USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

Transparent Attested DNS for Confidential Computing Services

Antoine Delignat-Lavaud, Cédric Fournet, Kapil Vaswani, Manuel Costa, and Sylvan Clebsch, *Azure Research, Microsoft;* Christoph M. Wintersteiger, *Imandra*

https://www.usenix.org/conference/usenixsecurity25/presentation/delignat-lavaud

This paper is included in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

Transparent Attested DNS for Confidential Computing Services

Antoine Delignat-Lavaud
Azure Research, Microsoft

Cédric Fournet
Azure Research, Microsoft

Kapil Vaswani
Azure Research, Microsoft

Manuel Costa
Azure Research, Microsoft

Sylvan Clebsch
Azure Research, Microsoft

Christoph M. Wintersteiger
Imandra

Abstract

Confidential services running in hardware-protected Trusted Execution Environments (TEEs) can provide higher security assurance, but this requires custom clients and protocols to distribute and verify their attestation evidence. Compared with classic Internet security, built upon universal abstractions such as domain names, origins, and certificates, this puts a significant burden on service users and providers. In particular, Web browsers and other mainstream clients do not get the same security guarantees as custom clients.

We present a new approach for users to establish trust in confidential services. We propose attested DNS (aDNS): a name service that securely binds the attested implementation of confidential services to their domain names. aDNS enforces policies for all names in its zone of authority: any TEE that runs a service must present a hardware attestation that complies with the domain-specific policy before registering keys and obtaining certificates for any name in this domain. aDNS provides protocols for zone delegation, TEE registration, and certificate issuance. aDNS builds on standards such as DNSSEC, DANE, ACME and Certificate Transparency. aDNS provides DNS transparency by keeping all records, policies, and attestations in a public append-only log, thereby enabling auditing and preventing targeted attacks.

We implement aDNS as a confidential service using a fault-tolerant network of TEEs. We evaluate it using sample confidential services that illustrate various TEE platforms. On the client side, we provide a generic browser extension that queries and verifies attestation records before opening TLS connections, with negligible performance overhead, and we show that, with aDNS, even mainstream Web clients benefit from confidential computing as long as some enlightened clients verify attestations to deter or blame malicious actors.

1 Introduction

Clients routinely trust Internet services based on their domain names, such as `usenix.org`. As they connect to the service (using, e.g., HTTP over TLS), their networking stack ensures that the server authenticates with a valid certificate for this name, and users more generally trust that the domain owner will grant such certificates only to authorized servers. This conveniently hides many implementation details, such as network addresses, certificates, and server configurations, and also allows the delegation of operations to third parties such as content delivery networks, caches, and other front-ends.

Confidential computing enables services to run in isolation in hardware-protected trusted execution environments (TEEs) with increased security both for the service owner and for its users. TEEs that run the service present attestation reports that jointly authenticate their hardware platform, their code, and their configuration. While this is much more informative than a domain name, the management, distribution, discovery, updating, and validation of attested evidence to establish the identity of a service remain largely open problems.

To illustrate the challenges of deploying confidential computing in practice, consider Signal contact discovery [39], one of the first successful confidential services with over 40M users. This directory service allows users to query the usernames associated with phone numbers. To protect user privacy, neither the cloud service provider (CSP) hosting the service TEEs nor the service owner should be able to learn these queries. When connecting to the service, and before querying any phone number, the Signal client checks the attestation report presented by the TEE to ensure it is running the correct code and configuration on an up-to-date platform. Service owners like Signal facilitate this check by making attestation fine-grained and providing the corresponding attestation-verification policy. Their TEE attestation includes a fresh authentication key to be used by clients to establish a secure channel to the service and encrypt their queries and responses. The service owner may additionally issue a certificate to endorse this attested public key. However, this is very brittle: any TEE code update or platform update requires updating clients with a new attestation-verification policy, which is hard to manage for scalable, long-lived services that involve many TEEs over their lifetime.

Verifying attestations for a given service also requires a custom client: the Signal chat application code embeds specific attestation-verification policy and code, and it is unclear how they would offer the same guarantees from a Web browser visiting `https://signal.org`. From a security viewpoint, the need for service-specific clients (and more generally for trusted mechanisms to discover them, review them, and keep them up-to-date) limits the benefits of confidential computing. Without independent scrutiny, for example, service owners could technically mount powerful targeted attacks by providing rogue clients that validate attestations of rogue TEEs.

We observe that clients heavily rely on DNS to identify services today, inasmuch as X.509, HTTPS and all Web security boundaries are ultimately rooted in DNS names. Hence, a refinement of the Signal certificate idea above is for the TEE to turn its attested key into a certificate for a well-known service name such as `directory.signal.org`.

The core novelty of this paper is to show how to systematically distribute attested keys by turning DNS itself into a confidential service, called attested DNS (aDNS). aDNS acts as a trusted controller for confidential services, i.e., it can authorize TEEs to participate in the service in a given role, and it can control the issuance of certificates to those TEEs. aDNS also makes the *governance* of the confidential service explicit by recording and enforcing policies that authorize its TEEs, and by making them discoverable to clients. Lastly, aDNS makes confidential services *auditable* by recording their history of policies, attestations, and certificates.

aDNS builds on standard security protocols such as TLS, DNSSEC, DANE, and ACME, which increases the security of records and certificates accepted both by enlightened clients (that verify attestations and can blame bad actors) and mainstream clients (that delegate this verification to aDNS), thereby providing a clear path for incremental deployment of confidential computing. Altogether, its features yield a new security foundation layer for the Web by providing a universal (i.e. service-agnostic) definition of confidential services: any service whose name is controlled by aDNS. Our definition comes with a familiar name-based user interface, supporting statements such as: "Every service whose name ends with `*.conf` runs in TEEs that meet its stated service policy."

Note that aDNS does not ascribe any semantics to the TEEs that participate in the service (in particular, it does not in itself guarantee that the data submitted to the service is confidential against the CSP and owners), but it enforces the intent of the service owners expressed as policies, which can be discovered and audited. As an example, a trading website may want to provide fairness of order execution, or a source code management website may want to guarantee the consistency of contributed commits using their TEEs. These are all valid examples of *confidential services*. We refer readers to the closely related problem of *code transparency* [3, 12] for understanding how to assign security properties to binary measurements of TEEs.

We implement an aDNS server and a browser extension client, and we show that they can be used to access different types of confidential services on various TEE platforms. We carefully optimize the connection protocol for aDNS clients so that checking attestation doesn't introduce any significant latency overhead. We also leverage the existing distributed DNS cache infrastructure to ensure scalability of aDNS, applying many of the lessons learned from similar DNS-based connection-time extensions in browsers, such as the recently deployed HTTPS/SVCB records. Our evaluation shows that the total connection latency overhead of aDNS (from the user intent to the first request packet) is negligible.

In summary, we make the following contributions:

1. a transparent attested DNS architecture to securely bind confidential service implementations to their domain names based on their registration policies (§3, §4);

2. protocols for querying and verifying attestation records, for registering TEEs as part of a service, and for obtaining certificates from an ACME-compliant certificate authority based on their attestation reports (§5);

3. protocols for controlled zone delegation, for bootstrapping trust, and for integration with DNSSEC PKI (§6);

4. implementations of an attested authoritative DNS service running as a network of TEEs, and of a lightweight browser extension for Web clients (§7);

5. sample confidential services (ML inference, token issuance, and privacy-preserving ad selection) integrated with aDNS, showing support for diverse TEEs: SGX enclaves, SEV-SNP confidential VMs, and confidential containers (§8.3);

6. an experimental evaluation of our implementation tested on the sample applications (§8);

In addition, §2 provides background on the technologies on which aDNS is built: DNS and DNSSEC, confidential computing, and transparency ledgers; §9 discusses related work on attested TLS and attested CAs; and an appendix provides additional details on sample applications. All artifacts are available from `https://doi.org/10.5281/zenodo.15611255`; see also `https://github.com/microsoft/ccfdns` for the latest source code.

2 Background

DNS, DNSSEC, and DANE. The Domain Name System (DNS), one of the most important foundations of the Internet, allows domain names to be associated with network addresses, service information, and other resources encoded as *resource records* (RR). Names range over series of labels such as `usenix` and `org`. The ability to delegate the authority for

names under a given suffix is the most defining feature of DNS, and is a major reason for its success as a distributed system: it offers full independent control of the delegated instance while simultaneously allowing a rough form of Internet governance at the upper layers of the hierarchy, which helps mitigate some of the problems such as namesquatting, typosquatting, Sybils, or name conflict resolution that plague other distributed identity systems [25, 58]. The concept of *(authority) zone* naturally emerges from delegation, and represents all the names controlled by a DNS instance that are not delegated. By policy, the root zone (controlled by ICANN) only contains delegations, whose names are called top-level domains (TLDs), such as .uk (delegated to Nominet) or .fr (delegated to AFNIC). Each TLD is responsible for the policies governing their zone. Most TLDs allow (for a fee) individuals and organizations to get a second-level name of their choice delegated to their own DNS zone.

DNS servers use a zone configuration file to describe their RRs. The Start of Authority (SOA) record declares a new authoritative zone, by indicating its primary DNS service. Conversely, a nameserver (NS) record delegates the authority for a name suffix to another name service. Common RR types include A and AAAA records for IPv4 and IPv6 addresses, CNAME records for aliasing, MX records for email services, etc. Records also include a time to live (TTL) indication, which indicates how long DNS clients may cache a given RR. This feature is essential to DNS scalability, as users are not expected to recursively contact authoritative DNS from the root to resolve a domain name. Instead, clients talk to a local resolver (managed by their network administrator or ISP) that will cache many of the upper layers of the DNS hierarchy. Then, if a client asks for the IP address of `www.usenix.org`, it may already have cached the NS record for `usenix.org` in the `org` zone and the SOA for `usenix.org`, and can thus directly query the right authoritative server.

DNSSEC was proposed in the late 90s to ensure the authenticity of DNS information. It introduces a new public key infrastructure (PKI) that follows the zone delegation structure of DNS. Each zone can declare a key signing key (KSK), whose hash is recorded in the parent zone as a delegated signer (DS) record. The KSK signs DNSKEY records containing the zone signing key (ZSK) effectively used to sign record sets (RRset), and these signatures are distributed in RRSIG records. To validate an RRSIG, a resolver needs the DNSKEY records for the ZSK and KSK and the parent DS records for all delegations in the name. Finally, DNSSEC supports proof of non-existence records (NSEC,NSEC3) to prevent censorship. Although DNSSEC adoption has been slow [33, 49] , it is now supported by most TLDs and resolvers, and gaining momentum, growing from 10,000 zones in 2010 to 1M in 2020 and 10M in 2024 [62].

DNS-Based Authentication of Named Entities (DANE) introduces TLSA records that indicate what public key or certificate should be used to communicate with a named service [21]. In TLS, DANE keys can be used either together with X.509 certificates for key pinning or directly for authentication [64], replacing PKIX with the DNSSEC PKI. DANE has gained significant adoption (750,000 zones found by SecSpider in 2024) especially for mail services. For a short time DANE was even enabled in Chrome [28].

Confidential Computing. Confidential computing leverages hardware capabilities to create TEEs that isolate their code and data from the rest of the system, including privileged components like the host operating system and the hypervisor. Because isolation is enforced in hardware, a TEE can be trusted to protect the confidentiality and integrity of the code and data it processes, even from the CSP that manages the cloud hosting software stack. To establish trust, confidential-computing hardware provides remote attestation: TEEs can ask the hardware to sign a message together with measurements of their initial configuration (code and data) with a key only accessible to the hardware. The signature is backed by a platform certificate issued by the hardware provider. Through remote attestation a client can thus verify a TEE's trusted computing base (TCB) and hardware platform.

Modern CPUs from Intel, AMD, and Arm provide confidential computing capabilities. Intel Software Guard Extensions (SGX) [22] support the creation of in-process isolated memory regions. AMD Secure Encrypted Virtualization (SEV-SNP) [4], Intel Trusted Domain Extensions (TDX) [9], and Arm Confidential Compute Architecture (CCA) [32] provide VM-based TEEs that isolate virtual machines. Accelerators such as Nvidia GPUs [13] and Graphcore IPUs [63] also provide confidential computing capabilities.

Most existing TEE applications like Signal contact discovery [24] and key recovery [10] rely on application-level attestation checks, using a simple locally configured attestation policy such as verifying the SGX enclave signer with a fixed key. One challenge with this approach is that the client opens a transport session (typically using TLS) with a service that is not yet trusted, and must thus request and check attestation evidence before sending any secret data over the same session. To stop a man-in-the-middle attack between the client and the TEE, for instance by a malicious service owner who can get a valid TLS certificate for the service, it is also necessary to bind the attestation to the TLS handshake, a mechanism generally referred to as *remotely-attested TLS* (RA-TLS) (see §9 for a survey). RA-TLS protocols fall in two groups: either they modify TLS to bind the attestation, breaking compatibility with legacy clients, or they embed attestations in certificates, breaking compatibility with existing CAs (and thus with legacy clients).

Relying parties can also delegate verification to attestation services managed by the hardware provider or the CSP [17,40], which are best positioned to maintain up-to-date attestation evidence for their platforms. This involves first gaining trust in these services, preferably on the basis of their own attestation [6], then verifying the evidence they produce upon

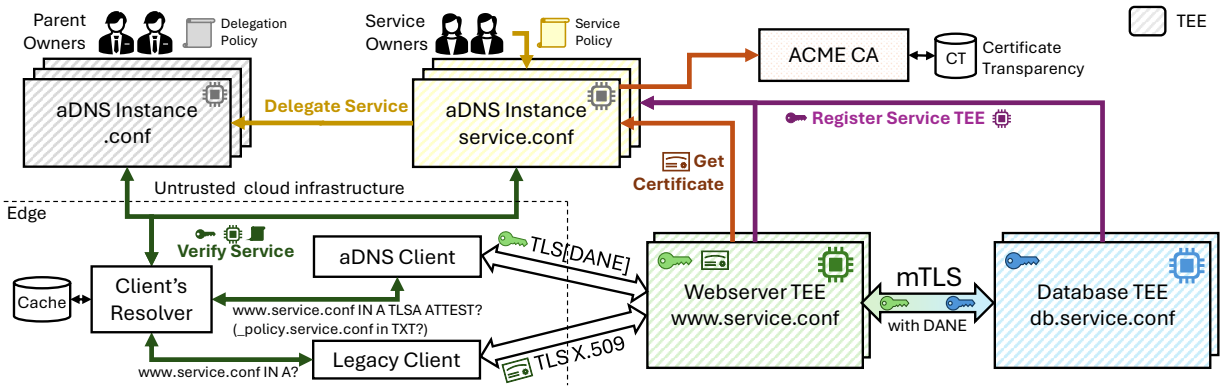

Figure 1: Architecture of Attested DNS

request, e.g., the entity attestation tokens (EAT) [36] they issue after verifying attestation reports (see §9 for a comparison with aDNS and §B for additional details on EAT).

Transparency. Transparency is a mechanism to force authorities to be consistent, auditable, and accountable. Its most successful implementation is the certificate transparency system [29] (CT) to keep certificate authorities from issuing bad certificates without getting caught. Transparency relies on a public verifiable append-only ledger to record information. This ledger can be routinely checked for consistency, or deeply audited in case an incident is suspected or discovered. Code transparency [3,12] similarly uses a transparency ledger to record information about how a given binary has been built, e.g. from which source files, in which build environment, etc.

In aDNS, we are interested in the transparency of DNS records to make confidential services transparent. Our implementation of aDNS is based on the Confidential Consortium Framework (CCF) [53], which operates a ledger between replicas hosted in SGX enclaves. CCF also offers independently verifiable proofs of ledger inclusion, which can be used for holding its replicas and their operator accountable, even in case of TEE compromise [57]. We use the CCF ledger to make the aDNS zone configurations transparent.

3 Architecture Overview

Figure 1 shows the high-level architecture of Attested DNS (aDNS). At the bottom right, a sample confidential service is composed of two kinds of nodes: a front-end Web server and a database, each running in its own trusted execution environment (TEE). There are many types and implementations of TEEs (see §2) but from an aDNS perspective, we only assume that the TEEs can produce remote attestation reports.

The first core idea of aDNS is to leverage existing DNS infrastructure to distribute and cache attested information, such that enlightened clients can obtain and validate them in parallel with TLS connection establishment to services, by piggybacking on existing DNS requests to avoid incurring extra latency. For simplicity, our presentation assumes that each confidential service is controlled by its own aDNS instance (that is, its own set of authoritative DNS servers), even though, in practice, many services can share the same zone. Hence, when our sample service is set up, a new aDNS instance is created for the zone `service.conf` (Figure 1, top center).

When a TEE starts for a given service, it must first register its attestation report in its zone. For example, a TEE running a Web server at IP address A with public key K^+ in a TLS certificate for domain $D = \texttt{www.service.conf}$ first generates an attestation report Q that measures its code and configuration including (A, K^+, D). The TEE then registers Q at its aDNS instance, which validates Q against the registration policy on record and then creates (or updates) the DNS records for D: it adds an address record $D \mapsto A$, an attestation record $D \mapsto Q$, and a TLSA record that pins the public key K^+, thereby endorsing its use to authenticate D.

When a client connects to `https://www.service.conf`, it must at least resolve the address of the service. Modern clients also query additional DNS records, such as SVCB records to check for HTTP/3 or QUIC support, or DANE records for the TLS public key of the server. We extend aDNS-aware clients in a similar way, such that they also query for attestation records in the same DNS request.

Since the attestation records are obtained from an otherwise untrusted DNS infrastructure, they need to be authenticated, so aDNS relies on DNSSEC to ensure its records are not tampered with. But the aDNS operator could also be malicious, and may tamper with the binding between domain names and attestation reports, or sign spurious records. Therefore, the aDNS server itself must run as a confidential service protected by TEEs, and enlightened clients must also check the aDNS server's attestation before trusting its records.

The second core idea is to make the TEE registration policies of confidential services discoverable and transparent. In general, if a client connects to a confidential service for the first time, it will be impossible to interpret the attestation re-

port, as the client has no idea what code the service is meant to run, or on which TEE platform. We introduce registration policies, which describe the conditions an attestation report should meet in order to be recorded on the DNS server. The registration policy of a service can also be fetched from DNS to be cached and inspected at the client, to help users decide if they trust a particular service or not, before the client even connects to the server.

The third core idea is to rely on hierarchical trust in the DNS tree to deal with DNS service attestation, so that clients only need to establish trust in a small set of root aDNS instances with well-known names. For simplicity, our presentation assumes a single root aDNS instance that operates the `.conf` top-level domain (TLD), but in practice many attested zones may co-exist in the name hierarchy. Hence, we have the root zone (managed by ICANN with the root DNSSEC keys), an attested TLD zone `conf` (top-left in the figure, controlled by an aDNS instance with its own attested KSK), and an attested service zone `service.conf` (top-center, controlled by another aDNS instance with its own attested KSK). Any time a sub-zone is delegated to a new aDNS instance (e.g. `service.conf`), it is the responsibility of the parent aDNS to verify the attestation of the new instance. Therefore, aDNS instances are configured with a *delegation policy* that describes what constitutes valid implementations of aDNS and valid TEE platforms to run them. Delegation policies are applied transitively through delegation, so they can only become more restrictive at each delegation step. For instance, the TLD instance may allow implementations of aDNS running on Intel, AMD, or Arm CPUs but the delegation policy for `service.conf` may only allow Intel or AMD CPUs.

Finally, a major benefit of aDNS is the ability for a TEE to take a dependency on other confidential services (inside or outside its DNS zone), for instance on a database at `db.service.conf`, without having to configure their credentials. Today, services that want to use mutually-authenticated TLS require a control plane (e.g. a service mesh [31] such as OpenShift and a proxy such as Envoy [38]) to provision client certificates and deal with authentication and authorization. With aDNS, both endpoints can use their DANE raw public keys, so that trust is established on the basis of the policy enforced for each service's name, without having to deal with any certificates or attestations in application code.

4 Security Goals and Threat Model

Core security goal. A confidential service comprises a DNS zone (e.g. `service.conf`) and a registration policy that specifies the rules that a TEE must satisfy to run the service under a given name in this zone (e.g. `www.service.conf`). The main goal of aDNS is to ensure that if a client successfully establishes a TLS session to a server with such a DNS name (e.g., a Web server), then this server must run in a TEE with an attestation that meets the associated registration policy.

Client Type	Web PKI	DNSSEC PKI	Service's aDNS	Parent's aDNS
aDNS Client	Untrusted	Untrusted	Untrusted	Trusted
aDNS Client + discovery	Untrusted	Untrusted	Trusted Transparent	Trusted
DANE	Untrusted	Trusted	Trusted	Trusted
X.509	Trusted Transparent	Untrusted	Trusted	Trusted

Table 1: Trust assumptions for different kinds of clients

The registration policy and service code are authored by a service owner who is responsible for authoring them in line with users' expectations (e.g., to ensure privacy of data sent to the service). The service owner is trusted, but their actions are auditable through aDNS's transparency log.

We assume the hardware-based TEE design, implementation, and manufacturing are correct; and we trust their PKI for endorsing device certificates and TCB updates (e.g. firmware). The service is deployed and operated by an untrusted service operator who may compromise any components outside TEEs (such as host operating systems, hypervisors, devices, networks, and non-confidential services).

We design aDNS to provide this core guarantee to a broad range of clients with different capabilities, detailed below. Table 1 summarizes the trust assumptions required for each type of client for our core security to hold, assuming the client itself is correctly implemented.

A fully-enlightened aDNS client (implemented, e.g., as a local resolver or a browser extension) fetches attestation from DNS and verifies it against a trusted local attestation policy. Although these clients rely on aDNS as an untrusted cache for availability and freshness, their security does not depend on the service owner or operator. In other words, they are as secure as today's service-specific clients that fetch and verify attestation at the application layer or using RA-TLS.

Next, we consider an aDNS client with the same capabilities but no *a priori* knowledge of the service it connects to, which we refer to as "discovery" because the client must discover the registration policy and either verify its authenticity (using, e.g., signed policies) or trust that the parent aDNS instance only accepts policies from authorized service owners. In the latter case, the client trusts both the service owner and aDNS, but it can rely on transparency and other mechanisms to mitigate that trust. The client may present the registration policy to the user to review, and retain it for future connections and audits. The client may also establish trust in aDNS instances it discovers the first time it needs to authenticate their records, by (1) requesting their attestation records, (2) verifying them against a trusted local policy for aDNS services, and (3) pinning their attested KSKs—this bootstrapping step is a special case of a fully-enlightened client connecting to aDNS as a confidential service. These different options are discussed in §5.3.

Finally, we consider two kinds of aDNS-unaware clients that also benefit from aDNS's attestation policy enforcement.

Clients that support DANE for TLS authentication will check that the peer's key is registered on aDNS, at the cost of additional trust in DNSSEC for the authenticity of TLSA records, and in aDNS instances for verifying attestation reports of TEE instances before publishing TLSA records. These clients also trust the parent aDNS for correct delegation to the service aDNS. However, no trust is required in Web PKI CAs since these clients authenticate services based on TLSA records instead of certificates.

Clients that solely rely on X.509 certificates issued by Web PKI CAs must additionally trust that these CAs issue certificates only after authenticating CAA records using DNSSEC and challenging TXT records (§5.2) as prescribed by the CA/Browser Forum Baseline Requirements [8]. While rogue certificate issuance cannot be prevented, CAs can be held accountable using Certificate Transparency and the certificate issuance log of aDNS.

Additional security properties. aDNS provides two additional properties to some clients.

Service transparency: all clients that use at least DNSSEC are guaranteed they are served the same records (including the service policy) as every other client as long as aDNS is not compromised. Each aDNS instance also maintains a tamper-proof ledger recording all changes to the records in the zone—this allows an audit of the full historical TCB of both the service and its aDNS instance, and also provides residual security in case of TEE compromise.

Freshness and linearity: aDNS clients (with or without discovery) are guaranteed to connect to up-to-date registered TEEs with fresh attestation evidence assuming clocks are synchronized and TEEs have access to a trusted time source (§7.2 discusses how we implement this assumption).

Assumptions on DNSSEC and PKIX. Our work builds on the two main PKIs for the Internet, namely DNSSEC and PKIX. While their security analysis is beyond the scope of this paper, we note that these two PKIs are interdependent (e.g., certificate issuance depends on DNS for domain validation, DoT and DoH rely on PKIX certificates, and HTTP3 combines DNSSEC HTTPS records and PKIX certificates for name authentication).

Our DANE and X.509 clients are fundamentally exposed to these PKIs and, as explained above, their security as they connect to confidential services requires additional trust assumptions. For example, DNSSEC protects DANE clients from record spoofing by network attackers, including cache poisoning in intermediate resolvers, but not from rogue authoritative DNS servers (for any name suffix of the service) that may issue authentic but spurious records. Similarly, X.509 clients are fundamentally exposed to rogue CAs. By design, aDNS complies with these PKIs' existing standards and security guidance, so at least it does not decrease their security, nor the effectiveness of existing countermeasures.

In contrast, enlightened clients independently verify evidence carried in records and certificates based on attestation reports and policies, or trust aDNS instances to perform these verifications on their behalf after verifying their own attestations, and thus are not directly exposed to PKI threats, although they may specifically depend on some hardware root certificates for attestation verification.

More generally, all clients remain exposed to high-level attacks based on names, e.g., they won't protect users tricked into clicking on a malicious link owned by an attacker.

5 Protocols for Confidential Services

5.1 Registering a TEE for a Service

We first describe the protocol to provision a new TEE to run as part of a confidential service, and more generally discuss the attestation-based authorization policies enforced by aDNS. The creation and registration of the service itself in an attested DNS zone are explained in §6.

Attestation reports presented by service TEEs are of the form $Q[platform, code, config, \vec{K}^+, time]$ where

- *platform* includes platform-specific information, such as its hardware identity and firmware patch level; we keep the details and the actual hardware signature abstract;
- *code* is a digest of the code running in the TEE: based on the attestation, the verifier can infer that any security invariant programmatically enforced by that code holds for the TEE that issues the report;
- *config* is the configuration of this TEE as part of an instance of the service; it notably includes the domain name and a unique identifier for the service instance, e.g., $D =$ `service.conf` and `v0.1`; a unique prefix for this TEE as part of this service, e.g., `node42.front-end`; and the list of origins to be served by this TEE, including their prefixes, IP addresses, protocols, ports, and key indexes, e.g., `www:{https, 443, 1}`.
- $\vec{K}^+$ is a list of public keys presented by the TEE. Each key is labelled with a signature algorithm and a usage—either DANE or X.509. The first key is always the DANE key used to uniquely identify this TEE;
- *time* is the attestation time presented by the TEE.

Intuitively, the attestation report transparently binds the name of the TEE within the service, e.g. `node42.front-end.`D, to its hardware platform, its software configuration, and its fresh identity key; and it similarly binds the origins it serves to fresh authentication keys.

Registration enables a TEE provisioned by an untrusted service operator to join a confidential service, as follows.

1. The service operator creates a new TEE for its choice of hardware *platform*, *code* and *config*.
2. The TEE starts executing this code, which presumably[1]
   (a) samples its fresh private keys, records its host time, obtains a hardware-attested report of the form given above, and gathers any supporting evidence, such as platform certificates.
   (b) retrieves and verifies the server keys and attestations of the aDNS instance of the service, as detailed in §5.3, based on the DNS name in *config*.
   (c) opens a mutually-authenticated TLS connection with aDNS, presenting its DANE identity key as client key and verifying that aDNS presents one of its retrieved attested key as server key.
   (d) posts a REGISTER request with its attestation report and supporting evidence as payload.
3. aDNS accepts the connection, verifies the attestation report, checks that its claimed service name is in the service zone it controls, that it passes the registration policy on record for this name, that its time is reasonably recent, and that its primary DANE public key matches the raw key presented in step 2c.

   If all these checks succeed, aDNS then installs new ATTEST, TLSA, and A/AAAA records for each of the origins declared in the attested configuration. In particular, it installs a TLSA record for DANE authentication using its primary key at its unique name. (Any clashes with existing records cause the registration to fail.)
4. Once aDNS acknowledges its registration, the TEE starts running on behalf of the service. For any subsequent connection with aDNS, it authenticates as a DANE client using the TLSA record for its identity key.

The TEE may then request certificates for some of its origins (see §5.2). The TEE may also update its registration with a fresh attestation report (e.g., to periodically rotate its keys), resuming from steps 2, subject to the service registration-update policy. The main difference is that old records are removed before installing new records for its updated origins.

Security. For any successfully registered TEE, aDNS must have verified and logged a corresponding attestation report that passed the service policy at the time of registration. Since the primary DANE key must be freshly sampled, the registration report uniquely identifies honest TEE instances. Further, since aDNS verifies possession of the DANE private key in the registration protocol (by requiring mutually-authenticated TLS), attackers cannot replay a registration request unless the TEE is compromised or its code is malicious. Service owners may want to limit how long a registration is valid for and require TEEs to re-attest regularly. We could require the report to include a random challenge by aDNS to ensure freshness of the report; however, since our client freshness guarantees are time-based, we instead rely on attested time. Most TEE implementations do not provide a secure time primitive, but we can rely on signed timestamps from an external trusted time source to limit how long stale reports can be used.

[1] Otherwise, attestation verification will fail at step 3.

5.2 Obtaining Certificates for Registered TEEs

Some TEEs need Web PKI certificates in addition to DANE to authenticate to X.509 clients, so aDNS also has a protocol to obtain certificates from any ACME-compliant CA. This protocol may run at any time after TEE registration. Its main security goal is to ensure that certificates for names in the service zone endorse only keys of its registered TEE instances, and thus to prevent the use of any other certificates to impersonate the service.

ACME and CAA (background). Automatic Certificate Management Environment (ACME, RFC 8555 [5]) is a standard that enables certificate authorities (CAs) to verify that a certificate request legitimately represents the domain names to be endorsed in the resulting certificate without human intervention. It is supported notably by Let's Encrypt and thus used to issue the majority of Web certificates, endorsing more than 500M domain names to date. ACME can be configured to verify ownership of domain names by requesting inclusion of a specific TXT record bound to a fresh random challenge for each of the requested domain names.

Certification Authority Authorization Resource Records (CAA, RFC 8659 and 8657) is a complementary standard that defines records to indicate which CAs and verification methods should be used for certificate issuance in a zone; it is considered best practice by most root CAs. The combination of ACME and CAA allows us to limit the TCB of certificate issuance for confidential services to their authoritative aDNS instance and their designated CA.

aDNS-based ACME. ACME is based on a series of JSON-encoded HTTPS requests from a client to the CA. We omit some details, such as error handling and request state polling. ACME clients sign all their requests with an account key, recorded by the CA when it creates the account.

For our purpose, the protocol (given in Figure 2) involves a requesting service TEE, its aDNS instance acting as ACME client, and the CA recorded in the service configuration, such as `letsencrypt.org`, acting as ACME server. As part of its initialization, the authoritative aDNS for the service generates a public-private keypair (K_a^+, K_a^-) and creates an account for all certificate requests in the service zone—the private account key K_a^- never leaves aDNS TEEs. aDNS also installs CAA records to prevent any standard-compliant CA from issuing certificates except for those it will explicitly request.

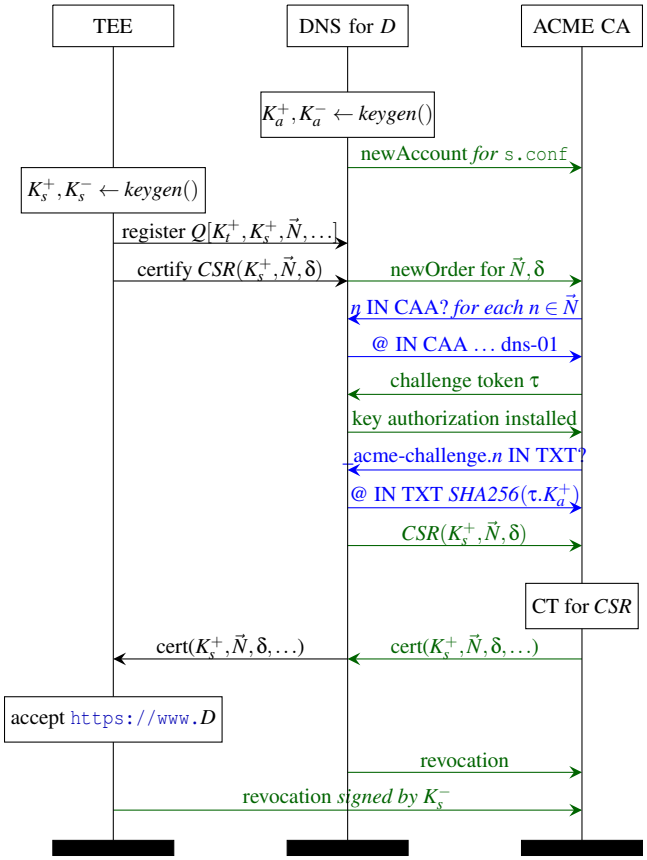

Figure 2: *Service registration and certification protocol, including ACME (green) and DNS (blue) messages.* K_a^+ *is the account key for the service implicitly used to authenticate ACME requests; for simplicity, we omit the identity key of the TEE used to authenticate its register and certify commands;* K_s^+ *and* $\vec{N}$ *are the Web public key and HTTPS origins included in the TEE attestation and authorized by its service registration for which the TEE requests a certificate;* δ *abbreviates the requested validity interval (the notBefore and notAfter fields in the certificate).*

For example, this can be achieved by adding a record

$D \mapsto$ `CAA 0 issue "letsencrypt.org;validationmethods=dns-01"`

stating that only `letsencrypt.org` can issue a certificate for any name that ends with D and that it must validate ownership of that name using a DNS challenge.

Assume a TEE registered a single X.509 keypair (K_s^+, K_s^-) configured to serve both `www.`D and D under a common certificate for K_s^+ (following common Web practice). To request the certificate, the TEE prepares and signs a PKCS#10 Certification Signing Request (CSR) for one of its attested X.509 keys and some of the origins recorded in its currently-registered attestation record. The CSR includes the public key, the requested names $\vec{N}$, and the notBefore and notAfter fields δ to be included in the resulting certificate; it is signed with the corresponding private key (K_s^-) as proof of its possession. The TEE then delegates certificate issuance to its aDNS by sending a certify command for this CSR.

aDNS verifies that the CSR matches the template for one of the X.509 keys in the current attestation record for the TEE and that every name $n \in \vec{N}$ ends with the service name. It then logs the CSR and sends an ACME newOrder request for $\vec{N}$ and δ signed by the confidential service account key.

The CA verifies that the request is legitimate: it first sends DNS requests to collect any relevant CAA records, starting with each full name $n \in \vec{N}$ and removing one prefix label at a time, until it gets a CAA record or it reaches the root. This crucially relies on DNSSEC to authenticate CAA records or their absence. Accordingly, the CA generates a fresh dns-01 challenge and sends it back to aDNS.

Following the standard `dns-01` verification method, aDNS proves that it controls every requested name n by computing an 'authorization key' k as the hash of the challenge concatenated with the hash of its public account key and by installing a '`TXT 300` k' record at every name `_acme-challenge.`n. It then signals the CA that it is ready to meet its self-assigned challenges. (As an invariant, to prevent interference with other sub-protocols, aDNS refuses to add TXT records with the reserved label `_acme-challenge` for any other purpose.)

The CA queries each of these records, authenticates them via DNSSEC, and verifies that their payloads match the authorization keys it expects for this order. It is now ready to receive the original self-signed CSR prepared by the TEE, verify its signature, match its contents against the pending order, and trigger the production of a certificate that endorses this contents, in two stages: (1) the CA publicly registers the CSR to ensure certificate transparency; (2) the CA signs the new certificate using its own private signing key, and returns this certificate to our aDNS client. aDNS verifies that the certificate matches the CSR and complies with certificate transparency. It logs this certificate, installs the corresponding TLSA records, and returns the certificate to the requesting TEE, to be presented to X.509 clients for any name in $\vec{N}$.

Security. The main property of interest is that a client accepts a certificate as valid for any origin with the name of a confidential service only if (1) the key has been attested by a TEE registered for the service; (2) the corresponding PKIX origin has been authorized by the service policy; (3) the leaf certificate has been endorsed by the CA recorded in the service configuration; and (4) the aDNS for the service has logged this certificate after successfully completing the issuance protocol above. The protocol crucially depends on the security of ACME and its use of DNSSEC by the CA to authenticate both CAA and challenge TXT records. ACME CAs are encouraged to implement additional mitigations to protect against an attacker corrupting a DNSSEC key or downgrading DNSSEC (if it doesn't chain to the root keys), such as performing DNS resolution from multiple network paths

to ensure they are consistent. Pinning aDNS KSKs can also protect CAs from the rest of the DNSSEC PKI.

If a CA that doesn't properly implement ACME or CAA issues a certificate to a name in a confidential service zone (for instance, using a weak validation method such as sending a challenge to the email address of the service owner), then the service owner may attack legacy X.509 clients. A Certificate Transparency monitor can detect the issuance of such certificates and hold the service owner (or bad CA) accountable.

5.3 Connecting to a Confidential Service

Assume a client attempts to connect to a confidential service at `https://D`. The full process for an enlightened aDNS-aware client, shown in Figure 3, relies on standard protocols. It consists of preparatory DNS queries to retrieve and validate the attested key of the server, followed by a standard TLS handshake authenticated with this key.

Enlightened clients. The attestation reports for `https://D` (see §5.1 for details) are available as `ATTEST` records at `_443._https.`*D*. As with `SVCB` service binding records (RFC 9460 [56]), the protocol and port are encoded in a prefix to enable multiple services and instances at a given name.[2]

If an aDNS client is connecting to a service for the first time and doesn't have a local service policy (discovery), it may also query for the service policy (available as a `TXT` record at `_policy.D`), and recursively, the attestation evidence of the service's aDNS instance. The client may either ask the user to explicitly review this policy, or trust the decision of the parent's aDNS to authorize the delegation.

The query for `ATTEST` and `TXT` records may be bundled with basic DNS address resolution (`A` and `AAAA` records) in a single packet. Reports and addresses are indexed the same way so the client knows which instance to expect when connecting to any address. Responses are recursively cached at every resolver (including at the client) and may return additional relevant records (with asterisks in the figure).

The client can initiate the connection to the service in parallel with the verification of the service's attestation report. However, clients must be careful to send 0-RTT data only after the attestation has been verified. The client must also check that TLS server key matches one of the keys in the attestation report. The client is expected to cache the result of attestation verification (and optionally, the authorized policy) for the minimum duration of either the TTL of the ATTEST record, or the maximum configured duration of the client.

Mainstream clients. Along with the DNS address resolution response, if the client resolver supports DNSSEC, it will receive additional records to authenticate the response, including the RRSIG record of the A (and optionally TLSA) RRset, the DNSKEY records of the service zone, and recursively, the chain of DS records up to either a pinned trusted aDNS root instance, or a global DNSSEC root key. If the client supports DANE, it will query the `TLSA` records and check one of their keys matches the TLS raw public key or certificate presented during the TLS handshake with the service. Otherwise, the client will validate the certificate using its set of trusted root CAs and ensure it includes the name of the service.

[2]We could also distribute attestations using key-value pairs in SVCB records for additional compatibility with legacy resolvers.

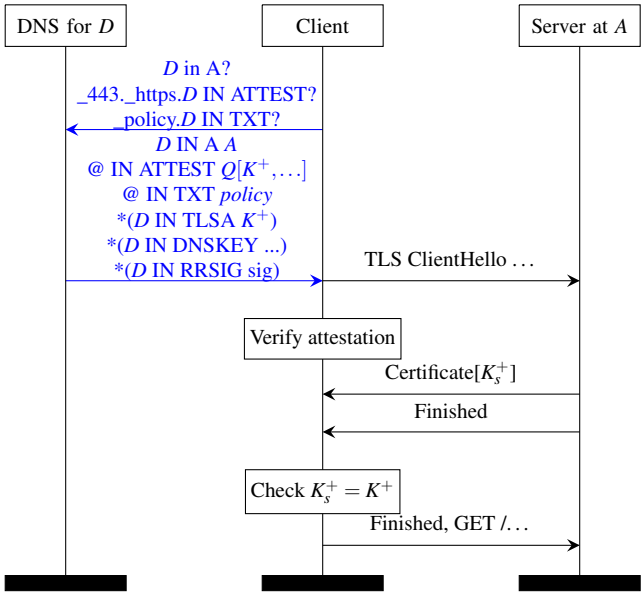

Figure 3: *Enlightened aDNS client connecting to an HTTPS service endpoint at D with unknown address and policy. All DNS messages are in blue.*

Security. We consider each of the clients in Table 1 in turn, first addressing how they meet our main security goal, then discussing additional freshness and transparency guarantees.

An **aDNS client with a local service policy** is as secure as a client using any variant of attested TLS. The main difference between the two is that the server attestation is received in a signed but *a priori* untrusted DNS record, instead of an *a priori* untrusted TLS connection, but it does not matter as long as this attestation complies with the service policy and authenticates the server TLS key. If the client and aDNS clocks are (roughly) synchronized and the client resolver checks the record has not expired, then the aDNS client also knows the instance it connects to has a fresh attestation for an up-to-date policy (up to the TTL on its attestation record).

An **aDNS client discovering the service policy** either knows how to locally verify the (a priori untrusted) service policy retrieved from DNS—in which case it is equivalent to the case above—or it must trust that aDNS correctly authorized and authenticated this policy for this service name, with different subcases depending on zone delegation.

If a client already trusts an aDNS instance (and in particular its delegation policy) for a prefix of the service name, then,

by recursion over each delegation step to the aDNS instance for the service, it can also trust this service policy. As with any other confidential service, trust in an aDNS instance can be achieved by fetching and verifying an attestation for this instance against a locally-trusted aDNS-verification policy, and then pinning its attested KSK for DNSSEC validation of any record in its zone of authority. Otherwise, if the client also discovers the whole attested zone for the service, then it must also trust the DNSSEC PKI to authenticate the KSK of the aDNS instance at the apex of this zone.

A **DANE client** effectively delegates attestation checking to aDNS, but as long as aDNS is trusted (as discussed above), this is equivalent for our main goal to an aDNS client that discovers then locally verifies the policy, and it is slightly weaker for transparency inasmuch as the DANE client cannot log or review the policy applied by aDNS.

An **X.509 client** additionally depends on the security of the certificate issuance protocol, discussed in §5.2 above. In other words, if its trusted CAs issue certificates for a confidential service only after CAA and ACME DNS-based name validation, then these clients get the same security as DANE clients, minus the freshness guarantees of DNSSEC.

In practice, the trust assumptions for mainstream clients are also mitigated when some aDNS clients verify attestation reports of aDNS instances, check aDNS and CT logs, and report inconsistencies. For example, by verifying attested evidence for aDNS, aDNS clients can ensure it serves the same records to all clients. In particular, any use of registration policies that do not meet end-user expectations or permit code with known vulnerabilities can be detected by aDNS clients or auditor.

6 aDNS Services and Zone Delegation

Creating an aDNS Instance. The protocol for starting an instance of aDNS in the DNS zone hierarchy is a refinement of the protocol for adding a TEE to a confidential service (§5.1). The operator creates a TEE that runs the initial primary DNS server, for its choice of code and configuration. (This may involve DNS requests to the intended parent aDNS instance to make sure the child and parent configurations are aligned, and prevent later attestation-verification failures.)

As it starts, the new aDNS TEE samples fresh keypairs and produces an attestation of the form $Q(platform, code, config, KSK^+, K^+_{ns0}, time)$ where *config* sets the intended full name of the zone (e.g. $D =$ `service.conf`), the new zone's policies, and the parent DNS presumably willing to delegate this zone. The configuration also indicates whether the parent runs aDNS or just DNSSEC, in which case the new zone is the apex of a island of attested names in the DNSSEC PKI.

The attestation includes two fresh public keys: its initial DNSSEC key-signing key KSK^+ and a TLS authentication key K^+_{ns0} used to serve both DNS queries over HTTPS and TLS (for reading resource records) and aDNS commands over HTTPS (for modifying them).

We name all authoritative DNS servers using reserved labels of the form `ns`n for $n \geq 0$ prefixed to the zone name. As outlined in the implementation (§7), the startup process may also involve mutually-attested communications with auxiliary TEEs that run backup DNS servers for the new zone.

The policies in the configuration are used to guard every command that may update the resource records in the zone or its authoritative DNS servers. This includes, for instance, commands for service TEE registration, sub-zone delegation, certificate issuance, code update, and policy update. In this presentation, we make the following simplifying assumption: aDNS instances are either 'leaf' zones for service TEE registration, or 'intermediate' zones that only support delegation. While not essential, this prevents conflicts between registration and delegations under the same name.

Unless it is the apex of an attested island, the new aDNS instance then sends a delegation request to its parent, as detailed below. Once accepted, it finally gets its Web key certified by the CA indicated in its configuration, as detailed in §5.2, acting both as requesting TEE and ACME client.

Delegating a Zone to a Child aDNS Instance. We now explain how the parent aDNS handles delegation requests. The request payload mostly consists of the child's attestation report, together with additional DNS information provided by the primary such as glue records containing the IP addresses of its DNS servers to be advertised by the parent. It is sent over mutually-authenticated TLS using DANE keys. Before opening the connection (see §5.3), the child requests and verifies the TLSA and ATTEST records to confirm the authenticity of the DANE key. In contrast, since the child DNS service is not yet part of the DNSSEC hierarchy, the parent provisionally accepts connections authenticated by *any* client key and then verifies that this key matches the attested DANE key in the report at the application level.

Upon receiving a new delegation request, the parent first verifies that it passes its own delegation policy. This involves verifying the child attestation report, checking that its code and configuration are correct for a new aDNS instance, checking that the name requested by the child is a subzone of the parent zone and that it has not yet been delegated. It also verifies that the child's delegation policy is compatible with its own (and ancestors') policies (see below). Assuming all these checks succeed, the parent updates its records to implement the delegation: following DNS and DNSSEC specifications, it creates a NS record for the child, a DS record for the child's attested key (KSK^+ in Q) at the child name, and glue records. It also creates an ATTEST record that logs the child attestation, making it transparent and auditable. The child waits for the confirmation that it delegation request has been accepted, then starts serving DNS requests and aDNS commands.

Later on, the child may send delegation-update requests to refresh its attestation report, its DNSSEC key, or just update its glue records. The parent similarly verifies then executes these requests; the main difference is that the child authenticates

with a DANE key on record in the attestation currently held by the parent. (This prevents conflicts with fresh delegations.)

Policies for Delegation and Code Updates We finally present a policy design that aims to balance the privileges of child and parent aDNS instances.

From a classic DNS viewpoint, the operator of a parent zone retains strong discretionary powers over its delegated sub-zones. In particular, it can modify or delete their delegation records, potentially endangering their security. With aDNS, in contrast, the owner of the parent zone only fixes its initial policies (including its ability to update them), which are then automatically enforced. This defines and limits the scope of discretionary updates, and ensures that these policies and their consequences will be transparently logged for audit.

To enforce meaningful security invariants across large sub-delegated zones, e.g. to enforce that all its aDNS TEEs meet minimal platform and code requirements, it is important to limit the scope of the policies that will be recursively enforced by child aDNS instances. To this end, we propose that the child delegation policy (including the policy to verify children aDNS attestations) be at least as strict as their parents'.

Our implementation enforces it by including in the child aDNS configuration the policies of all its parents up to the apex of the attested island, by checking this inclusion as part of the verification performed by the parent during delegation, and by independently evaluating *all* the delegation policies in the aDNS configuration before accepting delegation requests. In contrast, delegated instances have full control over their TEE registration policies for any other service besides aDNS.

7 Implementation

7.1 Attestation Library

To support a broad variety of TEEs, we need a library that can verify different types of reports and is highly portable to run both in the aDNS server and in many clients. We could delegate verification to attestation services provided by CSPs and hardware manufacturers, but this would add a highly trusted entity to our TCB. Instead, we implement Ravl, a new remote attestation validation library that only depends on OpenSSL. Ravl introduces a universal attestation format, consisting of a TEE type, a main report data, and a collection of tagged collaterals for certificates, CRLs and OCSP proofs, RIM bundles or custom collaterals such as Intel TCBInfo. Ravl is also capable of fetching outdated or missing collaterals, and offers historical validation (validity at a given timestamp).

7.2 aDNS Server

We implement an aDNS server using the Confidential Consortium Framework [53] (CCF), which runs in Intel SGX enclaves. We choose SGX as it remains the smallest TCB option among TEE implementations offered by mainstream cloud providers. CCF simplifies the implementation of aDNS thanks to several of its built-in features. Some of these features are mainly for convenience, such as the fact that CCF applications only need to implement application endpoints and the framework takes cares of mundane tasks such as managing host to enclave communication, TLS sessions, or HTTP requests. Other features solve difficult issues for us.

First, CCF provides operational reliability. Most DNS services require at least 2 instances for availability, but maintaining consistency between instances can be hard. Traditionally, one of the instances is the primary while backups use the zone transfer features of DNS to copy the primary's configuration. Authorizing and managing backups is even more challenging in the confidential computing threat model, where enclaves may be forked and kept alive by an attacker to keep stale configurations live. CCF implements a variant of the Raft [48] consensus to synchronize the leader with backup nodes. It manages mutual attestation between its replicas and takes care of reconfiguration automatically if the primary fails.

Second, CCF produces a tamper-proof ledger of transactions. The ledger is used internally by CCF to ensure the persistence and recoverability of the state of the application (especially if all replicas crash, a scenario referred to as disaster recovery). CCF maintains both the integrity (with a Merkle tree) and confidentiality (by encrypting entries with keys only accessible to TEEs) of its ledger. We take advantage of its ledger to provide auditability and transparency of confidential services served through aDNS. Every change to the zone configuration (e.g. registering a TEE) is recorded as a transaction in the ledger. Therefore, if an aDNS server publishes its ledger, an auditor can check the history of all TEEs that ever participated in a service. Another benefit of the ledger is that it enables *receipts*, consisting of a Merkle path and a signed root hash to prove that a given transaction was recorded in the ledger at a given state. We use receipts to verify a fresh aDNS instance is properly configured for delegation.

Third, CCF has an explicit concept of the *governance consortium* of a service, that is, the group of people that collectively represents the trusted entity operating and updating the service. In aDNS, an example of trusted configuration is the registration policy of a zone. Hence, updating the registration policy of a zone is considered a governance operation, which can only be initiated by *governance members*, and adopted by following a voting process directed by the *constitution* recorded at the start of the ledger. Each member is identified by a certificate, and each vote and decision is recorded in the ledger to ensure that the governance decisions are transparent.

CCF governance also helps us solve the trust bootstrapping problem with aDNS. Since delegation policies are applied transitively, the governance consortium of a root aDNS instance (at the apex of an attested zone) controls the baseline of what constitutes a valid aDNS instance, and thus is particularly trusted. A possible model for aDNS deployment is for a non-profit organization to operate a TLD, in a similar way

that the non-profit Public Internet Registry operates the `.org` and `.ngo` TLDs today. Alternatively, commercial entities such as CSPs may operate their own aDNS hierarchies, with their own root policies and aDNS implementations.

Server features. Our aDNS implementation is focused on serving authoritative zones, therefore it cannot act as a recursive resolver. It is capable of responding to requests over UDP, TCP (RFC7766), and HTTPS (RFC8484). However, our UDP interface always returns a response with the truncation flag (TC) to force the client to re-try over TCP, to avoid having to implement protections against DNS amplification and cache poisoning attacks. We implement the resource record types required by aDNS, including DANE (RFC7671) and CAA (RFC6844). We also introduce a new RR type for attestation (ATTEST) while policies are exposed as text (TXT).

We implement most DNSSEC features, including NSEC3. Expiration of RRSIG records and rollover of ZSKs requires careful time management which is challenging from an enclave application that can only access the (untrusted) host time. Standard DNS records have a relative TTL, but RRSIG records include both the start and end of validity of signatures as absolute timestamps. To mitigate this, we implement a notion of ledger time (inspired by blockchain techniques), so we can at least enforce monotonicity of host time. It is also possible to monitor time in replicas to detect the clock of the primary server drifting too far. We assume a cron job on the host periodically calls aDNS to trigger the re-keying and key rollover endpoint. If the host doesn't, then the signatures will expire and the service may become unavailable, but denial of service attack by the host is always possible anyway.

Our aDNS service has a built-in ACME client, which can be configured either to use a private Pebble CA (for testing) or Let's Encrypt [1]. When an aDNS instance starts, we automatically provision certificates for each nodes, which are by convention labelled `ns0`, ..., `ns`*N* where `ns0` is the primary and *N* the number of replicas. These are used both for DoH and for the endpoints required by the aDNS protocol.

Endpoints. The table below summarizes the JSON RPC endpoints exposed by aDNS over HTTPS:

Endpoint	Description
`/register_service`	Register a TEE to a name in the zone
`/register_delegation`	Delegate a sub-zone to a fresh aDNS instance
`/configure`	Configure a fresh aDNS instance
`/get_certificate`	Get an ACME certificate for a registered name
`/resign`	Refresh RRSIG records and rollover ZSK
`/acme_refresh`	Renew ACME certificates
`/endorsements`	Gets attestation evidence of this aDNS instance

Our implementation supports delegation and registration policies expressed as pure Javascript functions that return a boolean value. The functions have access to claims (key-value pairs) extracted from verified attestation reports.

7.3 Browser Extension

To illustrate how Web browsers might support aDNS, we create a Firefox extension that implements the client protocol defined in §5.3. We chose Firefox because, unlike Chromium-based browsers, it supports the `getSecurityInfo()` API for inspecting the TLS connection (including its certificates) on a request. We use the `WebRequest` API to intercept connections to websites in aDNS zones, captured by the expression `*://*.attested.name/*`. This assumes that all aDNS services fall under the `attested.name` suffix. In the future this may be replaced with a TLD, or there may be multiple subtrees of the DNS hierarchy with aDNS enabled.

We use the `onBeforeRequest` event to send the initial DNS queries, and the `onHeadersReceived` event to validate the TLS connection. We use either the Google or Cloudflare Trusted Recursive Resolver (TRR) to perform the additional DNS queries with DNS over HTTPS. This is available through `browser.dns.resolve()` API (when the `isTRR` flag is set in the response), but we prefer to query them directly with the Fetch API to ensure the Authentic Data (AD) flag is set, indicating successful DNSSEC validation. It is also possible to use the extension without a TRR, but this configuration can only resolve A and AAAA requests. To work around this, we implement a mechanism that encodes ATTEST and TLSA records as sequences of compressed AAAA records, e.g. when queried with the `_i._attest` prefix, where i is the fragment number. Compression is important to ensure the fragments are properly cached. Our evaluation (see §8.1) shows that 5 fragments are sufficient for most reports, so our extension pre-flights 5 fragment queries in parallel.

We compile our Ravl attestation library to WebAssembly so that the attestation can be verified in parallel with the request, which only blocks when it reaches the `onHeadersReceived` event. Empirically, we observe that the verification of attestation reports and collaterals can complete in under 20ms, which is lower than the connection latency for most websites. We compare the TLS certificate public key with the attested public key. If that check fails, we can interrupt the request and send the user to an error page.

For now, our extension discovers the registration policy and allows the user to inspect it using the extension's button, though we could allow the user to configure the policy instead. This raises questions about how users would react to new UI signals about attested websites, which are left to future work.

8 Evaluation

8.1 aDNS Server

Our implementation of aDNS is coded in ∼17K lines of C++, whose breakdown is shown in Table 2.

Figure 4 plots the throughput and latency of DNS queries using independent clients sending a single TCP request. Our

C++ Class	Description	LoC
Ravl	Remote attestation validation library	7117
RFC1035	Base DNS record types	1278
RFC4034	DNSSEC record types	1353
RFC*	NSEC3, ETLS, DANE, CAA	673
Resolver	DNSSEC implementation	2472
CCFDNS	Endpoints (query, register...)	3248
	Serialization and utilities	1146
CCF	Confidential Consortium Framework	∼64K
	CCF Dependencies (JSON, HTTP...)	∼125K
	Total Enclave TCB	∼200K

Table 2: TCB size of aDNS server

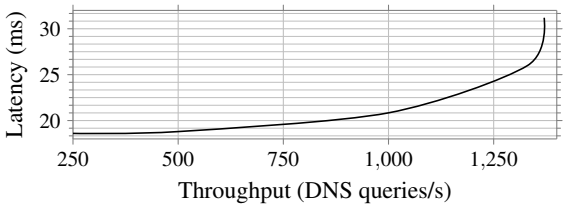

Figure 4: Throughput and latency of aDNS server

server runs on an Azure DCv3-series VM with 4 Xeon 8370C cores and 16GB of enclave-protected memory (EPC). Our aDNS server implementation is not designed to serve clients directly, and thus it is limited to about 1350 queries/s. In practice, we expect aDNS zones to be served by a set of untrusted, optimized front-end nameservers, such as Bind 9, that will cache the signed records from our authoritative server.

Figure 5 shows the scalability of an aDNS zone served by Bind 9.18.30 to different configuration of Dnspyre (a DNS benchmarking utility) running locally and configured to reflect the query profile of regular HTTPS client (querying only A records), DNSSEC clients (also querying DNSKEY, RRSIG and NSEC records), DANE clients (also querying TLSA records), and full-fledged aDNS clients (also querying ATTEST records encoded as AAAA record sets). This comparison shows that a typical nameserver can serve about half as many full aDNS clients as basic HTTPS clients. However, a partial aDNS client that checks DANE (and possibly discovers the service policy) uses a similar amount of resources as a regular DNSSEC client, and with gradual adoption of aDNS, the overall impact on DNS performance and scalability is unlikely to be noticeable by service operators.

Table 3 shows the latency of non-DNS endpoints. Incremental zone signing is not supported, so any record change causes all RRSIG to be re-signed. Due to NSEC3 and our fragmentation mechanism to encode ATTEST records into AAAA, our zones can be surprisingly large (up to 450 records per TEE registration), which explains the signing time. Our implementation currently starts the ACME protocol asynchronously as soon as a TEE is registered, which explains why the `/get-certificate` endpoint is fast, although it takes about 7 seconds for Let's Encrypt to return the certificate.

Finally, we evaluate the size of ATTEST records. Since

Endpoint	Context	Latency
`/register-service`	Register complete Intel SGX report	263ms
`/register-service`	Register SGX, missing collaterals	2.1s
`/register-service`	Register AMD SEV-SNP TEE	218ms
`/get-certificate`	Get Let's Encrypt certificate	72ms (+7.5s)
`/resign`	Re-sign zone with 454 records	180ms

Table 3: Non-DNS aDNS endpoint latency

	HTTPS	DNSSEC	RA-TLS	aDNS	+DoH
DNS Resolution	1ms	2ms	1ms	3ms	47ms
TLS Handshake	40ms	40ms	110ms	38ms	38ms
- Attestation (SGX)	-	-	-	27ms	27ms
- Attestation (SEV)	-	-	-	13ms	13ms
- Attestation (RAS)	-	-	71ms	-	-
Time to first byte	41ms	42ms	111ms	41ms	85ms

Table 4: Client latency of aDNS compared to RA-TLS

they are large and non-standard, they are not cached by most verifiers. Hence, we compress and split them into fragmented AAAA record sets, to fit into 512-byte UDP responses. We find that all reports can fit in 5 compressed fragments.

Attestation report type	Full size	Compressed	Fragments
Intel SGX	5951 B	2355 B	5
AMD SEV-SNP	3125 B	630 B	2

8.2 Client Evaluation

We implement aDNS in a browser extension for Firefox. Our Ravl library compiles to 2.9MB of WebAssembly and 130KB of JavaScript with Emscripten. We depend on pako for decompressing attestation records (46KB), basic CBOR libraries (409 lines) to interact with Ravl, and jQuery for UI elements (86KB). The rest of the code is 321 lines of JavaScript.

Table 4 compares the total connection latency (time to first request byte) of a regular HTTPS client with and without DNSSEC; an implementation of RA-TLS from Intel's Confidential Computing Zoo [23] using the Microsoft Azure Attestation [40] as an Attestation Service (AS); and two configurations of our aDNS client—one using our ISP's resolver and another using Cloudflare's public DNS-over-HTTPS (DoH), a configuration that provides better privacy against network observers and can bypass resolvers that either block or refuse to cache aDNS records. Critically, aDNS clients locally verify attestations in parallel with the handshake, which is why the attestation time does not add to the handshake time. In contrast, RA-TLS blocks the handshake until the certificate validation callback returns, hence the validation time is fully added to the handshake time. The latency of RA-TLS also suffers from the remote call to AS to enforce the service's policy. While this provides better freshness than aDNS (for which policies are enforced at registration time, and refreshed subject to the TTL policy of aDNS), the additional dependency on AS, the privacy loss, and the latency impact make RA-TLS a poor overall choice compared to aDNS.

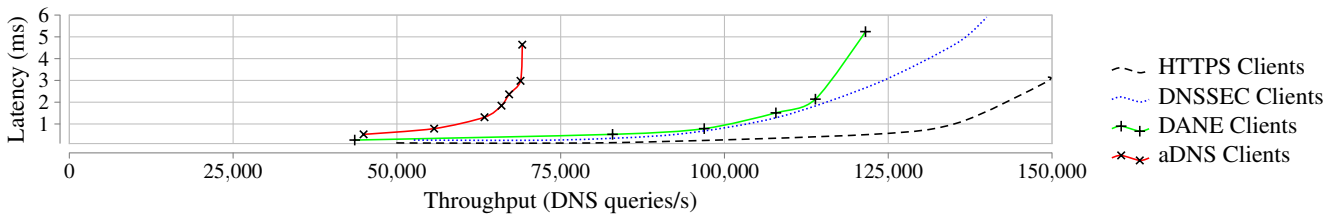

Figure 5: Client Scalability of aDNS Zones Served by Bind 9

8.3 Sample Applications

Inference Service. With the rising popularity of generative AI, concerns around data privacy, especially for the most performant commercial models that can only be queried through APIs, are harming adoption for sensitive scenarios. Confidential computing is the most scalable solution to ensure confidentiality of prompts and models in cloud-based AI systems.

We deploy a Triton [60] container from Nvidia's container registry as a confidential Azure Container Instance (ACI) with 4 AMD EPYC 7763v cores and 16GB of memory. We extend Ravl to support the additional Utility VM collateral of ACI attestation, used to prove that the OS image with the container runtime was built honestly by Azure. We fork Azure's open source attestation sidecar so that it can register the underlying TEE to aDNS at startup. We write an aDNS registration client in 367 lines of Go, which also obtains a Let's Encrypt certificate and runs an Nginx reverse proxy with the obtained certificate to access the application container service. Including all configurations, our sidecar adds 674 lines of code and can be adapted to any containerized application.

Privacy-Preserving Advertising. The Privacy Sandbox [19] is a set of browser technologies proposing to replace 3rd party cookies while providing advertisers with a privacy-preserving approach for targeted advertising and attribution measurement. A key component of the Privacy Sandbox are Protected Audience services [18], which utilize confidential computing to host advertiser-provided bidding and auction logic in an attested sandboxed environment while protecting private interest-group information collected by the browser. These services rely on a key management system (KMS) that releases private hybrid encryption keys only to known good TEEs. Since Azure's implementation of the Privacy Sandbox KMS [41] is a CCF application, we modify the built-in ACME client of CCF to register the nodes and get certificates from aDNS instead. Our patch uses 414 lines of C++.

9 Related Work and Discussion

Attested TLS. The state of the art for transport-level security with confidential computing is to verify attestation either during or after the TLS handshake using remotely attested TLS (RA-TLS). Many RA-TLS variants have been proposed [16, 20, 26, 46] (including a draft standard [61]) and implemented (e.g. in the Intel SGX SDK and libraries [14]). Sardar et al. [55] built a formal model and identified vulnerabilities in Intel's protocol [26]. All variants of RA-TLS share the downside of breaking compatibility with existing clients, either because they change TLS or use self-signed certificates. In contrast, since aDNS operates below the transport and application layers, it is compatible with all clients (their networking stack may be enlightened e.g. by adapting their DNS resolver) and it is more efficient by leveraging distributed DNS caches and allowing concurrent attestation validation and TLS handshake.

Akama et al. [2] propose a way to protect legacy Web clients of a confidential service using certificate transparency (CT) to detect conflicting attested and non-attested service certificates; aDNS goes further by preventing certificate issuance to non-TEE entities by ACME-compatible CAs, though it also relies on CT to detect mis-issuance by other CAs.

Attestation Services. Another related line of work focuses on delegating attestation verification to trusted third parties called attestation services (AS). AS operated by cloud providers (e.g. Azure [40] and GCP [17]) and hardware vendors (Intel, Nvidia, Arm) typically implement the Entity Attestation Token [37] standard, that is, they consume attestation reports and produce JWT tokens summarizing the contents of the report and the additional policies applied by the AS. The client's trust in the AS is a major concern—some AS run in TEEs but this still requires clients to understand their attestation. It may seem logical to depend on a TEE vendor's AS (given TEE security assumes the vendor is honest). However, enforcing service and cloud-specific policies in a global AS is challenging, and depending on them further creates latency, availability, and scalability issues. In comparison, aDNS can be described as a distributed, transparent, attested AS that can provide compatibility with EAT relying parties (as described in Appendix §B) while solving these issues. Auxiliary services from CSPs and hardware vendors may still be used to manage and cache attestation evidence such as platform and firmware certificates, but they need not be trusted with enforcing confidential service policies.

Attested DNS vs Attested CAs. Many confidential applications embed their attestation reports and other collaterals in their (self-signed) certificate, using X.509 extensions. Hence, a natural alternative to aDNS would be to have certification

authorities validate attestation against a universal policy (e.g., service runs on genuine Intel or AMD hardware), and issue certificates with embedded attestation evidence. While such an attested PKIX would sidestep DNSSEC and DANE deployment issues, we think attested DNS is preferable in the long term for several reasons.

1. PKIX is centrally governed (through the CA/Browser Forum) and getting all CAs and browser vendors to agree on global support for attestation seems difficult. Beekman et al. [6] approached CAs to request they sign an attestation extension, and concluded this was practically infeasible.

2. PKIX is not meant to be an online system, even with ACME. Although the situation is improving, certificates live longer than typical DNS records (90 days for Let's Encrypt, often years for other CAs) and have a broader scope (for classes of DNS names/suffixes rather than TEE instances). In comparison, DNS can support TEE-specific attestations of ephemeral keys, much as it already supports dynamic IP addresses.

3. DNS benefits from a distributed cache mechanism to reduce latency and bandwidth compared to evidence embedded in certificates sent on every connection.

4. On the client-side, validating this evidence in callbacks while processing certificate chains involves customizing the transport level, often shared between applications. It may also be hard to delay the use of the connection (including 0RTT traffic, cookies, etc.) until the attested evidence has been checked.

5. Together with certificate transparency and ACME, Let's Encrypt already provides an independent, trustworthy CA with strong practical security, which limits the value of a new attested CA, whereas to our knowledge we are the first to provide DNS transparency.

Even if CAs could be convinced to issue certificates with embedded attestation evidence, the problem of enforcing service-specific policies (e.g. the code of the service is built reproducibly from this open-source repository) would remain. In principle, CAs can be tasked with reviewing, approving and enforcing service-specific policies, but this is impractical.

In contrast, DNS is a decentralized hierarchical system, with controlled delegation of trust to subdomains. Hence we can, for instance, deploy an 'attested zone' without interfering with existing TLDs, and each layer in this zone (service, organization, TLD) can manage and enforce its own attestation policy. While most service guarantees are policy-dependent, some are generic (e.g., service transparency, code auditability) and meaningfully improve the security of all confidential computing users. The policies still map to DNS suffixes that users can recognize and trust by reputation.

10 Conclusion

In the past decade, many TEE-based applications have been proposed for Web search [44, 51], online payments [30, 35], video streaming [11], file sharing [15], password management [27, 34], and many other domains. Protecting data and model confidentiality of AI systems is by far the most popular type of applications [42, 43, 45, 52, 59, 63, 65, 66]. Paju et al. conducted a survey of 223 TEE applications [50], most of which rely on a custom client. However, as more applications appear, this approach cannot scale. aDNS is the first service attestation architecture to provide low connection overhead, compatibility with mainstream clients, and meaningful security properties for all clients in the face of various compromise scenarios.

Ethical Considerations

Our goal is to improve Internet security for service providers and their users, by providing a usable, transparent, and auditable way to distribute and verify the attestations of confidential service implementations. Even a trivial service policy that just checks the service runs in a valid TEE improves security against compromise of the service operator (e.g. public clouds). Confidential computing may be used by malicious service owners to build end-to-end encrypted platforms (e.g. to share or distribute illegal content). Our research does not amplify this risk—on the contrary, aDNS is designed to make service owners accountable for the policy and code they deploy under the service name. Existing DNS-based censorship mechanisms are largely unimpacted by aDNS.

Our design and prototype implementation build on open standards whenever possible. In particular, we comply with DNS,DNSSEC, ACME, DANE, and related standards, and we do not decrease their security for their existing users.

Regarding user privacy, the additional records used by aDNS do not reveal more information than the name of the service, which already appears in the IP address resolution query. Privacy conscious users can use DNS over HTTPS to ensure network observers can't track their service usage, though this only works if the resolver is shared by many users as network traffic to authoritative DNS servers leaks a lot of information about potential services. A confidential DoH resolver can also mitigate trust in the resolver, though this protection should be evaluated against the risks of side channels of their particular TEE platforms. Our research and its experimental evaluation did not involve any private data.

Open Science

The artifacts in this paper have been published on Zenodo and are available at https://doi.org/10.5281/zenodo.15611255 under the MIT license. They include the CCF-based

authoritative aDNS server, the remote attestation verification library, the Firefox client extension, and a command-line aDNS Python client.

References

[1] Josh Aas, Richard Barnes, Benton Case, Zakir Durumeric, Peter Eckersley, Alan Flores-López, J. Alex Halderman, Jacob Hoffman-Andrews, James Kasten, Eric Rescorla, Seth Schoen, and Brad Warren. Let's Encrypt: An automated certificate authority to encrypt the entire web. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, page 2473–2487, 2019.

[2] Kosei Akama, Yoshimichi Nakatsuka, Korry Luke, Masaaki Sato, and Keisuke Uehara. RA-WEBs: Remote attestation for web services, 2024.

[3] Mustafa Al-Bassam and Sarah Meiklejohn. Contour: A practical system for binary transparency. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 94–110, 2018.

[4] AMD. Strengthening VM isolation with integrity protection and more. *White Paper, January*, 53:1450–1465, 2020.

[5] Richard Barnes, Jacob Hoffman-Andrews, Daniel McCarney, and James Kasten. Automatic Certificate Management Environment (ACME). RFC 8555, March 2019.

[6] Jethro Beekman, John Manferdelli, and David Wagner. Attestation transparency: Building secure Internet services for legacy clients. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security (ASIACCS)*, pages 483–494. ACM, 2016.

[7] Yohan Beugin and Patrick McDaniel. Interest-disclosing mechanisms for advertising are privacy-exposing (not preserving). *Proceedings on Privacy Enhancing Technologies Symposium*, pages 41–57, 2024.

[8] CA/Browser Forum. Baseline requirements for TLS server certificates. https://cabforum.org/working-groups/server/baseline-requirements/documents/, 2024.

[9] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. Intel TDX demystified: A top-down approach. *ACM Computing Surveys*, 56(9), 2024.

[10] Graeme Connell, Vivian Fang, Rolfe Schmidt, Emma Dauterman, and Raluca Ada Popa. Secret key recovery in a Global-Scale End-to-End encryption system. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pages 703–719, Santa Clara, CA, July 2024. USENIX Association.

[11] Simon Da Silva, Sonia Ben Mokhtar, Stefan Contiu, Daniel Négru, Laurent Réveillère, and Etienne Rivière. Privatube: Privacy-preserving edge-assisted video streaming. In *Proceedings of the 20th International Middleware Conference*, pages 189–201, 2019.

[12] Antoine Delignat-Lavaud, Cédric Fournet, Kapil Vaswani, Sylvan Clebsch, Maik Riechert, Manuel Costa, and Mark Russinovich. Why should I trust your code? *Communications of the ACM*, 67(1):68–76, 2023.

[13] Gobikrishna Dhanuskodi, Sudeshna Guha, Vidhya Krishnan, Aruna Manjunatha, Michael O'Connor, Rob Nertney, and Phil Rogers. Creating the first confidential GPUs: The team at nvidia brings confidentiality and integrity to user code and data for accelerated computing. *ACM Queue*, 21(4):68–93, 2023.

[14] Zia Zhang et al. RATS architecture based TLS using librats. https://github.com/inclavare-containers/rats-tls, 2024.

[15] Benny Fuhry, Lina Hirschoff, Samuel Koesnadi, and Florian Kerschbaum. SeGShare: Secure group file sharing in the cloud using enclaves. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 476–488. IEEE, 2020.

[16] Kenneth Goldman, Ronald Perez, and Reiner Sailer. Linking remote attestation to secure tunnel endpoints. In *Proceedings of the First ACM Workshop on Scalable Trusted Computing*, STC '06, page 21–24, New York, NY, USA, 2006. Association for Computing Machinery.

[17] Google. Google cloud attestation. https://cloud.google.com/confidential-computing/docs/attestation.

[18] Google. Protected audience API overview. https://developers.google.com/privacy-sandbox/relevance/protected-audience.

[19] Google. What is the privacy sandbox? https://developers.google.com/privacy-sandbox/overview, 2023.

[20] Gilang Mentari Hamidy, Sri Yulianti, Pieter Philippaerts, and Wouter Joosen. Tc4se: A high-performance trusted channel mechanism for secure enclave-based trusted execution environments. In *International Conference on Information Security*, pages 246–264. Springer, 2023.

[21] Paul E. Hoffman and Jakob Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698, August 2012.

[22] Intel. Software guard extensions. https://software.intel.com/en-us/sgx (Accessed on 12/13/2019).

[23] Intel. Confidential computing zoo. https://github.com/intel/confidential-computing-zoo, 2023.

[24] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1447–1464, Santa Clara, CA, August 2019. USENIX Association.

[25] Harry A Kalodner, Miles Carlsten, Paul M Ellenbogen, Joseph Bonneau, and Arvind Narayanan. An empirical study of namecoin and lessons for decentralized namespace design. In *WEIS*, volume 1, pages 1–23, 2015.

[26] 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.

[27] Klaudia Krawiecka, Arseny Kurnikov, Andrew Paverd, Mohammad Mannan, and N Asokan. Safekeeper: Protecting Web passwords using trusted execution environments. In *Proceedings of the 2018 World Wide Web Conference*, pages 349–358, 2018.

[28] Adam Langley. Why not DANE in browsers? https://www.imperialviolet.org/2015/01/17/notdane.html, 2015.

[29] Ben Laurie. Certificate transparency. *Commun. ACM*, pages 40–46, sep 2014.

[30] Peng Li, Xiaofei Luo, Toshiaki Miyazaki, and Song Guo. Privacy-preserving payment channel networks using trusted execution environment. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2020.

[31] Wubin Li, Yves Lemieux, Jing Gao, Zhuofeng Zhao, and Yanbo Han. Service mesh: Challenges, state of the art, and future research opportunities. In *2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 122–1225, 2019.

[32] Xupeng Li, Xuheng Li, Christoffer Dall, Ronghui Gu, Jason Nieh, Yousuf Sait, and Gareth Stockwell. Design and verification of the arm confidential compute architecture. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pages 465–484, Carlsbad, CA, July 2022. USENIX Association.

[33] Wilson Lian, Eric Rescorla, Hovav Shacham, and Stefan Savage. Measuring the practical impact of DNSSEC deployment. In *22nd USENIX Security Symposium (USENIX Security 13)*, pages 573–588, 2013.

[34] Xueping Liang, Sachin Shetty, Lingchen Zhang, Charles Kamhoua, and Kevin Kwiat. Man in the cloud (MITC) defender: SGX-based user credential protection for synchronization applications in cloud computing platform. In *2017 IEEE 10th International Conference on Cloud Computing (CLOUD)*, pages 302–309. IEEE, 2017.

[35] Joshua Lind, Oded Naor, Ittay Eyal, Florian Kelbert, Emin Gün Sirer, and Peter Pietzuch. Teechain: a secure payment network with asynchronous blockchain access. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, pages 63–79, 2019.

[36] Laurence Lundblade, Giridhar Mandyam, Jeremy O'Donoghue, and Carl Wallace. The Entity Attestation Token (EAT). Internet-Draft draft-ietf-rats-eat-25, IETF, January 2024. Work in Progress.

[37] Laurence Lundblade, Giridhar Mandyam, Jeremy O'Donoghue, and Carl Wallace. The Entity Attestation Token (EAT). Internet-Draft draft-ietf-rats-eat-31, IETF, September 2024. Work in Progress.

[38] Lyft. Envoy proxy. https://www.envoyproxy.io/, 2024.

[39] Moxie Marlinspike. Technology preview: Private contact discovery for Signal. https://signal.org/blog/private-contact-discovery/.

[40] Microsoft. Azure attestation. https://azure.microsoft.com/en-us/products/azure-attestation/, 2022.

[41] Microsoft. Azure privacy sandbox KMS. https://github.com/microsoft/azure-privacy-sandbox-kms, 2024.

[42] Fan Mo and Hamed Haddadi. Efficient and private federated learning using TEE. In *Proc. EuroSys Conf., Dresden, Germany*, 2019.

[43] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. Darknetz: towards model privacy at the edge using trusted execution environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*, pages 161–174, 2020.

[44] Sonia Ben Mokhtar, Antoine Boutet, Pascal Felber, Marcelo Pasin, Rafael Pires, and Valerio Schiavoni. X-search: revisiting private web search using intel sgx.

In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*, pages 198–208, 2017.

[45] Arup Mondal, Yash More, Ruthu Hulikal Rooparaghunath, and Debayan Gupta. Flatee: Federated learning across trusted execution environments. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 707–709. IEEE, 2021.

[46] Arto Niemi, Vasile Adrian Bogdan Pop, and Jan-Erik Ekberg. Trusted sockets layer: A TLS 1.3 based trusted channel protocol. In *Nordic Conference on Secure IT Systems*, pages 175–191. Springer, 2021.

[47] Mark Nottingham. Playing fair in the privacy sandbox: Competition, privacy and interoperability standards. *Privacy and Interoperability Standards (February 3, 2021)*, 2021.

[48] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX annual technical conference (USENIX ATC 14)*, pages 305–319, 2014.

[49] Eric Osterweil, Michael Ryan, Dan Massey, and Lixia Zhang. Quantifying the operational status of the DNSSEC deployment. In *Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pages 231–242, 2008.

[50] Arttu Paju, Muhammad Owais Javed, Juha Nurmi, Juha Savimäki, Brian McGillion, and Billy Bob Brumley. SoK: A systematic review of tee usage for developing trusted applications. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*, pages 1–15, 2023.

[51] Rafael Pires, David Goltzsche, Sonia Ben Mokhtar, Sara Bouchenak, Antoine Boutet, Pascal Felber, Rüdiger Kapitza, Marcelo Pasin, and Valerio Schiavoni. CYCLOSA: Decentralizing private web search through SGX-based browser extensions. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 467–477. IEEE, 2018.

[52] Do Le Quoc, Franz Gregor, Sergei Arnautov, Roland Kunkel, Pramod Bhatotia, and Christof Fetzer. Securetf: A secure tensorflow framework. In *Proceedings of the 21st International Middleware Conference*, pages 44–59, 2020.

[53] Mark Russinovich, Edward Ashton, Christine Avanessians, Miguel Castro, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, Cédric Fournet, Matthew Kerner, Sid Krishna, et al. CCF: A framework for building confidential verifiable replicated services. *Microsoft, Redmond, WA, USA, Tech. Rep. MSR-TR-2019-16*, 2019.

[54] N. Sakimura, J. Bradley, M. Jones, and E. Jay. OpenID Connect discovery 1.0, December 2023.

[55] Muhammad Usama Sardar, Arto Niemi, Hannes Tschofenig, and Thomas Fossati. Towards validation of tls 1.3 formal model and vulnerabilities in intel's ra-tls protocol. *IEEE Access*, 12:173670–173685, 2024.

[56] Benjamin M. Schwartz, Mike Bishop, and Erik Nygren. Service Binding and Parameter Specification via the DNS (SVCB and HTTPS Resource Records). RFC 9460, November 2023.

[57] Alex Shamis, Peter Pietzuch, Miguel Castro, Cedric Fournet, Edward Ashton, Amaury Chamayou, Sylvan Clebsch, Antoine Delignat-Lavaud, Matthew Kerner, Julien Maffre, Olga Vrousgou, Christoph M. Wintersteiger, Manuel Costa, and Mark Russinovich. IA-CCF: Individual accountability for permissioned ledgers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 467–491, 2022.

[58] Janos Szurdi, Balazs Kocso, Gabor Cseh, Jonathan Spring, Mark Felegyhazi, and Chris Kanich. The long "taile" of typosquatting domain names. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 191–206, 2014.

[59] Florian Tramer and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. *arXiv preprint arXiv:1806.03287*, 2018.

[60] triton-inference-server/server: The triton inference server provides an optimized cloud and edge inferencing solution. https://github.com/triton-inference-server/server. (Accessed on 11/04/2021).

[61] Hannes Tschofenig, Yaron Sheffer, Paul Howard, Ionuț Mihalcea, Yogesh Deshpande, Arto Niemi, and Thomas Fossati. Using Attestation in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). Internet-Draft draft-fossati-tls-attestation-08, IETF, October 2024.

[62] George Mason University. Global dnssec deployment tracking. https://secspider.net/, 2024.

[63] Kapil Vaswani, Stavros Volos, Cédric Fournet, Antonio Nino Diaz, Ken Gordon, Balaji Vembu, Sam Webster, David Chisnall, Saurabh Kulkarni, Graham Cunningham, et al. Confidential computing within an AI accelerator. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 501–518, 2023.

[64] Paul Wouters, Hannes Tschofenig, John IETF Gilmore, Samuel Weiler, and Tero Kivinen. Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). RFC 7250, June 2014.

[65] Peterson Yuhala, Pascal Felber, Valerio Schiavoni, and Alain Tchana. Plinius: Secure and persistent machine learning model training. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 52–62. IEEE, 2021.

[66] Chengliang Zhang, Junzhe Xia, Baichen Yang, Huancheng Puyang, Wei Wang, Ruichuan Chen, Istemi Ekin Akkus, Paarijaat Aditya, and Feng Yan. Citadel: Protecting data privacy and model confidentiality for collaborative learning. In *ACM Symposium on Cloud Computing*, pages 546–561, 2021.

Appendix

As additional materials, we provide a sample confidential service consisting of multiple TEEs and of the corresponding TEE registration policies for this service (§A); an aDNS extension issuing EAT tokens based on registered verified attestations to relying parties (§B); and an overview of a confidential service for the Privacy Sandbox (§C).

A Sample service: Scalable L7 Load Balancing

To illustrate aDNS-enforced policies for registration and certificate issuance, let us consider a distributed service with two roles assigned to TEEs (see also Figure 1):

1. front-end servers that terminate connections to a shared URL `https://www.s.conf`, deal with client authentication and authorization, and dispatch their tasks to the back-end;

2. back-end servers that accept tasks from the front-end.

Each front-end TEE creates an X.509 key to accept client requests over TLS, and gets its endorsed by the CA, as explained in §5. The TEE also has a DANE key, used for client authentication with the back-end TEEs. Each back-end TEE creates a single DANE key to accept connection requests from front-ends. This limits the need for trusted Web certificates only to client-facing front-end TEEs.

The aDNS instance that controls the service registers both kinds of TEEs, and its policy (sample shown in Table 5) ensures they use pairwise-distinct names of the form `node`*n*`.front-end.s.conf`and `node`*n*`.back-end.s.conf` with code and configuration that match their role.

Each front-end regularly queries aDNS to get the names and DANE keys of every active back-end as part of their TLSA records. Symmetrically, each back-end caches the names and DANE keys of every active front-end. Hence, whenever a front-end opens a TLS connection to a back-end, both parties can present their names and raw keys in their ClientCertificate and ServerCertificate messages.

```
function validate(attestation, config) {
  const role = config["role"];
  const values = {
      "front-end": {
          hostdata: 0xDEADC0DE,
          measurement: 0xFEEDFACE
      },
      "back-end": {
          hostdata: 0xBAADF00D,
          measurement: 0x8BADF00D
      }
  };

  return role in values &&
      attestation["hostdata"] === values[role].
          hostdata &&
      attestation["measurement"] === values[role].
          measurement;
}
```

Table 5: Sample aDNS policy for a distributed service with front-end and back-end TEEs.

As long as they cache live TLSA records for these keys, they are guaranteed that their peers have been transparently attested and authorized for their role. They may simply rely on their recent verification by aDNS at registration time or, for additional assurance, fetch and verify the other party's attestation record. The latter approach may be useful, for instance, to confirm that the versions of the code at the two endpoint are compatible, or to ensure that the attested code of the front-end independently verifies the attestation of the front-end before trusting it with the client's data.

B A Discoverable, Transparent Entity Attestation Service

Entity Attestation Tokens (review) Attestation services enable clients to delegate the verification of attestation evidence to a trusted third party. They take as input attestation reports and product a special type of JSON Web Token called an Entity Attestation Tokens (EAT) [36], which encodes the properties of the TEE into a set of platform-agnostic claims. These claims can be be used as input of declarative authentication and authorization policies, while providing an abstract layer atop the underlying, hardware-specific attestation evidence and their cumbersome collaterals. Although these services are convenient, they grow the TCB and reduce auditability, as clients only authenticate tokens.

OpenID Discovery (review) The OpenID Connect Discovery [54] protocol is used by token issuers to advertise their capabilities to clients at a fixed, well-known URL, thereby enabling clients to dynamically obtain up-to-date information about the issuer's configuration, including the public keys used to sign ID tokens. This can similarly lead to a significant TCB extension, as the compromise of the discovery Web

server (or its TLS keys) can lead to the compromise of its discoverable services.

EAT as an auxiliary aDNS API Our main goal is to issue standard-compliant EAT tokens that mirror the verified, transparent attestation evidence recorded by aDNS for the services in its zone, for the benefit of relying parties that consume EATs instead of directly verifying attestation evidence. Clearly, fetching attestation from aDNS scales better than clients fetching an EAT token from a centralized attestation endpoint, but this is still a useful mechanism to support applications built on EAT. We ensure that the EAT issuance service has a minimal, well-defined TCB; that it strictly manages the TTL of the EATs it issues as part of the registration policies it enforces; and that it automatically manages the keys for issuance and for discovery.

Implementation Our EAT service is directly built into aDNS using about 200 lines of C++. When queried for a token for a service name, it fetches the latest unexpired attestation from its zone configuration, and translates the Ravl properties into EAT claims. To this end, we introduce the following new endpoints:

- `/common/v2.0/.well-known/openid-configuration` returns the configuration of the EAT discovery service, notably the discovery and token issuance URL;
- `/common/discovery/v2.0/keys` returns the JWKS of all public keys to verify recent and future EATs.
- `/common/oauth2/v2.0/token?service={S}` issues a token for the TEE(s) registered at name S.
- `/create-signing-key` samples a fresh keypair for token issuance, starts advertising the jwk for the public key, and keeps the private key within the CCF TEEs. This enables the (authenticated) service operator to trigger rotations without exposing private keys.

To prevent any confusion with core aDNS functionality, the EAT endpoint of aDNS can be served from a separate interface, using an independent certificate from the nameserver's primary name. Getting an EAT token for the inference service takes under 78ms (locally) and 160ms (remotely, with a RTT of about 15ms), which is much faster than any public attestation service we tested.

C Privacy Sandbox

Mozilla, Apple and Google have publicly committed to removing support for third party cookies in Firefox, Safari and Chrome by the end of 2024. This change is disruptive to the online advertising industry, which relies on third-party cookies to track users across websites in order to serve targeted ads (a feature that has been widely abused for many unauthorized purposes). In response, Google has introduced a new set of features in Chrome called the Privacy Sandbox designed to enable targeted advertising while limiting the ability for advertisers to track users. A key component of the Privacy Sandbox are Protected Audience services, which utilize confidential computing to run an auction between advertisers for each ad space based on the user's interest groups maintained by the browser. Interest groups are registered when visiting websites; to display an ad, the browser encrypts the list of interest groups under a public key whose associated private key can only be released to an attested TEE running the approved code. Advertisers participate in the auction by providing bidding scripts that can locally output a bid based on the interest groups, and some generic context (e.g. the user's location or language). The bidding and auction sandbox is meant to prevent any of this information from leaking to the advertiser. While Privacy Sandbox has been criticized both for its ineffectiveness at protecting privacy [7] as well as for the conflict of interest with Google's advertising business [47], we consider it a critical application irrespective of its technical merits because of its potential to become the most widely deployed confidential computing Web service.

In Google's current implementation, the key used to encrypt interest groups is generated in a TEE, before being split into two shares managed by independent organizations (called the *coordinators*). The threat model is that at least one of the coordinators is trusted, and if the coordinators decide to collude (e.g., by giving their share to each other, or by accepting a key not created by the key generation TEE), both of them can decrypt all user data without any risk of being blamed. We observe that this risk can be significantly reduced if instead of fetching the interest group encryption key from an untrusted HTTPS endpoint at the primary coordinator, this key was instead registered by the key generation TEE on aDNS and distributed through DANE records to browsers. Such an approach has the following benefits:

- the attested DNS service verifies and stores the attestation of the key generation TEE, so users can audit the generation of the key;
- the aDNS ledger makes the key transparent and it is no longer possible for the primary coordinator to do a targeted attack against a class of users;
- it separates the generation and distribution of the key, so that different browser vendors (e.g. Firefox, Edge) can distribute the key for Google's bidding and auction services while enforcing their own policies;
- it reuses the distributed caching infrastructure of DNS.