Evaluating Dynamic Tor Onion Services for Privacy Preserving Distributed Digital Identity Systems

Tobias Höller*, Michael Roland and René Mayrhofer

Institute of Networks and Security, Johannes Kepler University Linz, Austria
E-mail: tobias.hoeller@ins.jku.at; michael.roland@ins.jku.at;
rene.mayrhofer@ins.jku.at
**Corresponding Author*

Received 14 October 2021; Accepted 26 October 2021;
Publication 16 March 2022

Abstract

Digital identity documents provide several key benefits over physical ones. They can be created more easily, incur less costs, improve usability and can be updated if necessary. However, the deployment of digital identity systems does come with several challenges regarding both security and privacy of personal information. In this paper, we highlight one challenge that digital identity systems face if they are set up in a distributed fashion: Network Unlinkability. We discuss why *network unlinkability* is so critical for a distributed digital identity system that wants to protect the privacy of its users and present a specific definition of unlinkability for our use-case. Based on this definition, we propose a scheme that utilizes the Tor network to achieve the required level of unlinkability by dynamically creating onion services and evaluate the feasibility of our approach by measuring the deployment times of onion services.

Keywords: Tor, onion service, privacy.

Journal of Cyber Security and Mobility, Vol. 11_2, 141–164.
doi: 10.13052/jcsm2245-1439.1122

1 Introduction

Digital services have become an essential aspect of everyday life for many people in the last decade. In combination with the trend to embed computing capabilities in physical devices this has lead to an increased demand for digital identity systems. Their identities can be used in purely digital transactions like creating accounts for services or in physical interactions like verifying a driving license. Digital services need reliable digital identities to prevent fake accounts, limiting access for underage users, or holding their users legally responsible for their actions on a digital platform. Systems like the European e-ID provide this kind of functionality.

More physical applications require a stronger coupling between a digital identity and a physical individual. A recent example for this is the *EU Digital COVID Certificate* [8], a QR-code based approach to track both tests and vaccinations for the COVID-19 virus. These codes contain the information required to identify a person (name and birthday) along with a signed statement about their test result/vaccination. Since digital documents can easily be duplicated, the same document could be presented by multiple individuals that all claim to have the same name and birthday. This is prevented by mandating that a QR code is only valid in conjunction with a photo ID document that matches the name and birthday in the vaccination statement. Consequently, the EU COVID certificate must be considered a digital extension for a physical identity, as it does still depend on a physical ID being present.

There are also systems like Aadhaar [11] or the current mobile driving licence (mDL) standard [14] that include biometric information about the owner allowing verifiers to link a physical individual directly to a digital document without the need for any physical form of ID. These approaches cover the widest range of possible applications, but at the same time they also come with the most problems in regards to privacy.

Arguably the most important of which regards user linkability. If a single digital identity is used across a wide range of services, those services should not be able to link their users based on the digital identity they used to create their accounts. Aadhaar for example faces this issue because it reuses the same global identifier (12-digit Aadhaar number) for every service, while the European eID avoids that problem by deriving service specific sub-identifiers. While one approach is clearly preferable to the other, both concepts suffer a second linkability issue at the provider side. As long as digital identities are hosted by a central authority, this authority can track when, where, and how

a digital identity is being used. Therefore, these systems require significant trust from their users to be accepted and at least in the case of Aadhaar there have been numerous reports indicating that this trust might be misplaced [9, 23, 31].

The only way to mitigate this issue is to decentralize digital identities by storing them on user controlled devices like it is done with mDL. If verification happens locally between two devices, there is no central provider that could track the usage of a digital ID. However, mDL and similar systems have to include the biometrics of the identity holder so multiple verifiers can easily find out if they have seen the same ID or not. This brings us back to unlinkability as a key requirement for distributed digital identity systems.

The general unlinkability requirement can be split up into several more specific sub-requirements: First, we need a way to compare biometric features without disclosing enough information about them to make them linkable. Thankfully, there are multiple research efforts directed towards implementing privacy preserving biometrics, which demonstrate that unlinkability on a biometric level is achievable [19]. Second, the cryptography used to sign digital documents must also maintain unlinkability. Approaches based on zero-knowledge proofs like IBM's identity mixer [3] or the W3C's verifiable credentials [24] are promising solutions for this and should lead to future digital identity systems that provide unlinkability on a biometric and a cryptographic level. However, there is a third layer of unlinkability that must be considered when designing a distributed digital identity system, which brings us to the main issue discussed in this work:

In every decentralized identity system, specific devices have to interact whenever a digital identity is being used. Even if the communication is safely encrypted, information like source and destination addresses or timestamps cannot be concealed from attackers listening in on the network. By removing centralized digital identity providers that could track the usage of digital identities, that capability moves to network providers that can see the traffic going back and forth in a distributed identity system. Therefore, every distributed digital identity system that wants to protect the privacy of its users must find a way to ensure unlinkability on a network level.

In this paper, we present a specific definition of network unlinkability for distributed digital identity systems. Based on this definition we propose a way to instrument the Tor network to provide the required unlinkability. Finally, we investigate the time it takes to provision Tor onion services in detail because this time is of crucial importance for our proposed approach towards achieving network unlinkability.

2 Network Unlinkability

As stated in RFC 6973 [5] unlinkability only has meaning within a system that defines items of interest (IOIs), the systems anonymity properties, and entities interested in linking data (attackers). Once those have been defined, unlinkability is achieved by making sure that attackers cannot tell if two IOIs are related or not. Since some of our attackers might resort to statistical analysis, we clarify here that we need attackers to be unable to tell with reasonable certainty if two IOIs are related or not.

2.1 Items of Interest

When talking about network traffic, IOIs are usually derived from network packets. The most significant attributes from a privacy perspective are the IP addresses (IPv4 or IPv6) of both sender and destination, the time when the packet was sent and the amount of transferred data.

In order to properly assess the value and significance of this information, it is important to know when it is produced, meaning when network communication takes place. For a distributed digital identity system to work, there are a few network communication paths that have to be used. First, the user-controlled device holding the relevant identity information (= *identity agent*) must communicate with a biometric *sensor*, to confirm that the physically present individual is actually the holder of the provided digital identities. Second, the identity agent has to provide digital credentials to the entity that needs to verify the identity (= *verifier*). Although *sensor* and *verifier* are likely operated by the same entity, they have to be treated as distinct devices if they are not running on the same device, which we believe to be unlikely. Consider a simple example like validating passports at airports: There would be hundreds of sensors needed to measure biometric information, but verifying the provided digital identities could (and most likely would) be centralized at a single instance.

2.2 Attackers

When thinking about network metadata it is tempting to assume attackers are able to see everything. However, the assumption that potential attackers are able to see 100% of the network all the time, makes it impossible to conceal any metadata from them, meaning that the system must not produce any privacy relevant metadata. As this is almost impossible to achieve, we assume more realistic scenarios:

At the very least attackers need to be able to monitor the network traffic of a single device that participates in a digital identity system. Obviously, a dedicated attacker could also gain access to the network traffic of multiple devices to learn if and when those devices interact. On a larger scale organizations like internet providers which handle network traffic for their customers could abuse their access to build extensive behavior profiles on their customers. On a national level, governments could force all internet providers to secretly monitor the usage of digital identities to monitor their population. Finally, we do already know that on an international level it is common to monitor large quantities of network traffic by wiretapping larger internet exchange nodes [17, 18].

When thinking about the capabilities of attackers, it is usually worth considering the motivation of attacker as well. On an individual level, an attacker might abuse an identity system to stalk a victim via his or her identity agent. Alternatively, an attacker could target a verifier to learn how many customers a service handles. This would fall under the corporate espionage category. Organizations collecting data about their users, generally want to use that information themselves or sell it to other interested parties. While it could be argued that data collection by a company to improve its services is not a bad thing, we consider extracting information about individuals from their network patterns (without their explicit consent) malicious and categorize it as a potential attack. Attacks on privacy happening on a national level are usually motivated by obtaining information for law enforcement. While enforcing laws is a reasonable goal and information about specific individuals would certainly be useful to it, not all countries have law enforcement that respects and protects the freedom and privacy of citizens. Therefore, a privacy-preserving digital identity system must consider law enforcement trying to monitor the population as an attacker. Finally, international surveillance is almost always conducted as part of state sponsored intelligence gathering. If we consider local law enforcement as malicious then that must also apply to foreign governments, which are far less likely to respect the privacy of non-citizens.

2.3 Implications

Any attacker monitoring network traffic can easily decide if two connections are related if they use the same source or destination IP address. Consequently, a privacy preserving digital identity system must find a way to either conceal IP addresses from attackers, which is not supported by current

network implementations or reduce the information value provided by the collected IP addresses. This could be achieved by sending all messages via randomly selected proxies. An attacker would still see IP addresses and could still link them, but since they are selected randomly linking them provides no relevant information to the attacker. However, even if the IP address itself is taken care of, attackers able to monitor the traffic of more than a single client could use statistical analysis to link packets based on the data size and timing of the transmitted packet. While it would be possible to prevent traffic correlation based on packet size and timing by introducing padding and delays, doing so would lead to a system with significant latency issues that would most likely be rejected by users. Therefore, we are forced to relax our unlinkability requirement in this area to specify that attackers with access to a large set of network traffic should be unable to extract IOIs belonging to a specific individual and attackers must still be unable to tell for the majority of IOIs if they are related or not.

Finally, it should be noted that if an attacker is able to monitor the traffic of an individual device, there is no need for IOIs to identify one communication partner. The mere existence of traffic informs the attacker that the monitored device is interacting, revealing sensitive metadata in the process. Such traffic will always be linkable, making this a threat that distributed digital identity system will have to live with.

3 An Unlinkable Distributed Network via Tor

The standard solution for concealing network IP addresses is onion routing [10]. It protects traffic by encapsulating it in multiple layers of encryption and routing the traffic through a series of proxies, with each server stripping off one layer of encryption. Onion routing can conceal the IP addresses of communication partners and makes traffic correlation attacks significantly harder [1].

For the purpose of this research we evaluated different approaches towards onion routing to learn which would be most suited for a privacy preserving distributed digital identity system. Fundamental changes to basic networking protocols like proposed by HORNET [4] would most likely provide sufficient unlinkability, but we do not see such approaches being deployed at a larger scale in the near future. Instead we investigated approaches that implement onion routing as an overlay network on top of the current Internet architecture, as these approaches can already be used in practice. The two most important contenders in this field are Tor [29]

and I2P [13]. The design of I2P is intended for building privacy preserving distributed systems on top, while the Tor project is mainly focused on enabling users to browse the web privately. It would therefore be a reasonable assumption that a distributed digital identity system should be built on top of I2P, as its design clearly tries to support this kind of scenario. Ironically, this is not the case. The Tor project has focused on a much more requested use-case, causing it to accumulate far more users and supporters, which in turn enabled it to develop advanced features like onion services that also allow building distributed systems within the Tor network. Compared to I2P, Tor's implementation has more features, is better tested and has a much larger network of relays to work with, making the Tor project the best candidate for building a privacy preserving distributed digital identity system.

3.1 Tor

Tor is an onion routing technology that anonymizes network traffic by tunneling it through several nodes. A connection established via the Tor network is referred to as a *circuit* and usually consists of three nodes. The *guard node* (also called entry node), a middle node, and an *exit node*. Multiple layers of encryption enforce that only the guard node knows the origin of the traffic and only the exit node knows the destination [7].

The overwhelming amount of Tor traffic is used to connect to public websites. Access to Tor onion services makes up only $\sim$1.5% of the total traffic going through the Tor network [30].

Tor keeps track of all nodes currently existing within the Tor network in the consensus. The consensus is created by a group of highly trusted relays, called the directory authorities, and updated hourly. Important for onion services is the fact that every 24 hours a new shared random value is generated and published by the directory authorities. Unless otherwise specified, time periods mentioned in this paper refer to the 24 hours a shared random value is valid and when values are derived from time periods, they are derived using the shared random value of the time period.

Onion services were initially created as an example for systems built on top of the Tor network [6] and enabled users to provide services without disclosing their IP addresses or even having a public IP address. It found reasonable acceptance within the Tor community and several projects (for example SecureDrop [25] and Cwtch [20]) were built on top of it. Over time, several critical issues with the previous version (V2) of the onion service protocol were identified, like the use of 1024-bit RSA and SHA1 which

are no longer considered secure. Those ultimately lead to the publication of version 3 (V3) of the onion service protocol. Unless otherwise specified all following descriptions refer to V3 onion services.

Onion services enable the operation of globally accessible anonymous services. While the details of the onion service specifications [26,27] are quite complicated, the basic concept is relatively easy: Servers select introduction nodes, which can be used to contact them. They publish these introduction nodes within the Tor network, so other clients can find them. Clients can then request a connection with the onion service at a rendezvous point of their choice via an introduction point. Establishing the connection via such a rendezvous point provides anonymity to both the client and the server.

3.1.1 Creation of onion services

Every onion service is identified by a master keypair.

- The private master key grants full control over an onion service. This key is only used to derive blinded signing keys. Onion services use a new blinded signing key for each 24 hour time period. The fact that these keys can be precomputed allows offline storage of the master key, as it is only occacionally needed to precompute another batch of blinded signing keys.
- The public master key is encoded within the address of the onion service. Along with an optional secret, which can be used to protect an onion service with a password, it forms the onion service credential that clients need to know in order to connect to an onion service.

In order to create a new onion service, the host has to first pick three introduction points. Any nodes within the Tor network can be picked for this purpose, by default they are chosen at random. To hide the network location of the onion service, the host establishes connections to all introduction nodes via Tor circuits and keeps them open as long as the service persists.

In the next step, a hidden service descriptor is created to inform clients about the introduction points of a service. Every descriptor is identified by a blinded public key, which is derived from the hidden service credential and the shared random value of the current time period. Most of the descriptor is encrypted, only information needed to store and distribute the descriptor, like version and lifetime is transferred in plaintext. All other fields, like the list of introduction points, are encrypted with another key derived from the hidden service credential and the current time period. If client authentication

is enabled, a second layer of encryption is introduced to ensure that only selected clients can read the descriptor.

Once the descriptor has been correctly generated, it must be published so clients can access it. This is done via a distributed hash table (DHT) split across a subset of Tor nodes referred to as the hidden service directory (HSDir). The location of the descriptor within the DHT is determined by hashing the blinded public key, along with the current time period and the replica number. Replicas are used to distribute descriptors randomly across the HSDir. Additionally, a spread is defined to upload a descriptor not only to the single node determined by the location, but also to the closest nodes within the hash table. So, if one node fails the descriptor remains reachable. By default, onion services use two replicas and spread the descriptor over four nodes, resulting in 8 descriptor uploads for every onion service.

3.1.2 Access to onion services

To connect to an onion service, a client must first know the onion address, which contains the public part of the master keypair. With that information (and an optional secret) the hidden service credential can be derived. That enables the client to calculate the identifier of the service descriptor it is looking for, by blinding the credential with the current shared random value. Hashing the blinded key along with the time period and replica number tells the client the location of the descriptor within the HSDir. By default clients randomly contact one of the three HSDir nodes closest to the calculated location.

Any request to the HSDir must be made via Tor to ensure that clients remain anonymous to the operators of the Tor relays hosting the hidden service directory. Once clients receive the desired descriptor, they can decrypt it by deriving the decryption key using the credential and the current time period.

The client then picks a random node within the Tor network as a rendezvous point and establishes a circuit to this node. Afterwards it connects to one of the introduction points stated in the descriptor and asks the onion service to meet at the chosen rendezvous point. The introduction point can forward the request through the still open circuit maintained by the host responsible for the onion service. Upon receiving the request, the onion service creates a circuit to the rendezvous point and has its circuit connected directly to the circuit of the client. At this point a circuit across six relays connects the client to the onion service.

3.2 Privacy issue with onion services

Onion services are essential for a privacy preserving distributed system on top of the Tor network, because they are the only way to conceal destination addresses. Without them, devices within the system would have no way to accept incoming connections making it impossible to form a distributed network. However, if we assume that devices are accepting incoming connections via Tor onion services, the question arises if the onion addresses become a new IOI that can be linked. At first glance this seems not to be the case because onion addresses are never visible to attackers monitoring the network. Unfortunately, a distributed network on top of Tor has to assume that devices share their onion address with all other devices making it effectively public knowledge. That means that attackers would most likely be able to learn the onion addresses used by specific identity agents, which is problematic due to an inherent issue of Tor onion services:

Every connection to an onion service requires fetching the current descriptor from the hidden service directory. So the relays forming the hidden service directory can now keep track of when and how often an identity agent is contacted. Instead of monitoring their victims network traffic, attackers just have to expand resources to become part of the hidden service directory and they can monitor a random share of onion services every day. Previous research [2, 12, 21] has demonstrated that this issue can be exploited to obtain significant information about onion services. From the Tor project perspective this issue is acceptable, as it does not compromise the anonymity of the communication partners, which is their primary focus. However, for an identity system that expects attackers are able to link onion addresses to identities, information about when and how often an onion service has been used must be protected.

3.3 Dynamic onion services

One way to avoid onion services from being linked is by creating onion services dynamically and only using each onion service for a single purpose. This is possible because onion services can be created without any manual configuration. Every Tor client connected to the Internet can deploy a fresh onion service within seconds. Deleting an onion service is even faster, since it instantly stops working when the circuits to the introduction points are closed.

That makes it possible to dynamically create onion services that are only used for short periods of time before being discarded. The hidden service directory only leaks information about specific onion addresses known to

the attacker while information about two different onion addresses remain unlinkable. While dynamic onion addresses solve one linkability issue, they introduce a new one: How can we share those short-lived onion addresses with future communication partners? If they are published, they become easily linkable for attackers, so this is not an option.

Ideally, there is a secure channel to exchange an initial onion address and afterwards onion addresses can be rotated by including a new onion address in every message. If that is not possible, a device could run a static onion service that can be queried to request an initial onion address that can be rotated dynamically in the future. This comes with other issues like how to handle DOS attacks that request an unlimited amount of onion addresses, but would enable the distribution of onion addresses. Note that while the initial distribution would be linkable (i.e. an attacker could find out when and how often addresses are handed out) all future communication would remain unlinkable.

Another question to ask, is if the Tor network would be able to handle the additional load introduced by instrumenting onion services in this way. A positive property of onion services is that they do not require any exit relays, as their traffic stays within the Tor network. So the part of the Tor network that is most susceptible to overload can be ignored for our considerations. The hidden service directory would be hit with a significantly larger amount of upload requests, but since it is distributed across lots of nodes the additional load is unlikely to become an issue. What would significantly change is the amount of onion services existing in parallel, since most players in a distributed system communicate with more than one party. That in turn increases the amount of introduction points needed and with it the number of active circuits. While that definitely puts additional strain on the Tor network, that load should scale without issues as long as a successful digital identity system relying on Tor encourages the deployment of additional Tor relays, this should not be an issue.

Performance problems are more likely to arise when operators try to deploy onion services dynamically. While Tor does not limit the amount of onion services a client can deploy, the current implementation refuses to re-use Tor relays that are already used in an active circuit. At some point all available relays in the Tor network are used for circuits to introduction points leaving Tor unable to either connect to a rendezvous point or create additional onion services. During our testing we confirmed that a Tor client can maintain more than 100 onion services in parallel without issues. However, when having to create a new onion service regularly, the creation time of onion

services turns out to be a problem. Before a client request can be answered, a new onion services has to be deployed so it can be included in the response. This effectively increases the latency of a digital identity system that is already slowed down by sending messages via a six-hop circuit even more because it has to wait for the new onion service to be deployed.

4 Deployment Time Analysis

To find out if it is feasible for a client to create new onion services dynamically, we need to quantify the negative performance introduced by it. To do this we measure the time between instructing a host to generate an onion service and clients being able to access it. Only if this time is sufficiently low, it may be acceptable to generate onion services on-demand, which is required to build a digital identity system relying on this feature.

Our measurement setup is inspired by previous work of Loesing et al. [16] and Lenhard et al. [15], but instead of measuring the time it takes to access an onion service, we measure the time it takes to create one.

4.1 Measurement Setup

We use the Tor Stem[1] library to generate onion services. Timing information is extracted from the log file created by Tor and event listeners attached via the Stem library. This allows measuring the time of the following events:

- Start connecting to introduction point,
- circuit to introduction point established,
- introduction point ready,
- service descriptor created,
- start upload to HSDir, and
- finish upload to HSDir.

No good solution was found to measure the time it takes Tor to select introduction nodes when creating a new onion service. It seems reasonable to assume that this time is insignificant for the overall latency, but it could be speculated that one host running many onion services could experience deteriorating performance as Tor does not reuse introduction points[2].

[1] https://stem.torproject.org/

[2] The official documentation still has an open TODO on picking nodes. However, a review of the Tor implementation revealed that this is the case.

All our tests were conducted with version 0.4.3.5 of Tor and ran on a virtual machine running Debian 10, which was monitored to ensure that no local limitations regarding CPU, bandwidth, latency or memory would impact our measurements. The Internet connection (1GBit/s, low latency) was constantly monitored to be working within "normal" parameters, in order to assure that we do not accidentally measure latency effects or outages introduced primarily through our own Internet link.

To ensure that measurements do not influence each other, a new Tor process is completely bootstrapped within a fresh Docker container for every onion service. Our test system runs one test at a time to avoid different onion services impacting each other. To mitigate the effects of possible issues with our Internet connection or the Tor network, tests are conducted in a loop. Every iteration of the loop tests every configuration once. This loop ran more than 1500 times over a period of 10 days to obtain a sufficiently large sample size.

The Docker container specification with our measurement implementation is available at https://github.com/mobilesec/onion-service-time-measurement to enable other researchers to reproduce our measurements.

4.2 Measured Configurations

As already mentioned, onion services are still in development and can, therefore, currently be deployed in different configurations. To find out if the method of deployment has an impact on the provisioning time of onion services, four different types were measured:

1. V2: A V2 onion service with default parameters: Old, no longer recommended version, which was mainly included to enable comparisons with previous research.
2. V3: A persistent V3 onion service with default parameters.
3. Ephemeral: A V3 onion service with default parameters which can only be created via the control protocol and will only exist as long as the control connection to the Tor instance is maintained.
4. Vanguard: A V3 onion service with the Vanguard [22] extension to harden it against different deanonymization attacks.

4.3 Results

Figure 1 provides a good summary of the results of our analysis. The changes implemented by V3 of the onion service protocol have significantly improved

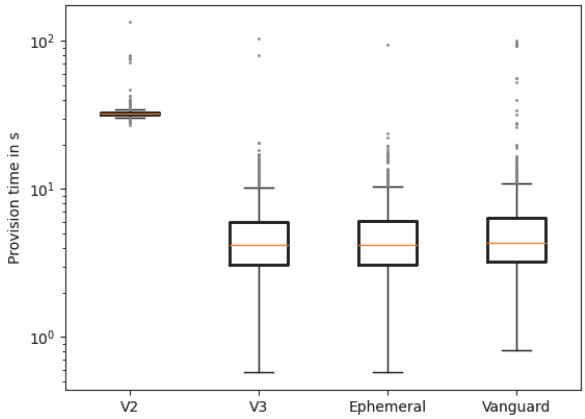

Figure 1 Overview of provisioning times.

deployment times from about half a minute to less than 10 seconds. There are no significant differences between normal and ephemeral onion services, which is no surprise considering that the only difference between those is the persistence of cryptographic keys on disk. The Vanguard extension also shows no significant changes in provisioning time, which is unexpected because modifying Tor's behavior via the control protocol should actually cause a performance overhead, but is apparently not relevant for our measured scenario.

4.3.1 Provisioning stages

A potentially interesting explanation for the significant differences in provisioning times between V2 and V3 is provided by Figure 2. It splits the provisioning into three stages:

1. The time it takes the host to establish the introduction points for the onion service,
2. the time it takes to generate a descriptor for uploading after introduction nodes have been established, and
3. the time it takes to actually upload the descriptor.

The first fact to note here is that V2 onion services appear to take much longer to generate their service descriptors. Since there was no obvious reason for such a significant performance difference, we investigated the source code and found that the current implementation of Tor V2 onion services waits 30 seconds before uploading a descriptor. There is no explanation in the specification [26] as to why this delay is necessary and the source code only

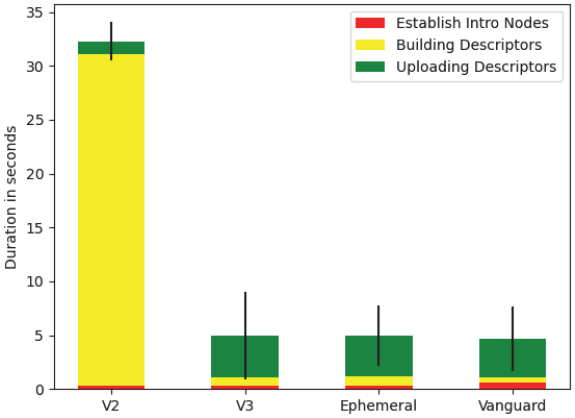

Figure 2 Composition of provisioning times.

comments that the delay is introduced to ensure that the descriptor is stable. Since V2 onion services are going to be disabled in October 2021 [28], we did not spend additional time on analyzing this issue.

Figure 2 also reveals other less obvious, but interesting, aspects. For example, it confirms a suspicion that is hard to verify on the logarithmic scale of Figure 1, namely the fact that for V2 onion services, upload times have not only less impact on the total provisioning time, but are also lower in absolute numbers. The exact reasons for this behavior is analyzed in Section 4.3.2.

Another interesting observation is the fact that establishing introduction nodes is insignificant to the provisioning time of an onion service across all configurations. This observation is however not fully correct because as already mentioned all our measurements were conducted with fully bootstrapped Tor instances. During the bootstrapping process, several circuits (in our experiments we encountered between 2 and 15 circuits during bootstrapping) are prepared, so they can be used for later connections. In our setup, these circuits are always used to connect to introduction points, so our measured time for the creation of introduction points does not include the circuit creation time. Since Tor already collects detailed metrics on circuit creation time [30], there was no reason to analyze them ourselves.

What is worth noting, is that the Vanguard plugin almost doubles the introduction node building time, without impacting the overall provisioning time. At first glance, this seems to imply that Vanguard is actually decreasing the descriptor creation time, which is unlikely considering the fact that Vanguard makes no changes to service descriptors. Instead, the difference is

caused by the fact that the generation and derivation of all keys required for creating a service descriptor take a constant amount of time and can already start before the introduction points have been selected. We verified this by deploying onion services with 10 introduction points. Naturally, they needed more time to establish their introduction points, but they still finished creating their descriptors at the same time as services with only three introduction points. This shows that the time needed to establish introduction points is currently irrelevant for the provisioning time of an onion service.

Our final observation is that the descriptor upload is the most significant factor for total provisioning time in current onion service configurations, so we look at them in more detail.

4.3.2 Descriptor upload times

Our results for descriptor upload times have to be put in context to be understood further: Every onion service uploads its descriptor to several nodes on the hidden service directory. The number can be configured by each service, but the defaults are three nodes for V2 descriptors and four nodes for V3. Both are uploaded in two replicas, so in total there are 6 and 8 uploads. Additionally, the V3 onion service specification requires them to always be valid in two time periods, the previous one and the current one. So when creating a new V3 service from scratch (as done by our test setup) 16 descriptors are uploaded initially.

When Tor clients try to access an onion service, they use their current time period. The previous one is only uploaded to avoid synchronization issues with clients that are still in the previous time period. Tor clients randomly pick one out of only three nodes from one of the two replicas. The fourth upload in V3 is only there to handle situations where a HSDir node goes offline. This means that a single upload could be sufficient to allow an incoming connection. Unless there are any issues with synchronization or failing nodes, six uploads already enable full connectivity. Our measurement setup was not designed to take this into account. Instead, we assume that a descriptor has been successfully published when half of all uploads (3 for V2 and 8 for V3) have been completed. The upload time in Figure 2 shows how long it took on average to complete half of all uploads. This decision removes the impact of very slow uploads and tries to find a middle ground between trying to find the earliest time when connections are possible and the time when connections are almost certain to succeed without retries.

Figure 3 shows the duration of individual descriptor uploads. The majority of upload requests finish in less than 5 seconds and almost all uploads

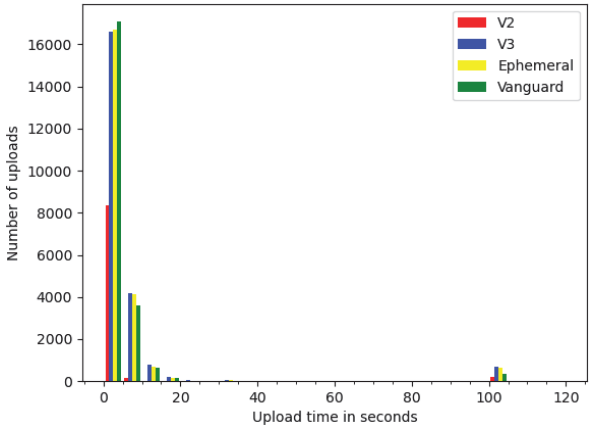

Figure 3 Time it took individual uploads to complete.

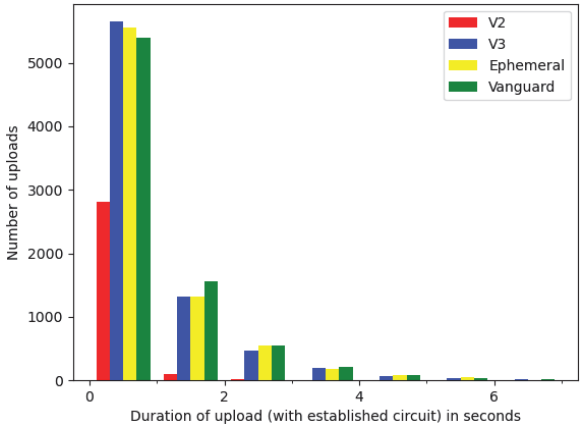

Figure 4 Time to upload descriptor via established circuit.

complete after 20 seconds. A noteworthy result of our measurement is an unexpectedly high number of upload requests that take between 100 and 105 seconds, which occurs for all measured configurations, but happens less often for onion services with the Vanguards extension. To further analyze this behavior we conducted a second smaller experiment by running the loop only 500 times and additionally tracking the time when upload circuits were completed. This allows us to split the upload time into the time it took to create a circuit and the time it took to actually upload the descriptor.

Figure 4 shows that plain uploads hardly ever exceed five seconds and even the slowest single upload we measured only took 12 seconds to

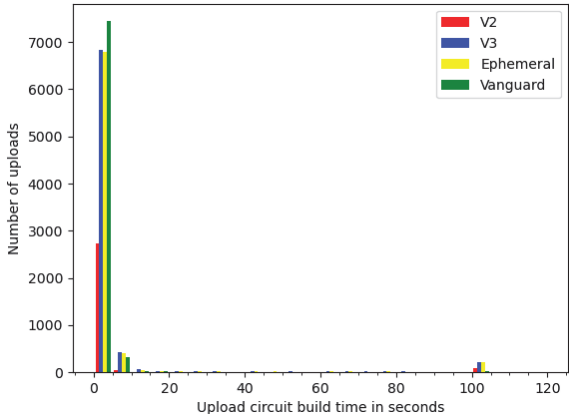

Figure 5 Time to create upload circuit.

complete. The unexplained 100 second delay is only present in the circuit creation time. This makes sense because this delay only happens when Tor fails to open a circuit to a hidden service directory. In this case a 100 second timeout occurs before another attempt is made. This also explains why Vanguards has a positive effect on this issue. It selects a subset of candidate nodes for the second and third hop of a circuit and tries to reduce the risk of picking a malicious node. Apparently, this also reduces the risk of picking nodes, that cause circuit creation attempts to fail, increasing the overall performance and reliability.

Another interesting result in this context is the fact that some circuits fail again after this 100 second timeout. In this case Tor does not wait and try for a third time, but instead abandons the upload attempt entirely. This does not result in any error displayed to the user, because the onion service concept is redundant and a single failed upload has no significant impact on the availability of an onion service. During our experiments we experienced an upload failure rate of about 1% for upload requests without Vanguard and a failure rate of about 0.8% for uploads with Vanguard.

Figure 4 confirms that V2 descriptors are published faster than V3, which is most likely caused by the much larger descriptor size in V3. Figure 6 provides a zoom-in on circuit build times below 8 seconds to facilitate comparison with Figure 4 and shows that the circuit creation time has more impact on how long it takes to publish a descriptor than the actual upload. An interesting observation is that our results seem to indicate that V2 upload circuits are created faster than V3 upload circuits. This effect is again caused by the fact that our Tor binaries were fully bootstrapped

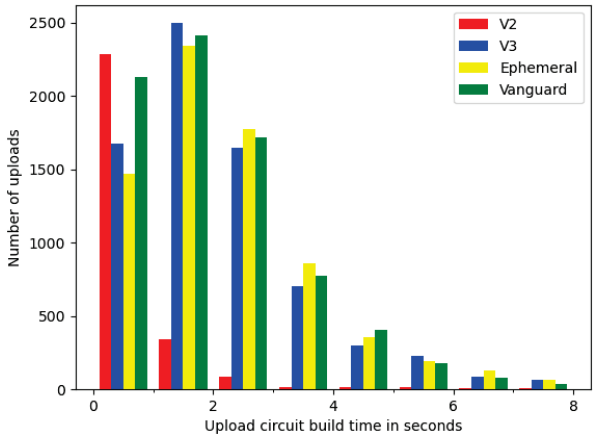

Figure 6 Zoom-in on uploads circuit build times below 8 seconds.

before any measurements were conducted, which allows Tor to cannibalize general circuits for uploads if there are any available. Since cannibalization is much faster than creating a circuit from scratch, this means that some upload circuits can be created faster than the rest. The lower number of uploads in the V2 onion service specification increased the relative impact of these cannibalized circuits, creating the impression that V2 upload circuits are created faster. Unfortunately, we could not properly quantify the impact of this issue, so we cannot say if there are any other factors contributing to the increased circuit creation time in V3.

5 Conclusion

In this article we discussed how network unlinkability can be defined for a privacy preserving distributed digital identity system. Evaluating several projects that try to provide anonymity on a network level led us to identify that the Tor project is the most suited platform to build upon. We suggested a concept that achieves the required network unlinkability by dynamically creating and removing onion services, making it impossible for an attacker to link them together.

Furthermore, we present one essential building block for instrumenting onion services dynamically, an in-depth investigation into the deployment times of onion services. At the moment the process of creating a new onion service takes about 5 seconds. This is clearly too much to dynamically create onion services on the fly. Since the majority of time is spent creating circuits

and timing is irrelevant for static onion services, it is unlikely that this number will improve in the near future.

However, there are two potential strategies to make dynamic onion services viable in practical applications. If the need for new onion addresses is predictable, clients could set them up in advance so they can hand them out when necessary. As long as onion services can be generated fast enough to always have some available, this effectively removes the latency impact of having to create onion services. Alternatively, clients could try to exchange service descriptors directly, instead of exchanging onion addresses. This would be interesting because the majority of onion service deployment time is spent on uploading descriptors to the hidden service directory. Removing this step would cut the onion service creation time down to about 1 second and also improve the time it takes the client to connect, because it does not need to fetch the service descriptor from the hidden service directory.

While it is still unclear at this point in time, which kind of digital identity systems will turn out to be the most successful, addressing the issue of network unlinkability is definitely one of the challenges that supporters of distributed digital identity systems must address. Our contribution highlights this challenge and provides some ideas how system designers might be able to overcome this issue.

Acknowledgments

This work has been carried out within the scope of Digidow, the Christian Doppler Laboratory for Private Digital Authentication in the Physical World. We gratefully acknowledge financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development and the Christian Doppler Research Association, 3 Banken IT GmbH, Kepler Universitätsklinikum GmbH, NXP Semiconductors Austria GmbH & Co KG, and Österreichische Staatsdruckerei GmbH and has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria..

References

[1] James Ball, Bruce Schneier, and Glenn Greenwald. Nsa and gchq target tor network that protects anonymity of web users. https://www.theguardian.com/world/2013/oct/04/nsa-gchq-attack-tor-network-encryption, 2013.

[2] Alex Biryukov, Ivan Pustogarov, and Ralf-Philipp Weinmann. Trawling for Tor Hidden Services: Detection, Measurement, Deanonymization. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, page 80–94, USA, 2013. IEEE Computer Society.

[3] Jan Camenisch and Els Van Herreweghen. Design and Implementation of the Idemix Anonymous Credential System. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS '02, page 21–30, New York, NY, USA, 2002. Association for Computing Machinery.

[4] Chen Chen, Daniele E. Asoni, David Barrera, George Danezis, and Adrain Perrig. Hornet: High-speed onion routing at the network layer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 1441–1454, New York, NY, USA, 2015. Association for Computing Machinery.

[5] A. Cooper, H. Tschofenig, B. Aboba, J. Peterson, J. Morris, M. Hansen, and R. Smith. Privacy Considerations for Internet Protocols. *RFC 6973*, 2013.

[6] Roger Dingledine. Next Generation Tor Onion Services. DEF CON 25, 2017.

[7] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*. USENIX Association, 2004.

[8] European Comission. eHealth and COVID-19. https://ec.europa.eu/health/ehealth/covid-19_en, 2021.

[9] Mengle Gautam. Major Aadhaar data leak plugged: French security researcher. https://www.thehindu.com/sci-tech/technology/major-aadhaar-data-leak-plugged-french-security-researcher/article26584981.ece, 2019.

[10] David Goldschlag, Michael Reed, and Paul Syverson. Onion routing. *Communications of the ACM*, 42(2):39–41, 1999.

[11] Government of India. Unique Identification Authority of India. https://uidai.gov.in/, 2009.

[12] Tobias Höller, Michael Roland, and René Mayrhofer. On the state of V3 onion services. In *Proceedings of the ACM SIGCOMM 2021 Workshop on Free and Open Communications on the Internet (FOCI '21)*, pages 50–56. ACM, August 2021.

[13] I2P. The Invisible Internet Project. https://geti2p.net/, 2021.

[14] International Organization for Standardization. Personal identification — ISO-compliant driving licence — Part 5: Mobile driving licence (mDL) application. Standard ISO/IEC TR 29110-1:2016, Geneva, CH, 2016.

[15] Jörg Lenhard, Karsten Loesing, and Guido Wirtz. Performance Measurements of Tor Hidden Services in Low-Bandwidth Access Networks. In *Applied Cryptography and Network Security*, pages 324–341, Berlin Heidelberg, 2009. Springer.

[16] Karsten Loesing, Werner Sandmann, Christian Wilms, and Guido Wirtz. Performance Measurements and Statistics of Tor Hidden Services. In *2008 International Symposium on Applications and the Internet*, pages 1–7, Turku, Finland, 2008. IEEE.

[17] Even MacAskill, Julian Borger, Nick Hopkins, Nick Davies, and James Ball. GCHQ taps fibre-optic cables for secret access to world's communications. https://www.theguardian.com/uk/2013/jun/21/gchq-cables-secret-world-communications-nsa, 2013.

[18] Andre Meister. How the German Foreign Intelligence Agency BND tapped the Internet Exchange Point DE-CIX in Frankfurt, since 2009. https://netzpolitik.org/2015/how-the-german-foreign-intelligence-agency-bnd-tapped-the-internet-exchange-point-de-cix-in-frankfurt-since-2009/, 2015.

[19] Iynkaran Natgunanathan, Abid Mehmood, Yong Xiang, Gleb Beliakov, and John Yearwood. Protection of privacy in biometric data. *IEEE Access*, 4:880–892, 2016.

[20] Open Privacy Research Society. cwtch. https://cwtch.im/.

[21] Gareth Owen and Nick Savage. Empirical analysis of Tor Hidden Services. *IET Information Security*, 10(3):113–118, 2016.

[22] Mike Perry. The Vanguards Onion Service Addon. https://github.com/mikeperry-tor/vanguards.

[23] Khaira Rachna. Rs 500, 10 minutes, and you have access to billion Aadhaar details. https://www.tribuneindia.com/news/archive/nation/rs-500-10-minutes-and-you-have-access-to-billion-aadhaar-details-523361, 2018.

[24] Manu Sporny, Dave Longley, and David Chadwick. Verifiable Credentials Data Model 1.0. https://www.w3.org/TR/vc-data-model/, 2019.

[25] Aaron Swartz. Securedrop. https://github.com/freedomofpress/securedrop.

[26] The Tor Project. Tor Rendezvous Specification. https://github.com/torproject/torspec/blob/master/rend-spec-v2.txt.
[27] The Tor Project. Tor Rendezvous Specification – Version 3. https://github.com/torproject/torspec/blob/master/rend-spec-v3.txt.
[28] The Tor Project. Onion Service version 2 deprecation timeline. https://blog.torproject.org/v2-deprecation-timeline, 2020.
[29] The Tor Project. The Tor Project. https://www.torproject.org/, 2021.
[30] The Tor Project. Tor Metrics. https://metrics.torproject.org, 2021.
[31] Srinath Vudali. Aadhaar details of 7.82 crore from Telangana and Andhra found in possession of IT Grids (India) Pvt Ltd. https://timesofindia.indiatimes.com/city/hyderabad/aadhaar-details-of-7-82-crore-from-telangana-and-andhra-found-in-possession-of-it-grids-india-pvt-ltd/articleshow/68865938.cms, 2019.

Biographies

Tobias Höller is currently a university assistant and PhD candidate at the Institute of Network and Security at Johannes Kepler University Linz. He obtained his Master's degree at Johannes Kepler University Linz. His research interests are onion routing (specifically Tor) and digital identity systems.

Michael Roland is a post-doc researcher at the Institute of Networks and Security at Johannes Kepler University (JKU) Linz, Austria. He is also a lecturer at the University of Applied Sciences Upper Austria in Hagenberg. His main research interests are digital identities, NFC, smart cards, and wireless technologies with focus on security and privacy. He holds a B.Sc. and a M.Sc. degree in Embedded Systems Design (University of Applied Sciences Upper Austria, 2007 and 2009) and a Ph.D. (Dr. techn.) degree in Computer Science (Johannes Kepler University Linz, Austria, 2013).

René Mayrhofer is currently heading the Android Platform Security team at Google and tries to make recent advances in usable, mobile security research available to the Billions of Android users. He is on leave from the Institute of Networks and Security at Johannes Kepler University Linz (JKU), Austria, where he continues to supervise PhD and Master students. His research interests include computer security, mobile devices, network communication, and machine learning, which he currently brings together in his research on securing mobile devices.