DID We Miss Anything?: Towards Privacy-Preserving Decentralized ID Architecture

Siwon Huh, Myungkyu Shim, Jihwan Lee, Simon S. Woo, Hyoungshick Kim, and Hojoon Lee

***Abstract*—Decentralized Identity (DID) is emerging as a new digital identity management scheme that promises users complete control of their personal data and identification without central authority involvement. The World Wide Web Consortium (W3C) has drafted the DID standard and provided reference implementations. We conduct a security analysis of the W3C DID standard and the reference universal resolver implementation, focusing on user privacy in the DID resolving process. The universal resolver is the key component in the architecture that processes DID requests and DID document retrievals. Our analysis demonstrates that privacy issues can arise due to the imprudent design of the universal resolver. Furthermore, we found that side-channels in the DID document caching schemes of real-world DID services can entail privacy concerns. Motivated by our security analysis, we present a novel DID resolving design, called Oblivira, to enable obliviously DID resolving. Oblivira is a secure resolving agent with a small footprint that enforces the universal resolver to resolve requests without knowing their content. We also propose a privacy-preserving DID document caching scheme that eliminates side-channels. Our evaluation results show that Oblivira only incurs approximately 2.6% of overhead on average with different resolver settings (3, 6, and 12 threads).**

***Index Terms*—Blockchain, decentralized identity, privacy on internet, trusted execution.**

I. Introduction

DECENTRALIZED *IDentity* (DID) is a new type of identifier that is globally unique, resolvable with high availability, and cryptographically verifiable. In principle, users can create their own DIDs to fully control and manage their identifiers on the internet without the involvement of any central authorities which can potentially lead to privacy issues and a single point of failure. Users can prove things about themselves using their DIDs by exposing the minimum personal information necessary to use online services. For example, through a DID document, a user can present cryptographically provable proof that the user is of legal adult age without revealing one's birth date, legal name, or any other personal information. For these reasons, DIDs are rapidly gaining attraction as the internet's standard promising identity management scheme. Initiatives such as the Decentralized Identity Foundation (DIF) [1] and DID working group of W3C are developing standards and reference implementations for the DID ecosystem. Several blockchain-based DID architectures [1], [2], [3], [4], [5] have been recently developed to enable DID services based on the standards, such as the W3C Decentralized Identifier Specification v1.0 [6] and the reference implementation that is openly available [7].

In DID services, the *universal resolvers* are the central entities that bind heterogeneous blockchain networks by providing a compatibility layer that connects DID queries to blockchain networks. A universal resolver operates as a single entry point to the service that ties diverse service providers and distributed ledgers together. The universal resolver's main task is to receive a DID request and return a DID document from the blockchain network specified in the request. A DID document selectively contains the DID owner's personal data such as the DID owner's public key, verifiable credentials, licenses, and services depending on a DID-based service that the DID owner wants to use.

In this work, we conduct a security analysis on the DID standard focusing on user privacy to contribute to the ongoing security discussion on the new digital identity standard [6]. Our security analysis demonstrates two privacy issues that may undermine user privacy in the current DID resolving architecture. The first privacy issue is that user privacy can be undermined by observing DID resolving processes at universal resolvers. A universal resolver can potentially associate DID requests with DID documents in DID resolving processes to infer a user's sensitive data (e.g., location) from the associated DID and DID documents. The second privacy issue is the presence of temporal side-channels that could also undermine user privacy. We discovered side-channels in the DID document caching schemes in open-source resolver implementations. The caching of DID documents is found to create a difference in response time for recently queried DIDs. We base our analysis on the ongoing standardization effort for DID [6] and the reference W3C-compliant universal resolver implementation [7] and DID resolver that directly or indirectly adopted the reference code. Through extensive analysis and experiments, we systematically categorize and provide a detailed analysis of the side-channels in the current DID resolving architecture in Section III.

Manuscript received 14 October 2021; revised 11 November 2022; accepted 8 January 2023. Date of publication 13 January 2023; date of current version 13 November 2023. This work was supported in part by the National Research Foundation of Korea (NRF) grant funded in part by the Korea government (MSIT) under Grant NRF-2022R1C1C1010494, in part by Institute for Information & communications Technology Promotion (IITP) grant funded in part by the Korea government (MSIT): under Grant 2020-0-00666, in part by Research on Security of 5G-data-intensive Computing Platforms Based on Disaggregated Architecture and under Grant 2019-0-00421 in part by AI Graduate School Support Program (Sungkyunkwan University)), and AI Platform to Fully Adapt and Reflect Privacy-Policy Changes under Grant 2022-0-00688. (Corresponding author: Hojoon Lee.)

The authors are with the Department of Computer Science, Engineering, Sungkyunkwan University, Seoul 16419, South Korea (e-mail: calvin0420@g.skku.edu; audrb30@g.skku.edu; hwan3526@g.skku.edu; swoo@g.skku.edu; hyoung@skku.edu; hojoon.lee@skku.edu).

Digital Object Identifier 10.1109/TDSC.2023.3235951

To address such user privacy issues in the existing DID resolving mechanism, we propose a novel oblivious DID resolving design called OBLIVIRA (**Obliv**ious **I**dentity **R**esolver **A**gent), a small footprint enclavized agent that can be incorporated into the reference universal resolver implementation for privacy-preserving resolving. OBLIVIRA manages the DID and DID documents in a secure enclave during the resolving process and forces the universal resolver to be oblivious of the content of what it processes. In particular, OBLIVIRA addresses unique challenges in building a privacy-preserving DID universal resolver. First, the reference universal resolver implementation is complex multi-component software subject to frequent changes according to updates to the DID standard or new features. Moreover, because the universal resolver is composed of the DID resolver, third-party drivers for blockchain networks, and web-based front-ends, it would be imperative to design OBLIVIRA such that it can be readily plugged into the current and future versions of the reference implementation with minimal changes. Thus, OBLIVIRA is designed to leave the existing software components nearly intact and only requires changing only one line of code in the resolver and minimal modifications to the drivers. Second, the presence of DID document caching in the DID resolvers introduces a formidable challenge in OBLIVIRA design. The caches are an essential element in blockchain-based DID resolvers, as we will explain. Besides the aforementioned cache side-channel in the current resolvers, we explain that a naive adoption of secure enclaves fails to mitigate DID cache side-channel in our strong attack model. Hence, we present a DID document caching scheme with robustness to side-channels in OBLIVIRA and evaluate its efficacy.

In this work, we provide the detailed architecture design in Section IV. We also build a DID caching scheme that is free of side-channels, as we became aware of the necessity of DID caching schemes for large-scale DID services to reduce strain on the distributed ledgers. Our caching scheme eliminates potential side-channels from the untrusted universal resolver's perspective as well as that of an outside observer. Our design is also non-intrusive in terms of necessary modifications to the reference resolver architecture and performance overhead; Only a few lines of code modification are necessary, and the performance degradation in resolving due to OBLIVIRA is about 2.6% with different resolver settings (3, 6, and 12 threads), as demonstrated in our thorough evaluation in Section VI.

We summarize the contributions of our work as follows:

- We present our analysis on the security of the reference DID universal resolver. We present our findings on the privacy concerns regarding the correlation of information and temporal side-channels, including a new real-world side-channel we found on the existing DID resolver designs.
- We propose OBLIVIRA, a novel small-footprint design for a non-intrusive resolving agent that can be embedded into the existing universal resolver to enforce the universal resolver to perform its task oblivious of the data it processes.
- We evaluate our prototype implementation of OBLIVIRA and demonstrate its security guarantees and performance efficiency in processing DID requests.

II. BACKGROUND

In this section, we explain the background information that is necessary for an understanding of our design and contributions. First, we summarize the key concepts of DID using the terminology from the DID standard [6]. However, we will narrow our scope on the critical concepts related to this work. Second, we will explain the functionality and role of the universal resolver. The detailed internal architecture of the DID resolver will be further explained in Section III along with our security analysis. Lastly, we explain the structure of the secure enclave, especially Intel SGX, and introduce the concept of oblivious RAM that prevents the inherent security issue of SGX.

A. DID Terminology

We briefly outline the terminologies defined by the W3C DID standard, focusing on the necessary concepts to understand our work.

DID and DID URL. A DID is an URI that has the following format: `did:method:identifier` (e.g., `did:btcr:xyv2-xzpq-q9wa-p7t`), where a method is usually synonymous with the blockchain network that the DID belongs to. For instance, `btcr` stands for Bitcoin Reference DID and `ethr` uses Ethereum as data registry. A *DID URL* may also contain *fragments* that are used to contain additional options or reference to a specific resource within the DID document (e.g., `did:example:123?service=agent&relativeRef = /credentials#degree`).

DID Subjects and Controllers. DID subject refers to the entity that a DID identifies and has ownership of the DID. DID controllers hold permission to update the DID documents stored in blockchain networks. Hence, a DID subject is also a DID controller in most cases, and DID controllers are those who have granted permission from the DID subject to have their public keys engraved in the DID document.

DID Document. DID document is constructed in JSON format and contains a set of information that describes the DID subject. Notably, it contains the public keys of DID subjects and controllers to prove their ownership or control of the document using their private keys to the requesters. A DID document can also contain *service endpoints*, which are usually URL that points to external sources that belong to the DID subject.

B. DID Resolving Process and Universal Resolver

The universal resolver is designed to serve as a hub for DID requests by enabling the resolution of DIDs that belong to different blockchain networks. The universal resolver has mainly two components: the resolver and third-party drivers. For each DID request, the resolver selects a driver that is responsible for the DID method specified in the DID request URL and forwards the request to the driver. Upon receiving the request, the driver performs vendor-specific procedures to retrieve the DID document. In most cases, the driver's role is to compose the vendor-specific query format, which we shall call *Document Request Format* (DRF), for the blockchain server, where most of the DID document formulation occurs.

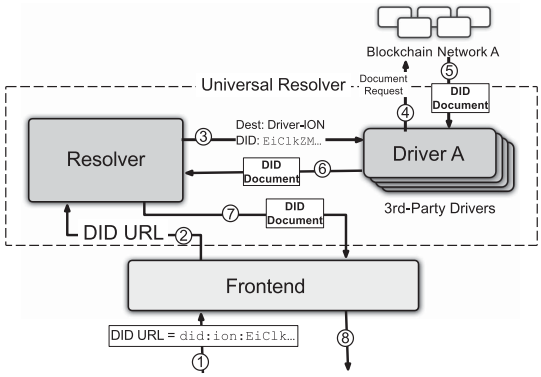


Fig. 1. Universal resolver infrastructure and data flow of DID resolution.

Fig. 1 illustrates the procedures that comprise the resolving process based on the universal resolver design and reference implementation proposed by DIF [7]. The resolver and each driver run as Docker container instances and communicate via HTTP over the local virtual network interface in the server. The resolving begins as a DID URL is delivered to the resolver through a front-end (e.g., a webpage query). The resolver takes the DID method field in the URL and finds the driver that is capable of handling that method. The driver then formulates a DRF to the service infrastructure or blockchain network using the requested DID. The driver forwards the DID document as it arrives in the resolver, and the resolver finally returns the document to the requester.

In addition, we found that the universal resolvers can collectively form a hierarchical structure. The implementation of third-party drivers is almost completely up to the vendors with minimal guidelines; a driver can be written to access the web service that interfaces directly with the distributed ledger or simply forward the request to the vendor's universal resolver. This is not necessarily the direction of the DID standard. However, it seems that this is the natural formation of the current DID ecosystem.

C. Secure Enclaves and Oblivious Computing

Secure Enclaves. Modern processor architectures often include hardware supports for isolating and protecting sensitive code and data, which are often called trusted execution environments or secure enclaves [8], [9]. OBLIVIRA is implemented using Intel Software Guard Extension (SGX) to protect its trusted computing base (TCB). In particular, SGX allows user programs to create an *enclave* in which sensitive code execution and data are protected in special memory pages called *enclave page cache (EPC)*. Pages in an EPC are protected from all privilege levels within the system, including OS and hypervisor.

SGX also provides hardware support and software components for *remote attestation* for in-cloud trusted services [10]. The attestation infrastructure in SGX allows the client of the service to cryptographically verify the authenticity of the SGX-enabled hardware and the integrity of the in-enclave program through the developer-signed known good hash value of the program's initial state. OBLIVIRA leverages Intel SGX to protect the DID and DID document contents, and allow client-bound attestation, to achieve trustworthy and oblivious DID resolution.

Side-Channels and Oblivious RAM. A number of works have revealed that the SGX enclaves can be susceptible to side-channel attacks due to their large attack surface [11], [12], [13]. These attacks have demonstrated that the adversary can profile the program behavior through fine-grained (e.g., cache line granularity) access trace collection and analysis. The key implication of the discovery of such attacks is that a naive adaptation of SGX only provides partial confidentiality of the processed data.

Oblivious RAM, or *ORAM*, is a primitive that hides memory access patterns from the untrusted server. ORAM recently got spotlighted as the cryptographically proven mitigation of access pattern-based side-channel attacks on SGX-enabled applications [14], [15]. ORAM is often implemented as a compiler-based tool that adds continuous shuffling and re-encryption of the data being accessed. The one downside of ORAM is its performance overhead, as the original work reported an overhead of 1400 $\times$ (O($(logN)^2$)). However, recent works have reduced the overhead significantly [16], [17], [18], [19].

OBLIVIRA leverages SGX enclave as a trusted anchor for protecting the resolving process of DID documents and therefore must consider possible leakage of sensitive information through side-channels. To this end, OBLIVIRA protects its DID document cache, a constantly updated data structure whose access pattern indeed leaks information, through adapting pathORAM [20]. As we will explain in Section IV, our ORAM-based DID document cache prevents information leakage from memory access patterns observed by the strong adversary with full system control.

III. SECURITY ANALYSIS OF DECENTRALIZED ID RESOLVING INFRASTRUCTURE

In this section, we present the potential privacy issues of the current DID universal resolver, which motivated our design of OBLIVIRA. Our analysis discovered several possible plausible attack scenarios in which the adversary may undermine user privacy through side-channels and information correlation. Moreover, the attacks can be amplified when the attacks are combined with multiple attack vectors. Our analysis reveals the concerning increasing privacy issues that may arise from identity resolving infrastructures. Universal resolvers, while built on top of public and decentralized data registry, provide a centralized DID resolving that resembles *Domain Name Systems (DNS)* resolving structure. Consequently, DID queries and DID documents inevitably aggregate at the resolver. Moreover, the resolving process also inadvertently reveals additional information, such as requester IP, browser agent string, and time of the requests, to the resolver. This additional information, combined with public information, is a potential privacy threat. In addition, temporal side-channels exist in some universal resolver implementations and can be leveraged as another source to correlate user information, as we will explain in more detail in the next section.

A. Threat Model and Assumptions

OBLIVIRA faces two types of adversary models. The first type of adversary is an *in-system* adversary with full control over the system that hosts the DID resolving service. The adversary controls all software on the system, including the system software (e.g., OS kernel), and hence the universal resolver itself can be considered malicious. Therefore, this type of adversary can observe the DID requests, resolved DID documents, and additional information about the requesters, such as the IP address of DID requester.

This adversary model represents several possible scenarios. The adversary may be a malicious insider (e.g., a disgruntled employee) who monitors all DID requests and document retrievals to pilfer user information. Alternatively, the model still holds the data-hungry cloud service provider or the DID service provider (who might be the same party) plot to accumulate the data so that they can learn additional information about the users through data analytics. Based on this adversary model, we discuss the user privacy issues in the current universal resolver design that mediates all DID interactions in a centralized manner.

The second type of adversary is among the requesters, an *outside* entity with no in-system leverage but access to the DID resolver's resolving service. Our findings indicate that there exists a temporal side-channel on the current resolver design that allows even such an outside observer to learn about the time of a DID document resolution. Therefore, if the adversary knows a victim's DID, the adversary can analyze the victim's service usage patterns. Moreover, when a DID-based service has ties to physical locations, the adversary may track a victim's locations through the attacks.

B. Information Correlation In the Resolver

We discuss the possible information correlation in the resolver with the aforementioned first type of adversary model, the malicious or compromised resolver. Unlike the original design principle of decentralized identity, the universal resolvers serve as a hub for DID queries and DID document fetches. When a large volume of seemingly public information is amassed, correlation and exposure of participating user information may occur. In other words, the universal resolver services running in the cloud can inherit privacy problems of the cloud. Malicious or compromised cloud providers, universal resolvers, and third-party provided drivers can examine, collect and possibly infer the relationships among the information. We outline the possible cases of such information correlations in the current DID-based identity services to demonstrate the motivations of the OBLIVIRA design here. Fig. 2 visualizes the sources of information correlation that are explained in this subsection.

DID and in-Document Correlation. While the DID itself and the contents of the DID documents are public information, it can still incur privacy issues in a general sense. First of all, DID requests on a certain DID reveal to the universal resolver that the DID is indeed active. Also, one or more services that are connected to the DID through the `service-endpoint` property in the DID document can be privacy-sensitive. The service-endpoint URLs inevitably reveal the nature of the

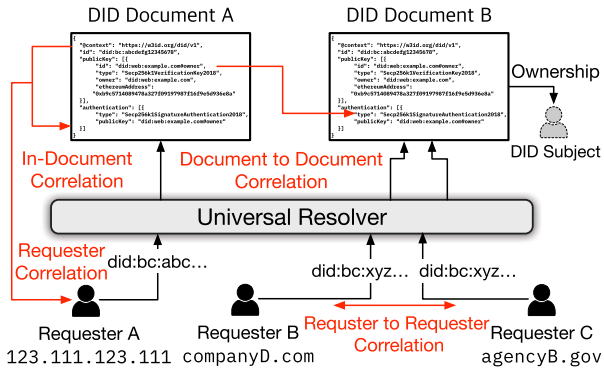


Fig. 2. Possible user information correlations during DID resolving process.

service that the DID subject uses (e.g., `www.linkedin.com`). Moreover, when more than one linked services are described in the document, their correlations may increase the sensitivity of the information. These issues are already under discussion in the standardization process [6]. While this issue alone may not be a profound privacy concern, we explain how it can be amplified when combined with other information.

Requester-to-Requester Correlation. Fingerprinting the requesters of a specific DID can also create a correlation of information. The source IP and access patterns may directly or indirectly expose region, time, and organization. Also, examining the DID requests reveals if the requester holds the permission of a DID controller. A malicious universal resolver may collect the list of source IP addresses that interact with a specific DID over time. The collected data can be further analyzed to fingerprint and uniquely identify the relationship between the DID subject and its requesters. The correlation grows as more requests that make queries or modifications on the same DID are collected. A possible correlation of information within a DID document, which we mentioned above, can produce another dimension of privacy leak, as the resolver can observe who makes changes on which part of the document.

Document-to-Document Correlation. Correlation of information present in two different DID documents that belong to two different DIDs can also occur. For example, a malicious universal resolver can run a keyword search on the DID documents that move through and harvest user information. This attack can identify two different DIDs that belong to the same entity or two entities with a linkage. This attack can be augmented with the DID and In-Document Correlation; a continuous collection of information may lead an attacker to induce a graph structure of the relation among requesters, the service endpoints in DID documents, and DIDs.

Perils of Centralized Resolving. Let us assume that the malicious universal resolver from our adversary model accumulates a large amount of DID requests and document contents over time. In order to put the risks of correlation that we have discussed thus far into concrete terms, we use the use cases of DID discussed in the W3C standard [6].

Consider a case where a financial institution must verify the user's bank statement and the person's educational credential (a digital diploma). A cryptographic proof issued by the bank that

does not reveal the person's identity but proves the account's balance is above the required amount is stored in the blockchain and can be retrieved by requesting DID A. The digital diploma is associated with DID B. As the financial institution makes subsequent DID requests for A and B, the universal resolver learns that the two DIDs are highly likely to belong to the same person, a case of fingerprinting of an individual through requester-to-requester correlation where the two requesters are the same party.

The resolver's access to DID document contents also gives rise to possible data collection that may prove valuable to advertisers. The services that authenticate users through DIDs inevitably embed their service endpoints (e.g., www.serviceA.com/auth?=..). Hence the resolver can, for instance, collect the unique user count of service A by searching DID documents with the given domain name, namely a case of document-to-document correlation. Furthermore, collecting the geographic location and time of access for the requesters of the DID entries would be trivially easy.

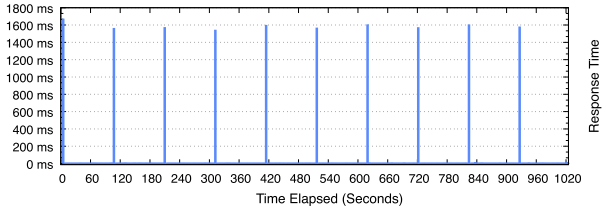
None of the additional information that the resolver can acquire is intended. The universal resolver arose out of necessity for a single entry for querying the data stored in the decentralized blockchain-based storage. However, its role in the current design is highly centralized and inherently prone to information correlation.

C. Temporal Side-Channel Observed From Outside

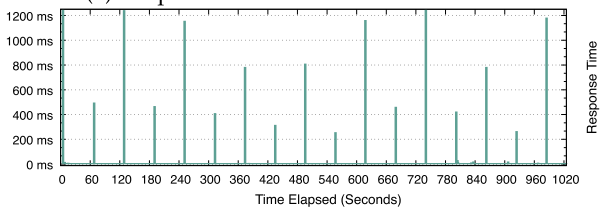
Time information opens another dimension of information correlation. As introduced in Section III-A, the temporal side-channel in the resolver allows an outside entity to undermine user privacy. We explain our findings on possible sources of temporal information gathered through the side-channels that exist in the current universal resolvers.

Time of Request as Private Information. The time of the DID query seen at the universal resolver can be a privacy concern when a connected service is closely tied to a specific location. An exemplary scenario is when DID is used as an authentication method for a physical entrance [21]. The contents of the DID document may reveal the entity that is requiring the authentication, thereby revealing the location, and further, it can be combined with the time of DID query.

Caching and Response Time Side-Channel. Our findings indicate that caching of DID documents is often involved in the DID resolving process. Such caching schemes can be easily found in the resolver, third-party driver, or blockchain network. We found that caching the retrieved DID documents is a prevalent way to alleviate the congestion under heavy loads. Due to blockchain networks' performance limitations, caching successfully retrieved DID documents can eliminate redundant DID document requests to blockchain networks. However, because of the caching, even an outside adversary who does not control the universal resolver can learn to estimate the time of DID requests by measuring the response time from the resolver. We found that DID document cache side-channels in multiple DID drivers and services show the example DID resolver cache [7], which is provided as a simple, generic interface for DID resolving.



(a) Response time side-channel in uPort[1]



(b) Response time side-channel in ION[1]

Fig. 3. Response time side-channel in DID universal resolvers (a) Response time side-channel in uPort[1]. (b) Response time side-channel in ION[1].

D. Real-World Resolver Cache Side-Channel

We identified side-channels introduced by DID document caching in the drivers included in the reference universal resolver. Fig. 3 illustrates the response time of the repeated DID request in two of such services, ION [2] and uPort [22], which are both self-sovereign identity management services, based on Bitcoin and Ethereum respectively. We hosted the universal resolver and the drivers in our local workstation to eliminate any caching in the network medium. We made DID requests locally to the running resolver using `curl` every second and measured the response time. Also, we prevented HTTP layer caching (e.g., `Cache-Control: no-cache`) in the web servers of the resolver and drivers that host the main programs written in the form of web apps. Then, we proceeded to analyze the driver code to pinpoint the exact locations of the DID document caching. The following code snippet is an excerpt from the driver code of Fig. 3(a).

```
...
var cache = require('js-cache')

const customCache: DIDCache = (parsed, re-
solve) =>
{
  if (parsed.params['no-cache'] === 'true')
    return await resolve()
  const cached = cache.get(parsed.didUrl)
  if (cached !== undefined) return cached
  const doc = await resolve()
  cache.set(parsed, doc, 60000)
  return doc
}
...
```

The code implements a DID document caching scheme using the `js-cache` package in node.js. The cache uses a Time-To-Live (TTL)-based eviction to keep the DID document for one minute. The peaks (around 1,500 ms) shown in Fig. 3(a) illustrate *cache misses* that occur every one-minute interval.

The DID request received at the universal resolver that has a method field of `ethr` is forwarded to the uPort driver. Then, the driver retrieves the DID document that belongs to the DID from the web service outlet of uPort's distributed ledger and saves the document in the caches. When the driver receives a DID request on the same DID, the cached document is returned without connecting to the distributed ledger, and hence produces the near 0 ms responses, as shown in Fig. 3(a).

We also investigated the code of the ION driver whose response time pattern is shown in Fig. 3(b), where the driver also employed a similar DID document caching strategy using `.Net` framework's HTTP response caching. We identified the factor that creates higher peaks (around 1,200 ms) every two minutes to be reconnecting to the distributed ledger server, and the lower peaks (around 500 ms) with an interval of one minute are the cache misses. Since both functionalities are built using the same framework and run in the same thread, we concluded that the two intervals coincide every two minutes.

Therefore, we expect that the time difference could reveal the time of successful retrieval of a DID document from the universal resolver. Factors such as the network connection quality may have an influence. However, the time difference is enough to distinguish between cached and uncached access. Therefore, the side-channel can be abused in many scenarios in which the DID is used for services tied with specific locations or privacy-revealing in other ways. We suspect that many services[1] choose to specifically cache the DID documents due to the performance and computation cost issues involved in using distributed ledgers as a data registry.

E. Analysis Implications and Security Requirements for OBLIVIRA

The privacy issues that we identified during our security analysis become the motivations for the OBLIVIRA design.

We concluded that keeping the requested DID and the DID document confidential from the universal resolver is essential. While this is a daunting task, it is yet a necessary task for oblivious DID resolving. The privacy issues we identified in DID universal resolvers pose profound implications because the information found in DID documents is anonymous but carries a high risk of correlation, and such information flows through a singular point. We foresee that this fundamental risk can be amplified as more services use DID as a gateway and more functionalities are added to the resolver as planned by the standards [6].

We also concluded that a secure DID document caching that is robust to side-channel attacks must be included in our design. The prevalence of DID document caching implemented in the resolvers and drivers indicates the necessity of such a caching scheme. In particular, based on the previous research on the performance of blockchain-based databases [23], [24], we expect that caching would significantly improve the viability of blockchains in large-scale services. In addition, while it is rather difficult to accurately profile the service capacity of current blockchain networks that depend on the infrastructure and implementation details, we were able to take a glimpse into the amount of workload imposed on the blockchain network while performing our experiments. For instance, retrieving a DID document from ION took up to 1,200 ms while the TCP ping (i.e., measuring the time it takes to establish a TCP connection) to the server was only around 200 ms from our end. Hence, our design seeks to provide a DID document caching scheme that is robust to side-channel attacks. This not only includes mitigating the side-channel that is observable from outside shown in Fig. 3, but also considering in-system side-channel attacks against enclave-protected cache that can be launched by a powerful adversary.

1.We found a total of 8 (`web`, `v1`, `key`, `uport`, `ion`, `sirius`, `unisot`, `sol`) out 34 drivers registered in the universal resolver to be using a caching scheme.

IV. OBLIVIRA DESIGN

OBLIVIRA is a design for a compact and portable secure enclave-enabled agent that can be incorporated into any universal resolver implementation to enable an oblivious DID resolving process. We create our design and develop our implementation based on the reference universal resolver. Our *Ephemeral DID (EphDID)* scheme is the fundamental idea that enables seamless integration of OBLIVIRA into the existing resolver. Also, OBLIVIRA's Oblivious DID Document Caches and the modified data flow prevent the untrusted existing universal resolver components from accessing the requested DID and the retrieved DID document.

A. Design Objectives

Based on the threat model and conclusion of our security analysis, we define the design objectives of OBLIVIRA as follows:

O1. Non-Intrusiveness. The first and foremost design objective of OBLIVIRA is non-intrusiveness, which has two aspects. First, our design must be readily incorporated into the reference universal resolver implementation with minimal modifications. The resolver has been adopted in many services, with over 30 third-party drivers developed specifically for the implementation [7]. Later versions of the DID standard, which is under active development, as evident in the frequency of updates, may not tolerate a heavily modified resolver design based on the current reference implementation. For this reason, OBLIVIRA is designed to reuse most of the reference implementation, as we will explain. Second, our design should minimize the performance impact on the reference implementation. We need to make sure that OBLIVIRA does not hinder the performance of the universal resolver under heavy load.

O2. Confidentiality of DID and Document. OBLIVIRA must force the untrusted universal resolver to perform the resolving process *without* knowing the requested DID and retrieved DID document. While OBLIVIRA reuses the existing resolver and driver code as much as possible (O1), it renders the resolver completely oblivious of the DID and DID document being processed. Confidentiality of the DIDs and documents is an imperative objective in thwarting information correlation as we explained in the previous section.

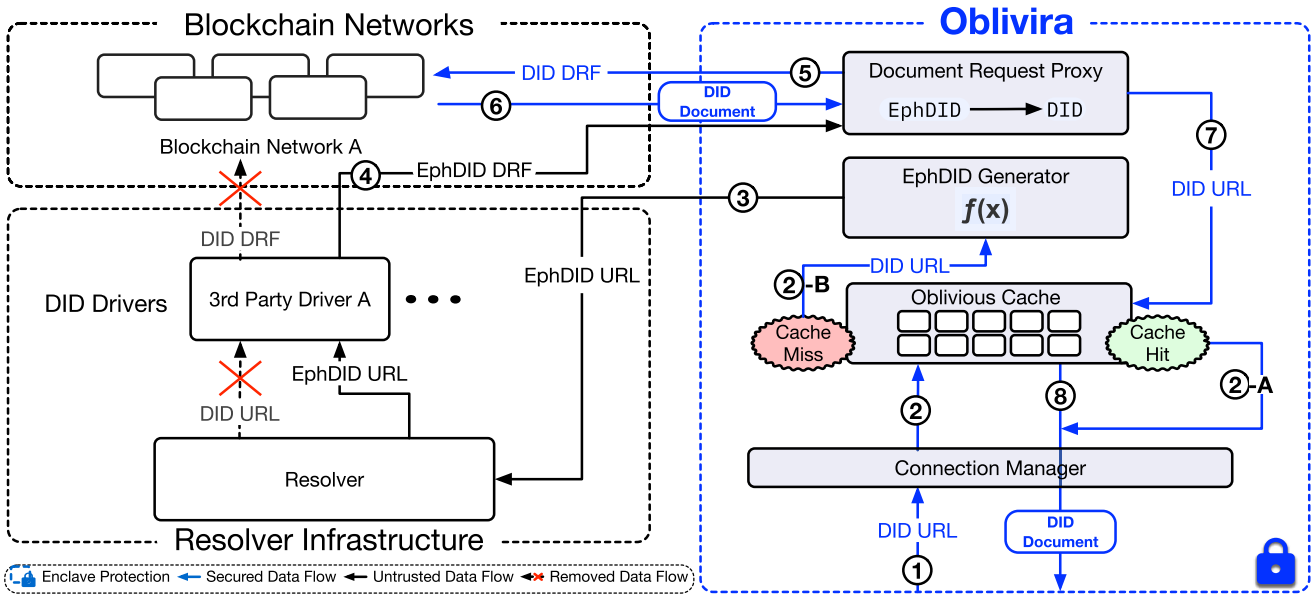


Fig. 4. Data-flow of OBLIVIRA-enabled universal resolver. OBLIVIRA maintains the confidentiality of requested DIDs and DID documents while reusing the existing resolving infrastructure.

O3. Response Time Side-Channel Mitigation. Our DID document caching scheme must also have a uniform response time. The current DID document schemes create query response time side-channel, as we explained in Section III-C. Our observation is that DID document caches' primary purpose is to reduce the restraint on the blockchain networks, not to shorten the query response time. Hence, we propose a new DID caching scheme that reduces DID document retrievals from blockchain networks, while effectively enforcing a uniform query response time to prevent side channels.

O4. Prevention of Cache-Entry Correlation in DID Document Caching. OBLIVIRA also must mitigate leakage of information through in-system side-channel attacks that can be launched by the malicious cloud service provider, universal resolver, and/or third-party drivers. More specifically, we must design a DID document caching scheme that does not reveal the specific entry accessed by the currently processed DID request. This effectively prevents the linking of two or more different requesters to a single cache entry.

B. OBLIVIRA Resolving Process Overview

We explain the changes in the resolving process that OBLIVIRA brings before explaining each component of our design. Fig. 4 illustrates the data flow of OBLIVIRA-enabled universal resolver design. OBLIVIRA intercepts two data flow points when adapted into the reference universal resolver: the incoming DID request from a client (①) and the driver's interaction with the blockchain network for retrieving DID documents (④–⑥). We explain the overall resolving process of a OBLIVIRA-enabled universal resolver by following the data flow from a DID request to the DID document return.

OBLIVIRA *Components and Data-Flow Changes.* Before explaining the resolving process data flow in the OBLIVIRA-enabled resolver, we briefly outline the key components of OBLIVIRA whose inner-workings will be detailed in the dedicated subsections (Sections IV-C, IV-D, and IV-E).

The *connection manager* that resides in the enclave takes over the client session handling of the universal resolver's HTTP server. This way, the DID requesters can establish a secure session directly with OBLIVIRA. The DID requests are delivered directly to OBLIVIRA through this channel (①), and also the same channel is used in the opposite direction when the retrieved DID document is returned to the requester (⑧).

OBLIVIRA's in-enclave *Oblivious DID document caches* replace the DID caching scheme of the universal resolver. OBLIVIRA's document caching scheme further secures the cached documents from information leakage through side-channel. It specifically mitigates two types of side-channels. The first possible side-channel is the DID document return time side-channel that we explained in Section III-C. Second, OBLIVIRA implements ORAM-based document storage to mitigate the known memory access pattern side-channel attacks [25], [26] against SGX enclaves that can be mounted by in-system adversaries.

The *EphDID Generator* and *Document Request Proxy* together facilitate *Ephemeral DID (EphDID)* scheme to force the existing resolving infrastructure to operate *without* having access to DID or DID Document. Our observation is that the existing universal resolver infrastructure does not need the real DID value for its operation; since the driver's job is usually limited to composing the vendor-specific DRF for the blockchain, as mentioned in Section II-B.

OBLIVIRA generates and maintains a list of EphDIDs for *each DID request* (i.e., not for each unique DID), and feeds the EphDID to the existing resolver infrastructure (③). The resolver and the drivers happily serve the request, and OBLIVIRA can reuse the resolver and vendor-maintained drivers with minimal

modifications. The Document Request Proxy collects the DRFs (④), which are ready to be sent to the blockchain servers, from the drivers and replaces the EphDID inside the document with its real DID.

From this point on, we traverse along the data-flow path of OBLIVIRA-enabled DID resolving process as shown in Fig. 4.

① *Remote Attestation and DID Request Arrival.* A DID query agent (e.g., a smartphone app) first attests the OBLIVIRA inside the server's DID resolver infrastructure before establishing a secure connection. In this process, the agent validates the identity of the enclave and a quote stating the integrity of the in-enclave code and data of OBLIVIRA. Agent sends a DID URL query, `http://{universal-resolver-ip}:{port}1.0/identifiers/did:ion:J9AnKD8jf-ZUjAbR`, to OBLIVIRA through the secure channel (Fig. 4-①).

② *DID Document Cache-Hit and Uniform Response Time.* Upon receiving the query request, OBLIVIRA first searches its oblivious DID document cache (`OCache`). In the case of a cache-hit (②-A), there is no need for further processing. At this point, the DID document in the `OCache` is ready to be sent as a response to the query. However, we inject an artificial delay before returning DID document to the requester in order to mitigate temporal side-channels(Section III-C). We explain the internals of our ORAM-based oblivious DID document cache later in Section IV-D.

②.*B*–④ *Ephemeral DID for Reusing Existing Resolver Infrastructure.* If a cache-miss occurs, OBLIVIRA proceeds to generate an *Ephemeral DID (EphDID)* for the given DID, then passes it to the existing resolver infrastructure (③). As such, the existing resolver and driver are forced to process an EphDID instead of the real DID of the received request. An EphDID is a randomly generated DID according to the vendor-specific length and format of the real DID. For instance, for the requested ION DID `did:ion:J9AnKD8jf-ZUjAbR`, we generate an EphDID with the matching format: `did:ion:EiClkZMDx-umQfTk`. From this point on, a series of processes performed by the existing resolver and drivers can generally operate with one crucial difference: the untrusted components now work with an EphDID (non-secret) instead of a DID (secret). The intermediate output produced by the driver, which we refer to as *DID Document Request Format* (DRF), will be interposed by OBLIVIRA. DRF is later used to retrieve the DID document from the blockchain.

⑤–⑥ *DID Document Retrieval With Document Request Proxy.* Since we cannot retrieve the intended DID document with the DRF generated with EphDID, we need to revise a small part of the driver to send DRF to our proxy in OBLIVIRA, Document Request Proxy, instead of the blockchain. Document Request Proxy receives the DRF generated with an EphDID that was initially intended to be sent to the blockchain network.

The proxy replaces the EphDID field in the DRF with the actual DID such that it can be sent to the blockchain network for the DID document. Below is an example of DRF, generated with an EphDID in ④:

After generating DRF, the proxy sends the DRF and receives a DID document through a trusted channel between the blockchain network server and the enclave in which OBLIVIRA runs. Upon receiving the document, the proxy forwards it to the Oblivious Document Cache.

```
//DRF with EphDID
GET /1.0/identifiers/did:ion: EiClkZMDx-umQfTk
HTTP/1.1
Host: beta.discover.did.microsoft.com
Connection: Keep-Alive
...
```

Algorithm 1: Resolving Process Inside OBLIVIRA Enclave.

```
Data: DID URL
Result: DID Document
Receive DID URL through Secure Channel;
Parse DID URL to DID & DID Method;
if OblivDIDCache[DID] == MISS then
    Generate EphDID;
        Search DID Syntax Rule(Length, Format) of the DID Method;
        Randomly generate EphDID based on DID Syntax Rule;
    Send EphDID to Universal Resolver;
    Generate DRF with EphDID;
        Find DID Driver correlated to DID Method;
        DID Driver generates EphDID_DRF based on its own rule;
    Send EphDID_DRF to OBLIVIRA;
    Generate DID_DRF with EphDID_DRF;
        Parse EphDID_DRF & Find EphDID-related elements;
        Replace EphDID to DID;
    Find Blockchain front-end address from DID Driver;
    Query Blockchain with DID_DRF;
    Receive DID Document from Blockchain;
    Write (DID, DID Document) in OblivDIDCache;
    Update average processing time for the DID Method;
else
    Read DID_Document from OblivDIDCache;
    Wait for the average processing time of the DID Method;
end
Send DID Document to Requester through Secure Channel;
```

⑦-⑧ *Caching and Returning of Retrieved DID Document.* The retrieved DID document is saved onto the caches to prevent unnecessary interaction with the blockchain network. Finally, the document is returned to the requester (client) through a secure channel established in ①.

Generalization of the Resolving Process. Here we revisit the resolving process of OBLIVIRA, which has been presented in chronological order following the passage of the DID request, in Algorithm 1.

C. Reusing Existing Infrastructure With EphDIDs and Proxy

OBLIVIRA's EphDID-based resolving scheme satisfies *O1. Non-intrusiveness* and *O2. Confidentiality of DID and document* in Section IV-A by forcing the existing resolving infrastructure to operate *without* having access to DID or DID Document, and at the same time, reusing their functionalities. Our strategy is to avoid creating a hard-to-maintain secure enclave adaptation of the reference resolver and design OBLIVIRA as a drop-in component that adds oblivious resolving. Our design choice is based on several observations and reasoning. First, the reference resolver implementation currently has been adopted in the services, with more than 30 third-party drivers in use. Second, future changes in the standard, derived services can easily render a complete enclave-enabled redesign of the resolver developed

based on the current version obsolete. By reusing unmodified resolver infrastructure, we can ensure and address the compatibility issue. Third, the drivers that retrieve DID documents from different blockchain networks are developed by third parties. The third-party developers contribute these drivers in the form of a container image; there is the minimal obligation to follow the standard API, and in fact, the drivers are written in diverse programming languages and frameworks. Hence, we concluded that including the drivers in the enclave TCB is insecure and infeasible due to portability and compatibility issues.

For this reason, we carefully examined the inner workings of the reference monitor and devised our EphDID-scheme that allows OBLIVIRA to respect the reference resolver's internal design and its interaction model with the drivers. OBLIVIRA hence can be incorporated into the resolver by a minimal modification that only changes that redirect the two data flow points to OBLIVIRA (shown in Fig. 4).

EphDID Generation. In order to generate EphDIDs for different DID methods, the EphDID Generator contains a list of DID method-specific EphDID generator functions. We generate a random number by using the SGX SDK-provided random number generator (i.e., `sgx_read_rand()`). It also maintains an EphDID-to-DID lookup table and shares it with the Document Request Proxy. The EphDID Generator queues a real DID-to-EphDID pair in a table, and the Document Request Proxy dequeues after DID document fetching has been completed.

Document Request Proxy. The proxy, which resides in the secure enclave, replaces the EphDIDs that appear in the DRF sent from the drivers with the real DID before relaying it to the blockchain networks. In order to forward the DRF generated with an EphDID and process it properly, we require two minimal changes to the drivers: First, the DRF must include the DID method of the request. This makes it easier to recognize which blockchain network the particular DRF was originally intended for. The Document Request Proxy could heuristically recognize the format. However, we concluded that it is more efficient to simply modify the driver to add an additional field in the request to include the DID method. Second, we modify the destination in the driver (e.g., `https://my-did-service.com/resolve?=xyv2-xzpq-q9wa-p7t...` to point to the proxy (e.g., `http://oblivira-container-ip:proxy-port`). In this way, the proxy can intercept the request and replace the EphDID with a real DID and retrieve the DID document.

D. Oblivious DID Document Caching

OBLIVIRA proposes an oblivious DID document caching scheme that prevents side-channels from two attack vectors simultaneously to achieve *O4*. First, it prevents information leakage from memory access patterns that can be observable by the untrusted software (e.g., the universal resolver). Second, it prevents DID request response time difference that an *outside* entity can observe.

ORAM-Based DID Document Caches. OBLIVIRA incorporates an adaptation of an ORAM scheme to its DID document caching to mitigate known side-channel susceptibility of SGX enclaves. In turn, a series of document cache accesses yield a uniform pattern in terms of the accessed memory address. The adversary observing the memory access pattern of OBLIVIRA enclave execution cannot make linkages between the DID requests and specific DID document cache entries.

Naive employment of SGX enclave may fail to ensure complete confidentiality of DID resolving process, as demonstrated in a number of existing works. As we discussed in Section II-C, several side-channels allow the adversary to monitor the memory access pattern within the enclave [13], [27]. Specifically for the OBLIVIRA design, we observe that the access pattern on the DID document caches indeed reveals information about the DID queries being processed. An adversary with full system control who possesses memory access monitoring capability can make a requester-to-requester correlation through the access pattern. More specifically, the adversary can learn of the accessed cache entry index(i) in both cases of a cache-hit and a cache-load following a cache-miss. With this side-channel primitive, the adversary can watch which cache entry is accessed for each DID request. A correlation that links the DID requesters rises when the adversary learns that different DID requests from different requesters induce access on the same cache entry i. This correlation can link multiple parties together or devices that belong to an individual.

OBLIVIRA's oblivious DID document caching scheme complements the SGX protection with side-channel elimination for a full confidentiality guarantee. Our approach is to prevent the untrusted in-system entities from pinpointing which cache entry was accessed through an ORAM-based DID document caching scheme. Implementing an ORAM-based DID cache complements the limited confidentiality guarantee of SGX; ORAM makes access to a specific DID cache index indistinguishable from access to other indexes, thereby preventing requester-to-requester correlation through DID caching. During our design process, we investigated solutions for stronger security guarantees: a design that hides not only accessed cache entry but also the occurrence of cache hits and misses altogether. However, we found that such a model is infeasible in the SGX security model. We further discuss this issue in Section VI-C. We explain the implementation of our oblivious DID document cache in Section V-B, and further discuss the security implications in Section VI-C.

Uniform Response Time DID Document Caching. We design the DID document cache to yield the uniform response time per each blockchain network from a DID requester's point of view. This is to directly address the attack that we explained in Section III-D and achieve *O3*. We internally maintain *real* average query retrieval time (T_{ravg}) for each blockchain network. Then, we carefully inject an artificial delay when the DID document retrieval from a blockchain has finished (T_{doc}) (right before returning the DID document to the requester.). OBLIVIRA offsets the response time in case of a DID document return from a cache-hit to achieve *O3*. Our observation is that the main purpose of a caching scheme in universal resolvers is to minimize the pressure on the blockchain networks, since traversing a large blockchain to retrieve a document can be relatively slower than typical databases. Therefore, we argue

that our uniform response time scheme effectively eliminates a side-channel with a logical trade-off, while retaining the purpose of the caching (*O3*). A more detailed description of the scheme is followed in Section IV-E. Mitigating the problem mentioned in our security analysis, OBLIVIRA also allows universal resolvers designs to apply caching for optimal resolving capacity safely.

E. Uniform Response Time Scheme

Retrieving a DID document from a blockchain network takes the longest time among many different steps involved in the resolving process, and different blockchain networks have different retrieval times due to the factors such as blockchain performance, the amount of load, and network latency with respect to the resolver. Hence, our uniform response time scheme was designed to apply different response offsets for each blockchain network. We measure the average time of DID Document retrieval for every distributed ledger corresponding DID method while the service is offline. These times will be used as a reference to create offset time for the requests which `hit` the cache. We internally maintain the offsets to retrieve the method-specific offset when a cache-hit occurs.

F. Portability and Performance

The non-intrusiveness *O1* in terms of necessary modifications to the existing resolver infrastructure is achieved by our overall design. Our design only requires a few lines of code in each driver to forward the DRF that is originally sent to the blockchain network to our proxy. The performance aspect of our objective *O1* will be further discussed in Section VI-A.

V. PROTOTYPE IMPLEMENTATION

In this section, we explain the prototype internals of OBLIVIRA. We implemented our prototype as a C/C++ application with Intel SGX SDK. For secure communication, we adapted *wolfSSL* [28] which has a built-in SGX support. Our prototype consists of approximately 3.2 K lines of C/C++ code, where the core functionalities of OBLIVIRA design near 1.7 K lines of code and the rest (1.5 K) are ORAM-based DID document cache implementation. OBLIVIRA leaves the existing universal resolver components nearly unmodified; we operate the resolver without any modifications and change the hardcoded destination address of the drivers to redirect the DRF to the Document Request Proxy in our design. The universal resolver and each third-party driver are deployed as Docker container images. They run web servers as their front-end, and their services are implemented as web apps. OBLIVIRA adapts SGX to protect its core functionalities and is deployed inside a Docker like the existing resolver infrastructure.

A. Enclavized DID Resolving Agent

Internally, three main services constitute OBLIVIRA. These are namely, *DID request handler* (DID handler), *DRF proxy*, and *DID document fetch handler* (doc handler). The handlers are implemented as thread functions that can be dispatched to the worker thread pool managed in our prototype. The threads enter the enclave mode through (`ECall`) to handle sensitive operations such as handling of DIDs and DID documents and establishing TLS connections. Each of the three services is implemented and runs as a thread, where more threads can be assigned to each service as needed. We evaluate the performance of our prototype with a varying number of threads for each service in Section VI.

DID Request Handler. DID request handler performs roles that correspond to the functions of the connection manager, oblivious DID document cache, and ephemeral DID generator shown in Fig. 4. The handler exposes the trusted service via port 443 as a web service. Upon receiving the user connection, it performs attestation of the enclave-side of OBLIVIRA and provides a quote to the challenger (i.e., the DID requester). The attestation scheme is taken from the wolfSSL-based remote attestation example from Intel [29]. DID request handler establishes an enclave-to-client TLS session inside enclave mode and receives a DID request (a DID URI) through a `HTTP GET` method. Also, for ephemeral DID generation, we used the in-enclave random number generator (i.e., `sgx_read_rand`). We manage an EphDID-to-DID map in the oblivious DID document caches (i.e., ORAM-managed and enclave-protected memory).

DRF Proxy. DRF listens on an arbitrary port (HTTP server), and the DRF Proxy thread enters enclave mode with the received ephemeral DID DRF upon receiving a `HTTP POST` from a driver. DRF enters enclave mode to look up the EphDID-to-DID map, then rewrites the DRF with the DID. The translated DRF, which now has a real DID, is forwarded to the DID document fetch handler explained in the next paragraph.

DID Document Fetch Handler. The first task of the doc handler is to contact the vendor's blockchain server to retrieve the document with the given DRF. Then, it enters the enclave to securely establish a TLS connection to the vendor's blockchain web interface to finally fetch the DID document. Upon successfully retrieving a DID document, the handler will store the DID document in the cache and return the DID document to the requester.

B. Oblivious DID Document Caching

Our ORAM-based DID document cache (`OCache`) mitigates in-system side-channels that may allow the untrusted universal resolver to pinpoint the accessed cache for a DID request. Our implementation is based on the pathORAM version of ZeroTrace [30]; we optimized the ORAM implementation to the size of the cache entry: a DID and a DID document. The bandwidth of ORAM and the size of storage space are inversely proportional to each other.

Also, our DID document cache resides within the EPC pages of an enclave and is essentially a key-value database where the DID is the key, and the corresponding DID document is the value. The size of the DID is set to a maximum size of 64 bytes, considering the length of the typical DIDs in deployed services. The DID document also has a maximum size of 2,048 bytes, which is also set conservatively according to the sizes of the DID documents we analyzed. Within the oblivious cache, there are several data structures. Outside enclave-protected memory,

Algorithm 2: OBLIVIRA's ORAM-Based DID Document Caching Algorithm.

```
Data: DID
Result: DID Document if cache HIT
 PathORAM & DID Map initialization;
 Converts DID to integer 'DID Index';
 Linearly scans DID Map;
 if DID Index exists in DID Map then
     ORAM.READ(DID Index);
          Read Leaf for DID Index in position map;
          Remap position map;
          Fetch Entire Path;
          Retrieve blocks and get DID Document;
          Write the Path back;
          Waits for the offset time;
     if Corresponding DID Document exists in ORAM then
         cache HIT;
     else
         goto cache MISS;
     end
 else
     cache MISS;
     Assign DID Index in DID Map;
     DID Resolution Process;
     ORAM.WRITE(DID Index, DID Document);
          Read Leaf for DID Index in position map;
          Remap position map;
          Fetch Entire Path;
          Retrieve blocks and write DID Document;
          Write the Path back;
 end
```

an encrypted binary tree mapping for the entire memory is stored, and also DID document is stored encrypted inside the tree. In the enclave, DID Map, position map, and stash are used. DID Map is a map that indexes DIDs, where the indexes allow searching DID documents in the position map of pathORAM. A position map is a DID index dictionary and leaf node of the encrypted memory tree, and a stash is a storage for temporally stored data, which is DID document in this case.

Next, Algorithm 2 illustrates the operation of `OCache`. When accessed with a DID Index, the path from the root node to the leaf node mapped to the DID Index is brought into the enclave via an `OCall`. The path is then decrypted inside the enclave, and the block containing DID Document corresponding to the DID Index is extracted to the stash. When the corresponding block is empty, in the case where OBLIVIRA receives a request for a DID that is just queried and requested but not yet resolved, the path is put back to the tree, and the tree gets reconstructed. After that, it follows the DID resolution process as there is no corresponding DID document in the cache. If the DID document is successfully retrieved from the ORAM, the path is put back to the tree, and the tree is reconstructed. In the case of cache eviction, an LRU strategy is adopted. In summary, 1) DID gets converted to DID Index and DID Index is used for searching position map, 2) the path containing a DID document in the binary tree is imported through the position map, 3) DID document is extracted into the stash, 4) the entire binary tree gets reconstructed in the way of pathORAM algorithm, and 5) finally, the stash is searched through `cmov` instruction to return DID document, which guarantees oblivious computation.

TABLE I
RESPONSE TIME COMPARISON OF OBLIVIRA VERSUS REFERENCE RESOLVER

Response Time Statistic	Reference Resolver	Oblivira
Avg. Response Time	760.1 ms	790.5 ms
Standard Deviation	24.54 ms	36.8 ms
Max	1,509 ms	1,894 ms
Min	749 ms	780 ms

Also, read and write operations are performed in the same way; extracting value or putting the value inside the tree is the only difference. By implementing ORAM on the cache, the adversary cannot infer anything on observing the memory access pattern but can only observe the random access on the memory.

Uniform Response Time. Uniform response time is implemented to mitigate temporal side-channel attack explained in Section III-C. We use the x86 `rdtsc` instruction as a source of high-precision timer. OBLIVIRA measures the time before the response to DID Controller, collects the response time in a certain time window, and sets the maximum difference as the offset time. Therefore, the response time difference between the cache hit and miss is unnoticeable. Additional delays or small variances caused by differing network conditions are outside the scope of this work, as we intend to make the cache hit and miss indistinguishable, not implementing strictly uniform response time. The `sgx_get_trusted_time()` (returns elapsed seconds) provided in the SGX SDK, does not provide enough resolution for our needs. Hence, we currently implement an `OCALL` to exit enclave to execute a `rdtsc` instruction to access high-resolution timer. This means that the source of time is insecure, and it is a limitation that can be solved with the upcoming version 2 of Intel SGX. We further discuss this issue in Section VIII.

VI. OBLIVIRA EVALUATION

In this section, we present our evaluation results on OBLIVIRA. We evaluated OBLIVIRA in an SGX 1.0-enabled machine with 8-core Intel Core i7-10700 2.90 GHz CPU and 16 GB RAM, running Ubuntu 20.04 with kernel version 5.8.0. We evaluate the performance of OBLIVIRA-integrated universal resolver against the reference universal resolver in Section VI-A. We provide the experimental result of our uniform response time method in Section VI-A. Lastly, in Section VI-C, we provide an in-depth security analysis of OBLIVIRA regarding its security guarantees against the major privacy issues we found.

A. DID Request Processing Performance

We conducted an experiment to evaluate and compare the performance of OBLIVIRA against the reference universal resolver to demonstrate the practicality of OBLIVIRA. In particular, we measured the number of DID Requests processed Per Minute (RPM) and presented the results in Fig. 5 and the average response time in Table I. We used the `ion` DID method. On the other hand, the caches such as the HTTP cache and DID driver's DID document cache are disabled to accurately measure the resolving performance. Each DID request was made using `curl` by sending an `HTTP GET` packet to the universal resolver.

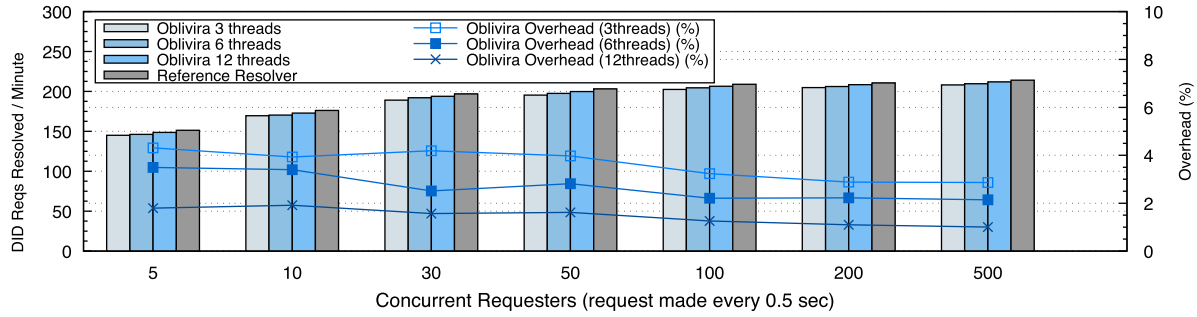


Fig. 5. Request-per-minute comparison of OBLIVIRA versus reference resolver with varying size of worker threads in OBLIVIRA. DID document caching is disabled for both approaches in all experiments.

Response Time Comparison. As shown in Table I, we measured the round-trip time for a DID request sent out to the reference resolver and OBLIVIRA-enabled resolver. We measured the average response time for reference resolver and OBLIVIRA-enabled resolver in 1,000 consecutive DID requests. The reference resolver took an average of 760.1 to return a DID document, and OBLIVIRA took 790.5 milliseconds, respectively. This result shows that the use of OBLIVIRA into the reference resolver has a minimal impact (4%) on the DID request response time.

Request-per-Minute Comparison. Fig. 5 illustrates DID requests processing performance of the OBLIVIRA-enabled resolver and the vanilla reference resolver. We ran multiple concurrent `curl` processes to generate synthetic DID requests to the reference universal resolver and OBLIVIRA-enabled universal resolver. We tested four different resolver settings, where one of them is a reference resolver without OBLIVIRA, and the other three approaches are OBLIVIRA with different numbers of worker threads (3, 6, and 12 threads). The worker thread count for the reference resolver and the driver in all four tested resolvers was set to the default value (200 for both the resolver and the driver). The requester processes make a DID request every 0.5 seconds. Also, we varied the number of concurrent requesters to create seven different levels of workload for the tested resolvers. Then, we measured the number of successfully returned DID documents during 20 minutes to calculate the number of RPM.

In the experiment (Fig. 5), we observe that the number of DID requests resolved per minute (RPM) saturates around 200 in all four resolver settings. In fact, we deliberately set the workload configuration in our experiment such that captures the saturation point. The saturation occurs because DID document retrieval from the blockchain network is a constant bottleneck during the resolving process (59.5% of the total response time to a request), as we will explain in our microbenchmark. The worse case was observed in the case of 3-threads and 5 requesters, which showed an overhead of 4.3%. The overhead of the 3-thread OBLIVIRA decreases as the load is increased with 2.88% in 200 concurrent requesters and 2.86% in 500 concurrent requesters. The average of 3-thread overheads across all load levels was 3.62%. In all, the average overhead in terms of RPM, with different resolver settings (3, 6, and 12 threads) under different loads, was 2.6%. Increasing the number of threads for OBLIVIRA had a limited improvement: the average overhead was measured to be 2.68% in 6-thread OBLIVIRA and 1.46% in 12-thread. Therefore, our experiment demonstrates that OBLIVIRA incurs only a negligible performance overhead to the operation of the reference resolver.

TABLE II
TIME MEASUREMENTS FOR EACH OPERATION DURING DID RESOLVING PROCESS. MEASUREMENT UNIT IS MICROSECOND(μS)

Receive DID Request	16,077	EphDID to RealDID in DRF	30
Cache Lookup	2,193	Send DRF to Blockchain	130
Generate EphDID	35	Retrieve DID Doc	772,440
Send EphDID to UR	30	Store to DID Doc Cache	2,980
Wait for UR DRF Gen.	499,985	Return DID Doc to Requester	2,655
Receive DRF from Drv	820	**Total: 1,297,375 μs**	

Measurement unit is microsecond(µs).

More specifically, OBLIVIRA shows its efficacy in terms of performance in two aspects. First, we find that OBLIVIRA does not hinder the performance overhead of the reference resolver even at the full load. Second, OBLIVIRA runs efficiently even with a small number of available worker threads. While the performance overhead does decrease with the increased thread counts, the benefit of increasing threads is limited; as the 3-thread setting already demonstrates a small overhead, and the resolved DID request count increase due to more threads is not significant. In conclusion, OBLIVIRA can effectively enable oblivious resolving to the reference resolver by adding only three additional threads to the resolver and the driver that are designed to use up to 200 threads.

Microbenchmarks. As shown in Table II, we evaluated microbenchmarks that measure the time consumption in eleven major operations that were explained in Section IV, during a resolving process in order to understand the composition of the performance overhead. The overall execution time was 1,297,375 μs. Note that the total time is higher due to all debugging codes enabled for the experiment. The major bottleneck was identified to be the retrieval of the DID document from the blockchain network (9. Retrieve DID doc, 59.5%). The operations performed with an EphDID by the resolver and driver took (5. UR to DRF, 38.5%). Note that this is time excluding receiving DID request (1) and retrieving the DID document from the blockchain network, which is now being performed by OBLIVIRA instead. The additional operations, such as (2,3,4,7,10) that are not present in the reference but added by OBLIVIRA, only add minimal performance overhead.

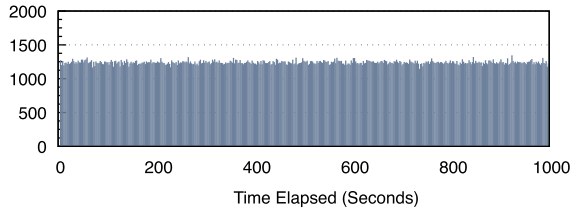


Fig. 6. Uniform Response Time of OBLIVIRA. Response time was measured with our proposed uniform response time scheme.

B. Uniform Response Time

To guarantee mitigation from the temporal side-channel, we implemented uniform response time in OBLIVIRA. As explained in Section IV-E, we add an artificial delay to render cache hits indistinguishable from cache misses. Fig. 6 presents the response time of a repeated DID request on the same DID. The experiment was conducted with the same DID request generation method as discussed in Section III-D, and we used the ION DID method for testing. This result shows that OBLIVIRA's DID document caching eliminates the side-channel that are observable from outside but still alleviates the strain on the blockchain networks. Fig. 6 presents the response time of around 1,200 milliseconds because the response time of a DID request to the ION network takes over one second.

C. Security Analysis of OBLIVIRA

We show how OBLIVIRA satisfies the security properties with respect to the threat model discussed in Section III-A through a thorough and point-by-point analysis. The goal of OBLIVIRA is to enforce that the possibly compromised or curious cloud infrastructure, which is inclusive of the untrusted resolver components, learn nothing about the DID queries and their results. Additionally, we mitigate a case of a requester-side side-channel that we discussed in Section III-C. Our analysis is structured with the following format: we outline the main objectives of OBLIVIRA design as *Objective N*, then we discuss the design properties (*Properties N.M*) that directly contributes to achieving the objective.

Objective 1. OBLIVIRA's operation can be verified by the remote DID requester.

The adaptation of SGX's remote attestation scheme [10], allows the DID requester to (1) confirm that the hardware on which the DID resolving is operating is an authentic SGX-enabled processor (2) an unmodified OBLIVIRA code has been loaded and intact in enclave memory.

Objective 2. In-memory contents and processing of DID and DID document are protected.

Property 2.1. Untrusted components, the universal resolver, and drivers are forced to operate only with EphDIDs.

In each DID resolving process, the DID requester must reveal three types of information: the DID, the retrieved DID document, and the source IP address from which the request is connecting. Among them, the most crucial information in the DID resolution is undoubtedly the DIDs and DID documents since all the additional information should be linked to a certain DID and DID document for the correlation of information. Through the ephemeral DID scheme, OBLIVIRA enforces the resolver to process DID request to be oblivious to this information. This means that the correlation of information that can undermine the privacy of the user is practically eliminated.

Property 2.2. Operations with DID and DID document are only performed within secure boundaries.

OBLIVIRA inherits all in-system security guarantees of SGX. OBLIVIRA design isolates the processing of DID and DID document. Hence, the code execution and the processed data are protected from all privilege levels within the system throughout the resolving process. Moreover, with SGX, all code and data are encrypted upon storage in memory, thereby preventing possible physical access to the DRAM.

Objective 3. DID document caching scheme in OBLIVIRA induces no observable temporal information.

Property 3.1. All observable memory accesses on DID caches are random and indiscernible.

ORAM-based DID cache in OBLIVIRA continuously remaps and re-encrypts its accessed data. The access patterns that the DID cache generates are mathematically proven to be random and indistinguishable. That is, DID cache in OBLIVIRA guarantees the prevention of linking a cache hit to a specific cache entry. Consequently, the adaptation of ORAM in OBLIVIRA design prevents the correlation of different IP addresses that are requested with the same DID, in addition to the DID and DID document protection provided by OBLIVIRA.

Property 3.2. All DID requests have equal response time for cache-hit and cache-miss from the requester's perspective.

OBLIVIRA guarantees uniform response time of OBLIVIRA-enabled resolver to DID requests. As we explain in Section IV, OBLIVIRA offsets the timing of DID document returned from a cache hit while even considering varying retrieval times for different blockchain networks. This effectively eliminates the side-channel observed from the point of view of an outside observer who has learned the DID, aiming to capture the time when the DID is requested.

As shown in the Table III, OBLIVIRA effectively prevents the sources of privacy leaks that exist in reference DID resolving architecture. However, there are some issues that we leave as orthogonal or future work. First, our design does not hide the source IP address or browser agent of the requester's connection to the resolver and regards it as an orthogonal issue. Users can choose to hide their IPs through conventional VPN services or Tor [31]. We believe that such protections would be sufficient. Preventing the OS kernel from knowing the source IP address is challenging with the security model under SGX. While SGX does not trust the kernel, it still depends on the kernel for access to system resources (e.g., through system calls such as `socket`). Hence, it is not feasible or desirable to include the source IP address in OBLIVIRA's privacy guarantees. On the other hand, we leave the temporal information leakage by the observation of system calls in OBLIVIRA as future work. Because an enclaved execution interacts with the OS kernel through system calls in the SGX security model, the untrusted kernel can observe a sequence of system calls and interrupts throughout the DID resolving process. In turn, if OBLIVIRA does not invoke the

TABLE III
PRIVACY LEAK PREVENTION COVERAGE OF OBLIVIRA

Privacy Leaks	Correlatable Information			Temporal Information		
	DID	DID Doc Content	IP/BrowserAgent	Cache-hit (Outsider PoV)	Cache-hit (Resolver PoV)	Cache-hits to Same Entry
Resolver w/ Oblivira	✔	✔	−	✔	▲	✔
Reference Resolver	✖	✖	▲	✖	✖	✖

✔ Prevented ✖ Exposed ▲ Multifaceted issue − Orthogonal issue

resolver with an EphDID or retrieve the DID document from the blockchain network, the kernel notices the difference in the sequence of system calls. We surmise that we can almost identically emulate the behavior of a program execution path in the case of a cache miss, except for sending a real DID document request, which would induce strain on the blockchain network.

VII. RELATED WORK

A. Securing Infrastructures

Our work presents a first detailed security analysis of DID that has not been explored in prior research, and we propose a novel design that achieves confidential resolving in a non-intrusive manner. Several works have previously discussed privacy-preserving information methods in various contexts, and we present the research that is directly relevant to ours.

Privacy and DNS. Several researchers have worked on the secure resolution of DNS queries to protect user privacy. Federrath et al. [32] introduced a dedicated DNS anonymous service to protect DNS queries using a distributed architecture. In addition, Lu et al. [33] proposed a privacy-preserving DNS that uses distributed hash tables, different naming schemes, and computational privacy retrieval methods.

Securing Infrastructures With Enclaves. Approaches that use secure enclaves such as Intel SGX [34] have been adopted for coping with a stronger threat model in which only the infrastructure's processor is trusted. For instance, Lightbox [35] proposes securing middleboxes in the network using SGX, which supports secure execution of network functions on protected traffic. Also, Pdot [36] proposes a recursive DNS resolver that operates inside SGX. In Pdot, DNS clients authenticate the enclave in DNS resolvers through remote attestation. Our work is similar to the existing works that secure the infrastructure on the internet. However, to the best of our knowledge, our work is first to perform a detailed security analysis of DID and universal resolver. That is, our work begins by defining the unique security problems of the new identity management scheme and proposes specific defenses against major privacy problems.

B. Decentralized ID and User Privacy

Research on user-controlled digital identification methods has been extensively explored, before the rise of DIDs [37], [38]. However, the existing user-controlled identification methods had shortcomings; the user-controlled identity management was still federated. Recently, blockchains have been explored as a verifiable and transparent identity data registry. Several works have explored blockchain for storing and managing authentication information [39], [40], [41]. The current development of Decentralized IDentity (DID) [1], [2], [6] adapts blockchains as the primary data registry to achieve decentralized identity management, allowing users to fully control their information. The W3C is in the process of standardizing DID since November 2019 [42], and Decentralized Identity Foundation (DIF), an industry consortium, is driving the standardization and building of DID-based services [2], [4], [22].

A number of recent works on the security and privacy guarantees in decentralized identity management systems have been proposed. Pennino et al. [43] proposed a protocol that leverages two blockchain systems for proving the ownership of a DID. In their scheme, users retrieve the proofs from one blockchain to claim their DID ownership for the other blockchain system. Omar et al. [44] examined the possibility of malicious attacks on DID-based decentralized IoT device management framework and proposed a new DID method supporting mutual authentication among devices. These works aimed to protect the availability of DID-based systems through the authentication of DID. In our work, we focus on identifying and preventing the potential privacy leakage in the current DID resolving architecture. Regarding the privacy issues in DID-based systems, Halpin [45] addressed issues of storing hashed credentials that contain personal medical information on the public blockchain. Kim et al. [46] conducted a comprehensive attack surface analysis on the DID infrastructure. However, the works are limited to a high-level assessment of the possible attack vectors. Compared to those existing works, our work reports concrete security concerns in the reference resolver implementation and also presents a practical TEE-assisted design for oblivious DID resolving.

C. Confidential and Oblivious Computing in the Cloud

Computation in the cloud faces multifaceted threats, including possibly malicious cloud administrators, co-tenants, etc. Modern processor architectures include hardware support [34], [47] for secure enclaves to protect data as well as execution of sensitive programs. However, side-channel attacks on such secure enclaves have undermined the promised trust boundaries. Side-channel attacks target patterns that may appear within the system to reveal secret information inside enclaves, such as code execution or data access patterns. Oblivious computing is a direction toward retrofitting confidential computing to have no discernible patterns produced as a side effect of computation inside secure enclaves. OBLIVIRA adapts the secure enclaves and oblivious computing techniques to realize its unique oblivious DID resolver design.

Confidential Computing in the Cloud. A number of works also have proposed various confidential data computation schemes for the cloud based on secure enclaves [48], [49], [50]. [49] adapted SGX to perform MapReduce in the cloud without exposing code and data to the cloud. [50] implements a database that uses SGX for protecting the data and querying process. Also, various cloud service providers now provide an enclave-enabled computation resources [51], [52].

Side-Channel Attacks on Secure Enclaves. Recent works have revealed that there exist side-channels that allow an adversary to extract protected information inside enclaves. [11] outlines memory-based side-channel attacks on Intel SGX. Controlled side-channel attacks [53], [54] exploit the interactions between the enclave and the untrusted kernel since SGX depends on the kernel for managing the system resources. For instance, page faults within the enclave can be abused to profile the memory access pattern of the enclave-protected program [13]. Cache side-channels can also affect secure enclaves, as explained in previous research [55]. Recent works have studied information leakage by the side-channel attack on secure enclaves. The classification of side-channel attacks on SGX varies, as the attack vectors are scattered on a wide range of potential attack surfaces. Attacks within monitoring page faults [25], timing interrupts [56], and exploiting memory segmentation features [57] are proposed. There are also several cache timing attacks against SGX [11], [27] using general side-channel attack techniques(e.g., Flush+Reload [58]). In addition, some micro-architectural attacks such as branch prediction attacks [59], speculative execution attacks [60], [61], rogue data cache load [62], and data sampling attacks [63] are proposed recently.

Oblivious Computing. To prevent side-channel attacks on secure enclaves, ORAM [64] is widely considered as a defense. ORAM hides memory access patterns from OS and hypervisor by continuously remapping and re-encrypting the data accessed [20], thereby preventing side-channel attacks by monitoring memory. As recent ORAM schemes [16], [19], [20] have significantly improved the performance, a wide range of applications aiming at oblivious services have been proposed lately. Prominent research works on ORAM include hiding code execution pattern [65], applying ORAM in the secure enclaves [15], [30], implementing an oblivious file system in the enclave [66], and ORAM application in the cloud computing [14], [67]. In particular, we applied ZeroTrace [30] for OBLIVIRA's OCache, which is an ORAM implementation inside SGX that can mitigate almost every attack vector, including controlled channel attacks. Moreover, other methodologies [68] have also been proposed for the mitigation of side-channel attacks within the enclave [67].

VIII. LIMITATIONS AND FUTURE WORK

Driver Compatibility. We successfully ported 10 different DID drivers to implement and test with OBLIVIRA. However, we found that some drivers may require more effort than others. More specifically, DID drivers for blockchains that do not have built-in support for DID (e.g., Bitcoin) often do not produce a straightforward DRF. They tend to retrieve information by performing complex operations and making multiple interactions with the blockchain server, then mold them into a DID document format. However, the typical drivers are seldom complex, and we expect that we can either port them to work with OBLIVIRA, or simply embed the driver's logic inside OBLIVIRA's enclave.

Oblivious Resolving in DID Standard. We wish to make our implementation public after the publication of the work in the hopes of aiding in stronger privacy considerations to be included in the W3C DID standard. Our implementation can be readily deployed with the reference universal resolver, and we expect that many service providers can easily integrate our design into their services. Our work leverages Intel SGX to enable oblivious DID resolving. Although the technology is only available on many Intel processors, we expect that many service providers can deploy their services using SGX-enabled cloud services such as Azure Confidential Computing.

Secure Time Measurements. Our uniform response time scheme requires a secure timer inside SGX. SGX version 1.0 does not support a trusted time source, and we had to resort to a `OCALL` that exits the enclave to measure elapsed time through `rdtsc`. A previous work points out the issue and proposes a solution [69]. However, the solution utilizes the *System Management Mode (SMI)* presented in x86 processors, and we concluded that it is not a portable solution for our design. Hence, we did not include it in our prototype. This limitation is expected to be resolved with SGX 2.0 by in-enclave support for `rdtsc` instruction.

Securing the Entire DID Resolver Hierarchy. We plan to investigate and apply the principles and design presented in this work to other systems involved in the DID resolving hierarchy. The universal resolver is analogous to the root name servers in the DNS system. Often the universal resolver forwards the DID request to the vendor-run DID resolving infrastructure that directly interacts with the blockchain network. However, we argue that the amount of information gathered at such leaf resolvers is relatively small, compared to that can be collected at the universal resolver. Considering that the universal resolver already supports nearly 50 different drivers at this point, it is reasonable to apply our design in the universal resolver as a precursory design proposal for the oblivious DID resolving mechanism. Also, we expect that our design can be readily applied to the leaf resolvers in a recursive manner.

Contributing to Standardization Efforts. We have already contacted the developers of DID resolvers that employ DID document caching. Along with the publication of our work, we wish to make our work available to the public. We hope that our design can be a reference to the developers of DID resolver and connected services. Also, we plan to introduce the high-level concepts of our design, such as the use of a trusted agent to mediate the resolving process and uniform response time caching, to the standardization efforts. Also, we plan to open-source our prototype so that it can be used as a reference for implementing a privacy-preserving universal resolver.

IX. CONCLUSION

In this article, we presented an extensive security analysis of the current state of DID standard implementations and our

privacy-preserving design called OBLIVIRA that enforces the universal resolver to resolve DIDs oblivious of their contents. Our security analysis provided systemized privacy issues regarding the correlation of information and temporal side-channels in the universal resolvers. We also demonstrated the side-channels through our attack examples on real-world DID services. We proposed OBLIVIRA, a small SGX-based agent that protects DIDs and DID documents, which effectively eliminates the side-channels that we discovered. Also, we evaluated the OBLIVIRA prototype and showed its efficacy in terms of necessary code modifications to the resolver and minimal performance overhead. The performance overhead was around 4.3% in the worst case in terms of DID requests resolved per minute and about 4% increase in response time. Also, we evaluated the security of OBLIVIRA against the privacy issues that we discovered. We demonstrated that our design prevents the correlation of information by enforcing the universal resolver to process DID requests oblivious of DID and DID document, and our oblivious caching scheme eliminated the side-channel that existed in the reference resolver. We hope that our research can contribute to the ongoing security discussion on the DID standardization efforts, and we plan to release our OBLIVIRA source code to the DID community to better protect users and services.

REFERENCES

[1] DIF, "Together we're building a new identity ecosystem," 2021. Last Accessed: Feb. 20, 2021. [Online]. Available: https://identity.foundation

[2] DIF, "ION:ION is an open, public, permissionless layer 2 decentralized identifier network that runs atop the bitcoin blockchain," 2021. Last Accessed: Feb. 25, 2021. [Online]. Available: https://identity.foundation/ion/

[3] serto, "Data meets identity," 2021. Last Accessed: Apr. 28, 2021. [Online]. Available: https://www.serto.id

[4] Sovrin, "Home - sovrin," 2021. Last Accessed: Feb. 25, 2021. [Online]. Available: https://sovrin.org/

[5] IBM, "IBM verify credentials:transforming digital identity into decentralized identity," 2021. Last Accessed: Feb. 27, 2021. [Online]. Available: https://www.ibm.com/blockchain/solutions/identity

[6] ERCIM MIT, Keio, and Beihang, "Decentralized identifiers (DIDs) v1.0: Core architecture, data model, and representations," 2021. Last Accessed: Feb. 25, 2021. [Online]. Available: https://www.w3.org/TR/did-core/

[7] D. I. Foundation, "Universal resolver," 2021. Last Accessed: Mar. 30, 2021. [Online]. Available: https://github.com/decentralized-identity/universal-resolver

[8] F. McKeen et al., "Innovative instructions and software model for isolated execution," in *Proc. 2nd Workshop Hardware Architectural Support Secur. Privacy*, 2013, Art. no. 10. [Online]. Available: https://doi.org/10.1145/2487726.2488368

[9] ARM, "Arm trustzone technology," 2021, Last Accessed: Apr. 2, 2021. [Online]. Available: https://developer.arm.com/ip-products/security-ip/trustzone

[10] I. Corporation, "Strengthen enclave trust with attestation," Last Accessed: Jun. 28, 2022. [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/attestation-services.html

[11] W. Wang et al., "Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 2421–2434. [Online]. Available: https://doi.org/10.1145/3133956.3134038

[12] S. P. Johnson, "Intel sgx and side-channels," 2018. Accessed: Jul. 21, 2021. [Online]. Available: https://software.intel.com/content/www/us/en/develop/articles/intel-sgx-and-side-channels.html

[13] J. V. Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx, "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 1041–1056. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/van-bulck

[14] S. Lai et al., "OblivSketch: Oblivious network measurement as a cloud service," in *Proc. 28th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2021. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/oblivsketch-oblivious-network-measurement-as-a-cloud-service/

[15] A. Ahmad, B. Joe, Y. Xiao, Y. Zhang, I. Shin, and B. Lee, "OBFUSCURO: A commodity obfuscation engine on Intel SGX," in *Proc. 26th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2019. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/obfuscuro-a-commodity-obfuscation-engine-on-intel-sgx/

[16] X. Wang, H. Chan, and E. Shi, "Circuit ORAM: On tightness of the goldreich-ostrovsky lower bound," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, 2015, pp. 850–861. [Online]. Available: https://doi.org/10.1145/2810103.2813634

[17] X. S. Wang, Y. Huang, T.-H. H. Chan, A. Shelat, and E. Shi, "SCORAM: Oblivious RAM for secure computation," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 191–202. [Online]. Available: https://doi.org/10.1145/2660267.2660365

[18] X. S. Wang et al., "Oblivious data structures," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2014, pp. 215–226. [Online]. Available: https://doi.org/10.1145/2660267.2660314

[19] L. Ren et al., "Constants count: Practical improvements to oblivious RAM," in *Proc. 24th USENIX Secur. Symp.*, 2015, pp. 415–430. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/ren-ling

[20] E. Stefanov et al., "Path ORAM: An extremely simple oblivious RAM protocol," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 299–310. [Online]. Available: https://doi.org/10.1145/2508859.2516660

[21] W. P. Rachel Lerman, "Vaccine passport apps are here. but the technical challenges are still coming," 2021. Last Accessed: Apr. 2, 2021. [Online]. Available: https://www.washingtonpost.com/technology/2021/04/02/vaccine-passports-apps-faq/

[22] serto, "Home - uport," 2021. Last Accessed: Feb. 25, 2021. [Online]. Available: https://www.serto.id/

[23] P. Thakkar, S. Nathan, and B. Viswanathan, "Performance benchmarking and optimizing hyperledger fabric blockchain platform," in *Proc. 26th IEEE Int. Symp. Model. Anal. Simul. Comput. Telecommun. Syst.*, 2018, pp. 264–276. [Online]. Available: https://doi.org/10.1109/MASCOTS.2018.00034

[24] S. S. Hazari and Q. H. Mahmoud, "A parallel proof of work to improve transaction speed and scalability in blockchain systems," in *Proc. IEEE 9th Annu. Comput. Commun. Workshop Conf.*, 2019, pp. 916–921. [Online]. Available: https://doi.org/10.1109/CCWC.2019.8666535

[25] J. V. Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, and R. Strackx, "Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution," in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 1041–1056. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/van-bulck

[26] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller, "Cache attacks on Intel SGX," in *Proc. 10th Eur. Workshop Syst. Secur.*, 2017, pp. 1–6.

[27] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A. Sadeghi, "Software grand exposure: SGX cache attacks are practical," in *Proc. 11th USENIX Workshop Offensive Technol.*, 2017, Art. no. 11. [Online]. Available: https://www.usenix.org/conference/woot17/workshop-program/presentation/brasser

[28] wolfSSL, "Embedded TLS library for applications, devices, IoT, and the cloud," 2021. Last Accessed: May 1, 2021. [Online]. Available: https://www.wolfssl.com

[29] I. L. C. S. R. Projects, "cloud-security-research/sgx-ra-tls," 2021. Last Accessed: Apr. 28, 2021. [Online]. Available: https://github.com/cloud-security-research/sgx-ra-tls

[30] S. Sasy, S. Gorbunov, and C. W. Fletcher, "ZeroTrace: Oblivious memory primitives from Intel SGX," in *Proc. 25th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2018. [Online]. Available: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_02B-4_Sasy_paper.pdf

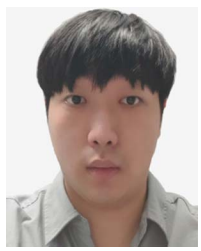
[31] R. Dingledine, N. Mathewson, and P. F. Syverson, "TOR: The second-generation onion router," in *Proc. 13th USENIX Secur. Symp.*, 2004, pp. 303–320. [Online]. Available: http://www.usenix.org/publications/library/proceedings/sec04/tech/dingledine.html

[32] H. Federrath, K.-P. Fuchs, D. Herrmann, and C. Piosecny, "Privacy-preserving DNS: Analysis of broadcast, range queries and mix-based protection methods," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2011, pp. 665–683.

[33] Y. Lu and G. Tsudik, "Towards plugging privacy leaks in the domain name system," in *Proc. IEEE 10th Int. Conf. Peer-to-Peer Comput.*, 2010, pp. 1–10. [Online]. Available: https://doi.org/10.1109/P2P.2010.5569976

[34] Intel, "Intel software guard extensions," 2021. [Online]. Available: https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html
[35] H. Duan, C. Wang, X. Yuan, Y. Zhou, Q. Wang, and K. Ren, "LightBox: Full-stack protected stateful middlebox at lightning speed," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 2351–2367. [Online]. Available: https://doi.org/10.1145/3319535.3339814
[36] Y. Nakatsuka, A. Paverd, and G. Tsudik, "PDoT: Private DNS-over-TLS with tee support," *Digit. Threats: Res. Pract.*, vol. 2, no. 1, Mar. 2021, Art. no. 3. [Online]. Available: https://doi.org/10.1145/3431171
[37] J. G. Faísca and J. Q. Rogado, "Decentralized semantic identity," in *Proc. 12th Int. Conf. Semantic Syst.*, 2016, pp. 177–180. [Online]. Available: https://doi.org/10.1145/2993318.2993348
[38] D. J. Weitzner, "Whose name is it, anyway? Decentralized identity systems on the web," *IEEE Internet Comput.*, vol. 11, no. 4, pp. 72–76, Jul./Aug. 2007.
[39] X. Zhu and Y. Badr, "Identity management systems for the Internet of Things: A survey towards blockchain solutions," *Sensors*, vol. 18, no. 12, 2018, Art. no. 4215.
[40] M. Lücking, C. Fries, R. Lamberti, and W. Stork, "Decentralized identity and trust management framework for Internet of Things," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency*, 2020, pp. 1–9. [Online]. Available: https://doi.org/10.1109/ICBC48266.2020.9169411
[41] H. Shafagh, L. Burkhalter, S. Duquennoy, A. Hithnawi, and S. Ratnasamy, "Droplet: Decentralized authorization for IoT data streams," 2018, *arXiv:1806.02057*. [Online]. Available: http://arxiv.org/abs/1806.02057
[42] ERCIM MIT, Keio, and Beihang, "Decentralized identifiers (DIDs) v1.0: Core architecture, data model, and representations," 2021. Last Accessed: Feb. 25, 2021. [Online]. Available: https://www.w3.org/TR/2019/WD-did-core-20191107/
[43] D. Pennino, M. Pizzonia, A. Vitaletti, and M. Zecchini, "Binding of endpoints to identifiers by on-chain proofs," in *Proc. IEEE Symp. Comput. Commun.*, 2020, pp. 1–6.
[44] A. Sghaier Omar, "Decentralized identity and access management framework for Internet of Things devices," 2020.
[45] H. Halpin, "Vision: A critique of immunity passports and W3C decentralized identifiers," in *Proc. Int. Conf. Res. Secur. Standardisation*, 2020, pp. 148–168.
[46] B. G. Kim, Y.-S. Cho, S.-H. Kim, H. Kim, and S. S. Woo, "A security analysis of blockchain-based did services," *IEEE Access*, vol. 9, pp. 22894–22913, 2021.
[47] A. Limited, "Building a secure system using trustzone technolog," 2009. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf
[48] S. Brenner et al., "SecureKeeper: Confidential zookeeper using Intel SGX," in *Proc. 17th Int. Middleware Conf.*, 2016, Art. no. 14. [Online]. Available: http://dl.acm.org/citation.cfm?id=2988350
[49] F. Schuster et al., "VC3: Trustworthy data analytics in the cloud using SGX," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 38–54. [Online]. Available: https://doi.org/10.1109/SP.2015.10
[50] C. Priebe, K. Vaswani, and M. Costa, "EnclaveDB: A secure database using SGX," in *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 264–278. [Online]. Available: https://doi.org/10.1109/SP.2018.00025
[51] M. Azure, "Azure confidential computing," 2021. Last Accessed: Apr. 28, 2021. [Online]. Available: https://azure.microsoft.com/en-us/solutions/confidential-compute
[52] G. Cloud, "Confidential computing," 2021. Last Accessed: Apr. 28, 2021. [Online]. Available: https://cloud.google.com/confidential-computing
[53] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *Proc. IEEE Symp. Secur. Privacy*, 2015, pp. 640–656. [Online]. Available: https://doi.org/10.1109/SP.2015.45
[54] J. Van Bulck, F. Piessens, and R. Strackx, "SGX-step: A practical attack framework for precise enclave execution control," in *Proc. 2nd Workshop Syst. Softw. Trusted Execution*, 2017, Art. no. 4. [Online]. Available: https://doi.org/10.1145/3152701.3152706
[55] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "CacheZoom: How SGX amplifies the power of cache attacks," in *Proc. 19th Int. Conf. Cryptographic Hardware Embedded Syst.*, 2017, pp. 69–90. [Online]. Available: https://doi.org/10.1007/978--3-319-66787-4_4
[56] J. V. Bulck, F. Piessens, and R. Strackx, "Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 178–195. [Online]. Available: https://doi.org/10.1145/3243734.3243822
[57] J. Gyselinck, J. V. Bulck, F. Piessens, and R. Strackx, "Off-limits: Abusing legacy x86 memory segmentation to spy on enclaved execution," in *Proc. 10th Int. Symp. Eng. Secure Softw. Syst.*, 2018, pp. 44–60. [Online]. Available: https://doi.org/10.1007/978--3-319-94496-8_4
[58] Y. Yarom and K. Falkner, "FLUSH RELOAD: A high resolution, low noise, L3 cache side-channel attack," in *Proc. 23rd USENIX Secur. Symp.*, 2014, pp. 719–732. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom
[59] D. Evtyushkin, R. Riley, N. CSE and E. C. E. Abu-Ghazaleh, and D. Ponomarev, "BranchScope: A new side-channel attack on directional branch predictor," *ACM SIGPLAN Notices*, vol. 53, no. 2, pp. 693–707, 2018.
[60] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. Lai, "SgxPectre: Stealing Intel secrets from SGX enclaves via speculative execution," *IEEE Secur. Privacy*, vol. 18, no. 3, pp. 28–37, May/Jun. 2020. [Online]. Available: https://doi.org/10.1109/MSEC.2019.2963021
[61] E. M. Koruyeh, K. N. Khasawneh, C. Song, and N. B. Abu-Ghazaleh, "Spectre returns! speculation attacks using the return stack buffer," in *Proc. 12th USENIX Workshop Offensive Technol.*, 2018, Art. no. 3. [Online]. Available: https://www.usenix.org/conference/woot18/presentation/koruyeh
[62] J. V. Bulck et al., "Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution," in *Proc. 27th USENIX Secur. Symp.*, 2018, pp. 991–1008. [Online]. Available: https://www.usenix.org/conference/usenixsecurity18/presentation/bulck
[63] C. Canella et al., "Fallout: Leaking data on meltdown-resistant CPUs," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 769–784. [Online]. Available: https://doi.org/10.1145/3319535.3363219
[64] O. Goldreich, "Towards a theory of software protection and simulation by oblivious RAMs," in *Proc. 19th Annu. ACM Symp. Theory Comput.*, 1987, pp. 182–194. [Online]. Available: https://doi.org/10.1145/28395.28416
[65] A. Rane, C. Lin, and M. Tiwari, "Raccoon: Closing digital side-channels through obfuscated execution," in *Proc. 24th USENIX Secur. Symp.*, 2015, pp. 431–446. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/rane
[66] A. Ahmad, K. Kim, M. I. Sarfaraz, and B. Lee, "OBLIVIATE: A data oblivious filesystem for Intel SGX," in *Proc. 25th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2018. [Online]. Available: http://wp.internetsociety.org/ndss/wp-content/uploads/sites/25/2018/02/ndss2018_06A-2_Ahmad_paper.pdf
[67] O. Ohrimenko et al., "Oblivious multi-party machine learning on trusted processors," in *Proc. 25th USENIX Secur. Symp.*, 2016, pp. 619–636. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko
[68] F. Brasser et al., "DR.SGX: Automated and adjustable side-channel protection for SGX using data location randomization," in *Proc. 35th Annu. Comput. Secur. Appl. Conf.*, 2019, pp. 788–800. [Online]. Available: https://doi.org/10.1145/3359789.3359809
[69] H. Liang and M. Li, "Bring the missing jigsaw back: Trustedclock for SGX enclaves," in *Proc. 11th Eur. Workshop Syst. Secur.*, 2018, Art. no. 8. [Online]. Available: https://doi.org/10.1145/3193111.3193119

Siwon Huh received the BS degree in mathematics and computer science from Sungkyunkwan University, in 2021. He is currently working toward the master's degree in computer science and engineering with Sungkyunkwan University, Suwon, Korea. His research interests include blockchain identity management and secure machine learning computation.



Myungkyu Shim received the BS degree in computer science from Sungkyunkwan University, in 2021. He is currently working toward the MS degree with the Dept. of Computer Science and Engineering, Sungkyunkwan University, Korea. His current research interests include trusted execution environments and software security.

Jihwan Lee received the BS degree in computer science from Sungkyunkwan University, in 2021. He is currently working toward the MS degree with the Dept. of Computer Science and Engineering, Sungkyunkwan University, Korea. His main research interests are trusted execution environments and program analysis.

Simon S. Woo received the BS degree in electrical engineering from the University of Washington (UW), Seattle, the MS degree in electrical and computer engineering from the University of California, San Diego (UCSD), and the MS and PhD degrees in computer science from the University of Southern California (USC), Los Angeles. He was a member of technical staff (technologist) for 9 years with the NASA's Jet Propulsion Lab (JPL), Pasadena, California, conducting research in satellite communications, networking and cybersecurity areas. Also, He worked with Intel Corp. and Verisign Research Lab. Since 2017, he was a tenure-track Assistant Professor with SUNY, South Korea and a research Assistant Professor with Stony Brook University. Now, he is a tenured-track assistant professor with the SKKU Institute for Convergence and Department of Applied Data Science and Software in Sungkyunkwan University, Suwon, Korea.

Hyoungshick Kim received the BS degree from the Department of Information Engineering, Sungkyunkwan University, the MS degree from the Department of Computer Science, KAIST, and the PhD degree from the Computer Laboratory , 'The University of Cambridge, in 1999, 2001, and 2012, respectively. He is an associate professor with the Department of Computer Science and Engineering, College of Software, Sungkyunkwan University. He is also working as a distinguished visiting researcher with CSIRO Data61. After completing his PhD, he worked as a post-doctoral fellow with the Department of Electrical and Computer Engineering, University of British Columbia. He previously worked for Samsung Electronics as a senior engineer from 2004 to 2008. He also served as a member of DLNA and Coral standardization for DRM interoperability in home networks. His current research interest is focused on usable security, blockchain, security vulnerability analysis, and data-driven security.

Hojoon Lee received the BS degree from The University of Texas at Austin, and the PhD degree from KAIST, in 2018, advised by Prof. Brent Byunghoon Kang. He is currently an assistant professor with the Department of Computer Science and Engineering, Sungkyunkwan University since September 2019. Prior to his current position, he spent one year as a postdoctoral researcher with CISPA under the supervision of Prof. Michael Backes. His main research interests lie in retrofitting security in computing systems against today's advanced threats. His research interests include but are not limited to operating system security, trusted execution environments, program analysis, software security, and secure AI computation in cloud.