moreover, the existing Internet architecture lacks essential security protections; for instance, users are required to trust third parties, like DNS recursive resolvers, with sensitive information about their networking patterns.

To compensate for the rigidity of the current architecture, large cloud providers have begun to deploy custom in-network services — caching, load balancing, etc. — that are not available on the public Internet. The proliferation of these private networks threatens to fracture Internet architecture into a messy, ad-hoc patchwork of proprietary services. The Extensible Internet (EI) is an initiative to make the sorts of in-network services traditionally offered by private networks available across the public Internet. EI operates in a new "Layer 3.5" service layer and builds on a network of *service nodes* — commodity servers running at the network edge. Because service nodes will be operated by a broad range of different parties, hardware enclaves are used to ensure that each service node is running the correct open-source software stack, and that service node operators cannot monitor or tamper with the execution of services. In this work, we leverage the new EI deployment model to address two security– and privacy-related challenges of using web applications: DNS lookups and server integrity.

The first challenge stems from the fact that, currently, most DNS queries are handled by a recursive resolver, which learns clients' communication partners and access patterns. The Oblivious DNS (ODNS) protocol [13] aims to address this privacy issue by splitting the work of the recursive resolver across two separate parties: one that communicates with the client and forwards encrypted queries, and another that decrypts queries and performs the actual name resolution. The first party does not learn the plaintext of the client's DNS query, while the second party does not learn which client the DNS query originated from. However, the privacy guarantees of ODNS are broken if these two parties collude. Furthermore, ODNS exhibits substantially lower throughput than standard DNS due to its reliance on computationally expensive public-key cryptography.

The second challenge is that the server that the client connects to may be compromised; web application operators may outsource the distribution of their application resources to third-party hosting providers that suffer from vulnerabilities that compromise the integrity of the application itself. Remote attestation mitigates this issue by allowing clients to verify the integrity of remote applications' code. However, this requires that clients always have access to up-to-date hashes of the code of the applications they wish to integrity-check. The client would need to obtain these hashes through a separate secure channel, which can itself be compromised, or the web application operator would need to distribute them to clients upon every application code update. These existing solutions pose scalability and latency concerns.

1.1 Contributions

To address the shortcomings of existing solutions for private DNS lookups and remote attestation, we designed and developed new EI services that facilitate the general adoption of these security practices. The main contributions of this paper are:

- An EI service for private DNS name resolution, based on the

ODNS protocol, in which service nodes act as verifiably private DNS resolvers for encrypted DNS queries. Our approach demonstrates lower latency and higher throughput than the original ODNS protocol.

- EI services that facilitate the deployment of enclaved applications that can easily be integrity-checked. These services make it possible for application operators to distribute code hashes to fewer parties, making code updates easier. We offer two types of services to clients: hash distribution and outsourced attestation-checking.
- An evaluation of the performance overheads of running services in AMD SEV enclaves, considering the costs of network operations.
- An evaluation of the performance of our new services. Our benchmarks of both the optimized ODNS service and the remote attestation services demonstrate reasonable performance in comparison to their non-EI counterparts.

2 Background

In this section, we detail the threat model that our services aim to protect against, as well as some of the core technologies that our work builds upon.

2.1 Threat Model

Our services will be deployed on servers operating on the network edge, so they are vulnerable to attacks by privileged attackers with control over the hardware and software stack, including the operating system, hypervisor, platform and device firmware, and other privileged software. Our work aims to protect the confidentiality and integrity of the in-network services to securely provide functionality to connecting clients.

We do not consider side-channel attacks because mitigations for most common side-channel attacks have been proposed in the academic literature. Additionally, we view access pattern–based attacks as out of scope in the context of Internet infrastructure due to the natural noisiness of network traffic.

2.2 Hardware Enclaves

In recent years, hardware enclaves have become an attractive option for running applications on untrusted host platforms. Enclaves provide the security guarantee of *shielded execution*, which protects the confidentiality and integrity of an application from the underlying operating system and hypervisor. This makes hardware enclaves a useful tool for securely outsourcing application execution to third-party cloud providers. A variety of hardware enclave implementations have been available for a number of years; among these are Intel's Software Guard eXtensions (SGX) [7] and AMD's Secure Encrypted Virtualization (SEV) [9] technologies.

2.2.1 Intel Software Guard eXtensions (SGX)

Intel's Software Guard eXtensions (SGX) [7] technology isolates a region of memory from the underlying operating system. It can be used to separate entire applications, or just specific high-security components of applications, from an untrusted operating system. The confidentiality and integrity of enclave memory is protected by the CPU hardware, which blocks memory accesses from outside the enclave, including accesses from the operating system and the hypervisor. Transitions into and out of the enclave are handled by special SGX instructions, which set aside enclave state in a thread control structure and scrub register state before transitioning to protect secrets. In addition to ensuring the confidentiality and integrity of enclave memory, SGX also provides remote attestation functionality, which allows a remote user to verify the integrity and authenticity of an enclaved application.

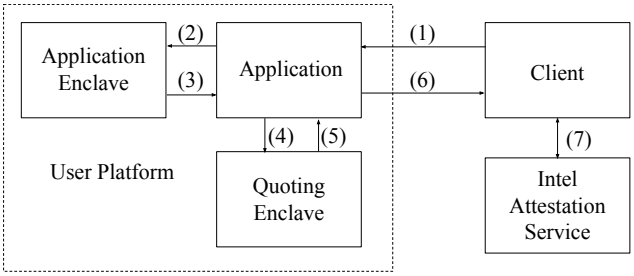


Figure 1: Intel SGX remote attestation flow

2.2.2 AMD Secure Encrypted Virtualization (SEV)

AMD Secure Encrypted Virtualization (SEV) [9] provides shielded execution through encrypted virtual machines. AMD SEV combines AMD Secure Memory Encryption (SME) technology and the AMD-V virtualization architecture to shield entire guest VMs from a potentially malicious hypervisor and from any other virtual machines running on the same hardware. Because SEV enclaves work at the VM level of granularity, applications do not have to be modified in order to run under SEV protection.

SEV provides three main security guarantees: confidentiality of the guest VM's data, authentication of the platform, and attestation of the guest VM's contents. First, SEV guarantees the confidentiality of the guest VM's via AMD Secure Memory Encryption (SME), which uses dedicated hardware on memory controllers to encrypt data before it is written to DRAM and decrypt it before it is accessed by the CPU. Second, SEV provides proof of the authenticity of the platform to prevent malicious devices from impersonating legitimate SEV-enabled platforms. The platform must make available an identity key, signed by AMD and the owner of the platform, to prove to the user that it is an authentic AMD platform and reveal the owner of the machine. Finally, the shielded guest VM can attest to end users that its integrity has not been compromised. Once the guest VM image is provided to the SEV firmware, SEV provides a signed measurement that the VM user can verify the integrity of before sending any sensitive data to the VM. These protocols ensure the confidentiality, integrity, and authenticity of both the guest image and the host platform.

2.3 Remote Attestation

Remote attestation allows a client to verify that an application is running in a valid enclave and has not been tampered with. Clients

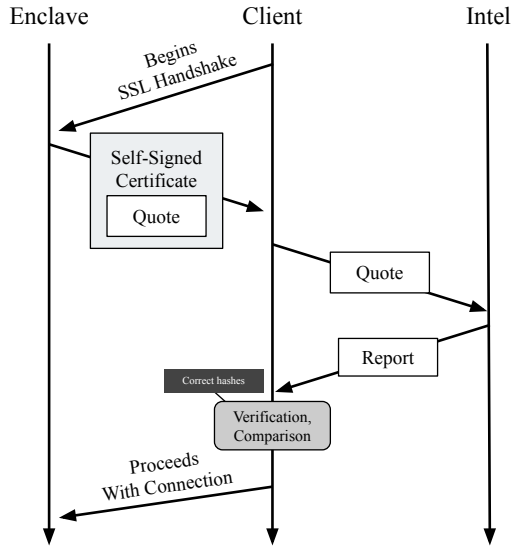


Figure 2: RA-TLS remote attestation flow for Intel SGX

perform remote attestation before sending any sensitive data to an enclaved application. Because our work is tightly integrated with Intel SGX remote attestation flows, we will present an overview of the remote attestation process for SGX enclaves [7].

In the attestation process outlined by Intel (Figure 1), (1) the client first issues an attestation challenge to the application it is trying to connect with. The application then (2) requests that the SGX enclave generate a local attestation report, which contains hashes of the enclave's code and data segments. The local attestation report (3–4) is sent to a separate Quoting Enclave on the same machine, which verifies the report before (5) signing it and returning it to the application. The application (6) sends the signed report (called a Quote) to the client, which compares the hashes in the Quote with the expected hashes. If the hashes match, the client (7) forwards the Quote to the Intel Attestation Service (IAS) to verify its signature. If the Quote is successfully verified, the client can conclude that the remote application is actually running within an enclave and is executing the expected code.

2.3.1 RA-TLS

The RA-TLS [10] protocol combines the Intel SGX remote attestation process with a standard TLS handshake to create a secure, authenticated channel between a client and an enclave. The protocol is shown in Figure 2. On every launch, the enclave generates a new self-signed TLS certificate containing a newly generated Quote that is cryptographically bound to the public key in the certificate. During the TLS handshake, the certificate is sent to the client; rather than verifying it with a Certificate Authority, the client performs the SGX remote attestation process using the embedded Quote. Once verified, the client can use the TLS certificate to resume the TLS handshake with the enclave.

This construction leverages the chain of trust from the TLS certificate to the Intel Attestation Service. Because the Quote is cryptographically bound to the TLS certificate's public key, a verified Quote implies that the public key in the same certificate belongs to a legitimate enclave. Furthermore, a matching Quote hash implies that the legitimate enclave is running known, trusted code. Because a new key pair is generated on every enclave launch, the certificate can only be used to establish a secure channel with the specific enclave instance that generated it (assuming it is not accidentally leaked outside the enclave).

2.3.2 Limitations

A scalability limitation of remote attestation is that it requires the client to have the expected hashes of the enclaved application's code. This would require the client to go through a separate secure channel, which is itself a target of attack, to obtain up-to-date hashes before using any external resource. Alternatively, the publisher of an external resource could somehow securely distribute new hashes to all clients after every code update. These solutions pose potential scalability and latency barriers to the general adoption of remote attestation.

2.4 The Extensible Internet

To address the growing functionality disparity between public and private Internet infrastructure, the Extensible Internet (EI) [5] has been developed to be an open architecture that can be used to deploy of a universal set of services available across public and private networks. EI adds a new "Layer 3.5" service layer to the existing Open Systems Interconnection (OSI) model [16]. This layer is comprised of service nodes — commodity servers on the network edge that run a set of public services chosen through an IETF-like [1] governance process. Service nodes will generally be deployed by Last Mile Providers (LMPs), like traditional ISPs, to enable tighter integration with existing Internet infrastructure. Clients connect to service nodes through a discovery process, which associates them with one or more nearby service nodes.

EI service nodes run within AMD SEV enclaves, which isolate an entire guest VM from the platform operator and provide protection against internal actors with access to the physical service node hardware. AMD SEV was chosen over Intel SGX as the enclave environment for service nodes because the performance overheads that it imposes are generally smaller than those of SGX [11]. In Section 6.1, we present network benchmarks that demonstrate the feasibility of running service nodes within SEV enclaves.

Because the services running on each service node are open-source, anyone can easily verify the integrity of a service node's software before communicating with it. Remote attestation allows each client to verify that a service node has not been compromised before using its services. Clients may choose to maintain a long-term secure channel with the service node to enable quicker communication. Integrity-verified service nodes can peer with one another and establish long-term secure channels, forming a network that can be used to securely propagate data through the EI infrastructure.

2.5 Domain Name System (DNS)

The Domain Name System (DNS) is a core part of the Internet's infrastructure, used daily by almost every Internet-connected device. Clients use the DNS protocol to perform *name resolution*,

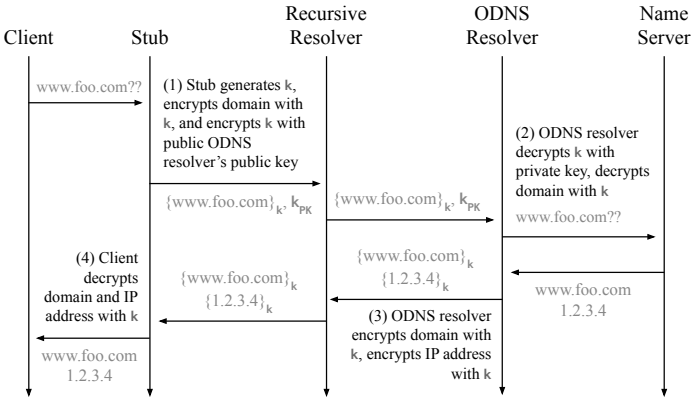


Figure 3: The original ODNS protocol.

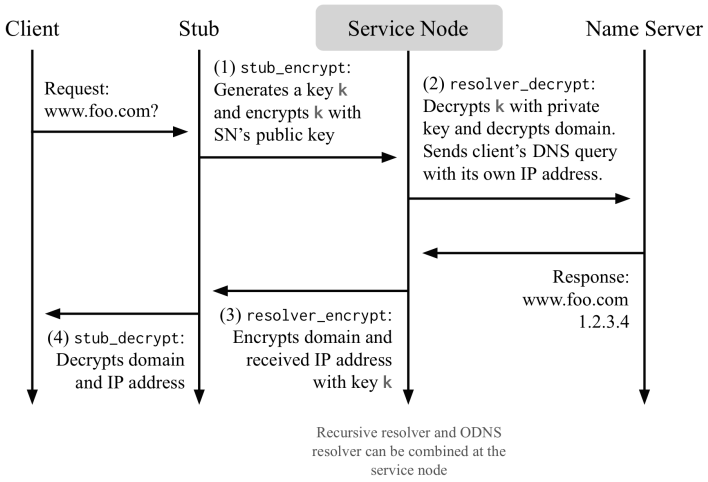


Figure 4: Our optimized ODNS protocol, in which the DNS recursive resolver and the ODNS resolver are both subsumed into a single ODNS resolver service running on an EI service node.

mapping human-readable domain names (e.g. `berkeley.edu`) to the IP addresses used to identify hosts in packet routing.

To perform a DNS lookup, the client sends a query containing a domain name to their local DNS *recursive resolver*. The recursive resolver is usually operated by either the client's ISP or a DNS provider like OpenDNS or Google Public DNS. The recursive resolver then contacts a series of *name servers* to determine the *authoritative server* responsible for maintaining the domain-to–IP address mappings for the requested domain name. Finally, it queries the authoritative server to retrieve the correct IP address for the requested domain name, and forwards the IP address back to the client. Because DNS queries are sent in plaintext, recursive resolvers can observe which domain names are requested by which clients, and can thus compile a detailed picture of a client's browsing history.

2.5.1 Oblivious DNS (ODNS)

To alleviate these privacy concerns, Schmitt *et al.* proposed the Oblivious DNS (ODNS) protocol [13], which introduces additional parties into the name resolution process to prevent any one entity from linking a client with a DNS query. The ODNS protocol is shown in Figure 3. A "stub" resolver program running on the client first generates a symmetric session key and uses it to encrypt the requested domain name. The stub encrypts the symmetric key with the ODNS resolver's public key and concatenates it to the end of the encrypted domain name. Finally, the stub appends the `.odns` top-level domain (TLD) to this encrypted blob and sends a standard DNS query for `[encrypted key|encrypted domain].odns` to their local recursive resolver. The recursive resolver, seeing the `.odns` TLD, routes the query directly to the ODNS resolver. The ODNS resolver decrypts the symmetric session key and domain name and performs the normal name resolution process. After obtaining the IP address corresponding to the requested domain name, the ODNS resolver encrypts both the domain name and IP address and sends them back to the recursive resolver, which forwards them back to the client. Assuming that the two do not collude, the recursive resolver never learns the clients' plaintext DNS queries, and the ODNS resolver never learns which clients made the queries that it receives.

While the ODNS protocol makes great strides towards ensuring the privacy of DNS queries a practical and incrementally deployable way, the protocol has several limitations:

1. **ODNS requires the addition of a new party to the name resolution process.** This requires more infrastructure, introduces new points of failure for DNS queries, and also increases the latency of DNS requests.

2. **The privacy guarantees of ODNS are completely broken if a recursive resolver and an ODNS resolver decide to collude.** It is difficult to prove non-collusion, and even if both parties are initially honest, a sufficiently powerful adversary, like a government, can compel the resolvers to reveal query data.

3. **Each request requires expensive public-key cryptographic operations.** The need to perform computationally expensive public-key cryptographic operations limits the number of requests per second that each ODNS resolver can handle. While the cost of encrypting requests is negligible on the client side, our experimental results show that the added overhead of public-key decryption severely limits the throughput of each ODNS resolver relative to a standard DNS resolver.

3 Oblivious DNS Service

To address the limitations of the Schmitt *et al.*'s ODNS protocol, we designed a version of the protocol optimized for deployment on EI. Our design leverages the fact that service nodes run in enclaves to provide security and privacy guarantees similar to those of the original ODNS protocol.

The basic version of our optimized ODNS protocol is shown in Figure 4. The major difference from the original ODNS protocol is that the standard DNS recursive resolver and the ODNS resolver are both subsumed into a single resolver service running on an EI service node. Because clients can verify the integrity of the open-source code running on the service node through remote attestation, clients can ensure that the service nodes they use for oblivious DNS lookups are not storing or leaking information about their

requests. This eliminates the need for an additional ODNS server and the additional network hops required to resolve an ODNS query. Thus, our optimized ODNS service provides security and privacy guarantees that are similar to those of the original ODNS protocol but that can be verified in practice. This service does not require any new ODNS-specific infrastructure or the use of a new TLD, so it integrates even more seamlessly into the existing DNS infrastructure

In our optimized protocol, like the original ODNS protocol, the client's "stub" first generates a random symmetric session key and uses it to encrypt the requested domain name. It then encrypts the symmetric key with the public key of the ODNS resolver service running in a local service node. The client then sends the encrypted symmetric key and encrypted domain name directly to the resolver service. The resolver service decrypts the symmetric key and domain name and performs the recursive lookups to obtain the IP address mapped to the domain name. Finally the ODNS resolver service encrypts the domain name and corresponding IP address with the symmetric session key and sends them back to the client. We use an authenticated symmetric encryption algorithm to ensure that the symmetric-encrypted domain name and IP address cannot be tampered with on the return trip to the client.

3.1 Long-Lived Symmetric Key Variant

A streamlined variant of our protocol reduces the cryptographic overheads of encrypted DNS queries by taking advantage of the existing EI communication primitives. In an eventual large-scale EI deployment, each client maintains a long-lived secure channel to its local service node. The ODNS resolver service and the client can reuse this existing secure channel to set a long-lived symmetric key for encrypting DNS queries. Future DNS queries can be encrypted with this shared symmetric key to avoid public-key encryption altogether. This optimization reduces the computational load on the service node and allows it to handle a higher rate of incoming DNS queries.

4 Remote Attestation Services

To address the shortcomings of existing remote attestation services, we designed new services to assist clients as they perform remote attestation to verify the integrity of a server. We developed two different services that can run on all EI service nodes to facilitate widespread remote attestation: a hash distribution service and an attestation-checking service. These services aim to alleviate the scalability and latency issues that preclude the deployment of applications supporting remote attestation. The motivation behind these services is that they enable application developers to distribute hashes to fewer entities; whenever there are any code updates, publishers only need to communicate these changes with the EI servers rather than with each client individually.

Once the correct hashes are known within the EI infrastructure, service nodes can share these hashes with one another through the existing network of secure channels. The specific details of how the correct hashes are propagated throughout the EI infrastructure are out of scope for this paper. For the purposes of this work, we make the assumption that service nodes already have a means of obtaining and distributing the correct hashes beforehand.

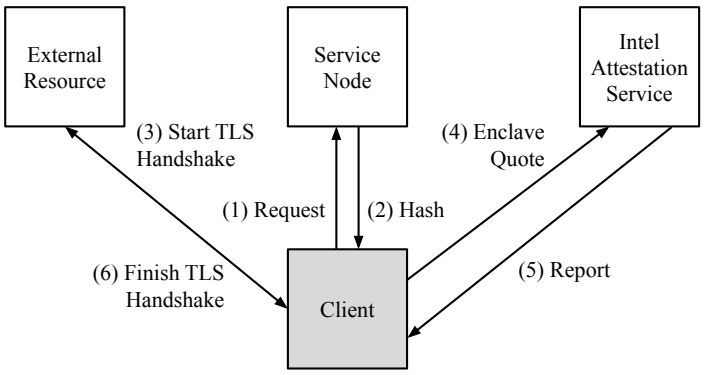


Figure 5: The architecture of the hash distribution service. The service node distributes up-to-date code hashes to clients to use in remote attestation. It is assumed that the service node already has the correct hash for the external resource.

Leveraging EI enables the deployment of remote attestation to be more scalable; each party now only talks to a small subset of the network. Application publishers distribute updated hashes to EI servers rather than clients, and each service node assists only a subset of EI clients with remote attestation.

These services have been designed specifically to work with Intel SGX enclaves and SGX remote attestation flows, but could be extended to other enclave implementations.

4.1 Hash Distribution Service

The basic idea behind the hash distribution service is that each service node is treated as a storage system for correct enclave hashes. As EI already has a process for establishing a secure communication channel between a client and a service node, the client can simply reuse this secure channel to retrieve hashes. The client no longer needs to worry about obtaining hashes from a separate channel or about ensuring that they are able to update their local hashes whenever there are code changes to the external resource.

Figure 5 illustrates the high-level design of the hash distribution service. To begin the remote attestation process, a client first sends a request to the service node for the external resource it wishes to verify the integrity of. The service node retrieves the correct hash from a local database and returns it to the client. Once the client has the correct hash, it performs an RA-TLS handshake to verify the authenticity of, and establish a secure channel to, the external resource.

4.2 Attestation-Checking Service

An issue with the hash distribution service is that it requires the client to perform remote attestation itself before connecting to an external resource. Clients would need to establish a secure connection with a third-party attestation service, like the Intel Attestation Service (IAS), which may not always be feasible on devices with limited network connectivity. To address this issue, we introduce the attestation-checking service, in which the service node performs remote attestation itself to verify the integrity of the external resources on behalf of clients.

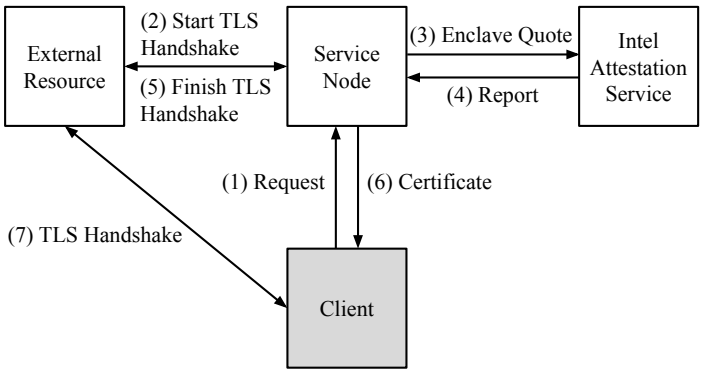


Figure 6: The architecture of the attestation-checking service. The service node verifies the integrity of an external resource on behalf of the client by performing RA-TLS remote attestation. It is assumed that the service node already has the correct hash for the external resource.

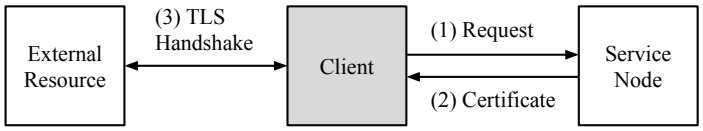


Figure 7: The architecture of the caching workflow for the attestation-checking service. It is assumed that the service node had previously performed RA-TLS remote attestation for the external resource and already has its certificate cached locally.

Figure 6 illustrates a high-level overview of this service. To verify the integrity of an external resource, the client sends a request to the service node. Upon receiving the request, the service node performs the RA-TLS remote attestation process with the external resource and returns the TLS certificate back to the client after sending it to Intel for verification. The client can use the certificate to establish its own secure channel with the same enclave instance without needing to contact the third-party attestation service for additional verification.

This construction leverages the chain of trust from the TLS certificate to the Intel Attestation Verification Service. Because the Quote is cryptographically bound to the TLS certificate's public key, a verified Quote implies that the public key belongs to a legitimate enclave. Furthermore, a matching Quote hash implies that the legitimate enclave is running trusted code. Because a new key pair is generated on every enclave launch, the certificate can only be used to establish a secure channel with the specific enclave instance that generated it.

With this attestation-checking service, the client only needs to verify the integrity and authenticity of one entity (its local service node) relatively infrequently, while the service node can verify the integrity of all external resources that the client wishes to communicate with. The client receives all of the security benefits of remote attestation for external resources without needing to repeatedly perform the attestation process themselves.

4.2.1 Caching

The attestation-checking service can be further optimized by introducing caching. The idea behind this optimization is that the service node can cache all of the TLS certificates it has previously verified to avoid performing redundant RA-TLS handshakes. To fulfill a request to verify a previously verified resource, the service node can simply return the cached certificate without going through the entire remote attestation process. Figure 7 demonstrates the architecture of this optimization.

Caching further reduces the latency of remote attestation, because the service node no longer needs to contact the third-party attestation service before providing the client with a certificate. This can be especially useful if network latencies occupy most of the application's execution time, as is the case with short-lived workloads or microservices that are created on demand but are used by multiple clients. Caching also reduces the reliance on third-party attestation verification services, like the IAS, which may artificially rate-limit verifications. Avoiding these types of services whenever possible allows us to eliminate a potential throughput bottleneck.

However, caching introduces new security concerns. Clients can learn which resources have already been accessed by the service node by measuring the response latency of an attestation request. If some external resource is used infrequently by only a small set of users, this timing may reveal sensitive access information specific to those users. To grant users flexibility regarding these risks, the attestation-checking service allows clients to opt-out of having the certificate they requested to be cached for later use.

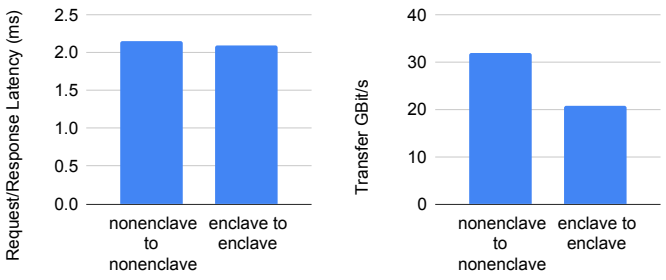
5 Implementation

5.1 ODNS Services

We implemented a prototype ODNS resolver service running on top of the existing EI framework. We also wrote a corresponding client-side library for creating and sending encrypted DNS requests, which we used in our evaluation. Lastly, to compare our new protocol against an approximation of the original ODNS implementation, we also implemented a minimal recursive resolver implementation that forwards ODNS queries to the ODNS resolver and sends the corresponding replies back to the correct client. Together, these modules consist of approximately 900 lines of additional C++ code.

Our prototype private DNS resolver service builds on top of the Knot DNS resolver [4], a popular and robust caching DNS resolver implementation. In our prototype implementation, incoming private DNS requests are decrypted and arranged into standard DNS query packets that are passed to the local Knot resolver daemon. The relevant components of the responses provided by the Knot daemon are encrypted and arranged into private DNS response packets that can be parsed by the client-side library.

The services use the cryptography components of the Mbed TLS library [2] for both symmetric-key and public-key encryption. Specifically, we chose AES-GCM as our symmetric-key encryption algorithm, as it is designed to be highly efficient while also providing both integrity and confidentiality guarantees. For public-key encryption, we used 2,048-bit RSA keys.



(a) Network Latency (b) Network Bandwidth

Figure 8: Microbenchmark results for network transfers.

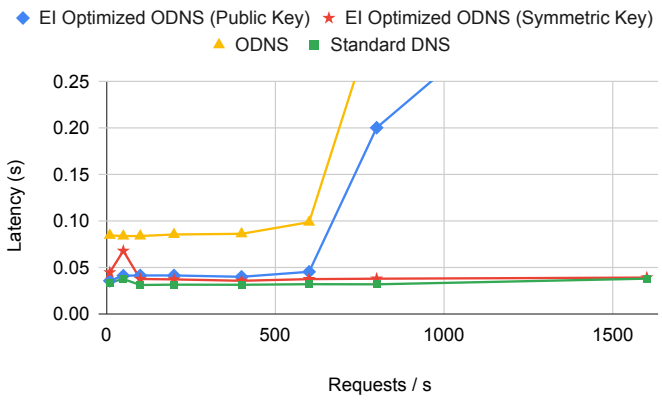


Figure 9: A comparison of the latencies of four name resolution setups at different request rates.

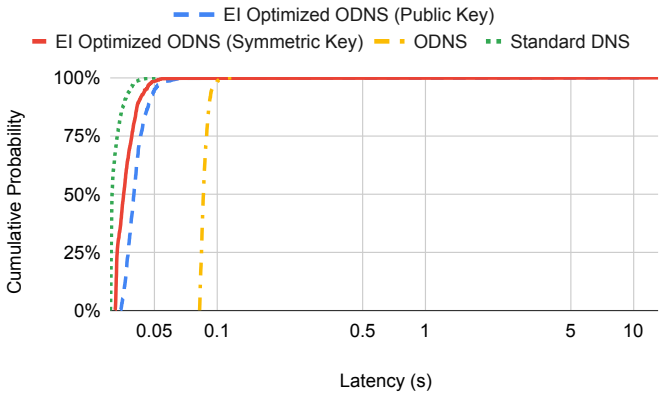


Figure 10: A comparison of the cumulative distribution of the latencies of four name resolution services at a request rate of 400 requests per second.

5.2 Remote Attestation Services

We implemented prototypes of all of the services detailed in Section 4 — the hash distribution service, the attestation-checking service, and the attestation-checking service with caching. The services are implemented on top of the existing EI framework, using approximately 670 additional lines of C++ code. The services use the Mbed TLS library [2] for TLS operations and the RA-TLS libraries from the Gramine project [3] to verify certificates with Intel.

6 Evaluation

6.1 AMD SEV Networking Overheads

To understand the performance impacts of running EI service nodes in secure enclaves without additional optimizations, we ran microbenchmarks to evaluate the performance differences between enclaved and non-enclaved VMs for networking tasks. Our test VMs were hosted by Google Compute Engine. For both enclaved and non-enclaved VMs, we used `n2d-standard-48` machines operating in the same zone. Figure 8 shows the results of our networking benchmarks.

Latency. We used the `netperf` tool to benchmark the network latencies between two VMs. We recorded the TCP request/response time latencies, which measure the time between a VM's sending of a 1-byte request and its receiving a 1-byte response. A new request is sent immediately after a response is received, and the experiment lasts for 10 seconds. We observed no increase in network latencies for enclave-to-enclave communication compared to nonenclave–to–nonenclave communication.

Bandwidth. To measure network bandwidth, we used the `iperf` tool to determine the maximum achievable bandwidth between two VMs. In our setup, `iperf` constructs 16 concurrent data streams and saturates each stream with as much data as possible. We observe a bandwidth drop of approximately 35% for data transfers between enclaved VMs compared to data transfers between non-enclaved VMs. Given that our services do not require transferring significant amounts of data over the network, this slowdown is acceptable for our use cases.

6.2 ODNS Services

We evaluated the performance of our EI-assisted private DNS service by comparing four different name resolution setups:

EI Optimized ODNS (Public Key) Names are resolved according to the protocol described in Section 3, where the public key of the ODNS resolver is used for encrypting queries.

EI Optimized ODNS (Symmetric Key) Names are resolved according to the protocol in Section 3.1, where a symmetric key is used for encrypting queries.

ODNS Names are resolved using an approximation of the original ODNS protocol described in section 2.5.1.

Standard DNS Names are resolved by sending standard DNS requests as EI packets to an EI service (running on a special non-enclaved service node) that acts as an extremely lightweight interface to the Knot DNS resolver, without the encrypted DNS packet handlers in front.

The service nodes used in the two EI-assisted setups ran on `n2d-standard-4` virtual machines hosted by Google Compute Engine, with 4 CPU cores, 16 GB of RAM, and AMD SEV enabled.

The ODNS resolver and the recursive resolver used in the ODNS setup setups each run on their own `n2d-standard-4` virtual machines. Likewise, the bare Knot DNS resolver in the Standard DNS setup ran in an `n2d-standard-4` virtual machine without AMD SEV protections. In all of the setups, the client request load was modeled by a `n2d-standard-16` Compute Engine virtual machine that had 16 CPU cores and 64 GB of RAM and was not configured with AMD SEV protections. We wanted to ensure that this machine was over-provisioned to prevent it from becoming an artificial bottleneck. The network latency between every pair of machines was roughly 35 ms.

For each of the four setups, we repeatedly measured the end-to-end latency of 1,000 name resolutions while varying the request rate. We sent queries following a Poisson arrival process to simulate the burstiness of real-world traffic. Figure 9 shows, for all four systems, how the median name resolution latency changes as the request rate increases. Figure 10 shows the cumulative distribution of the name resolution latencies for all four setups at a request rate of 400 requests per second.

Throughput. The public-key version of the EI-assisted setup provides only minimal throughput improvements relative to the original ODNS protocol, which, unlike Standard DNS, hits a throughput wall at around 600 requests/s. One of the key limitations of ODNS identified in Section 2.5 is that the overhead of public-key cryptography significantly limits the system's throughput. The symmetric-key version of the EI-assisted setup eliminates this bottleneck, allowing it to maintain the same throughput as Standard DNS up through the highest request rates that we were able to reliably measure before significant packet drops.

Latency. Both of the EI-assisted setups demonstrated significantly improved latency relative to the original ODNS protocol, narrowly approaching that of standard DNS. This is because the extra network hops that are required for the privacy guarantees of the original ODNS protocol to hold are eliminated in the EI-assisted protocols, which provide very similar privacy guarantees by using secure enclaves. The baseline latencies of the ODNS setup, observable at lower request rates, hovered around 0.085 s. The baseline latencies of the EI-assisted setups were less than half those of the ODNS setup; the baseline latencies for the public-key variant were around 0.040 s, while those of the symmetric-key variant were around 0.037 s. The baseline latencies of the Standard DNS setup were only slightly lower, at around 0.031 s.

6.3 Remote Attestation Services

We evaluated the performance of the different EI-assisted remote attestation services detailed in Section 4. We compared four different remote attestation setups:

Non-EI The client is assumed to already have the correct hashes for the external resources it wishes to verify the integrity of. It performs the remote attestation process without using any EI services.

Hash Distribution The construction detailed in Section 4.1, in which the service node supplies the correct hashes to the client, and the client performs the remote attestation process itself.

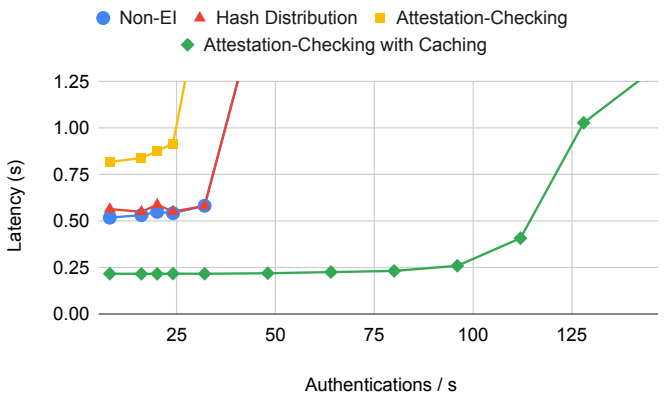


Figure 11: A comparison of different remote attestation setups.

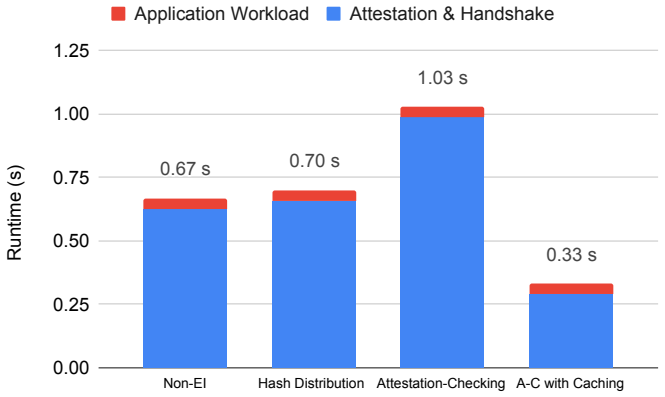


Figure 12: A breakdown of remote attestation and data transfer time for a small microservice workload.

Attestation-Checking The construction detailed in Section 4.2, in which the service node performs remote attestation on behalf of the client.

Attestation-Checking with Caching The construction detailed in Section 4.2.1, in which the service node caches verified TLS certificates from past remote attestation requests to avoid performing remote attestation twice for the same resource.

First, we varied the request rate and recorded the latencies observed for a round of authentication — specifically, we measured the interval of time between when the client begins the request and when a secure channel to the external resource has been established. As in the ODNS experiments, we sent requests following a Poisson arrival process. For each request rate, we recorded 512 sample latencies, and report the average request latency for each request rate.

Second, for each of the remote attestation setups, we recorded the breakdown of the runtime of a client's request to a third-party application. The synthetic application used for this benchmark receives a 128-bit string from the client and echoes back a response. This application is intended to simulate low-computation applications such as a key-value store or single sign-on authentication. We divided the runtime of the request into two phases: the TLS handshake (which encapsulates remote attestation) and the applica-

tion workload. For the TLS handshake, we measured the interval of time between when the client starts to initiate the connection and when a secure channel has been fully established. For the application workload, we measured the interval between when the client begins sending its request and when it has received the entirety of the response. We averaged the results of 64 subsequent trials, executed with a 0.5-second break between trials, and report the average latencies for both the TLS handshake and application workload.

The external resource used in both these experiments ran in an Intel SGX enclave that used a single core of a `Standard_DC2s_v2` virtual machine hosted by Microsoft Azure, which has 2 CPU cores and 8 GB of RAM in total. The service node used a single core of a `n2d-standard-2` VM hosted by Google Compute Engine, with 2 CPU cores, 8 GB of RAM, and AMD SEV enabled. Requests from multiple clients are emulated using multiple cores of an `e2-standard-8` Compute Engine VM, with 8 CPU cores, 32 GB of RAM, and no AMD SEV protections. The network latency between the clients and the service node was approximately 20 ms, the network latency between the external resource and the clients was approximately 30 ms, and the network latency between the external resource and the service node was approximately 40 ms.

The results of the attestation comparison experiments are shown in Figure 11, and the results of the runtime breakdown experiments are shown in Figure 12.

Throughput. The attestation-checking setup achieved the lowest throughput of approximately 28 rounds of authentications per second. The achieved throughput of this setup is bounded by the service node, which must interleave interactions with the external resource, IAS, and the clients. The non-EI and hash distribution setups achieve approximately the same throughput of 34 rounds of authentications per second. These setups are artificially rate-limited by the IAS, so the service node VM was never fully utilized. This bottleneck demonstrates the downside of relying heavily on a centralized third-party attestation service during runtime. The last setup — attestation-checking with caching — achieves the highest throughput of around 112 rounds of authentications per second. It does not rely on communication with IAS during runtime and is able to better utilize the VM's resources to achieve greater throughput.

Latency. At lower request rates, we observe the baseline latencies for all four setups. Out of the setups that communicate with IAS, the non-EI setup has the lowest latency of 0.54 s. The hash distribution setup has a constant increase in latency (0.03 s) over the non-EI setup due to the additional round trip communication with the service node. The attestation-checking service (without caching) has the highest latency (0.80 s) out of the three. This high latency is due to the fact that the setup requires two RA-TLS handshakes per round of authentication: one between the external resource and the service node and one between the external resource and the client itself. The attestation-checking service with caching has the lowest latency (0.25 s) out of all of the setups. It does not need to communicate with IAS on every request, so we do not see the latency increases associated with interacting with IAS.

Attestation Breakdown. In all four attestation scenarios, we observe in Figure 12 that the TLS handshake takes a majority of the request runtime. In the non-EI case, the attestation and TLS handshake use 0.62 s, while the actual application only takes 0.04 s, which adds a significant overhead to the overall runtime of a client request. We observe that the proportion of time for the TLS handshake is above 90% for all scenarios, except for the attestation-checking with caching setup where it is reduced to 87% of the total runtime with a handshake of 0.29 s and application runtime of 0.04 s. For low-computation applications, such as key-value stores and SSO authentication, that may be limited by network communication, the latency improvements we see in our attestation-checking service with caching provide significant speedup to client requests.

7 Discussion

Our prototype optimized ODNS service leverages the unique properties of EI's deployment model to facilitate private DNS lookups with latencies comparable to those of standard DNS lookups, without requiring any new ODNS-specific infrastructure. It also improves on the throughput of ODNS by avoiding the use of public key cryptography during DNS lookups.

Our prototype remote attestation services also leverage the EI infrastructure to address the scalability and latency problems faced by existing approaches to remote attestation. Using EI allows application publishers to distribute up-to-date hashes of application code to fewer parties after code changes. The services give clients the choice of either performing remote attestation themselves or outsourcing the process to a trusted, tamper-proof service node, making remote attestation more accessible to different clients.

Together, our optimized ODNS and remote attestation services address some of the security and privacy concerns faced when connecting to web applications in the existing Internet infrastructure. Both services are only made possible by EI's security-focused infrastructure, demonstrating the unique and immediate value that EI can provide to Internet users. Our work shows how EI's security-focused infrastructure can be used to broaden the adoption of, and simplify the deployment of, security– and privacy-enhancing services in the future.

7.1 Limitations and Future Work

While our prototype services demonstrate reasonable performance compared to their non-EI equivalents, they still have noteworthy limitations. Our services, as currently implemented, do not fully take into account possible data leakage to adversaries that are eavesdropping on the network and executing timing attacks to link clients to specific requests. This problem is not an inherent limitation of our services, and can be addressed in future work by introducing noise into service nodes' outbound traffic. A service node can randomly send extraneous remote attestation requests or DNS queries to obscure clients' actual requests from a network eavesdropper.

Other directions for future work include designing a secure information sharing system for the EI network. There exist secure channels between service nodes that can be leveraged to propagate information like application code hashes. Designing and optimizing such a system would make services, like the EI-assisted attestation services, more feasible to deploy. Another direction for future work

is implementing load balancing services to enable better utilization of network resources. The benchmarks for the attestation-checking service demonstrate how the service node itself can eventually become a throughput bottleneck. Automatically balancing requests across multiple service nodes would help improve services' performance.

8 Related Work

8.1 Secure Middleboxes

Software middleboxes, or network function virtualization (NFV), have played an important role in providing additional functionality to private networks. They have improved network security through firewalls, reliability through load balancers, and performance through caches [6].

While middleboxes can be used to deploy services similar to our EI-assisted ODNS and remote attestation services, they are not as tightly integrated into the Internet architecture, resulting in proprietary, ad-hoc designs that do not fit well into the public Internet infrastructure. The key idea behind EI service nodes is that they would be managed by network operators and better integrated into the Internet infrastructure. EI services would be available for general public use across both public and private networks.

8.2 Distributed DNS

While our optimized ODNS protocol aims to protect clients' privacy by decoupling client IP addresses from their domain queries through cryptographic means, another approach taken to increase the privacy of DNS lookups involves further decentralization of the name resolution process. K-resolver [8] seeks to remove the recursive resolver from its single-point-of-trust position by distributing each client's DNS queries across K separate recursive resolvers, such that each recursive resolver only learns $1/K$ of a client's queries, as opposed to a single recursive resolver learning the domain name of every server to which a client connects. However, this solution still reveals a fraction of a user's internet queries to each one of a set of untrusted recursive resolvers. In contrast, our optimized ODNS protocol does not allow any user information to be accessible to the recursive resolver, since we encrypt all DNS queries and only reveal the plaintexts within enclaves that are known to be running trusted open-source code.

8.3 Scalable Remote Attestation

Previous work on the problem of making remote attestation more practical and scalable has focused largely on solutions for use within a single administrative unit, and in particular on distributing the work of integrity-checking resources across different the different VMs or applications that are themselves being integrity-checked.

As recounted in [15], TCCP [12], a scheme first presented in 2009 for use with virtual machine monitors, involves the creation of a single trusted "coordinator node" that verifies the integrity of a set of VMs, one by one. A distributed version of the TCCP protocol [14] has also been proposed, in which the coordinator randomly selects a small set of VMs or applications, verifies their integrity, and compels them to act as "helper nodes." Each helper node is assigned a cluster of VMs and a collection of correct hashes by the coordinator node; it iterates through each of the VMs in its cluster and verifies each one's integrity before finally reporting the complete set of results back to the coordinator. Even in the distributed version of TCCP, however, the number of helper nodes that the coordinator node needs to manage scales linearly with the number of VMs that need to be monitored, making it a poor choice for a system of the scale that EI aims to attain. Furthermore, although the throughput of the distributed version of TCCP is higher, the communication links between the single coordinator node and numerous helper nodes remain a bottleneck.

Song et al. have introduced Collective Attestation scheme towards Cloud System (CACS) [15] with the aim of eliminating the aforementioned bottleneck. In CACS, all shielded applications are organized into an Attestation Relationship Tree (ART), which is a rooted tree of arbitrary degree and structure that can be organized in any way (the authors note that it could be made to align with the network or application structure). Each node in the tree is responsible for verifying the integrity of all of its direct children. The attestation service can send a collection of correct hashes to the root node, which are recursively distributed to the lower levels of tree. Finally, the (trusted) attestation service itself verifies the integrity of the ART's root node.

As noted above, both of these protocols delegate the work of integrity-checking resources to the resources that are themselves being integrity-checked. This design is infeasible for an attestation service like ours, which is intended to assist in integrity-checking applications operated and hosted by a wide range of third parties.

9 Conclusions

In this work, we introduce two types of EI services to assist with private DNS queries and remote attestation. We analyze the performance of different workloads and optimizations for these services and show that use of the EI infrastructure can yield throughput and latency improvements because of EI's unique position on the network. We believe that these services will improve the overall security and privacy of the public Internet.

References

[1] About — ietf.org. https://www.ietf.org/about/.

[2] ARMmbed/mbedtls: An open source, portable, easy to use, readable and flexible SSL library. https://github.com/ARMmbed/mbedtls.

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

[4] Knot Resolver. https://www.knot-resolver.cz/.

[5] 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.

[6] Brian Carpenter and Scott Brim. Middleboxes: Taxonomy and issues. Technical report, RFC 3234, February, 2002.

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

[8] Nguyen Phong Hoang, Ivan Lin, Seyedhamed Ghavamnia, and Michalis Polychronakis. K-resolver: Towards Decentralizing Encrypted DNS Resolution. *arXiv preprint arXiv:2001.08901*, 2020.

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

[10] 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.

[11] Saeid Mofrad, Fengwei Zhang, Shiyong Lu, and Weidong Shi. A comparison study of Intel SGX and AMD memory encryption technology. In *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, pages 1–8, 2018.

[12] Nuno Santos, Krishna P. Gummadi, and Rodrigo Rodrigues. Towards Trusted Cloud Computing. In *Proceedings of the 2009 Conference on Hot Topics in Cloud Computing*, HotCloud'09, San Diego, California, 2009. USENIX Association.

[13] 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.

[14] Partha Sen, Pritam Saha, and Sunirmal Khatua. A distributed approach towards trusted cloud computing platform. In *2015 Applications and Innovations in Mobile Computing (AIMoC)*, pages 146–151, 2015.

[15] Yuan Song, Wenchang Shi, Bo Qin, and Bin Liang. A collective attestation scheme towards cloud system. *Cluster Computing*, September 2020.

[16] William Stallings. *Handbook of computer-communications standards; Vol. 1: the Open Systems Interconnection (OSI) model and OSI-related standards*. Macmillan Publishing Co., Inc., 1987.