T^{2}DNS: A Third-Party DNS Service with Privacy Preservation and Trustworthiness

Paper # 342, 13 pages

***Abstract*—We design a third-party DNS service named T^{2}DNS. T^{2}DNS serves client DNS queries with the following features: protecting clients from channel and server attackers, providing trustworthiness proof to clients, being compatible with the existing Internet infrastructure, and introducing bounded overhead. T^{2}DNS's privacy preservation is achieved by a hybrid protocol of encryption and obfuscation, and its service proxy is implemented on Intel SGX. We overcome the challenges of scaling the initialization process, bounding the obfuscation overhead, and tuning practical system parameters. We prototype T^{2}DNS, and experiment results show that T^{2}DNS is fully functional, able to serve a typical enterprise network, and scalable to the number of clients.**

***Index Terms*—Third-Party DNS, Privacy Preservation, Trustworthiness, Intel SGX, Attestation**

I. Introduction

Domain Name System (DNS) is a basic public service[1] that resolves an Internet client's query of a name (e.g., a URL or an ENUM[2], [3]) to an IP address. In recent years, DNS is more and more served in a flat way. Third-party DNS service providers (SP, e.g., Google public DNS[4], OpenDNS[5], or even local IPS DNS service) are providing non-authoritative services to their clients[1] for various reasons such as parental control and customer attraction. However, *whether these non-authoritative third-party DNS services could faithfully keep the clients' privacy becomes a potential concern for the client.*

Eliminating such concerns from clients requires the third-party service to be designed with various security and practicality requirements. In details, (R1) the SP of DNS service should be able to defend clients from privacy-violating attacks on the communication channel; (R2) the service should be able to defend clients from malicious, compromised or semi-honest SP operators; (R3) the service needs to convince the clients that it is trustworthy (i.e., faithfully execute DNS logic without "peeking" into clients' data); (R4) the service should be compatible with the existing Internet architecture (especially the DNS protocol); and finally (R5) the service had better introduce acceptable overhead to the existing Internet DNS system.

Few existing solutions can satisfy all these five requirements together. Targeting privacy preservation, a set of solutions apply secure communication protocols[6], [7] to encrypt DNS queries into ciphertext. These solutions are hard to be compatible with existing Internet DNS infrastructure and protocol. Solutions like TOR[7] use an anonymous proxy to relay DNS messages, but their cost is expensive (e.g., building an anonymous network). Another set of solutions apply obfuscation protocols[8], [9], [10] which hide a client's true query into several dummy ones. These solutions are easy to be compatible with the existing Internet DNS infrastructure, but would cost extra traffic volume and processing workload for the Internet DNS server. Targeting trustworthiness, trusted execution environment (TEE) can protect a runtime software from untrusted OS operators and also show the client the attestation of this trustworthiness. But this TEE approach still needs to be integrated with the privacy-preserving approaches, and faces the challenge to serve the attestation from a large number of clients.

We propose a third-party DNS service named T^{2}DNS, which is a *hybrid* solution that satisfies the five requirements above. T^{2}DNS constitutes an agent on each client and a proxy between the clients and the Internet DNS service. The channel between the agent and the proxy uses a secure communication protocol, and that between the proxy and the Internet DNS servers uses obfuscation protocols. Thus, a client query would be encrypted and sent to the proxy, and the proxy obfuscates the client queries with several dummy ones and relays the obfuscated set to the Internet DNS server; the responses from the Internet DNS server are resolved and the corresponding response to each client is sent back via the secure communication channel. This hybrid protocol can keep clients' privacy from channel attackers (R1) and be compatible with the Internet DNS infrastructure (R4). T^{2}DNS implements the proxy in a TEE (Intel SGX in this paper), and thus can protect clients from SP server attackers (R2) and providing trustworthy proof to the client (R3). In the design of T^{2}DNS, we overcome the following challenges.

First, the traditional process for an SP to provide a trustworthy proof to a client (e.g., remote attestation in Intel SGX) does not scale for a public service like DNS, because a public service may have many clients (many parties), and having each client to attest the SP would cause too much overhead for the SP proxy. In T^{2}DNS, we decouple the process of attesting the proxy (e.g., platform genuineness and proxy software integrity) from showing the proof to each client. We use a *ticket* to record the attestation result and reuse the result for each client, saving out the cost of each client attesting by themselves.

Second, obfuscating client queries with dummy ones should not overload the Internet DNS servers (R5, both traffic volumes and the computation). We customize the obfuscation algorithm — instead of obfuscating each individual query, we batch queries and obfuscate them together. And the ratio of

[1]Presumably, they should interact with authoritative DNS servers and serve their own clients.

client queries to dummy queries is tuned to avoid flooding the Internet DNS servers.

Third, the system parameters (e.g., obfuscation degree, batch size) need to be tuned to defend various attacks. We formally model the system behavior and consider various possible attacks; defending clients from each attack is described as the constraints to the system parameters in the model. And finally, we find feasible parameters that can satisfy all constraints.

We prototype T^{2}DNS using Crypto++ and SGX library, each of which has the acceleration from AES-NI. Our evaluation shows that T^{2}DNS is compatible with the existing Intel SGX framework and the Internet DNS service. While the SGX implementation introduces some processing overhead, a small Intel NUC is sufficient to provide the T^{2}DNS service for a typical enterprise network (500 QPS in T^{2}DNS v.s. 10 QPS in the network), and T^{2}DNS can be easily scaled by deploying many instances. In this paper, we make the following contributions:

(1) We design and formalize a third-party DNS service that provides privacy preservation and trustworthiness. We propose the method of tuning the system parameters to defend from various attacks.
(2) We scale the SP attestation protocol by decoupling the process of verifying the SP proxy with that of showing the proof to the client.
(3) The prototype and evaluation of T^{2}DNS show that T^{2}DNS is feasible to be deployed on the Internet and serve typical enterprise networks.

II. Background

Third-party DNS services become a new trend for Internet clients. They need to satisfy various security requirements to resolve clients' concerns and practicality requirements to be deployable on the Internet. However, few of existing solutions can satisfy all these requirements.

A. Problem Statement

DNS is an Internet service to the Internet users, which translates names (e.g., URL, ENUM[2], [3]) to IP addresses, and then the Internet users can find the named service by the IP address. DNS servers (including root servers, top-level servers, domain authoritative servers, and local servers) are deployed on the Internet to provide this service, and a client can query these Internet DNS servers iteratively or recursively to resolve a name to an IP address.

Recently, there is a trend that the DNS service is provided by non-authoritative third-party service providers (SP). Google Public DNS[4], OpenDNS[5], and even campus network DNS servers are examples of such non-authoritative service providers. These third-party services are preferred by the DNS clients by providing features such as access control (e.g., parental control) or performance acceleration (e.g., caching and providing DNS resolution results, geographically distributed deployment to reduce latency).

However, without security enhancement, a third-party service would potentially be questioned by their clients, which would consequently affect the adoption of the service. A client may be concerned whether the SP can protect him from external malicious attackers on the communication channel and from internal untrusted (either compromised or semi-honest) service operators. The SP not only needs to protect the client but also shows a proof to the client that he is protected. Thus, our goal in this work is to build *a third-party DNS service with privacy preservation and trustworthiness.*

B. Requirements

The security goal to protect a client's privacy is defined as making the attacker difficult to get the queried content of the client (i.e., associating a client IP with the queried names[11]). The target SP's solution should protect clients' privacy (requirement 1 and 2), make itself trustworthy (requirement 3), and be compatible with the existing Internet infrastructure (requirement 4 and 5).

R1 (Channel privacy): Protecting clients from channel attackers. The channel lies between the client and the service provider (SP) and between the SP and the Internet DNS servers. The solution should protect the channel from typical passive attacks (e.g., eavesdropping) and active attacks (e.g., modifying or dropping queries). **(Threat model:)** we assume that the attacker can and only can access (passively and actively) all the messages exchanged between the client, the SP, and the Internet DNS server. And the attacker's goal is to obtain the name queried by a client, that is, the association between client and query. We do not consider Denial of Service (DoS, e.g., dropping all queries or responses to make the service unavailable).

R2 (Server privacy): Protecting clients from SP operators. A third-party service is a place where all queries and responses are aggregated and distributed, which is hosted on a physical server. Thus, the SP operator could also possibly violate clients' privacy. For example, a semi-honest (i.e., being honest but curious) operator or a compromised server can have the access to the service data (i.e., in runtime memory), and further get the content of clients' queries. **(Threat model:)** we assume that the third-party service is an executing application on a physical machine, and the attacker has the privilege to access the physical machine's OS as well as the application's virtual memory, and the attacker's goal is to steal the data in the service application's memory. We do not assume that the attacker could physically access the hardware.

R3 (Trustworthiness): Showing proof to clients that they are protected. While SP can design various mechanisms to defend the two kinds of attacks above, another task is to make the service itself be trustworthy, i.e., making clients trust that they are protected, so that the clients would be willing to use the service.

R4 (Compatibility): The solution's protocol should interact with the client and the existing DNS infrastructure correctly. Since the current DNS system is already deployed on the Internet and it is impossible to rebuild and deploy a ground-up system, the solution should be compatible with the state-of-the-art Internet DNS infrastructure, including the client side and the server side. That is, the third-party service

should adapt to the existing deployed infrastructure without assuming to replace the existing Internet DNS servers.

R5 (Lightweight): The solution should cause acceptable computation and communication overhead to the client and the existing Internet infrastructure. The solution needs to interact with the client and the Internet DNS servers. If too much communication (and consequent computation) overhead is introduced, the interaction would either cost too much resource or be regarded as malicious attack and banned.

C. State of The Art

Referring to existing solutions, few of them can satisfy all the five requirements. We list several existing works and analyze the scope that they can be applied (Table I). Privacy-preserving protocols use either encryption or obfuscation protocols to protect client privacy on the channel, but they are hard to be both compatible and lightweight; a trusted execution environment could protect client privacy on servers and provide trustworthiness, but needs to be integrated with other mechanisms.

1) Privacy-Preserving Protocols: **Encryption Protocols.** A set of works allow clients to use encryption protocols (e.g., TLS) to connect with the Internet DNS server directly without the third-party service, which is out of the scope of this paper (i.e., providing extra third-party service). In the scenario of a third-party service provider delegating the original Internet DNS service, a proxy stands between clients and the server, the clients communicate with the proxy using encryption protocols, and the proxy relays the queries and the responses between the clients and the server. In this scenario, a metric named *unlinkability* is defined to quantify the security — for a proxy P and an adversary A, the probability that A outputs a correct association of a client and its query is defined as the unlinkability $adv[A, P]$. For example, there are d users issuing d queries, and a random guessing adversary R would have an advantage of $adv(R, P) = 1/d$.

Several existing protocols fall in this category (e.g., using TOR, TLS), but they need to be improved or complemented for all the five requirements above. For example, TorDNS[7] and PDoT[6] need server-side deployment, which is hard to achieve on the existing Internet; [7] relies on TOR system, which requires a large anonymous overlay network and introduces extra end-to-end latency (from the TOR nodes).

Obfuscation Protocols. In this kind of protocols, clients' queries can be mixed with many dummy queries so that an adversary cannot distinguish the true and dummy queries (e.g., X-Search[9]). A metric named *indistinguishability* is defined to quantify the security — the metric is similarly denoted as $adv[A, O]$ (for an obfuscation mechanism O and an adversary A to win the association attack game). If there are d true queries from one client and n dummy ones, a random guessing adversary can output a correct association with probability $adv[A, O] = d/(n + d)$.

Obfuscation mechanisms are easy to be compatible with the existing system, but would introduce extra workload (both communication and computation). In DNS systems, if each client's query is obfuscated with many dummy queries, the Internet DNS server would be overloaded (possibly trigger DoS defense), thus, it is not wise to directly obfuscate each client query. [12] proposes a semantic classification algorithm for generating dummy queries, but the computation cost is not acceptable. X-Search[9] is a recent work (a hybrid of TEE below and privacy preservation) that applies obfuscation protocol; if applied directly for DNS, its obfuscation protocol would overload the Internet DNS server.

2) Trusted Execution Environment: Trusted Execution Environment (TEE) is an approach to provide SP privacy and trustworthiness, and examples of such platforms are Intel SGX[13] and ARM TrustZone[14]. We use Intel SGX as an example of TEE, which protects the application by two meanings. (1) In the *runtime*, Intel SGX hardware can stop privileged software (e.g., OS) to access an SGX-protected application memory. Thus, even if the host machine of the application is compromised (by malicious attackers or semi-honest operators), the adversaries still have no access to the applications' data. (2) In the *initialization*, the SGX hardware and library make a procedure called *remote attestation* to check the genuineness of the platform (an SGX CPU from Intel) and the loaded application program (not a forged software).

Several solutions port the end-point software to TEE (e.g., X-Search[9], PDoT[6]) and inherit existing remote attestation for the same reason (privacy and trustworthiness). While for a public service like DNS, it is not wise to apply the original remote attestation directly, and *the remote attestation must be customized*. In the original SGX remote attestation[15], there are three parties — a client, a verifier, and Intel Attestation Service (IAS). In the scenario of the third-party DNS service, there are more parties — many clients (many parties), the service provider, the TTP service provider (who delegates the original service), and IAS. The remote attestation protocol must take the security and the scalability of each party into consideration. We design such a protocol in §IV.

III. Overview and Challenges

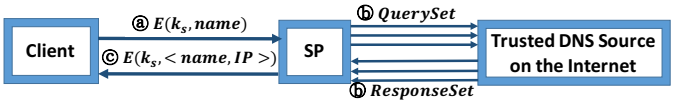


Fig. 1. Workflow

We propose to take a *hybrid* approach for all the five requirements, and design a system named T^{2}DNS whose workflow is shown in Figure 1. T^{2}DNS consists of an agent on the DNS client and a proxy on the SP, both of which are developed by a party recognized by the clients and the SP (e.g., open-source community). The agent can be implemented as a browser extension or a system module, which handles the client's DNS request; the proxy is implemented on Intel SGX which provides the service to the clients in public (R2 and R3).

TABLE I
APPROACHES TO THE DNS PRIVACY PROBLEM

	Obfuscation (§II-C)	Encryption (§II-C)	TEE (§II-C)	Batch Obfuscation (§V)	T^{2}DNS (§IV, §V, §VI)
Channel Privacy	✓	✓	✗	✓	✓
Server Privacy	✗	✗	✓	✗	✓
Trustworthiness	✗	✗	✓	✗	✓
Compatibility	✓	✗	✗	✓	✓
Lightweight	✗	✓	✓	✓	✓

The proxy of the SP stands in the middle between clients and the Internet DNS server, making the whole channel to be two segments. In the segment between the SP and the DNS server, obfuscation protocols should be applied instead of encryption protocols, because it is difficult for the SP to push the Internet DNS server to upgrade DNS protocol to use encryption protocols (R1 and R4). In the segment between clients and the SP, encryption protocols should be applied instead of obfuscation protocols, since an obfuscation algorithm on the client side would amplify and overload the DNS SP and the Internet DNS server (R1 and R5).

Each client first initializes a connection to the proxy, where a secure channel is established (i.e., key exchange) and the authenticity of the SP proxy is verified based on SGX remote attestation. Then the agent intercepts and reissues the client's DNS query and resolves the DNS responses. The proxy accepts DNS queries from clients, recursively or iteratively queries a trustworthy DNS service source[2], and forwards the DNS responses to the DNS clients. Targeting the specific requirements of the DNS system, the design of T^{2}DNS overcomes the following challenges.

C1: Scaling the remote attestation process. When each client initializes, it needs to attest the genuineness of the T^{2}DNS proxy on the SP server. While the original remote attestation process[15] assumes that ONE verifier to attest one service, which is not a reasonable choice for a public service with many clients. With many clients issuing the remote attestation request, the T^{2}DNS proxy and the attestation infrastructure (i.e., Intel Attestation Server) would be overloaded. In T^{2}DNS, we customize the attestation process (§IV) — an authenticated verifier attests the genuineness of the T^{2}DNS proxy and issues a ticket as the proof, then each client just needs to request for the ticket and uses verifier's certificate to further verify the T^{2}DNS proxy's genuineness. This new initialization protocol is lightweight to support a large number of clients.

C2: Bounding the overhead of the obfuscation mechanism. While the proxy has to deploy the obfuscation mechanism to be compatible with the Internet DNS servers, naively obfuscating each true query with many dummy ones would overload the Internet DNS servers. To bound the overhead introduced by the T^{2}DNS proxy, we propose a *batch obfuscation* algorithm (§V). The batch obfuscation is a hybrid approach — it first mixes true queries and then adds a bounded number of dummy queries, avoiding obfuscating each query and flooding the Internet DNS server.

C3: Deciding the system parameters. Deploying T^{2}DNS needs to tune several parameters, and the choice of these parameters should make a tradeoff between security and performance. These parameters include batch size, obfuscation degree, and obfuscation candidate set size. To solve this problem, we formalize the process of the obfuscation mechanism, several possible attacks, and system capacity, each of which introduces formal constraints on the system parameters. By finding a feasible solution with the constraints, we find the best system parameters. (§VII)

IV. INTIALIZATION

We elaborate the problem in applying the original attestation process for public DNS service, and give new attestation protocol in T^{2}DNS.

A. Problem in Applying SGX Remote Attestation

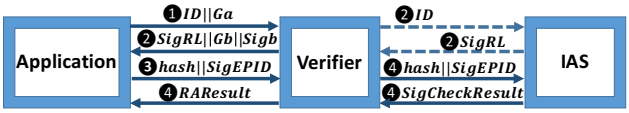


Fig. 2. Three parties in Intel SGX remote attestation

The original remote attestation assumes three parties: an application, a verifier, and the Intel Attestation Service (IAS) (Figure 2)[3]. In the original attestation, there are three goals — the verifier checks whether the application and its platform are genuine, and the verifier and the application build a secure communication channel. And the whole process mainly has the following steps. After the initialization, the verifier continues to send data and delegate the computation to the application (supposed on another remote platform).

(1) Application sends the verifier an EPID GID `ID` [4], which denotes the platform (i.e., CPU unique ID) that runs the application. The application then sends its public key `Ga` (from Diffie-Hellman Key Exchange protocol, DHKE) to the verifier.

(2) The verifier queries to the IAS to retrieve the Signature Revocation List (`SigRL`, which can be offline) for the

[2]They can be either Internet DNSes (root, top-level, and authoritative DNS) or local network DNS (e.g., an ISP or a campus network).

[3]This process is in [15], we change the terminology (e.g., SP to verifier) to avoid the confusion with the terms in this paper.

[4]The extended group ID (GID) of the Intel® Enhanced Privacy ID[16]

ID and sends the application a message with its public key Gb, SigRL, and its signature Sigb.

(3) The application checks the validity of the the platform (SGX enhanced without being revoked using SigRL and ID) and the public key (using Gb and Sigb). If both are valid, the application generates a quote and sends it to the verifier. The quote has a hash of the binary executable loaded to the protected SGX memory (also called enclave) and a signature SigEPID by the private key of the EPID (i.e., an asymmetric key pair sealed in the CPU).

(4) The verifier uses hash and SigEPID to query the Intel Attestation Server (IAS) to confirm the signature is valid (i.e., the application CPU has the secret key), and check the hash with a pre-computed value. If both verifications succeed, the verifier replies to the application to confirm its trustworthiness. In the meanwhile, both the verifier and the application compute the session key Ks using Ga and Gb.

However, in the public DNS service scenario, each individual client needs to verify the genuineness of the service provider (SP). If we let the client play the role of the verifier as above, there would be too much interaction between clients and the SP proxy and IAS, which does not scale. Essentially, there are many clients and they are actually different parities; allowing each client to conduct remote attestation individually is not a reasonable way.

In addition, once the SP proxy is upgraded, each client needs to download the new hash value and redo the remote attestation (comparing the two hash values). It is not easy to have all clients consistently updated.

B. Initialization in T^{2}DNS

In the original remote attestation, the verifier plays two roles — the application identity verifier (who verifies the application's genuineness) and the application user (who sends data and delegates computation to the application after the attestation). While in T^{2}DNS's initialization, we *decouple* the two roles. The service provider in T^{2}DNS plays the role of application above, and a separate verifier is set up for attestation, but each client is the application user and the client gets the attestation result from the verifier. The whole initialization is the following phases.

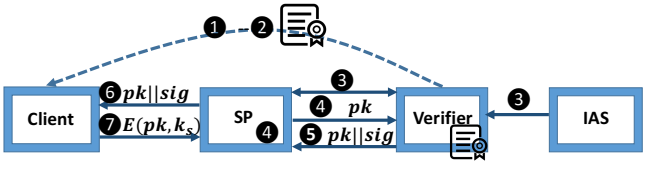


Fig. 3. Initialization and Key Distribution

Phase 0: Offline. Clients set up the agent and certificates. And T^{2}DNS verifier is set up on the Internet.

(0) The T^{2}DNS authority[5] sets up a verifier (i.e., the *trust anchor*) on the Internet to serve remote attestation for all runtime T^{2}DNS services.

(1) Each client downloads and installs the T^{2}DNS agent.

(2) Each client downloads the certificate of the verifier CertVeri.

Phase 1: SP Initialization. The service provider starts the T^{2}DNS proxy and performs the remote attestation.

(3) The SP proxy starts and performs the remote attestation with the verifier (the same as the original remote attestation introduced in §IV-A). If the attestation is successful, SP proxy continues to apply for a *service provision ticket* to the verifier.

(4) The SP application generates a public-secret key pair <PK_SP, SK_SP>. It keeps the SK_SP within the enclave and sends PK_SP to the verifier.

(5) The verifier signs the PK_SP with its own secret key, and sends a ticket PK_SP ‖ SIGVeri to the SP.

Phase 2: Client Initialization. The client starts, fetches the remote attestation results from the SP (and verifies the result), and builds a secure communication channel to the SP.[6]

(6) A client starts and fetches the ticket from the SP. It verifies the validity of the ticket to confirm the genuineness of the SP (using PK_SP ‖ SIGVeri and CertVeri).

(7) The client chooses a symmetric session key ks, encrypts it by PK_SP, and sends it to the SP to build the secure communication channel.

In this whole initialization process, the attestation between the SP, verifier, and IAS is performed only once, and each client reuses and verifies the attestation result (i.e., a valid PK_SP ‖ SIGVeri) to confirm the genuineness of the SP. Thus, an increasing number of clients would not cause more workload to the Intel IAS and the T^{2}DNS proxy's attestation. Until now, the client is ready to send DNS queries and the SP is ready to resolve and reply to the queries. In practice, the task to set up the verifier on the Internet could be undertaken by the T^{2}DNS authority (e.g., the organizer or the developer in the opensource community).

V. Handling Runtime Queries

A. Batch Obfuscation

The proxy needs to preserve the privacy of the clients. Applying any of encryption or obfuscation alone cannot guarantee both compatibility and lightweight, and T^{2}DNS proposes a hybrid approach (see in Table I). In T^{2}DNS, the proxy would batch client queries; when the queries accumulate to a batch size or a timer expiration triggers the deadline to process batched queries, the queries would be first mixed (like encryption protocols) and then obfuscated with dummy queries (like obfuscation protocols).

[5]It can be the opensource community.

[6]Only when the SP upgrades or the client uses the service for the first time, a client needs to do the initialization.

TABLE II
NOTATIONS AND TERMINOLOGIES

Symbol	Type	Meaning
T	const	The privacy threshold
M	const	No. of entries in database (to sample dummy queries)
N_c	const	Max No. of client queries in a batch
t	const	Time to batch client queries
n_c	var	Actual No. of client queries in a batch
n_d	var	No. of dummy queries (varying with n_c and T)
R	const	The max rate that the Internet DNS can accept queries

The whole process is formalized as follows. The proxy batches client queries. When the batch size reaches N_c or query process deadline (time t) elapses, the proxy would fetch all batched queries ($n_c \in [1, N_c]$), sample n_d queries from a candidate set (with M pre-stored dummy queries), and obfuscate the two sets and send them to the trusted Internet DNS server.

Meanwhile, the proxy records the association between the query source (i.e., the client IP) and the query (i.e., the queried name). The proxy would receive the DNS response results from the Internet DNS server and relay them to the corresponding client.

B. Privacy Analysis

We use $adv[A, E]$ to denote the *probability that an adversary A could infer a correct association of a client and its query under the privacy-preserving protocol E*. We consider the following possible attacks to T^{2}DNS and assume a privacy threshold T that the clients/SP would like to guarantee (i.e., $adv[A, E] < T$). In all the following attacks, we assume the adversary could observe the set of clients that issue queries and the obfuscated queries/responses between the proxy and the Internet server.

Random guess attack. A random guessing adversary A_r observes $n_c + n_d$ queries, and the probability for him to associate a query with a client correctly is $adv[A_r, P] = \frac{1}{n_c+n_d}$[7]. To guarantee privacy, we have

$$\frac{1}{n_c + n_d} < T, 1 \le n_c \le N_c. \tag{1}$$

Active attack. DNS is a public service, therefore an adversary can send several queries to take the slot among the N_c client queries. If an honest client sends a query, the adversary can exclude its own queries from the obfuscated set (narrowing down the range of the set that contains the client's query). In the worst case, the adversary excludes $N_c - 1$ queries from the obfuscated set and guesses the client's query in the remaining ones. To guarantee privacy, we have

$$\frac{1}{1 + n_d} < T. \tag{2}$$

Posterior-observation attack. The obfuscation set is generated by randomly sampling a database with M entries. If the database size is too small, the sampled queries could have a collision with the client queries. For an adversary A_p that observes the $n_c + n_d$ obfuscated set and outputs the collided query as the client query, the advantage $adv[A_p, P] = Pr[\text{sampled queries collide with the client queries}]$. Thus, we have

$$\begin{aligned} &Pr[n_d \text{ sampled queries collide with } n_c \text{ client queries}] \\ =&1 - Pr[n_d \text{ sampled queries are not from } n_c \text{ client queries}] \\ =&1 - \left(1 - \frac{n_c}{M}\right)^{n_d} < T, 1 \le n_c \le N_c. \end{aligned}$$

This is equivalent to

$$1 - \left(1 - \frac{n_c}{M}\right)^{n_d} \approx \frac{n_c \times n_d}{M} < T, 1 \le n_c \le N_c. \tag{3}$$

DoS Attack. For the reason of reliability, a DNS client could retransmit dropped queries, which can be abused by an adversary. If an adversary drops the whole obfuscated set S_1, forcing the clients to retransmit the queries and proxy to generate a new obfuscated set S_2, by computing $S_1 \cap S_2$ the adversary can infer the clients' queries. There is no effective algorithm to defend this kind of attack, and we categorize this kind of attack as a DoS attack. Once a DoS attack is detected, the T^{2}DNS proxy stops the service.

C. Deployment Constraints

The obfuscation mechanism would generate extra queries to the Internet DNS server. To avoid flooding the Internet DNS server, the number of extra queries should be bounded. Assume the clients' query rate is r and the Internet DNS server's processing capacity is R. If queries are fully batched, they should satisfy

$$\frac{N_c}{r} \times R > n_d. \tag{4}$$

And if queries are not fully batched, the partial query set should be

$$\frac{n_c}{t} \le R, 1 \le n_c \le N_c. \tag{5}$$

VI. T^{2}DNS PROXY ARCHITECTURE

A. Architecture

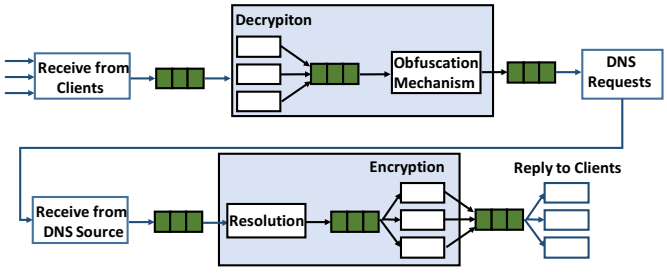


Fig. 4. T^{2}DNS Architecture

Workflow. Figure 4 shows the architecture of the T^{2}DNS proxy, and Algorithm 1 is the pseudocode of each step. Client queries are processed in the following steps. `ReceiveFromClient()` receives client queries and puts them in a queue (line 4). `Decryption()` would decrypt each

[7] P is the T^{2}DNS proxy.

Algorithm 1 Algorithms in T^{2}DNS Proxy

Input:
1: N: the number of dummy queries
2: **function** MAIN
3: Intialize $NameCache$
4: Receive queries from clients to $CipherInQuery$
5: DECRYPTION
6: $NameSet \leftarrow$ OBFUSCATION
7: Query Trustworthy DNS with $NameSet$ to $NameIP$
8: RESOLVE($NameIP$)
9: $Response \leftarrow$ ENCRYPTION
10: Send $Response. < name, IP >$ to $Response.clientIP$
11: **end function**
12: **function** DECRYPTION
13: **for** each Q in $CipherInQuery$ **do**
14: $key \leftarrow$ ClientKey[$Q.clientIP$]
15: $PlainName \leftarrow$ DECRYPT($key, Q.cipherName$)
16: $NameInQuery$.append($PlainName$)
17: **end for**
18: **end function**
19: **function** OBFUSCATION
20: $NameSet \leftarrow$ RANDOMSAMPLE($N, NameCache$)
21: **for** each Q in $NameInQuery$ **do**
22: $NameSet$.RandomInsert($Q.name$)
23: **end for**
24: **return** $NameSet$
25: **end function**
26: **function** RESOLUTION($NameIP$)
27: **for** each Q in $NameInQuery$ **do**
28: **if** $Q.name == NameIP[i].name$ **then**
29: $Q.IP \leftarrow NameIP[i].IP$
30: **end if**
31: **end for**
32: **end function**
33: **function** ENCRYPTION
34: **for** each Q in $NameInQuery$ **do**
35: $key \leftarrow$ ClientKey[$Q.clientIP$]
36: $CipherRes \leftarrow$ ENCRYPT($key, < Q.name, Q.IP >$)
37: $Response$.append($< CipherRes, Q.clientIP >$)
38: **end for**
39: **return** $Response$
40: **end function**

query and store the association of `<clientIP, name>` (line 5, line 12-18). `Obfuscation()` mixes the client queries with randomly sampled queries (line 6, line 19-25). `QueryTrustedDNS()` would send the obfuscated query set and receive the `responses` (line 7). `Resolution()` would look up the `responses` to find the IP (`NameIP`) for the `name` of each `<clientIP, name>` association. `Encryption()` would encrypt the `NameIP`. Finally, `ReplyToClient()` sends the encrypted `NameIP` to corresponding `clientIP`.

Parallelization. Among all the modules, `Encryption()` and `Decryption()` process multiple independent client queries, and they can be implemented in a multithreading program structure. Parallelizing these two modules can reduce the encryption/decryption computation time on queries, but would also incur SGX enclave transition overhead (one transition for each thread). We *optionally* implement the parallel encryption and decryption, and conduct experiments to evaluate whether the multithreading acceleration can cover the extra overhead.

In the parallel implementation, encryption/decryption is implemented as two thread pools respectively, as well as each thread consumes queries in its predecessor and produces to its successor, and the buffer/queue between threads has concurrency control using conditional variables.

B. SGX Enhancement

To preserve client privacy in the proxy, the proxy software is enhanced by SGX. In SGX, a piece of software needs to specify the variables and functions in the protected memory zone named enclave, and in the runtime, the enclave can deny unauthorized accesses. When deciding which part of the proxy is put into enclaves, we follow the principles below.

- SGX enclaves can only protect user space memory, and cannot protect system calls to the OS kernel. The functions `ReceiveFromClient()`, `ReplyToClient()`, and `QueryTrustedDNS()` interact with the OS for packet I/O, and cannot be put into enclaves.
- Functions that process client queries could observe the plaintext of client IP and query, and should be in the enclaves. These functions are `Decryption()`, `Obfuscation()`, `Resolution()`, and `Encryption()`.
- The transition between enclave and non-enclave code costs CPU cycles, thus, calling enclave functions from non-enclave ones and the vice vera should be avoided to the best effort. Therefore in the implementation of the four enclave functions above, we put all their submodules into the enclave.

VII. IMPLEMENTATION

In the initialization protocol (§IV), step 3 is implemented as described in standard Intel SGX remote attestation[13]. All steps that require a symmetric cipher (`Encryption()` and `Decryption()`) are implemented in AES-GCM mode (an authenticated encryption cipher), and the asymmetric cipher in step 4, 5, and 6 is implemented using RSA Key exchange (not generation) that relies on TCP connections, which is implemented outside the enclave (which needs socket and system calls).

In the implementation of T^{2}DNS proxy, we follow the principles introduced in §VI-B and put trusted modules (i.e., `Decryption()`, `Obfuscation()`, `Resolution()`, and `Encryption()`) into enclaves (i.e., ECall); then we put untrusted modules (i.e., `ReceiveFromClient()`, `ReplyToClient()`, and `QueryTrustedDNS()`) outside enclaves (i.e., OCall). Specifically, the `Encryption()` and `Decryption()` are implemented with SGX Crypto library which has CPU hardware acceleration (i.e., AES New Instruction, AES-NI[17])

We build T^{2}DNS on Intel NUC Kits[18] which provides bare metal machine supporting hardware SGX, and the server with four 2.40GHz cores, 8GB memory, and 100Mbps bandwidth. We set up another machine to run the DNS agent and

issue DNS requests to the proxy, and the proxy is configured to use the DNS service in a cloud as the trustworthy Internet DNS source. In the experiment, we choose the DNS query dataset from [19], which includes 10 million DNS name queries from the real world.

We combine formulas (1)-(5) and get the following requirements on the system parameters. To avoid overloading the Internet DNS server, we choose $R = 500$; we assume the most tolerable latency $t = 10ms$, and set N_c to be 5. We tune T to be 0.1 and 0.05 with n_c varying from 1 to 5, and then we get corresponding n_d — $n_d = 9$ for $T = 0.1$ and $n_d = 19$ for $T = 0.05$. Finally, we set M to be 100000. We use these parameters in the following evaluation.

VIII. Evaluation

Our experiment shows that T^{2}DNS can interact with the Internet DNS server correctly, and we show the evaluation results of performance and overhead in this section. Our evaluation shows that T^{2}DNS has good performance and low overhead to support scalable clients' initialization and typical network DNS query volume.

A. Performance

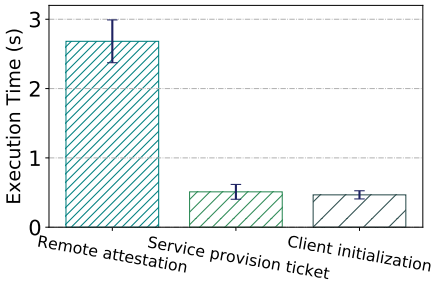
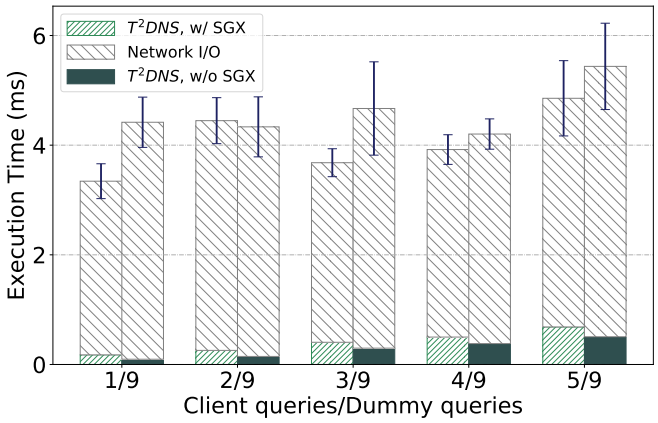


Fig. 5. Execution time for each initialization

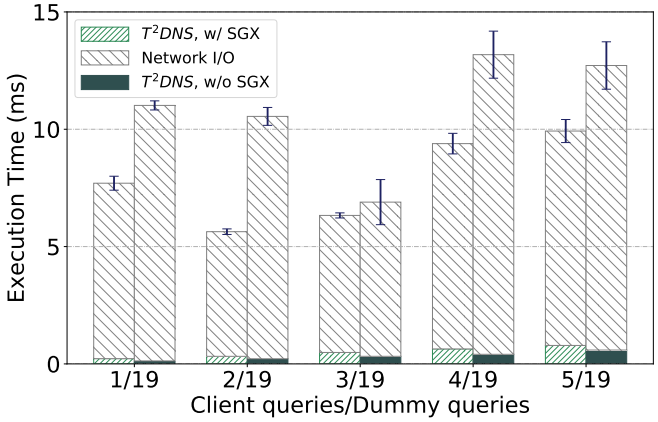
1) Initialization: We measure the execution time of remote attestation (step 3 in phase 1), service provision ticket application (step 4 and 5 in phase 1), and client initialization (step 6 and 7 in phase 2) in the whole initialization protocol. Note that remote attestation is performed by interacting with IAS on the Internet, which contains the Internet RTT time (about 100X ms); the other two procedures are performed on our testbed (in an Ethernet) and the RTT time is negligible (less than 1ms). The experiment is repeated for 1000 times and the result is in Figure 5. And we get the following observations.

In each SP proxy initialization, the remote attestation costs 2.68s, and the service provision ticket application takes 0.51s. The SP proxy does not boot/reboot frequently, so the cost is acceptable.

In each client's initialization, it would take less than 1 second (0.46s in the experiment) to verify a service provision ticket and build the secure channel. Each client would reuse the channel for further queries. More importantly, verifying the service provision ticket avoids each client to remotely attest the T^{2}DNS proxy (2.68s), making the client initialization to be scalable.



(a) $T = 0.1$



(b) $T = 0.05$

Fig. 6. Overall execution time to handle one batch query

2) Query Service Benchmark: Figure 6 shows the execution time for T^{2}DNS to serve client queries. In the experiment, we tune the privacy threshold to be 0.1 and 0.05, and the number of client queries in each batch to range from 1 to 5. We compare T^{2}DNS and T^{2}DNS without SGX enhancement to evaluate the SGX overhead. In the execution time benchmark, we specifically measure the time spent in interacting with the Internet DNS server (i.e., sending the obfuscated set and waiting for the replies), and show the percentage of this time in the total execution time (i.e., the backslash segment). The reason is that this interaction time depends on the Internet network quality and the Internet DNS server's processing capability, and we need to exclude it to evaluate the capability of T^{2}DNS proxy software. We repeat the experiment for 10000 times and get the following observations.

First, the time to interact with the Internet DNS server dominates the whole execution time. In all experiments, the interaction time is from 86% to 95%. Second, excluding the interaction time with the Internet DNS server, the local execution time at the proxy server increases linearly with the number of client queries. For example, when the privacy threshold is 0.1, the local execution time is 0.18ms for 1 client

query and 0.69ms for 5 ones. Third, comparing T^{2}DNS with and without SGX enhancement, the SGX protection costs extra 26% - 45% time (e.g., with threshold 0.05, the time is 0.23ms with SGX and 0.19ms without SGX for 1 client query, and 0.79ms v.s. 0.59ms for 5 ones).

We conclude that a commodity SGX server is sufficient to complete the local processing (at least 130 QPS[8] in our prototype v.s. 10 QPS in an enterprise network[20]), but the proxy itself introduces extra one-hop latency between the client and the Internet DNS servers. Thus, in practical deployment, T^{2}DNS can be scaled horizontally (i.e., more instances) to handle a large volume of DNS queries, but had better to be deployed near to the clients to reduce latency (e.g., in the same campus network, or in the same ISP region).

3) Micro Benchmark: We evaluate the performance of each invidual module in T^{2}DNS, including `Encryption()`, `Decryption()`, `Obfuscation()`, and `Resolution()`. Specifically, we would evaluate how the following factors influence the execution time of each module — parallelization, SGX enclave transition, and SGX enclave computation.

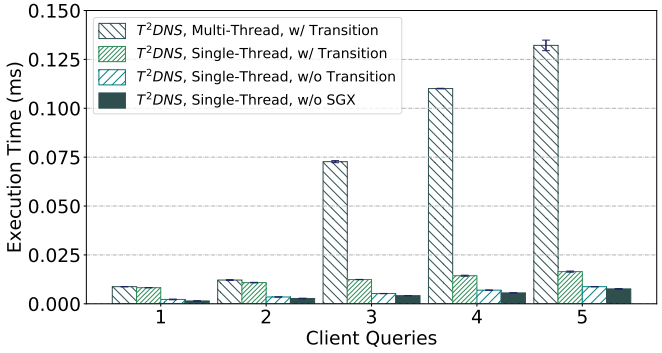


Fig. 7. Execution time to decrypt one batch query

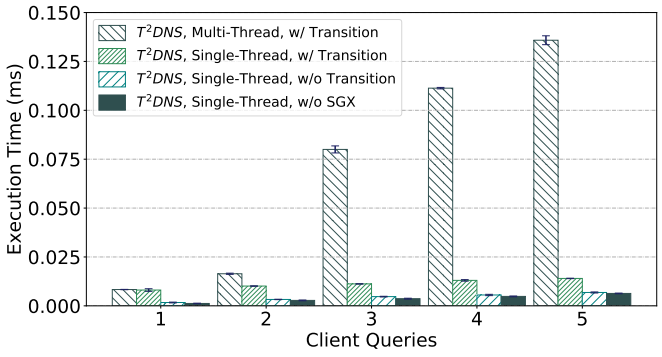


Fig. 8. Execution time to encrypt one batch query

Encryption/Decryption `Encryption()` and `Decryption()` would handle a batch of client queries but no dummy queries. In the experiment in Figure 7 and 8, we only tune the number of client queries in the X-axis (no dummy queries), repeat each experiment 1 million times, and compute the average execution time to process one query batch (X-axis). Among the experiments, "T^{2}DNS w/o SGX" is a pure non-SGX implementation and we use it as the baseline; we can tune T^{2}DNS in single-thread or multi-thread mode ("Single-Thread" v.s. "Multi-Thread"); by default, the code to encrypt/decrypt each batch needs making two transitions between the enclave mode and non-enclave mode (i.e., "w/ Transition"), thus, we specifically test a "w/o transition" mode which encrypts/decrypts all batches in the enclave without transition to evaluate the overhead of SGX enclave transition. The experiment results of encryption and decryption are similar, and we get the following observations.

First, implementing encryption/decryption in SGX enclaves does not increase the execution time significantly. With a privacy threshold of 0.05 and 5-query batches, SGX implementation causes 54% more time in encryption and 46% in decryption. And the total encryption/decryption time is 0.03ms, which is only 4% of the whole local execution time (0.79ms in §VIII-A2).

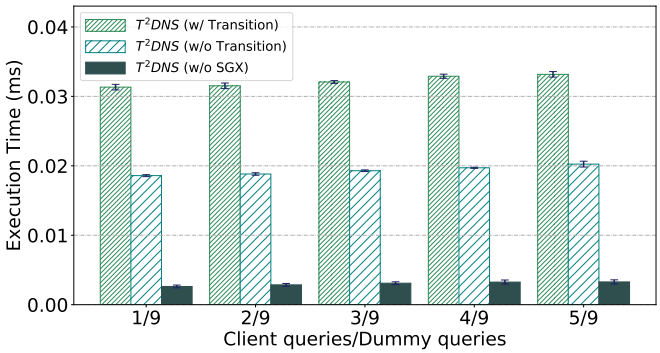
Second, the SGX enclave transition increases execution time significantly. Comparing the results of encryption/decryption in the single thread mode, starting the computation outside the enclave (then transit to the enclave to encrypt/decrypt) takes 0.006ms-0.008ms extra execution time than putting all computation inside the enclave, which is about 1 to 5 times of the execution time of encryption/decryption themselves. For example, the execution time of decryption with a batch of size 4 takes 0.013ms and 0.006ms respectively for the case with and without transition (2X).

Third, surprisingly, enabling multithreading in encrypting/decrypting multiple client queries does not reduce the execution time. Due to the limitation of SGX hardware or library, a thread cannot be created/waked up inside SGX enclaves; thus, in the current multithread mode in T^{2}DNS, we prepare multiple threads (one thread for one query) outside the enclave and let each thread transit into the enclave by themselves. With this implementation, we also observe that the total execution time increases with the number of threads. One potential reason is that the internal implementation of SGX enclave transition does not support multithreading well (e.g., concurrency control causes multithread enclave transition to be sequential), and the significant portion of execution time for enclave transition (26% - 45% of the local execution time according to the 2nd observation above) further amplifies the effect.

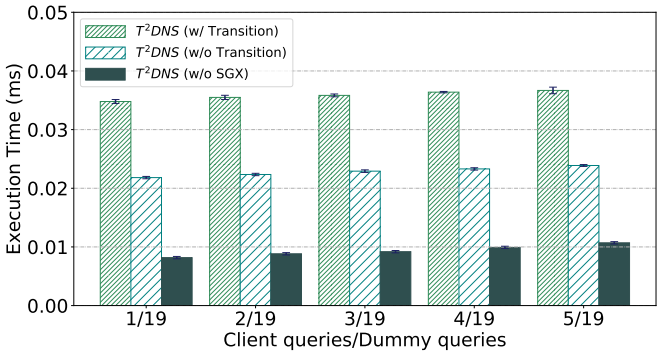
In summary, with the state of the art SGX implementation (hardware and library support), T^{2}DNS should be implemented in a single thread mode. And the SGX implementation itself introduces 2.2-5.8 times more execution time than pure encryption/decryption, but it is still an acceptable portion of time in the overall execution time.

Obfuscation/Resolution Similarly, we measure the performance of `Obfuscation()` and `Resolution()`, and show the results in Figure 9 and 10. These two modules process a batch instead of individual queries, and thus, they have

[8]It was evaluated under T 0.05 and 1-query batch.

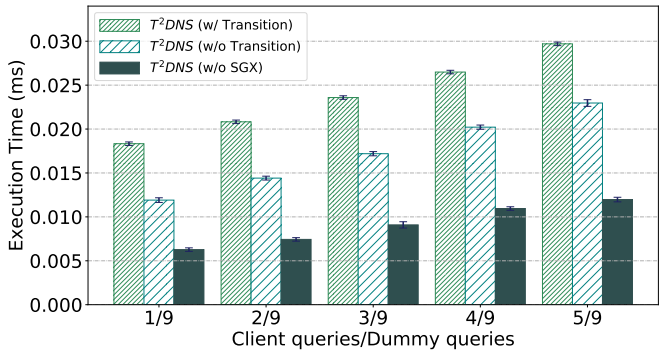


(a) $T = 0.1$

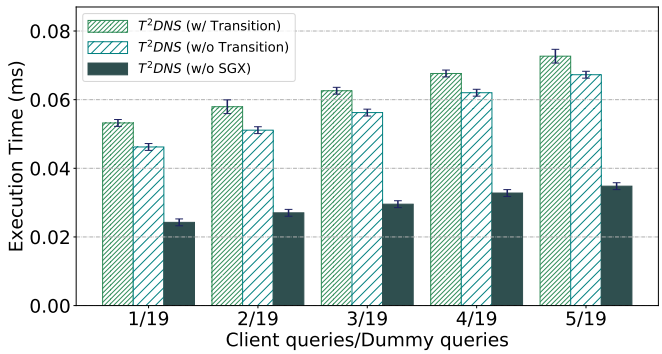


(b) $T = 0.05$

Fig. 9. Execution time to obfuscate a client query batch



(a) $T = 0.1$



(b) $T = 0.05$

Fig. 10. Execution time to resolve a batch

no multithreading mode; and we tune the privacy threshold to be 0.1 and 0.05, which decides the batch size. We have the following observations.

First, a lower privacy threshold (stricter privacy constraint) causes longer execution time, because a lower privacy threshold leads to a larger batch size, which takes more time to obfuscate and resolve. Second, obfuscation adds a constant number of dummy queries (9 dummy queries for $T = 0.1$ and 19 for $T = 0.05$), and the execution time does not increase significantly with the number of client queries. While resolution needs to match each client query with its response among the obfuscated response set, thus, the execution time increases linearly with the number of client queries.

Second, computing obfuscation and resolution in the SGX enclave introduces more significant execution time compared with computing encryption and decryption. With a privacy threshold 0.05, executing obfuscation inside SGX causes 2X of the time than that outside SGX; while encryption/decryption is about 1.5X. The reason is that Intel CPU has special instruction set to compute encryption and decryption (AES New Instructions, i.e., AES-NI[17]), which is integrated in both Crypto++ and SGX trusted Crypto library; and our implementation of `Obfuscation()` and `Resolution()` does not have hardware optimization. Finally, transition into/out of the SGX enclave causes extra CPU time. Each obfuscation and resolution would make two transitions (in and out), which costs about 0.005ms-0.013ms.

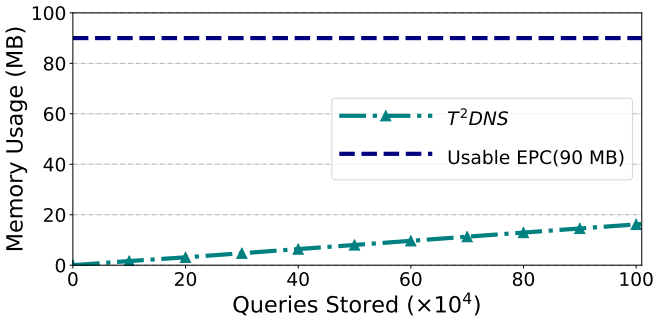


Fig. 11. Memory Usage in EPC Pages

4) Overhead: **Memory.** We further measure the memory overhead of T^{2}DNS. Figure 11 shows the memory usage with a varying number of entries in the obfuscation candidate set. The memory cost of T^{2}DNS is linearly proportional to the number of entries. From the results, we can observe that T^{2}DNS can support 1 million cache entries with 16.2MB memory, and yet is still far from the scope of SGX EPC page size (90 MB). If the memory usage exceeds the EPC size, there would be page swap causing performance degradation; T^{2}DNS could support large obfuscation candidate set (to defend posterior-observation attack) without suffering from this performance degradation.

TABLE III
T^{2}DNS QPS WITH 100% CPU

Privacy Threshold	Query Per Second	CPU Utilization
$T = 0.1$	1025	100%
$T = 0.05$	500	100%

CPU. Our implementation is on an Intel NUC, which is a small computing unit. In the experiment, we make the NUC run at full speed (i.e., handling 5-query batvch) to test how many queries can be served per second (Table III). In practice, an NUC is sufficient to support the service for a typical enterprise network (10 QPS)[20], and if the coming rate of client queries exceeds the processing capacity of an NUC, T^{2}DNS can be scaled by deploying more instances.

IX. Related Work

Privacy-Perserving Protocols. As discussed in §II-C, the encryption-based protocols (e.g., T-DNS[21], DNSCrypt[22], DNS over TLS[23], [24]) aim to achieve unlinkability. [25] requires the deployment of the secure communication protocol at the DNS server side, which is not feasible in the near future. And TOR-based approaches (e.g., aeon[26], [27]) suffer from the high resource cost of the TOR network and is also proved to have risks [7]). In DNS over TLS (DoT), the plaintext of a client's encrypted query can still be identified by applying machine learning methods[28], and thus, the encryption had better be further obfuscated to eliminate the statistical characteristics (e.g., interaction rounds, response time, etc.).

Obfuscation protocols are used in various applications, e.g., database[29], web search[30], [12], location-based service [31], context privacy[32], hidden branch decision[33], and machine learning[34]. DNS query has its unique characteristics, and obfuscation algorithms cannot be applied directly — first, multiple DNS queries cannot be merged as one query (unlike web search) [9]; second, DNS query volume must be bounded to avoid flooding the Internet DNS server (or falsely trigger DoS defense). And T^{2}DNS uses a customized batch obfuscation protocol.

Privacte Information Retrieval (PIR). PIR approaches[35], [36] are based on crypto frameworks (e.g., multi-party computation, or homomorphic encryption), and these solutions suffer from high communication cost. They have not been ready for practical deployment.

Intel SGX. Intel SGX is a trusted platform for many applications, e.g., network functions (SafeBricks[37] and ENDBOX[38]), functional encryption[39], web searching[8], and dynamic function[40]. T^{2}DNS refers to their design and implements a DNS proxy, where the influence of multithreading, SGX enclave transition, and SGX execution efficiency are carefully measured and evaluated.

Hybrid approaches. There are also works proposing a hybrid approach. For example, PEAS[41] combines encryption with obfuscation; X-search[9] combines SGX with obfuscation; and PDoT[6] uses TLS+SGX, and designs its client-specific attestation. T^{2}DNS is different from them by using hybrid protocols (both encryption and obfuscation) for compatibility, customizing obfuscation algorithm for lightweight, and customizing client initialization protocol for scalability.

Other DNS security issues. With the development of the Internet, the original DNS infrastructure[3] exposes several security risks. DNSSEC[42] is designed to protect the DNS data integrity, i.e., defending from DNS forgery[43], [44]; Liu et al. discussed and measured the risk of dangling DNS records[44]; DNSIntercept[45] describes the threat from attackers to intercept the DNS resolution path. T^{2}DNS is compatible to work with these solutions in practical deployment.

The DNS client has a reliability requirement — if one DNS query fails (i.e., no response), it would retry the query. However, this makes T^{2}DNS no able defend differential privacy analysis[10], [46] and re-identification attack[47]. We suggest complementing T^{2}DNS with a DoS attack detector. Once such an attack is identified, stop T^{2}DNS's service.

Distortion privacy metric[48] is discussed a lot for obfuscation algorithms. It assumes that ONE party with many true queries is obfuscated with many dummy queries, and the metric quantifies the probability for an adversary to find the dummy queries. In DNS queries, the scenario is different — there are many clients (many parties). Thus, we do not consider this metric in the privacy analysis.

Security solutions without a third party. There are decentralized solutions to protect DNS query privacy. For example, PPDNS[11], secret sharing[49], mix algorithm[50], and group messaging (Dissent[51], [52]) have a group of clients organized (e.g., Distributed Hash Table) and delegate each other's query to achieve anonymity. Organizing individual clients to collaborate is not an easy task; moreover, thus T^{2}DNS aims to enhance the security and trustworthiness of the third party service, which is not in the same scope of these works.

X. Conclusion

To make the Internet DNS service provision more flexible, we propose T^{2}DNS which is a third-party DNS service with privacy preservation and trustworthiness. T^{2}DNS uses a hybrid protocol of encryption and obfuscation to defend clients from channel attackers and be compatible with the Internet DNS protocol. T^{2}DNS is implemented on Intel SGX, thus, it can defend clients from server attackers and is able to provide proof of its trustworthiness. T^{2}DNS has a customized initialization protocol to allow many clients to attest the T^{2}DNS proxy in a scalable way; T^{2}DNS has a batch obfuscation algorithm that can bound the obfuscation communication overhead; T^{2}DNS models the system behavior and many attacks, and finds the feasible parameters for the practical deployment. Our prototype and evaluation of T^{2}DNS show that T^{2}DNS is fully functional, able to serve typical enterprise networks, and scalable to serve more clients.

[9]Although the DNS protocol is designed to support multiple names in one query, we find the Internet DNS server does not support multi-name queries.

REFERENCES

[1] S. Hao, H. Wang, A. Stavrou, and E. Smirni, "On the dns deployment of modern web services," in *2015 IEEE 23rd International Conference on Network Protocols (ICNP)*. IEEE, 2015, pp. 100–110.

[2] S. Castillo-Perez and J. Garcia-Alfaro, "Anonymous resolution of dns queries," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 2008, pp. 987–1000.

[3] "Rfc 1034," https://tools.ietf.org/html/rfc1034, accessed May 1, 2019.

[4] "Homepage of google public dns," https://dns.google.com/, accessed Nov. 1, 2019.

[5] "Homepage of opendns," https://www.opendns.com/, accessed Nov. 1, 2019.

[6] Y. Nakatsuka, A. Paverd, and G. Tsudik, "Pdot: Private dns-over-tls with tee support," *arXiv preprint arXiv:1909.11601*, 2019.

[7] B. Greschbach, T. Pulls, L. M. Roberts, P. Winter, and N. Feamster, "The effect of dns on tor's anonymity," *arXiv preprint arXiv:1609.08187*, 2016.

[8] R. Pires, D. Goltzsche, S. B. Mokhtar, S. Bouchenak, A. Boutet, P. Felber, R. Kapitza, M. Pasin, and V. Schiavoni, "Cyclosa: Decentralizing private web search through sgx-based browser extensions," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2018, pp. 467–477.

[9] S. B. Mokhtar, A. Boutet, P. Felber, M. Pasin, R. Pires, and V. Schiavoni, "X-search: revisiting private web search using intel sgx," in *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference*. ACM, 2017, pp. 198–208.

[10] A. Boutet, D. Frey, R. Guerraoui, A. Jégou, and A.-M. Kermarrec, "Privacy-preserving distributed collaborative filtering," *Computing*, vol. 98, no. 8, pp. 827–846, 2016.

[11] Y. Lu and G. Tsudik, "Towards plugging privacy leaks in the domain name system," in *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*. IEEE, 2010, pp. 1–10.

[12] E. Balsa, C. Troncoso, and C. Diaz, "Ob-pws: Obfuscation-based private web search," in *2012 IEEE Symposium on Security and Privacy*. IEEE, 2012, pp. 491–505.

[13] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

[14] A. ARM, "Security technology building a secure system using trustzone technology (white paper)," *ARM Limited*, 2009.

[15] "Intel software guard extensions remote attestation end-to-end example," https://software.intel.com/en-us/articles/code-sample-intel-software-guard-extensions-remote-attestation-end-to-end-example, accessed Nov. 1, 2019.

[16] S. Johnson, V. Scarlata, C. Rozas, E. Brickell, and F. Mckeen, "Intel® software guard extensions: Epid provisioning and attestation services," *White Paper*, vol. 1, pp. 1–10, 2016.

[17] "Intel data protection technology with aes-ni and secure key," https://www.intel.com/content/www/us/en/architecture-and-technology/advanced-encryption-standard-aes/data-protection-aes-general-technology.html, accessed Nov. 1, 2019.

[18] "Intel nuc kits," https://www.intel.com/content/www/us/en/products/boards-kits/nuc/kits.html, accessed Nov. 1, 2019.

[19] "Homepage of fdns," https://opendata.rapid7.com/sonar.fdns_v2/, accessed Nov. 1, 2019.

[20] D. Liu, Y. Zhao, K. Sui, L. Zou, D. Pei, Q. Tao, X. Chen, and D. Tan, "Focus: Shedding light on the high search response time in the wild," in *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[21] L. Zhu, Z. Hu, J. Heidemann, D. Wessels, A. Mankin, and N. Somaiya, "T-dns: Connection-oriented dns to improve privacy and security," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 379–380, 2015.

[22] "Homepage of dnscrypt," https://www.dnscrypt.org/, accessed Nov. 1, 2019.

[23] J. Dickinson and S. Dickinson, https://dnsprivacy.org/wiki/display/DP/DNS+Privacy+Implementation+Status/, accessed Nov. 1, 2019.

[24] Z. Hu, L. Zhu, J. Heidemann, A. Mankin, D. Wessels, and P. Hoffman, "Specification for dns over transport layer security (tls)," *IETF RFC7858, May*, 2016.

[25] A. Pfitzmann and M. Hansen, "Anonymity, unlinkability, unobservability, pseudonymity, and identity management-a consolidated proposal for terminology," 2005.

[26] D. Haslinger, "Towards an anonymous domain name system," Ph.D. dissertation, 08 2011.

[27] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The second-generation onion router," Naval Research Lab Washington DC, Tech. Rep., 2004.

[28] R. Houser, Z. Li, C. Cotton, and H. Wang, "An investigation on information leakage of dns over tls," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*. ACM, 2019, pp. 123–137.

[29] J. Domingo-Ferrer, A. Solanas, and J. Castellà-Roca, "h (k)-private information retrieval from privacy-uncooperative queryable databases," *Online Information Review*, vol. 33, no. 4, pp. 720–744, 2009.

[30] H. Nissenbaum and H. Daniel, "Trackmenot: Resisting surveillance in web search," 2009.

[31] M. Duckham and L. Kulik, "A formal model of obfuscation and negotiation for location privacy," in *International conference on pervasive computing*. Springer, 2005, pp. 152–170.

[32] R. Wishart, K. Henricksen, and J. Indulska, "Context privacy and obfuscation supported by dynamic context source discovery and processing in a context management system," in *International Conference on Ubiquitous Intelligence and Computing*. Springer, 2007, pp. 929–940.

[33] N. G. Tsoutsos and M. Maniatakos, "Obfuscating branch decisions based on encrypted data using misr and hash digests," in *2017 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2017, pp. 115–120.

[34] T. Zhang, Z. He, and R. B. Lee, "Privacy-preserving machine learning through data obfuscation," *arXiv preprint arXiv:1807.01860*, 2018.

[35] Y. Lindell and E. Waisbard, "Private web search with malicious adversaries," in *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2010, pp. 220–235.

[36] H. Pang, J. Shen, and R. Krishnan, "Privacy-preserving similarity-based text retrieval," *ACM Transactions on Internet Technology (TOIT)*, vol. 10, no. 1, p. 4, 2010.

[37] R. Poddar, C. Lan, R. A. Popa, and S. Ratnasamy, "Safebricks: Shielding network functions in the cloud," in *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*, 2018, pp. 201–216.

[38] D. Goltzsche, S. Rüsch, M. Nieke, S. Vaucher, N. Weichbrodt, V. Schiavoni, P.-L. Aublin, P. Cosa, C. Fetzer, P. Felber *et al.*, "Endbox: Scalable middlebox functions using client-side trusted execution," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2018, pp. 386–397.

[39] B. Fisch, D. Vinayagamurthy, D. Boneh, and S. Gorbunov, "Iron: functional encryption using intel sgx," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 765–782.

[40] R. Silva, P. Barbosa, and A. Brito, "Dynsgx: A privacy preserving toolset for dinamically loading functions into intel (r) sgx enclaves," in *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2017, pp. 314–321.

[41] A. Petit, T. Cerqueus, S. B. Mokhtar, L. Brunie, and H. Kosch, "Peas: Private, efficient and accurate web search," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, 2015, pp. 571–580.

[42] O. Kolkman and R. Gieben, "Dnssec operational practices," Tech. Rep., 2006.

[43] D. Dagon, M. Antonakakis, P. Vixie, T. Jinmei, and W. Lee, "Increased dns forgery resistance through 0x20-bit encoding: security via leet queries," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008, pp. 211–222.

[44] D. Liu, S. Hao, and H. Wang, "All your dns records point to us: Understanding the security threats of dangling dns records," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 1414–1425.

[45] B. Liu, C. Lu, H. Duan, Y. Liu, Z. Li, S. Hao, and M. Yang, "Who is answering my queries: Understanding and characterizing interception of the {DNS} resolution path," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1113–1128.

[46] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of cryptography conference*. Springer, 2006, pp. 265–284.

[47] A. Petit, T. Cerqueus, A. Boutet, S. B. Mokhtar, D. Coquil, L. Brunie, and H. Kosch, "Simattack: private web search under fire," *Journal of Internet Services and Applications*, vol. 7, no. 1, p. 2, 2016.

[48] R. Shokri, "Privacy games: Optimal user-centric data obfuscation," *Proceedings on Privacy Enhancing Technologies*, vol. 2015, no. 2, pp. 299–315, 2015.

[49] G. Di Bella, C. Barcellona, and I. Tinnirello, "A secret sharing scheme for anonymous dns queries," in *AEIT Annual Conference 2013*. IEEE, 2013, pp. 1–5.

[50] H. Federrath, K.-P. Fuchs, D. Herrmann, and C. Piosecny, "Privacy-preserving dns: analysis of broadcast, range queries and mix-based protection methods," in *European Symposium on Research in Computer Security*. Springer, 2011, pp. 665–683.

[51] H. Corrigan-Gibbs and B. Ford, "Dissent: accountable anonymous group messaging," in *Proceedings of the 17th ACM conference on Computer and communications security*. ACM, 2010, pp. 340–350.

[52] D. I. Wolinsky, H. Corrigan-Gibbs, B. Ford, and A. Johnson, "Dissent in numbers: Making strong anonymity scale," in *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, 2012, pp. 179–182.