Dont get caught, keep your Onions in a Vault

Humza Ikram*
Carnegie Mellon University, Pennsylvania, US

Rumaisa Habib*
Stanford University, California, US

Muaz Ali*
University of Arizona, Arizona, US

Zartash Afzal Uzmi
LUMS University, Pakistan

Abstract

When web applications wish to operate anonymously, they routinely host themselves as 'Hidden Services' in the Tor network. However, these services are frequently threatened by deanonymization attacks, whereby their IP address and location may be inferred by the authorities. We present `VaulTor`, a novel architecture for the Tor network that ensures an extra layer of security for the Hidden Services against deanonymization attacks. In this new architecture, a volunteer (vault) is incentivized to host the web application content on behalf of the Hidden Service. The vault runs the hosted application in a Trusted Execution Environment (TEE) and becomes the point of contact for interested clients. This setup can substantially reduce the uptime requirement of the original Hidden Service provider, thereby significantly decreasing the chance of deanonymization attacks against them. Using a vault node in place of the hidden service node does not cause any noticeable performance degradation when accessing the hosted content.

2012 ACM Subject Classification Security and privacy → Pseudonymity, anonymity and untraceability

Keywords and phrases Tor, anonymity, Hidden Services, Trusted Execution Environments

Digital Object Identifier 10.4230/OASIcs.NINeS.2026.17

Funding *Rumaisa Habib**: Stanford Graduate Fellowship

1 Introduction

The enormous expansion of the world wide web is coupled with growing demands for anonymity and privacy. Besides a huge end-user client base, an increasing number of web services—legal and illegal—also choose to remain anonymous [2], fearing closure, or even prosecution, by the government and law enforcement agencies [8]. The Onion Router (Tor) [5] network has emerged as one of the most popular solutions for providing anonymity: nearly 3 million clients connect to Tor daily [9], and hundreds of thousands of anonymized web addresses are published each day, with over 150,000 currently serving traffic to end users [17].

Tor Hidden Services (aka. Onion services) aim to uphold freedom of speech in repressive regimes and offer circumvention in regions of undue and excessive internet censorship, thus bringing benefits to the public [58]. At the same time, Hidden Services pave the way for criminal activities such as selling illegal drugs and weapons [28]. All in all, there are incentives for governments and law enforcement to deanonymize Hidden Service (HS) providers[1] and curb their operations. In 2014, the authorities of 6 European countries and the United

* These authors contributed equally.
[1] Hidden Service Provider is an individual that owns the Hidden Service. They also create and serve the content of the Hidden Service.

States collaborated in *Operation Onymous* through which they cracked down on 416 Hidden Services, accused them of foul play, and arrested 17 individuals [8].

We focus on the *technical* challenges of reinforcing the anonymity of hidden service providers. We present `VaulTor`, a novel overlay architecture that builds upon Tors existing HS design by introducing a *vault* node between the HS provider and clients. The HS provider offloads service operation to the vault, which serves clients on the providers behalf. This setup strengthens the anonymity of HS providers in three key ways:

1. After offloading their service, the HS provider may sporadically connect to the vault to update content as and when needed, without the need to be online all the time. This significantly reduces the attack surface against long-running deanonymization attacks on the HS provider.
2. The HS provider may connect to the vault from widely varying locations, further reducing deanonymization risk.
3. An adversary cannot make an on-demand connection with an HS provider. Communication opportunities with the HS provider are now solely available to the vault, and even those channels are initiated by the HS provider.

The service offloaded by the HS provider to the vault is hosted inside a Trusted Execution Environment (TEE), such as the one provided by Intel-SGX [40]. Using a TEE brings two additional benefits: (i) the vault owner (even with root access) cannot snoop on the application code/data while it is being uploaded, served, or stored in the vault, and (ii) since client and the HS provider contact the vault through similar encrypted channels, the vault is never sure if it is an HS provider or a client it is communicating with, ruling out a myriad of attacks against the HS provider (details in Sec 9.1).

The `VaulTor` architecture, which has no bearing on the client anonymity (discussed in Sec 8.2), promises enhanced anonymity for the HS provider by shifting the deanonymization risk to the vault owner (see Sec 8.3 for details). This design choice is suitable as the HS providers may only be interested in content creation without taking the deanonymization risk that leaps up when serving the content, even as a Tor Hidden Service. In contrast, the vault owners are willing to serve the content in exchange for monetary benefits (such as crypto payments[2]) or social incentives (serving content they wish to support and propagate). A motivating example for our scenario might be a journalist in an oppressive regime who can anonymously upload content banned in their regime to a vault located in a neutral country. Another example may be providing 'on-the-ground' information from within a region where internet outages frequently occur (and the HS provider cannot remain online for extended periods).

`VaulTor` offers various attractive features not ubiquitous in alternate design choices: (a) allows for hosting dynamic services, in contrast to data hosting services such as IPFS or `pastebin.com`, (b) offers content isolation from the server administrator, in contrast to a simple (non-TEE based) virtual machine-based approach, and (c) ensures that the server administrator cannot snoop on private keys in order to serve arbitrary content on behalf of the HS provider.

`VaulTor` also brings robustness to the HS operation. If a vault node is shut down by the action of authorities, the HS provider can use a different one to make the content available. Alternatively, if the HS provider goes offline permanently, the TEE can continue hosting the content as long as the vault owner remains incentivized.

In `VaulTor`, the vault simply replaces and acts on behalf of the HS provider, responding

[2] We discuss a secure way to do crypto transactions in Section 10.3

to clients through network paths containing the same number of Tor nodes as if the content is served by the HS provider in the existing architecture. Thus, the VaulTor design bears no negative impact on the network latency. We do observe a slight average increase in the time to first byte and time to last byte (a maximum of 5.7% and 2.9% respectively when using Intel-SGX as the TEE), attributed to hosting the content inside a TEE which incurs its own performance overhead (9.2.2).

To understand the operation of VaulTor, we note that a typical Tor circuit is an overlay path through three volunteer relay nodes (the entry guard node, a middle node and the exit node) on the Tor network [32]. Such a circuit provides *one-way anonymity* to a client trying to connect via the circuit to any non-anonymous server on the web. If the server wants to remain anonymous as well, it must also choose another Tor circuit with three nodes [24]. The existing HS architecture anonymizes both the client and the server (two-way anonymity) using a mechanism that stitches the two Tor circuits together (see Sec 2.2). With the VaulTor architecture, a server that desires to be anonymous is facilitated to replicate, and infrequently update, its web content, and service on the vault. This is similar to a CDN node offering content replication to a server. The two-way anonymity continues to exist between the vault node and the client, delivering similar delay performance as observed in the existing hidden service architecture. Indeed, our results in Sec 9.2.2 confirm this. We also consider a sample of popular deanonymization attacks and provide an outline of how VaulTor offers enhanced HS provider anonymity under those attacks (see Sec 9.1). We also discuss legal and deployability considerations for both HS providers and vault operators in Sec 10.5 Altogether, this paper makes the following contributions:

- A novel architecture of Tor nodes, used by VaulTor to provide robust anonymity to host Hidden Services (HS).
- Step-wise description of the protocol VaulTor uses to ensure anonymity of the HS provider, vault, and the clients.
- A thorough security analysis and description of the attacks that are mitigated by VaulTor.
- A working prototype and its performance measurement over the actual Tor network. The prototype is available at: https://github.com/RumaisaHabib/vaultor

2 Background

2.1 Trusted Execution Environments (TEEs)

TEEs provide a platform for *secure remote computation* which allows *securely* executing an application in a remote *untrusted* system without compromising the *Integrity* and *Confidentiality* of the application data. Several hardware architectures provide TEE implementations [40, 31, 16, 48]. Our prototype implementation leverages the TEE provided by Intel-SGX [40] to create a secure execution channel between the vault and the HS provider. We host an HS inside the TEE which itself is set-up inside a vault (Sec 7). An instance of a program running inside a TEE is called an enclave.

TEEs provide many guarantees. These include:

1. **Sealing:** A program running inside a TEE can encrypt and write data to the disk for persistent storage. Only the *same program* running on *the same device* in a TEE can decrypt this data.
2. **Isolation:** A program inside a TEE can not access the memory of its host and vice versa.
3. **Remote Attestation:** A piece of code running inside a TEE can prove, to an outside observer, what piece of code it is and that it is running inside a TEE.

A local trusted device may use the following *simplified flow* for remote attestation to verify that a remote untrusted device is running the desired piece of code:

1. The local device builds the code and gets a measurement of the memory space[3].
2. The local device sends the code to the remote device.
3. The remote device runs this code inside a TEE.
4. When running inside a TEE, the code can then request the CPU to generate a cryptographic hash of the program's memory space. This hash is then signed by the CPU using a secret hardware attestation key embedded in the CPU.
5. The local device can request this signed measurement from the remote device. It then verifies this signature by matching the measurement hash against one computed locally and by verifying the signature against its public key.

We will stick to Intel-SGX terminology, in which the signed cryptographic hash is called a *quote*. Furthermore, it should be noted that a small amount of arbitrary data (produced by the code inside the TEE) can be embedded in the quote as well. This data is called the REPORTDATA field. Different TEE implementations employ different means for signature verification. Intel-SGX, for example, requires sending the quote to Intel's online Attestation Service which verifies that the quote is valid. The local device also has the option to perform this attestation via a proxy (such as Tor).

The REPORTDATA field is crucial for establishing a secure connection with an enclave (program running within the TEE). This field routinely contains a public key whose corresponding private key is known only to the enclave. This public key can then be used to create a secure connection with the enclave using any form of key exchange such as Diffie-Hellman.

2.2 Conventional Tor architecture

In the current Tor protocol, there are 6 Tor nodes between a client and a hidden service (HS). This ensures two-way anonymity, i.e., both the client accessing the HS and the HS provider itself remain anonymous. Hidden services are identified using an onion URL. Figure 1 and the following points detail the protocol for the establishment of communication between an HS and a client in the *current* architecture.

1. An HS provider contacts a relay and asks them to act as an Introduction Point (IN)[4]. The HS provider receives an acknowledgment from the IN.
2. The HS provider creates an "Onion service descriptor" which includes the public key for the HS and the IP addresses of its INs. This descriptor is signed by the public key of the HS. The HS provider sends this to a HSDir. The HS provider gets an acknowledgement from one of many Hidden Service Directories (HSDirs)[5].
3. A client asks the HSDir for the service descriptor of the HS. They receive and verify the signature of the HS.
4. The client picks a Tor relay to act as a Rendezvous Point (RP) and establishes a Tor circuit to it. The client gives the RP a Rendezvous cookie.
5. The client sends the same Rendezvous cookie and the IP address of the RP to an IN.
6. The IN forwards the cookie and Rendezvous address to the HS provider.
7. The HS provider makes a Tor circuit with the Rendezvous Point and sends the cookie.
8. The Rendezvous Point compares the two cookies and, if they match, relays communication from both sides to each other.

[3] This includes the code itself, the data, the stack, and the heap.
[4] IN is chosen to be distinct from Internet Protocol (IP)
[5] HSDirs are special relays that store and provide hidden service descriptors to the clients.

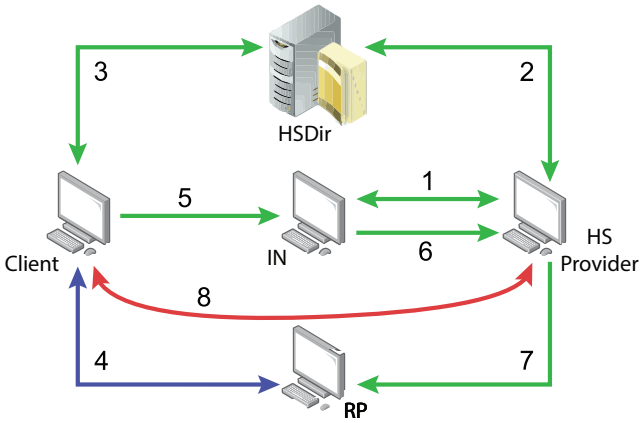


Figure 1 *The current implementation of hidden services. Red paths represent information flow through Tor circuits with 6 nodes. Green paths represent information flow through Tor circuits with three nodes. Blue paths represent information flow through Tor circuits with only two nodes.*

3 Related work

There are two notable prior works that leverage TEE technology to improve Tor. In SGX-Tor [45], different nodes (such as Directory authority, HSDir, and middle nodes) run the Tor software in TEEs so that they cannot be modified to collude and launch deanonymization attacks. In contrast, `VaulTor` reduces the deanonymization threats by not requiring the HS provider to be online. Furthermore, `VaulTor` is an architectural solution that does not require any Tor node to run their service within TEE (unless they volunteer to be a vault), thus maintaining backward compatibility for Tor nodes.

Another approach, SmarTor [14], utilizes TEEs and smart contracts to decentralize directory authorities (which are currently a few trusted and centralized servers) in the Tor network. The authors argue that this would increase the security of the Tor network as it would reduce the need to trust particular servers and make it more difficult for governmental authorities to crack them down. `VaulTor`, on the other hand, leverages the privacy guarantees provided by the TEEs to create trust by allowing a separate, new entity (the vault) to host web application content, thus making it harder to deanonymize the Hidden Service provider.

Additional prior works [7, 20, 55, 62] have also suggested modifications to the Tor network to enhance HS anonymity. Sec 10.1 discusses these solutions and their potential use cases. There exists a whole body of work aiming to improve the Tor network in general. Security improvements include a layered mesh topology [33] for circuit formation and prevention of route capture attacks [51]. There has also been work to improve client-side stability in censored regions using WebRTC [25, 35, 19].

Performance improvements include optimizing the Tor path selection algorithm [12, 15, 59, 65] and load balancing techniques [13, 30, 44]. In particular, CenTor [17] considers serving content from multiple CDN-like content replication nodes to improve the content delivery time. These improvements can be applied in conjunction with `VaulTor` to further optimize the anonymity network (see Sec 10.1).

4 Design Goals

Our design goals are meant to ensure enhanced anonymity for HS-provider, while ensuring that anonymity guarantees for other parties (i.e., client, vault node) remain intact.

We will outline our design goals succinctly here:

1. Ensure that the anonymity guarantees for the HS provider in `VaulTor` are *at least* as strong as an HS provider in the conventional Tor architecture.
2. Minimize the uptime for the HS provider, shrinking the attack surface against them.
3. Ensure that the anonymity guarantees for the newly introduced Vault node are *at least* as strong as those for an HS provider in the conventional Tor architecture.
4. Ensure that the anonymity guarantees for the client remain *at least* as strong as those for a client in the conventional Tor architecture.
5. Ensure that the HS provider retains full control of any content that is being served on its behalf.
6. Ensure that the system is able to serve rich content (not just limited to static content).
7. Ensure that the system is backward compatible and is built using off-the-shelf components and technology.

5 Alternate hosting options

The `VaulTor` architecture meets all our design goals specified in Sec 4. Other simple alternatives for anonymous content hosting may not satisfy this requirement. This section considers two such possible alternatives and describes why they are unable to achieve the full set of our design goals.

5.1 Using a static content hosting service

An HS provider may upload static content to a data hosting service like an IPFS. While it is possible that the HS provider is able to upload content securely, this content can not be modified dynamically and, thus, fails to meet our fifth design goal. For example, an HS provider trying to a run a forum online will not be able to get posts or comments by clients. `VaulTor` permits dynamic content to be served which allows the HS provider to host complicated websites on an external server.

5.2 Hosting on the cloud

An HS provider may host data dynamically on a virtual machine present on an external server (or a public cloud). They may use a remote login tool such as SSH (and the Tor proxy) to anonymously access the server and upload their content. However, the administrator of the server (with root access, or the cloud operator) may be able to access and modify this data even if the HS provider wishes to restrict access. This means the provider does not maintain full control over the content, violating the fourth design goal.

In `VaulTor`, the administrator cannot access data that is present inside the TEE. This data exists either in an encrypted manner on the disk or is only accessible to the TEE if the data is in the main memory. Lastly, as both the client and HS provider make an encrypted channel with the TEE, content is safe while in transit.

An HS provider that uploads their content to an external server (via SSH or through other means) is not guaranteed that this content will be served without modification. The administrator of this server may modify this content (while keeping the same URL and x509

certificate) and serve content that the administrator desires. This is possible because the administrator can snoop the private key from main memory or storage, presenting a major threat to client anonymity. On the other hand, `VaulTor` guarantees that the content served to a client is the content intended by the HS provider. The enclave creates a certificate and is the only entity (alongside the HS provider) that can serve this content.

An HS provider may choose to provision cloud services that enable TEEs and host their content inside one. In this scenario, the cloud would act as an intermediary, reducing the uptime requirement for the HS provider. However, provisioning cloud services isn't anonymous. Know-your-customer (KYC) and/or payments via fiat currency can compromise the anonymity of the HS provider. It would also necessitate online transactions, which may be untenable in regions of instability, where volunteer vaults may be the only option.

6 Threat Model

We make realistic assumptions about the objectives and capabilities of adversaries. An adversary may have access to a small fraction of Tor relays and sufficient resources to qualify as a guard relay, HSDir, or a vault. Moreover, a malicious vault may be able to target specific HSes to host in their machine for the explicit purpose of deanonymizing the HS provider and clients of that service or for the purpose of modifying that service. Even a strong adversary will have a limited ability to acquire network traffic from ISPs and monitor traffic patterns between an end user and a guard node. We wish to protect the anonymity of the HS provider from this adversary and retain the control of the HS provider over any content that the HS is providing. In addition, a malicious HS may attempt to deanonymize a specific vault by hosting their web content on a TEE that the vault runs. As discussed in sec 2.1, we do not consider side-channel attacks.

We also assume that any two entities (amongst the vault, HS provider and client) can collude to launch an attack against the third. For example, an HS provider and a client may work together in an attempt to deanonymize a vault.

Moreover, we assume a careful HS provider. That is, we assume that the HS provider does not leak identifying information in the content it provides. Furthermore, at time t, it will refuse to offload any content to a vault unless it verifies that, at time t, it is connecting to a valid enclave (this is a realistic assumption as verification is an easy task). Similarly, we assume a careful vault. That is, a vault inspects the program provided by an HS provider for malicious code, and will not run this program if it deems it malicious.

7 VaulTor

7.1 Architecture

Building upon the conventional Tor hidden services architecture (Sec 2.2), `VaulTor` introduces three new entities: a device which we refer to as a *vault*, a TEE which will host an *enclave* and an optional external attestation service. The enclave is present inside the vault (see Figure 2) and utilizes its computational resources.

In `VaulTor`, a willing device may offer to host content by advertising itself as a vault. This can be achieved without modifications to the current Tor architecture. The vault creates an onion address for itself (we shall refer to this onion website as the *Vault Contact Hidden Service* (VCHS)) to facilitate correspondence with potential HS providers. Vaults may ask for compensation for hosting an HS (further discussion on incentives is given in Sec 10.3).

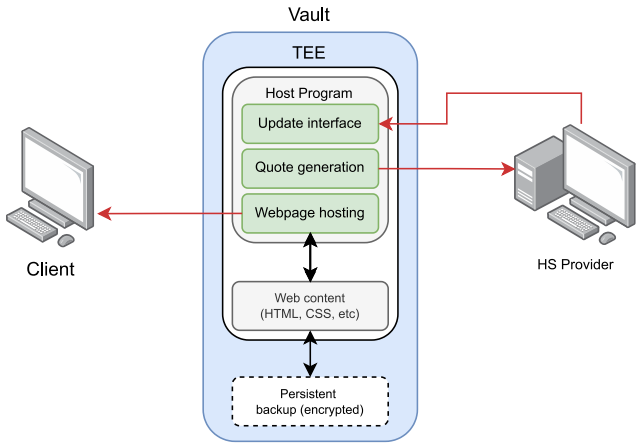


Figure 2 *Our proposed implementation. Red paths represent information flow through Tor circuits with 6 nodes. The data inside the enclave is secure and information flow through the arrows is encrypted i.e., the vault owner can not interpret it.*

In this scenario, the vault has the same privacy guarantees that the Hidden Services have in the current Tor architecture.

The HS provider can reach out to a vault through the vault's VCHS and provide a basic program which we shall refer to a *host program* (HP) that 1) hosts a web server and 2) provides an interface through which (only) the HS provider can upload or remove content when they wish to update their website[6]. This program must be running inside an enclave to provide the privacy guarantees detailed in this architecture. Thus, the HS provider must verify that the enclave has been correctly set up (i.e., the program is running without any modifications to the code and within a TEE) *before* it provides all of the content it wishes to host and any incentives to the vault owner. Future content is also provided through the same update interface described earlier. This process is outlined in detail in Sec 7.2.2.

The TEE guarantees security and privacy for any content the HS provider transfers into the vault. In addition, it removes the requirement for continuous up-time of the original HS provider as the enclave can continue servicing the clients. While the vault must remain online to serve web content, the HS provider does not have this obligation and is hence protected from various deanonymization attacks (Sec 8).

Our architecture ensures increased anonymity and flexibility for the HS provider, minimal decrease in performance for a client (while maintaining the same security guarantees that Tor provides), and a level of anonymity for the vaults that is comparable to that of HSes in the current Tor architecture.

7.2 Protocol

7.2.1 Host Program (HP) Creation

Before an HS provider contacts a vault, it must write a host program that is intended to run within the TEE. This program should provide the following key functionalities:

[6] The interface asks for a secret and after it is verified, the content can be uploaded or removed as desired. Since the secret verification occurs within the script that is running within an enclave, the vault cannot tamper with it without being noticed by the HS provider.

- It should host a server[7] bound to a port. This server should be able to handle POST and GET requests. Furthermore, the code should be able to handle requests dynamically. For example, it should be able to store files uploaded by a client in separate directories.
- The HP should, by default, provide an interface on the server to input an authentication secret. If the secret (which is known to the HS provider) passes the hardcoded verification in the HP, the HS provider should be allowed to modify the contents of the enclave directory. The authentication key can either be a password that is hashed and compared, or it could be the HS provider's private key whose corresponding public key is written in the HP.
- Upon starting the server, the program should generate a *quote* file, that can be verified by the HS provider or a client (details given in Sec 7.2.2). This quote file is available in the web directory and can be accessed by the HS provider or the client for verification through remote attestation as described in Sec 2.1.
- The HP should provide some functionality to create backups of the hosted content in case the vault crashes. Backups are stored on the disk but encrypted using the enclave's *sealing key*. This sealing key is deterministically generated by the CPU depending on the program running inside the TEE and the key burnt into the CPU. This key is only available to the enclave.

Running an arbitrary application inside a TEE is not straightforward. However, a library OS (libOS), such as Gramine-SGX, facilitates this process with minimal modifications to the application. Furthermore, since the entire libOS is contained inside the TEE, no inspection of the application code is necessary. To this end, we assume that a libOS like Gramine-SGX is available to the vault.

7.2.2 Bootstrapping

The vault owner hosts and advertises the onion URL of its VCHS. The HS provider, vault owner and the HP (running inside a TEE) take the following steps to host their service in the vault (also shown in Figure 3):

(i) The HS provider creates the HP (with the functionality described in Sec 7.2.1).
(ii) The HS provider uploads the HP at the VCHS. It is important to note that the host program is uploaded in plaintext (either as a script or a binary).
(iii) The vault owner runs the HP inside a TEE, hosts a hidden service (we shall refer to this as `yourHS.onion`), and binds it to the network port on which the host program will handle incoming requests.
(iv) The host program will generate a public-private key pair (P_{srv}, S_{srv}) for the hidden service it provides. P_{srv} is made available to anyone who connects to `yourHS.onion` while S_{srv} only exists as a variable inside the enclave's memory (and is sealed to the disk for persistent storage).
(v) The host program generates a quote which represents P_{srv}[8] and the program. This quote is available to anyone accessing `yourHS.onion`.
(vi) The HS provider accesses `yourHS.onion`, retrieves the quote, and verifies that the quote is legitimate[9].

[7] Common servers such as Apache or Nginx can be used.
[8] If the size of P_{srv} is too large to fit inside the REPORTDATA of the quote, a hash of P_{srv} may be used. If P_{srv} is embedded in a certificate, then typically the certificate hash is used.
[9] In Intel-SGX, this may involve sending this quote to Intel's online attestation service. In RISCV Keystone, the end user can verify the quote themselves.

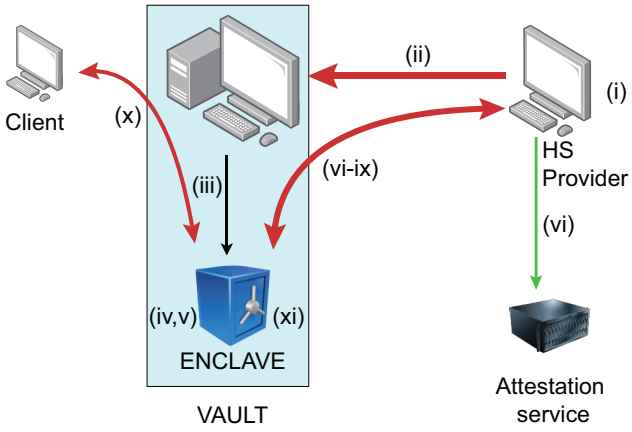


Figure 3 *Our proposed implementation. Red paths represent 6-node circuits and green paths represent 3-node circuits.*

(vii) The HS provider creates a secure connection with the enclave using P_{srv} and any form of key-exchange such as Diffie-Hellman. This public key (P_{srv}) may be embedded in a self-signed x509 certificate in order to facilitate *https* connections.

(viii) The HS provider supplies the authentication secret over this secure connection, after which it can securely upload content (such as HTML, CSS, PHP, and JavaScript files).

(ix) The enclave hosts the uploaded web application content.

(x) Any client that connects to `yourHS.onion` can interact with the hosted web application.

(xi) The enclave regularly encrypts and backs up these files into non-volatile storage using the sealing key.

This procedure ensures increased anonymity for the HS provider. Their data is hosted in the vault and the vault owner can not access data inside the enclave or read the traffic in or out of the enclave. Figure 4 provides a flow diagram of this process detailing some decisions the HS provider must make while uploading to the vault.

7.3 Enclave Isolation

To ensure the protection of the vault, the enclave must have limited privileges. The following conditions, at minimum, are necessary:

1. The enclave has access to a fixed, and limited, amount of RAM. One possible way to achieve this is by running the enclave within a virtual machine[10] configured with limited memory. This prevents an enclave from occupying all available RAM, thus safeguarding the performance of other programs on the vault.
2. The enclave can only use a fixed amount of persistent memory. This can be achieved by isolating the enclave in a separate disk partition. This prevents an enclave from completely filling up persistent disk space that should be available to other programs on the vault.
3. The enclave can only use a fixed amount of network resources. This is possible by controlling the network traffic rate through Trickle [6].
4. Any connections going out from the enclave *must* be restricted to only go via the Tor Proxy. This can be done by creating rules in iptables [4]. This is necessary to ensure

[10] The maximum ram must be set in Intel-SGX at enclave creation time.

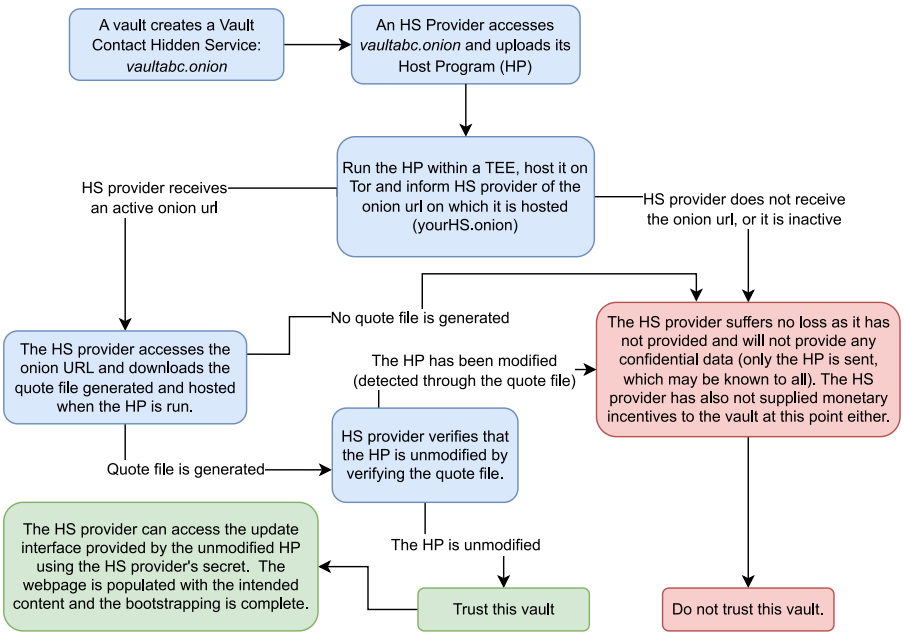


Figure 4 *The steps taken in order for an HS provider to trust and upload to a vault.*

that the IP of the vault is not made available to the HS provider. Any traffic that is *not* going out through port 9050 (the default Tor Proxy port) is blocked by a firewall. Furthermore, any traffic going to the Host Program must originate from the Tor client [11].

7.4 Client connection

A client must ensure that it is connected to the correct HS identified by its P_{srv} (embedded in an x509 certificate). To this end, the HS provider distributes not only the onion URL but also the hash of the x509 certificate when it wishes to advertise its service (similar to how conventional onion URLs are advertised). When connecting to an HS hosted on a vault, the client only needs to verify that the x509 certificate supplied by the service matches its advertised hash to ensure that it is connected to the appropriate entity. The client may maintain a list of valid certificate hashes[12]. Note that the client *never* has to perform remote attestation themselves.

The client in `VaulTor` is exactly the same as a client in the conventional Tor architecture – they use the same connection protocol. Layered on top of this is the ability to match an x509 certificate (or its hash) with the one that is advertised by the HS provider. This matching can be done trivially with a browser extension to the Tor browser.

8 Attack surfaces

In this section, we specify the anonymity guarantees the `VaulTor` provides to each of the three entities: the client, the vault, and the HS provider. We consider the scenarios where

[11] This is to done to preserve vault anonymity from a malicious HS provider as discussed in 8.3.
[12] Checking of certificate hash is trivial and may be added as a subroutine in the client's Tor browser.

each of these can be malicious as well as the scenario in which two of them collaborate to deanonymize the third. VaulTor enhances HS provider anonymity and leaves the client anonymity as it is. We further show that our new actor—the vault—is as protected as an HS provider in the current Tor Hidden Services architecture.

8.1 HS provider Anonymity

Scenario 1: Malicious client

The HS provider no longer interacts with the clients directly. To access the web application content, the clients now establish a connection with the vault instead. Thus, unless the HS provider leaks identifying information in their content, they are safe from deanonymization at the hands of a client.

Scenario 2: Malicious vault

In the traditional Tor design, where the client and HS provider communicate over a two-way anonymous channel initiated by the client, the attack scenarios in Table 1 render the client a harder anonymity target by a malicious HS provider than an honest HS provider by a malicious client. A mirror situation exists in the VaulTor design where the HS provider uploads and updates the content on a vault over a two-way anonymous channel. Thus, even in the worst case, an HS provider in VaulTor is as anonymous as an HS provider in the traditional architecture. Furthermore, the minimal uptime requirement enhances the anonymity of the HS provider in VaulTor architecture. The use of TEE at the vault offers additional guarantees of data integrity and data confidentiality to the HS provider.

Scenario 3: Vault and client collude

The attack opportunities open to a client are a subset of the attack opportunities possible for a vault (since a vault has the same privileges as a client and more). Thus, the protection guaranteed for an HS provider from the vault applies in a scenario where the vault and client may collude.

8.2 Client privacy

We now show that a client is as protected in VaulTor as they are in the current Tor architecture.

Scenario 1: Malicious HS provider

The HS provider is completely disconnected from the client, and hence is unable to launch attacks on the client directly.

Scenario 2: Malicious vault

A malicious vault may attempt to a) serve modified content or b) launch a deanonymization attack on the clients. We now show why these attacks are not feasible in our architecture:

a) As the content is being hosted inside an enclave, clients can ensure that any content being served by the vault has not been maliciously modified. Since the x509 certificate is generated by the HP running inside a TEE and the corresponding private key (S_{srv}) is *only* available to the TEE and the HS provider, a secure connection established using the certificate is guaranteed to be serving content vetted by the HS provider. Another consideration is that the content being hosted inside a vault is regularly encrypted and backed up to the disk. While this backed-up data can not be modified, a vault owner can selectively delete this backed-up data and restart the program in the enclave. This may result in the enclave accidentally serving outdated data to clients. However, if pieces of

content are properly timestamped, the TEE can refuse to serve content that is outdated or add warnings while serving this content.

b) In VaulTor, a client's perspective of the Hidden Service architecture remains the same. A client still accesses content through a 6-node connection–except that instead of connecting to the HS provider, it connects to a vault. Thus the client enjoys the same privacy guarantees as a client in the conventional Tor architecture. Furthermore, as discussed in sec 10.8, the client may enjoy enhanced data privacy.

Scenario 3: Vault and HS provider collude

A colluding vault and HS provider in the VaulTor architecture have the same attack opportunities against a client as a malicious HS provider in the conventional Tor architecture. Thus, the deanonymization risk for the client is the same as when only the vault is malicious.

8.3 Vault privacy

VaulTor introduces a new entity in the Tor architecture: a vault that assumes a role similar to that of an HS in the current Tor design, thus maintaining a similar level of protection against deanonymization attacks.

Scenario 1: Malicious client

A vault is as vulnerable to a client as a Hidden Service is in the traditional Tor network. One may even argue that the vault has stronger anonymity guarantees due to the fact that the web application content is hosted within an enclave (a secret, protected environment the vault cannot modify). This may provide plausible deniability to the vault as it would be blind to the traffic that enters and leaves its enclave.

Scenario 2: Malicious HS provider

A vault is protected from attacks launched by an HS provider through the Host Program by ensuring that the safety criteria specified in Sec 7.3 are satisfied.

Consider a malicious HS provider who uploads code that tries to obtain identifying information about the vault by leveraging the fact that the enclave is utilizing the vault's hardware. For example, this code may try to read files belonging to the vault or the vault's OS in order to *directly* find identifying information or it may try to obtain its IP *indirectly* by pinging an external server. The security guarantees provided by TEEs make direct attempts impossible; the host is *also* isolated from the TEE just as the TEE is isolated from the host. Furthermore, most applications for TEEs run in a VM (as is the default in gramine-SGX [1]), adding to the isolation. Indirect attacks are also mitigated using a firewall. By ensuring that all outgoing traffic is ported through port 9050 (the default port for Tor), only the IP of the exit node is available to the Host Program.

Furthermore, by ensuring that all requests originate from the Tor client software on the vault, the vault is protected from pinging based attacks. If this is not done, a malicious HS provider may upload a simple Host Program that replies with a unique phrase to the HS provider's IP when this Host Program is pinged. The HS provider may then ping various candidate IPs in the hopes of stumbling upon the vault's IP which would reply with the phrase. This sort of attack is only possible if the Host Program can be pinged directly, from outside the Tor client software.

Scenario 3: Client and HS Provider collude

The HS provider is in a unique position as it directly provides code that the vault runs within an enclave. If an entity controls both the HS provider and a client node, we consider an attempt to launch a watermarking attack (described in Sec 9.1). This attack, in particular, only requires the control of two entities connected to the third. Moreover, the fact that the

HS provider and client can make repeated on-demand requests to the vault further benefits the viability of this attack.

To attempt to launch a watermarking attack (similar to what [38, 39] describe), the HS provider would add some watermark to the Tor traffic that can be identified at the client end. Despite controlling both the HS provider and a client, an attacker would not have the ability to successfully launch a watermarking attack on the vault. This is because of a missing component that this attack requires: the control of a guard relay. If we also assume the control of a guard relay, the HS provider is no longer necessary, as the guard relay can be the entity that watermarks the traffic. A guard relay and client could potentially launch this attack on their own without the requirement of an HS provider. Hence, the vault is as protected from a watermarking attack as an HS is in the current architecture. Scenario 3 is thus akin to having two malicious clients in the conventional Tor architecture.

9 Evaluation

In this section, we will qualitatively evaluate the effect of `VaulTor` in deflecting various families of existing attacks on HSes from the HS provider to our new Vault node. Afterwards, we will quantitatively measure the performance impact of `VaulTor` on client side network performance.

9.1 Known Attacks Deflected

We will list various attacks and briefly explain how the `VaulTor` architecture deflects them from the HS provider to the vault. This list is non-exhaustive yet exemplifies the prominent attacks in recent years.

Clock Skew: These attacks rely on repeatedly sending requests to an HS in order to heat up its CPU which has some tangible effect on the timestamp of incoming packets [60, 52]. In `VaulTor`, this is impossible. *No one* can repeatedly send packets to the HS provider.

Congestion: This attack relies on an adversary congesting existing guard nodes in the network [42], forcing the HS to connect to their compromised guard node long enough for the adversary to correlate traffic. This attack is completely deflected in `VaulTor`; the HS provider is sporadically online for limited periods and an adversary would have to congest the network indefinitely.

Fingerprinting: These attacks rely on learning the traffic patterns of an HS and referencing this against the traffic of a candidate set of guard nodes [46, 57, 53, 36]. In our architecture, the vault is serving the traffic while the HS provider is taciturn. Thus, this type of attack will not work on the HS provider. Similar attacks that rely on compromised middle nodes [41] are similarly deflected.

Guard Node Discovery: The Tor developers currently consider this the most potent threat against hidden services [7]. This attack relies on making multiple connections with the HS provider such that their malicious middle node is next to the HS provider's guard node. Repeated, on-demand connections with the HS provider are impossible in `VaulTor`. As such, this attack is eliminated.

Location Leaks: Such attacks rely on the negligence of the HS provider and are out of the scope of this paper [49].

Watermarking: In this type of attack, an adversary watermarks traffic on the client side in order to detect it at the malicious guard node of the HS provider [38, 39]. If a malicious vault node tries to launch this attack on the HS provider, this attack would be

rendered less effective because HS-provider would have minimal uptime connection with the vault code instead of a constant connection.

Table 1 shows the various scenarios in which an adversary can launch a deanonymizing attack on Tor Hidden Services along with the impact of VaulTor on these attacks.

Scenario	Attack Categories	VaulTor impact
Adversary can send arbitrary requests to the HS provider	Clock Skew, Watermarking, Guard Node Discovery, Fingerprinting	Scenario eliminated
HS Provider has high uptime	Clock Skew, Watermarking, Congestion, Fingerprinting	Scenario diminished
High volume of traffic coming from the HS provider	Watermarking, Congestion, Fingerprinting	Scenario diminished

Table 1 *Various scenarios that lead to categories of contemporary attacks on Tor Hidden Services along with the impact of* VaulTor *on these scenarios.*

9.2 Performance

It is important that the security improvements VaulTor brings do not significantly degrade client side network performance. Important client side performance metrics include the network latency and the throughput. In this section, we detail our experimental setup (which includes our implementation of a vault) and our evaluation of these metrics. Our experiments measure the time experienced by the client to retrieve the data from an HS; the registration and bootstrapping processes in VaulTor occur only once and have a negligible[13] performance impact in the overall lifetime of the HS.

9.2.1 Experimental setup

To measure and compare the network performance of Hidden Services when hosted within and outside a TEE, we ran two instances of a Host Program on the same machine. One of these HPs ran inside a TEE (facilitated by the gramine-SGX library OS [1]) while the other HP (which we shall refer to as a *vanilla* HP) ran outside a TEE. A Tor client[14] was also launched on the same machine which generated two onion URLs: one for the enclave and one for the vanilla HP. The Tor client directs traffic for each of these onion URLs to their respective HPs, allowing them to serve content via Tor.

Both the enclave and the vanilla HP ran webservers and, in order to ensure consistency, served the same landing webpage simultaneously. In addition, the HP running inside a TEE had the ability to generate a quote in order to facilitate remote attestation. Both web servers were written in Python3 and regularly backed up data to persistent memory. Moreover, these webpages were hosted on the same device with an SGX-enabled Intel processor (Core-i5 10210U).

We conducted these experiments with three webpages, each with varying page sizes (0.5kB, 50kB, and 5000kB). The content on these webpages included HTML and JavaScript.

[13] This performance impact will only be negligible if uploading content does not significantly increase the uptime of HS provider.

[14] This client is not the same as a client in Tor architecture which we have discussed above. This is a program necessary to interact with the Tor network.

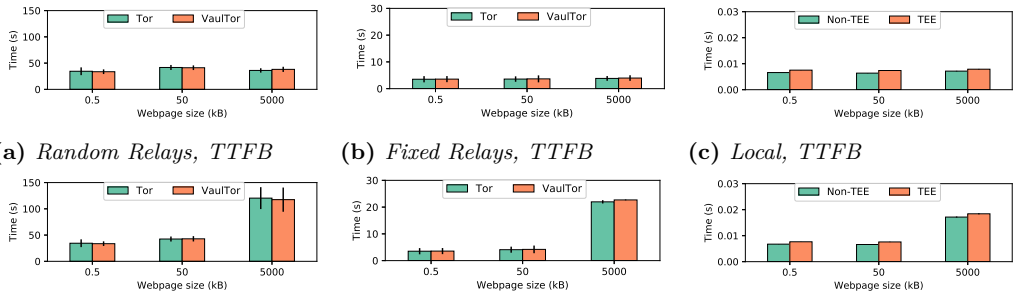
We measured the performance using three methods:

1. *Random Relays:* We restarted the Tor application between each measurement to establish fresh circuits. This gave us three random relays for every measurement.
2. *Fixed Relays:* We used fixed/constant relays[15] across webpages for both `VaulTor` and Tor. We report the average performance of three different fixed circuits.
3. *Local:* We locally accessed the webpages.

Each webpage was loaded 250 times in each of the methods, save for the Fixed Relays method, for which we loaded each webpage 250 times on each circuit (a total of 750 measurements) and took the average of the results. Methods (1) and (2) were conducted on the actual Tor network.

We thus quantified any overall changes in performance caused specifically by hosting an HS within a TEE.

9.2.2 Results



(a) *Random Relays, TTFB* **(b)** *Fixed Relays, TTFB* **(c)** *Local, TTFB*

(d) *Random Relays, TTLB* **(e)** *Fixed Relays, TTLB* **(f)** *Local, TTLB*

Figure 5 *Time to first byte (TTFB) and time to last byte (TTLB) for webpages with varying page sizes without and within a TEE. Error bars represent 99% confidence intervals.*

Figure 5 shows the time to first byte (TTFB) and time to last byte (TTLB) for the 3 webpages hosted in the 2 architectures (the current architecture and `VaulTor`).

We note a minimal difference in performance across webpages and testbeds. Note that, for most of the results collected over Tor, the average TTFB and TTLB in the `VaulTor` architecture fall within the 99% confidence interval of those of the current architecture. The only result (collected over the Tor network) that lied outside the confidence interval was for the TTLB of a 5000kB webpage routed through fixed relays. This had an average increase of 2.9%.

If we consider all the results, including those that lie within the confidence intervals, we note a maximum increase in TTFB and TTLB of 5.7% (5000kB, Random Relays) and 2.9% (5000kB, Fixed Relays), respectively.

It should be noted that running an arbitrary program inside an Intel-SGX TEE may have a non-negligible computation overhead. When accessing the webpages locally (and hence, not over the Tor network), we note a maximum percentage increase in time in the case of the TTFB of a webpage of size 50kB (15.9%). However, this delay is negligible compared to delays caused by Tor's network latency. As such, it is not surprising that the percentage

[15] These were chosen randomly from advertised Tor relays here: `https://www.dan.me.uk/tornodes`

performance overhead of VaulTor over the conventional Tor architecture is minimal when measured over the real Tor network.

We believe this nominal decrease in performance is justified considering the major anonymity benefits VaulTor brings to the HS provider.

9.2.3 Ethics

We had ethical considerations while conducting our performance measurements. We solely collected timing information and the size of the files we downloaded. We did not store the IP addresses of the entry and exit nodes, so as to preserve their anonymity. In addition, our load on the Tor network was negligible. We ensured this by conducting the experiments serially, and not in parallel, to minimize the load at any given time. To the best of our knowledge, we did not hinder any other users' experiences on the Tor network.

10 Discussion

10.1 Additional Anonymity Measures

There exist a number of proposals (such as [7, 20, 55, 62]) that enhance the anonymity of the HS provider. These solutions, however, result in degraded network performance (longer delays and lower throughput) for the client, when used in the conventional HS architecture. This reduction in network performance renders these solutions less attractive today. With the VaulTor architecture, the vault serves the content to the clients, and any retrofitting at the HS provider side has no bearing on the network performance experienced by the client.

For example, the Vanguard add-on [7] (which inserts additional hops in the connection) may be used by the HS provider without affecting client-side performance. Similarly, privacy-preserving path selection methods [20, 55, 62] may incur latency costs but are a non-issue for an HS provider that connects with the vault infrequently.

Furthermore, techniques like temporary proxies are completely compatible with our system and may be used by clients and the HS provider to connect to vaults to obfuscate their traffic [25, 35, 19].

10.2 HS Provider Flexibility

In the current Tor architecture, the HS provider must remain static in order to serve content. The flexibility offered by VaulTor can be leveraged by the HS provider to communicate from secure and variable locations. This would especially be beneficial in the context of activists or journalists who want to report their content from secure *intermediate* locations in oppressive regimes without the risk of getting caught. Moreover, VaulTor would allow the HS provider's content to remain accessible during Internet outages, which is commonplace in regions with political instability and censorship [21, 22].

10.3 Incentives for Vault Node

10.3.1 Monetary Incentives

The vault owner proxies for the HS provider and, on its behalf, serves content to the clients. This act must be incentivized for the vault owner. These incentives may be social incentives – similar to how users of Tor run relays and nodes.

However, if incentives are monetary, they must be exchanged in a secure and private manner. Towards this end, a blockchain may be used to ensure that the vault owner receives cryptocurrency rewards for hosting content for the HS provider. One approach to this is that the vault owner supplies the HS provider with their address on a public blockchain such as Ethereum [67]. Only if regular cryptocurrency payments are made to this public address does the vault owner continue hosting. This allows both the HS provider and the clients to "crowdsource" an HS on a vault.

This previous approach does necessitate timely payments from the HS provider. This requirement can be removed via the use of smart contracts. A smart contract can lock the cryptocurrency that it receives from the HS provider and clients. The smart contract can then use an oracle to verify that the HS is being hosted properly and perform remote attestation. If the HS is being hosted properly and the remote attestation is successful, the smart contract releases the cryptocurrency to the vault's blockchain address. In order to preserve privacy, zero knowledge enabled cryptocurrency such as Zcash [23] can be used. Furthermore, a Decentralized Exchange (DEX) may be used to trade cryptocurrency. These measures reduce the possibility of profiling based attacks.

For example, a vault owner may make their zero-knowledge blockchain address (such as for monero [56]) available on VCHS and the HS provider may anonymously transfer this cryptocurrency by publishing their transaction by connecting to an external blockchain node through the Tor proxy. In a zero-knowledge blockchain, the transaction itself will have no identifying information present in it that may be used for social engineering (such as, by using chain analysis tools).

10.3.2 Altruistic Reasons

Vault operation may also be motivated by altruistic considerations rather than direct financial compensation. This model closely mirrors the operation of Tor relays, where participants voluntarily contribute resources to enhance privacy, censorship resistance, and overall network resilience without receiving monetary rewards. While operating a Tor relay may incidentally support unlawful activities, it also enables many socially beneficial uses, such as protecting free expression and providing access to information under restrictive conditions. Similarly, vault nodes may be used for both benign and potentially abusive purposes; however, their primary value lies in supporting privacy preserving and censorship resistant services.

Similar altruistic participation models exist in other systems, including running VPN or proxy servers for community use, and operating public Network Time Protocol (NTP) servers. In these cases, operators are motivated by a desire to support open infrastructure, enhance collective security, or contribute to public-good Internet services.

10.4 Plausible Deniability for Vault

In the `VaulTor` architecture, a vault owner is not privy to the content present inside the TEE. We believe that this adds an extra layer of plausible deniability, greater than the plausible deniability of conventional data hosting services that are aware of the content being served. When hosting content on behalf of an HS provider, the only thing the vault owner knows is the Onion URL of the Hidden Service. This has a parallel with Guard Nodes in the conventional Tor architecture that know what Onion URL's traffic is routed through them. Both the vault and the guard node can not read this traffic or compromise its integrity, only help move this content. The only difference is that the physical storage resources of the

Vault owner are being used. However, even this physical storage is encrypted and opaque to the Vault owner which is not privy to the information being served, just like a guard node.

In our future work, we can enable vault owners to run a Tor client inside an enclave and run the HP inside another enclave on the same machine. The Tor client generates a new onion URL and shares it (only) with the Host Program. The HP then serves content on this onion URL using the Tor client as a proxy. The HP also forwards the onion URL to the HS provider who can then advertise it as before. In this scenario, the client does not need to modify their browser. As such, the vault owner is not aware of the content they are serving. In this scenario, they can not be held liable for the content they are serving as the information about which machine is serving what content is available to *"no one"*. And *"no one"* includes the vault owner and the HS provider.

10.5 Legal and deployability considerations

`VaulTor` is a technical design that aims to improve the anonymity and availability of hidden services. While it does not change Tors underlying trust or threat model, legal and regulatory constraints may nonetheless become a practical barrier to deployment. For example, a vault operator could face legal risk for hosting third-party content that is alleged to be unlawful (e.g., infringing or prohibited content), even if the operator cannot readily inspect the hosted state. In addition, hidden service operators remain responsible for the services and content they publish, regardless of whether hosting is delegated to a TEE-hosted vault node.

For vault operators, legal exposure is jurisdiction-dependent and may hinge on how local law treats third-party hosting, infrastructure provision, and duties arising from notice or investigation. Thus, even though `VaulTor` is designed so that the vault cannot inspect the encrypted hidden service content state, prospective vault operators should treat legal risk as non-negligible and evaluate deployability under applicable local law (e.g., jurisdictional choice, operational policies, and whether participation is restricted to vetted deployments).

10.6 Incremental deployment:

The `VaulTor` architecture supports incremental deployment (albeit its strength is fully utilized when there are many vaults present on Tor). Vaults can register their VCHS themselves to an HSDir similar to how Hidden Services are currently already registered. This allows for a slow, optional adoption of `VaulTor`.

The client *does* need to install a small extension (as discussed in Sec 7.4) that compares P_{srv} (or its hash) with the one advertised by the HS provider but this is a trivial add-on and does not affect traditional HSes.

10.7 Multiple Vaults:

An HS provider may commission multiple vaults to hold their data. To this end, they may download the S_{srv} and the x509 certificate from the HP of one vault and upload it to an HP they have hosted on another vault. As clients use the certificate hash provided by an HS to validate its identity (as described in Sec 7.4), they can be certain they are being served by the same HS provider even if the onion URL of the HS is different. This will add redundancy and fault tolerance to the HS provider's content: if one vault becomes inactive, the other vaults can continue to serve content.

10.8 Strengthening client data privacy

In the VaulTor architecture, the host program is present inside a TEE and its measurement (see Sec 2.1) is available to the client (in addition to the HS provider). The HS provider may elect to make the code itself available to the client, allowing the client to inspect this code. If the code is simple (for example, the code only stores and serves content to password authenticated requests), then the client can upload private data to the server *without having to trust the HS provider* as is necessary in the current Tor architecture.

10.9 Attacks against TEEs:

A wide variety of side-channel attacks exist that can target TEEs [68, 64, 61, 27, 54]. These attacks aim to discern secrets contained inside the TEE, such as private keys, through various means such as leveraging page faults. Vendors are prompt in mitigating side-channel attacks [10] as the community uncovers those. Considering the research and development that focuses on mitigating side-channel attacks [66, 43, 47, 37], architectural designs discount such attacks from their threat models [63, 11, 50]. We also follow take the same course of action (Sec 6).

10.10 Advancements/variations in TEE technology:

Although our current implementation utilizes Intel-SGX, VaulTor is a generic solution that could theoretically support any TEE service. As newer TEE services (such as Intel-TDX [3]) emerge and improve, the strength and flexibility of VaulTor improve as well. In the future, a diverse set of vaults utilizing differing TEE technologies could be built and tested. Other promising implementations of TEEs are also being proposed [18, 26, 29, 34, 48]. We believe that TEEs will become increasingly resistant to side-channel attacks.

11 Conclusion

We present VaulTor as an architectural solution that leverages TEE technology to reduce the threat of deanonymization attacks against HS providers on the Tor network. To this end, VaulTor introduces a new actor: the vault, which serves content on the HS provider's behalf. We show that VaulTor prevents several HS deanonymization attacks by utilizing the vault, whilst preserving the same level of client anonymity as in the current architecture. This is achieved without any noticeable performance degradation experienced by the client. We also argue that vaults have the same security guarantees as HS providers in the conventional Tor architecture.

References

1 Gramine,https://gramineproject.io/.

2 How Do Onion Services Work?, https://community.torproject.org/onion-services/overview/.

3 Intelő Trust Domain Extensions (Intelő TDX), https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html.

4 iptables(8) - Linux man page, https://linux.die.net/man/8/iptables.

5 Tor, https://www.torproject.org/.

6 trickle(1) - Linux man page, https://linux.die.net/man/1/trickle. 2002.

7 Announcing the Vanguards Add-On for Onion Services, https://blog.torproject.org/announcing-vanguards-add-onion-services/. 2014.

8 Global action against dark markets on Tor network, https://www.europol.europa.eu/newsroom/news/global-action-against-dark-markets-tor-network. 2014.

9 Users - Tor Metrics, https://metrics.torproject.org/userstats-relay-country.html. 2022.

10 2023.

11 Adil Ahmad, Juhee Kim, Jaebaek Seo, Insik Shin, Pedro Fonseca, and Byoungyoung Lee. Chancel: Efficient multi-client isolation under adversarial programs. 01 2021.

12 Masoud Akhoondi, Curtis Yu, and Harsha V. Madhyastha. Lastor: A low-latency as-aware tor client. *IEEE/ACM Trans. Netw.*, 22(6):17421755, dec 2014.

13 Mashael AlSabah, Kevin Bauer, Tariq Elahi, and Ian Goldberg. The path less travelled: Overcoming tor's bottlenecks with traffic splitting. In Emiliano De Cristofaro and Matthew Wright, editors, *Privacy Enhancing Technologies*, pages 143–163, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

14 Greubel Andre, Dmitrienko Alexandra, and Kounev Samuel. Smartor: Smarter tor with smart contracts: Improving resilience of topology distribution in the tor network. In *Proceedings of the 34th Annual Computer Security Applications Conference*, ACSAC '18, page 677691, New York, NY, USA, 2018. Association for Computing Machinery.

15 Robert Annessi and Martin Schmiedecker. Navigator: Finding faster paths to anonymity. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 214–226, 2016.

16 ARM Limited. Security technology: building a secure system using trustzone technology http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf. 2018.

17 Arushi Arora and Christina Garman. Improving the performance and security of tor's onion services. *Proceedings on Privacy Enhancing Technologies*, 2025:531–552, 01 2025.

18 Raad Bahmani, Ferdinand Brasser, Ghada Dessouky, Patrick Jauernig, Matthias Klimmek, Ahmad-Reza Sadeghi, and Emmanuel Stapf. CURE: A security architecture with CUstomizable and resilient enclaves. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1073–1090. USENIX Association, August 2021.

19 Diogo Barradas, Nuno Santos, Luís Rodrigues, and Vítor Nunes. Poking a hole in the wall: Efficient censorship-resistant internet communications by parasitizing on webrtc. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS '20, page 3548, New York, NY, USA, 2020. Association for Computing Machinery.

20 Armon Barton and Matthew Wright. Denasa: Destination-naive as-awareness in anonymous communications. *Proceedings on Privacy Enhancing Technologies*, 2016(4):356–372, 2016.

21 David Belson. A recent spate of Internet disruptions, https://blog.cloudflare.com/a-recent-spate-of-internet-disruptions-july-2024/. 2024.

22 David Belson. Q2 2024 Internet disruption summary, https://blog.cloudflare.com/q2-2024-internet-disruption-summary/. 2024.

23 Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized anonymous payments from bitcoin. Cryptology ePrint Archive, Paper 2014/349, 2014. https://eprint.iacr.org/2014/349.

24 Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. Trawling for tor hidden services: Detection, measurement, deanonymization. In *2013 IEEE Symposium on Security and Privacy*, pages 80–94, 2013.

25 Cecylia Bocovich, Arlo Breault, David Fifield, Serene, and Xiaokang Wang. Snowflake, a censorship circumvention system using temporary WebRTC proxies. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 2635–2652, Philadelphia, PA, August 2024. USENIX Association.

26 Ferdinand Brasser, David Gens, Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. Sanctuary: Arming trustzone with user-space enclaves. 01 2019.

27 Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. Telling your secrets without page faults: Stealthy page Table-Based attacks on enclaved execution. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1041–1056, Vancouver, BC, August 2017. USENIX Association.

28 Nicolas Christin. Traveling the silk road: a measurement analysis of a large anonymous online marketplace. *Proceedings of the 22nd international conference on World Wide Web*, 2013.

29 Victor Costan, Ilia Lebedev, and Srinivas Devadas. Sanctum: Minimal hardware extensions for strong software isolation. In *Proceedings of the 25th USENIX Conference on Security Symposium*, SEC'16, page 857874, USA, 2016. USENIX Association.

30 Hussein Darir, Hussein Sibai, Chin-Yu Cheng, Nikita Borisov, Geir Dullerud, and Sayan Mitra. Mleflow: Learning from history to improve load balancing in tor. *Proceedings on Privacy Enhancing Technologies*, 2022:75–104, 01 2022.

31 Tom Woller David Kaplan, Jeremy Powell. AMD memory encryption, `https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf`. 2016.

32 Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The Second-Generation onion router. In *13th USENIX Security Symposium (USENIX Security 04)*, San Diego, CA, August 2004. USENIX Association.

33 Joan Feigenbaum, Aaron Johnson, and Paul Syverson. Preventing active timing attacks in low-latency anonymous communication. pages 166–183, 07 2010.

34 Andrew Ferraiuolo, Andrew Baumann, Chris Hawblitzel, and Bryan Parno. Komodo: Using verification to disentangle secure-enclave hardware from software. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 287305, New York, NY, USA, 2017. Association for Computing Machinery.

35 Gabriel Figueira, Diogo Barradas, and Nuno Santos. Stegozoa: Enhancing webrtc covert channels with video steganography for internet censorship circumvention. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '22, page 11541167, New York, NY, USA, 2022. Association for Computing Machinery.

36 Jamie Hayes and George Danezis. k-fingerprinting: A robust scalable website fingerprinting technique. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1187–1203, Austin, TX, August 2016. USENIX Association.

37 Zhou Hongwei, Ke Zhipeng, Zhang Yuchen, Wu Dangyang, and Yuan Jinhui. Tsgx: Defeating sgx side channel attack with support of tpm. In *2021 Asia-Pacific Conference on Communications Technology and Computer Science (ACCTCS)*, pages 192–196, 2021.

38 Alfonso Iacovazzi, Daniel Frassinelli, and Yuval Elovici. The DUSTER attack: Tor onion service attribution based on flow watermarking with track hiding. In *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*, pages 213–225, Chaoyang District, Beijing, September 2019. USENIX Association.

39 Alfonso Iacovazzi, Sanat Sarda, and Yuval Elovici. Inflow: Inverse network flow watermarking for detecting hidden servers. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 747–755, 2018.

40 Intel. Intel software guard extensions programming reference, `https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf/`. 2014.

41 Rob Jansen, Marc Juarez, Rafael Galvez, Tariq Elahi, and Claudia Diaz. Inside job: Applying traffic analysis to measure tor from within. 01 2018.

42 Rob Jansen, Florian Tschorsch, Aaron Johnson, and Björn Scheuermann. The sniper attack: Anonymously deanonymizing and disabling the tor network. In *NDSS*, 2014.

43 Yuan Jinhui, Zhou Hongwei, and Zhang Laishun. F-sgx: Next generation sgx for trusted computing. In *2021 IEEE Asia-Pacific Conference on Image Processing, Electronics and Computers (IPEC)*, pages 673–677, 2021.

44 Aaron Johnson, Rob Jansen, Nicholas Hopper, Aaron Segal, and Paul Syverson. Peerflow: Secure load balancing in tor. *Proceedings on Privacy Enhancing Technologies*, 2017, 04 2017.

45 Seongmin Kim, Juhyeng Han, Jaehyeong Ha, Taesoo Kim, and Dongsu Han. Sgx-tor: A secure and practical tor anonymity network with sgx enclaves. *IEEE/ACM Transactions on Networking*, 26(5):2174–2187, 2018.

46 Albert Kwon, Mashael AlSabah, David Lazar, Marc Dacier, and Srinivas Devadas. Circuit fingerprinting attacks: Passive deanonymization of tor hidden services. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 287–302, Washington, D.C., August 2015. USENIX Association.

47 Fan Lang, Wei Wang, Lingjia Meng, Jingqiang Lin, Qiongxiao Wang, and Linli Lu. Mole: Mitigation of side-channel attacks against sgx via dynamic data location escape. In *Proceedings of the 38th Annual Computer Security Applications Conference*, ACSAC '22, page 978988, New York, NY, USA, 2022. Association for Computing Machinery.

48 Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

49 Srdjan Matic, Platon Kotzias, and Juan Caballero. Caronte: Detecting location leaks for deanonymizing tor hidden services. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 14551466, New York, NY, USA, 2015. Association for Computing Machinery.

50 J. Menetrey, M. Pasin, P. Felber, and V. Schiavoni. Twine: An embedded trusted runtime for webassembly. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 205–216, Los Alamitos, CA, USA, apr 2021. IEEE Computer Society.

51 Prateek Mittal and Nikita Borisov. Shadowwalker: Peer-to-peer anonymous communication using redundant structured topologies. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, page 161172, New York, NY, USA, 2009. Association for Computing Machinery.

52 Steven Murdoch. Hot or not: Revealing hidden services by their clock skew. pages 27–36, 01 2006.

53 Milad Nasr, Alireza Bahramali, and Amir Houmansadr. Deepcorr: Strong flow correlation attacks on tor using deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 19621976, New York, NY, USA, 2018. Association for Computing Machinery.

54 Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. A survey of published attacks on intel sgx, 2020.

55 Rishab Nithyanand, Oleksii Starov, Phillipa Gill, Adva Zair, and Michael Schapira. Measuring and mitigating as-level adversaries against tor. *ArXiv*, abs/1505.05173, 2016.

56 Shen Noether. Ring signature confidential transactions for monero. *IACR Cryptology ePrint Archive*, 2015:1098, 2015.

57 Andriy Panchenko, Asya Mitseva, Martin Henze, Fabian Lanze, Klaus Wehrle, and Thomas Engel. Analysis of fingerprinting techniques for tor hidden services. In *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*, WPES '17, page 165175, New York, NY, USA, 2017. Association for Computing Machinery.

900 **58** phobos. Using Tor hidden services for good, https://blog.torproject.org/
901 using-tor-hidden-services-good/. 2012.
902 **59** Florentin Rochet, Ryan Wails, Aaron Johnson, Prateek Mittal, and Olivier Pereira. *CLAPS:*
903 *Client-Location-Aware Path Selection in Tor*, page 1734. Association for Computing Ma-
904 chinery, New York, NY, USA, 2020.
905 **60** Komang Oka Saputra, Wei-Chung Teng, and Yi-Hao Chu. A clock skew replication attack
906 detection approach utilizing the resolution of system time. In *2015 IEEE/WIC/ACM Interna-*
907 *tional Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, volume 3,
908 pages 211–214, 2015.
909 **61** Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. Preventing
910 page faults from telling your secrets. In *Proceedings of the 11th ACM on Asia Conference on*
911 *Computer and Communications Security*, ASIA CCS '16, page 317328, New York, NY, USA,
912 2016. Association for Computing Machinery.
913 **62** Yixin Sun, Anne Edmundson, Nick Feamster, Mung Chiang, and Prateek Mittal. Counter-
914 raptor: Safeguarding tor against active routing attacks. In *2017 IEEE Symposium on Security*
915 *and Privacy (SP)*, pages 977–992, 2017.
916 **63** Yuanyuan Sun, Sheng Wang, Huorong Li, and Feifei Li. Building enclave-native storage
917 engines for practical encrypted databases. *Proc. VLDB Endow.*, 14(6):10191032, feb 2021.
918 **64** Jo Van Bulck, Frank Piessens, and Raoul Strackx. Sgx-step: A practical attack framework
919 for precise enclave execution control. In *Proceedings of the 2nd Workshop on System Software*
920 *for Trusted Execution*, SysTEX'17, New York, NY, USA, 2017. Association for Computing
921 Machinery.
922 **65** Tao Wang, Kevin Bauer, Clara Forero, and Ian Goldberg. Congestion-aware path selection
923 for tor. In Angelos D. Keromytis, editor, *Financial Cryptography and Data Security*, pages
924 98–113, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
925 **66** Jan Wichelmann, Anna Pätschke, Luca Wilke, and Thomas Eisenbarth. Cipherfix: Mitigating
926 ciphertext side-channel attacks in software, 2023.
927 **67** Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum*
928 *project yellow paper*, 151(2014):1–32, 2014.
929 **68** Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic
930 side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and*
931 *Privacy*, pages 640–656, 2015.