Virtual Private Services

Karsten Loesing and Guido Wirtz
karsten@torproject.org, guido.wirtz@uni-bamberg.de

Tor Tech Report 2008-07-001*
July 25, 2008

Abstract

Providing an Internet service is no longer only a domain of organizations and enterprises, but becomes increasingly popular for private persons. Virtual private services are defined as services that are offered by private users over public Internet connections and for a limited, possibly changing set of other private users. Virtual private services demand extended security properties, e.g. hiding service activity from unauthorized clients, that are less important in public services. Two common approaches exist to utilize Tor hidden services for offering virtual private services. These are examined for security problems that might be privacy-relevant for the service provider. Subsequently, an extension of the hidden service protocol is presented that is designed to better support virtual private services. This design is not meant to replace the existing hidden service design, but should complement it.

1 Problem Statement

Providing an *Internet service* has first and foremost been a domain of organizations and enterprises. Dedicated and well-equipped servers handle thousands of service requests simultaneously. This makes sense for a large number of applications ranging from websites to Internet relay chat. In the area of privacy-enhancing technologies most effort has been dedicated on protecting the privacy of clients of such public Internet services.

In recent years, the decreasing costs of processing power, memory, and network bandwidth have enabled private persons to set up and operate their own Internet services. In most cases these *private services* are not meant to replace public services at all. A possible scope of private services is to offer them over the public, untrustworthy Internet, but only to a limited, possibly changing set of clients. In the following these services will be referred to as *virtual private services* following virtual private networks that allow creation of a secure private network on top of an untrustworthy public network.

In contrast to public services, virtual private services exhibit specific security requirements that stem from the fact that they are provided by private users and not by enterprises or organizations:

*This report was presented at Hot Topics in Privacy Enhancing Technologies (HotPETs 2008), Leuven, Belgium, July 2008.

Hide Server Location The location of the server reveals a lot of information about the service provider and should therefore be hidden. If a service is provided on a laptop or mobile device, the service provider might even be tracked while being under way as demonstrated in [3].

Prevent Unauthorized Access Access attempts performed by non-authorized users need to be blocked as early as possible. Since most private services still have limited resources, unauthorized access attempts may not be exploited to perform denial-of-service attacks.

Conceal Service Activity Non-authorized users may not be able to learn about service activity. In some cases service activity is consistent with the service provider's online activity. An adversary could exploit knowledge about service activity over time to learn useful information like timezone or personal online behavior about the service provider. One particular requirement of virtual private services is simplicity to remove authorization from clients. In such a case, the service should appear unavailable again to the former client. This is useful for temporarily granting access to a service.

Impede Service Usage Profiles Network components should not be able to generate profiles of client requests to a private service, even if requests cannot be attributed to specific clients.

One example for private services is presence-aware real-time communication. In a serverless instant messaging system the end user can be considered as service provider who offers a presence and messaging service to his communication partners. Usually, instant messaging users want to hide their presence from the public and avoid being annoyed by unsolicited messages from strangers. When it comes to communication, the possibility to gently remove authorization by letting the service disappear gains particular important, rather than explicitly explaining removal to a communication partner. [6, 7] contain proposals of such a privacy-protecting instant messaging system.

The anonymous communication system Tor [2] provides anonymity to its users by relaying traffic over a network of relay nodes. A client that wants to communicate anonymously creates a *circuit* consisting of three randomly selected Tor relays. The last relay in a circuit connects to the public server that the client actually wants to talk to. All messages between client and the last relay are encrypted in multiple layers, which is the reason for Tor's name: *The Onion Routing*. The idea is that no single entity but the client learns where a circuit starts and where it ends.

In addition to providing anonymity for clients, Tor enables users to offer *location-hidden services* that conceal the location of a server. Tor implements this by connecting two circuits—one created by a client, the other by a hidden server—at a commonly agreed Tor relay, the so-called *rendezvous point*. A rendezvous point passes end-to-end encrypted messages from client to hidden server and vice versa. In order to protect rendezvous points from attacks, a hidden service picks a set of Tor relays as *introduction points* which work like rendezvous points, but only transfer a single message containing the rendezvous request. While rendezvous points are established for a single connection, introduction points accept multiple connection requests and are set up for an interval of hours or even days. Finally, a hidden service publishes a *hidden*

service descriptor, containing a signed list of introduction points, to a set of *directory servers* from which it can be downloaded by clients.

When using Tor hidden services for virtual private services, the first goal of hiding server location is achieved. However, Tor hidden services—just as little as any other pseudonymous communication system—offer no built-in support for client authorization. The reason is that the primary purpose of hidden services is to offer services to the anonymous public, at what client authorization seems to be contradictory. The earliest time to perform client authorization is after an anonymous client has successfully established a connection to a hidden service. Ideally, authorization does not leak any information that could be used by unauthorized users to identify the hidden service.

There are two rather different approaches to implement virtual private services with Tor hidden services: In the first approach, a service provider sets up a single hidden service for all users and performs user authorization after successful connection establishment. Albeit being simple, there is no way to hide the existence of a service once issued to a client, putting the private service at risk on being attacked or profiled. A second approach is to configure a separate hidden service for each client in addition to a later authorization when the connection is established.[1] Although this gives a service provider much more control over visibility of his service, it uses up lots of resources from the Tor network and leaks links between provided hidden services to the network.

The approach taken here is to extend the hidden service protocol to support both, public services to anonymous users and private services to authorized clients only. The main idea is to keep the Tor hidden service protocol sequence unchanged, but modify single steps to fulfill the extended security requirements of virtual private services.

The next section contains a description of the cryptography behind the Tor hidden service protocol. The security issues of either of the two approaches to implementing virtual private services using Tor hidden services are discussed in Section 3. The new approach to extend the hidden service protocol for virtual private services is presented in Section 4. Section 5 contains an analysis of security properties of the new approach. Section 6 concludes.

2 Hidden Service Protocol

A description of the Tor hidden service protocol [1, 2] and of the underlying cryptography is necessary to understand the current security problems and proposed extensions. Figure 1 shows an overview of exchanged messages while Figure 2 contains the corresponding cryptographic message contents. Table 1 lists all symbols used in Figure 2 and throughout the rest of this report. All messages (besides message 5) are sent over Tor circuits and are therefore end-to-end encrypted.

Suppose that a user Bob wants to establish a new hidden service. For this purpose, Bob sends his signed public permanent key PK_B to a small number of Tor relays requesting them to act as

[1] A superficially neat idea is to advertise a separate hidden service for each user, but omit subsequent client authorization in the belief that hiding a service prevents other people from accessing it. The hidden service design does not conceal service existence from directory servers and introduction points. These could easily access a hidden service that performs no further authorization. Therefore, separate client authorization is taken for granted here.

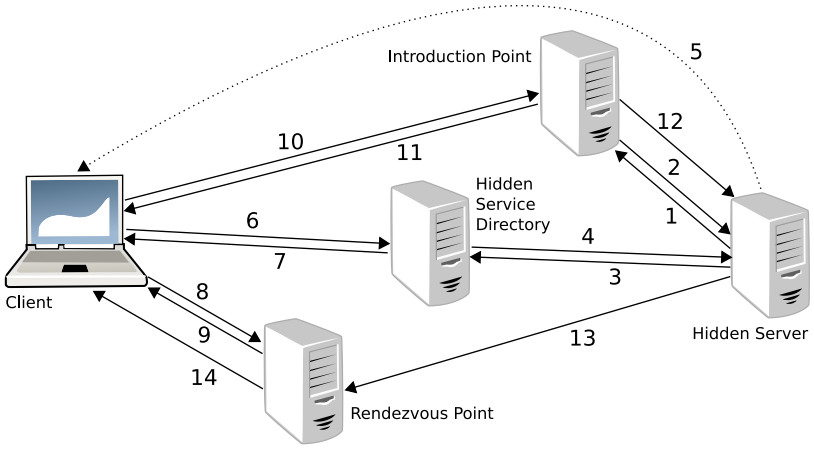

Figure 1: Overview of the hidden service protocol

his introduction points. This happens in step 1 whereupon Bob receives an acknowledgement in step 2. Next, the hidden service creates a hidden service descriptor including his public permanent key, a timestamp T, a list of introduction points $IPO_{1...n}$, and a signature of these data, created with his private permanent key PK_B^{-1}. The hidden service publishes this descriptor to the directory servers in step 3 and receives an acknowledgement in step 4. Then, the hidden service can tell his service address which is equal to the permanent key identifier $H(PK_B)$ to his clients out of band in step 5.

Whenever a user Alice wants to access Bob's hidden service, she looks up the current descriptor from a directory server by requesting an entry for the service's permanent key identifier in step 6. Assuming that she receives a correct descriptor in step 7, she picks a random relay and requests it to act as rendezvous point on her behalf by sending a single-use rendezvous cookie RC to it in step 8 which is acknowledged in step 9. Next, Alice picks one introduction point from the descriptor and sends to it an introduction request in step 10 consisting of the unencrypted permanent key identifier and an encapsulated message. The latter is encrypted using the service's public permanent key and contains contact information of the client's rendezvous point, the rendezvous cookie, and the first half of a Diffie-Hellman handshake g^x. If the introduction point recognizes the permanent key identifier, it acknowledges the request in step 11 and forwards the encrypted part of the message to its hidden service in step 12. Bob contacts the rendezvous point and sends to it the rendezvous cookie and the second half of the Diffie-Hellman handshake consisting of g^y and $H(g^{xy})$ in step 13. The rendezvous point, upon recognizing the rendezvous cookie, forwards the request to the client in step 14, finally establishing an end-to-end encrypted connection between Alice and Bob.

3 Security Analysis of Private Tor Hidden Services

A service provider can take (at least) two different approaches to offer a virtual private service to a limited set of clients using Tor hidden services. These two possibilities are discussed here together with the possible security issues they raise.

$$\text{Bob} \rightarrow \text{IPO} : PK_B, N, S_{PK_B^{-1}}(PK_B, N) \quad (1)$$

$$\text{IPO} \rightarrow \text{Bob} : - \quad (2)$$

$$\text{Bob} \rightarrow \text{Dir} : H(PK_B), M = (PK_B, T, IPO_{1\ldots n}), S_{PK_B^{-1}}(M) \quad (3)$$

$$\text{Dir} \rightarrow \text{Bob} : - \quad (4)$$

$$\text{Bob} \rightarrow \text{Alice} : H(PK_B) \quad (5)$$

$$\text{Alice} \rightarrow \text{Dir} : H(PK_B) \quad (6)$$

$$\text{Dir} \rightarrow \text{Alice} : M = (PK_B, T, IPO_{1\ldots n}), S_{PK_B^{-1}}(M) \quad (7)$$

$$\text{Alice} \rightarrow \text{RPO} : RC \quad (8)$$

$$\text{RPO} \rightarrow \text{Alice} : - \quad (9)$$

$$\text{Alice} \rightarrow \text{IPO} : H(PK_B), E_{PK_B}(RPO, RC, g^x) \quad (10)$$

$$\text{IPO} \rightarrow \text{Alice} : - \quad (11)$$

$$\text{IPO} \rightarrow \text{Bob} : E_{PK_B}(RPO, RC, g^x) \quad (12)$$

$$\text{Bob} \rightarrow \text{RPO} : RC, g^y, H(g^{xy}) \quad (13)$$

$$\text{RPO} \rightarrow \text{Alice} : H(g^{xy}) \quad (14)$$

Figure 2: Hidden service cryptographic protocol

3.1 Single Hidden Service for All Users

In the first scenario a service provider sets up a single hidden service for all his users and performs user authorization after connection establishment. Whenever the service provider wants to grant access to a new user, he tells her his hidden service address and hands out new credentials for the subsequent authorization. The other way round, when he wants to remove authorization for a specific client, he removes her credentials from the list of permitted clients.

There are a number of *security problems* in this protocol for virtual private services. The main reason for most of these problems is the unrestrained propagation of the hidden service's permanent key identity:

Unauthorized Access Attempts Non-authorized clients can open an unrestricted number of connections to the hidden service and attempt to access it before failing at subsequent client authorization. Even worse, with every established connection a non-authorized client wastes some of the hidden service's resources that has to build or extend a circuit to the client's rendezvous point. This attack can be performed by all former clients that have been removed by the service provider but still know the hidden service address. Even though introduction points and directory servers also learn about a hidden service address and could attack the service, they cannot relate it to a specific virtual private service and therefore have no such incentive.

Service Activity Former clients can track the service's activity even if they are not authorized to access it any more. The easiest way to do so is to periodically request the service's descriptor. Further, an adversary could run one or more relays and wait to be picked as

Table 1: Symbols used in cryptographic messages

Alice	authorized client
Bob	hidden service
Carol	another authorized client
CK_{AB}	A's client key to access hidden service B
Dave	formerly authorized client
DC_{AB}	A's descriptor cookie to access hidden service B
Dir	hidden service directory
$E_{PK}()$	encryption with public key PK
g^x, g^y, g^{xy}	Diffie-Hellman public keys and shared secret key
IK_i	introduction key generated for introduction point i
IPO	introduction point
$K_{SK}()$	symmetric encryption with secret key SK
N	nonce
PK_B, PK_B^{-1}	public and private permanent key of hidden service B
PK_{AB}, PK_{AB}^{-1}	A's public and private permanent key to access hidden service B
RC	rendezvous cookie
RI	replica index
RPO	rendezvous point
$S_{PK^{-1}}()$	signature with private key PK^{-1}
T	timestamp
TP	time period
$H()$	secure hash function

introduction point. At last, the adversary might be able to collaborate with one of the directories to passively monitor descriptor publications.

Anonymous Service Usage A formerly authorized client could monitor anonymous service usage by setting up one or more relays and wait to be picked as introduction point for the service. She might even attack other introduction points that work on behalf of the service until one of her relays is selected. Likewise, if she is able to collaborate with one of the directories, she can passively track requests for a service's descriptor and derive anonymous service usage from that.

3.2 Separate Hidden Service For Each Client

An alternative approach is establishment of a separate hidden service for each client. In this setting the service provider hands out a new hidden service address together with authorization credentials to a new user. In order to remove a client's authorization, the service provider simply stops advertising the corresponding hidden service.

From this follows that the attacks as described for the first approach do not apply. The only entities that can relate a hidden service address to a service provider are the service and one client. As soon as the hidden service is stopped, the removed client cannot learn anything

about the service, because she doesn't know the other hidden service addresses that the service provider uses. However, two problems remain:

Network Load The effort that is required to set up a separate hidden service for each client is tremendous. The service provider needs to establish a separate set of introduction points and publish separate hidden service descriptors. This limits the number of clients that can reasonably be authorized and puts significant load on the network.

Links Between Pseudonyms A service provider who offers more than one hidden service at the same time might occasionally establish multiple introduction points on the same relay. Further, he is likely to publish several hidden service descriptors in quick succession. These events give hints to introduction points and directory servers that two or more hidden services are run by the same provider. A once-authorized client that learns about this relation can attack or profile a service provider when her authorization is removed by using one of the other hidden service addresses.

4 Extension of the Hidden Service Protocol

The main idea of the approach taken here is to reduce the amount of information that directory nodes and introduction points learn about a service. Similarly to the second approach described above, access to hidden services is issued on a per-user basis, but without the massive overhead of keeping a number of circuits open that is proportional to the number of authorized clients. The sequence of exchanged protocol messages remains unchanged, but single steps are modified to fulfill the extended security requirements of virtual private services. The specification details can be found in [4]. Figure 3 contains the extended hidden service protocol that is developed in the course of this section.

4.1 New Introduction Key for Introduction Points

In the first place the information that an introduction point learns about a hidden service is reduced. An introduction point is not required to learn *the* identity of a hidden service, but only *an arbitrary* identity created by the hidden service to perform its task, i.e. match incoming client requests with a previously registered hidden service. A client needs to learn about that identity before contacting the introduction point and be assured that it belongs to the hidden service. But there is no need for the identity that an introduction point learns to be equal to the hidden service's identity or even persist for longer than a single introduction point establishment. This prevents an introduction point from recognizing a hidden service.

In this new approach a hidden service creates a new asymmetric *introduction key* for every introduction point establishment and sends the public key to the introduction point instead of his public permanent key in step 1 of the hidden service protocol. The hidden service includes the public introduction keys of all established introduction points in the descriptor that it sends to the directory server in step 3. The client uses the introduction key that she obtains in step 7 from the directory to create her message to the introduction point in step 10. Finally, the hidden service uses the introduction key instead of his permanent key to decrypt client requests in step 12.

$$\text{Bob} \rightarrow \text{IPO} : IK_i, N, S_{IK_i^{-1}}(IK_i, N) \quad (1)$$

$$\text{IPO} \rightarrow \text{Bob} : - \quad (2)$$

$$\text{Bob} \rightarrow \text{Dir} : H(H(CK_{AB}), H(DC_{AB}, TP, RI)), \quad (3)$$
$$M = (CK_{AB}, H(DC_{AB}, TP, RI), T, K_{DC_{AB}}(IPO_{1\ldots n}, IK_{1\ldots n})),$$
$$S_{CK_{AB}^{-1}}(M)$$

$$\text{Dir} \rightarrow \text{Bob} : - \quad (4)$$

$$\text{Bob} \rightarrow \text{Alice} : H(CK_{AB}), DC_{AB} \quad (5)$$

$$\text{Alice} \rightarrow \text{Dir} : H(H(CK_{AB}), H(DC_{AB}, TP, RI)) \quad (6)$$

$$\text{Dir} \rightarrow \text{Alice} : M = (CK_{AB}, H(DC_{AB}, TP, RI), T, K_{DC_{AB}}(IPO_{1\ldots n}, IK_{1\ldots n})), \quad (7)$$
$$S_{CK_{AB}^{-1}}(M)$$

$$\text{Alice} \rightarrow \text{RPO} : RC \quad (8)$$

$$\text{RPO} \rightarrow \text{Alice} : - \quad (9)$$

$$\text{Alice} \rightarrow \text{IPO} : H(IK_i), E_{IK_i}(RPO, RC, g^x, DC_{AB}) \quad (10)$$

$$\text{IPO} \rightarrow \text{Alice} : - \quad (11)$$

$$\text{IPO} \rightarrow \text{Bob} : E_{IK_i}(RPO, RC, g^x, DC_{AB}) \quad (12)$$

$$\text{Bob} \rightarrow \text{RPO} : RC, g^y, H(g^{xy}) \quad (13)$$

$$\text{RPO} \rightarrow \text{Alice} : g^y, H(g^{xy}) \quad (14)$$

Figure 3: Proposed revised hidden service protocol

4.2 Client Key as Replacement for Permanent Key

An important requirement for the new approach is the ability to withdraw client authorization at any time. Therefore, a hidden service issues separate hidden service descriptors for each client, so that he can later stop publishing descriptors for removed clients. The service creates a separate client key CK_{AB} ($CK_{CB}, CK_{DB}, \ldots$) for each client replacing the permanent key PK_B. When advertising his service, the service provider uses the client keys to generate and upload descriptors for all authorized clients in protocol step 3. All service descriptors may contain the same set of introduction points, which is why this approach scales better than the second approach using the unmodified hidden services. Bob tells the client key identifiers to Alice in step 5 out of band. Alice requests Bob's descriptor using the client key identifier in step 6 and obtains it in step 7.

4.3 Client-specific Descriptor Identifiers

An important step of this approach is to hide relations between descriptors that are issued by the same service but for different clients. Otherwise, a removed client who can link the client key identity of another, still authorized client to the service could attack or profile the service. The approach taken here is to store descriptors on a periodically changing subset of a

potentially large number of *hidden service directory nodes* as opposed to the central directory servers. A further objective of distributing the hidden service directory is better scalability, as the new approach requires to store and serve a number of descriptors that is proportional to the number of authorized clients. [5] contains a possible design of a distributed hidden service directory that is not further discussed here. In the following it is assumed that descriptors with different identifiers will be stored on different directories with a certain probability.

There are a couple of new requirements to hidden service descriptor identifiers. They need to change periodically, so that a descriptor is stored on changing directory nodes over time. Descriptor identifiers shall not change at the same periodic time for all descriptors to avoid re-publication bursts. A directory node needs to be able to verify that a hidden service that creates a descriptor is allowed to store it under a claimed descriptor identifier; otherwise, an adversary could claim that his descriptor has a certain descriptor identifier in order to occupy the descriptor space of a legitimate hidden service. Nobody but the authorized client may be able to determine future descriptor identifiers from a given descriptor.

These requirements lead to three new components of descriptor identifiers besides the client key identifier $H(CK_{AB})$:

Descriptor Cookie The secret descriptor cookie DC_{AB} is meant to prevent anyone who observes a descriptor from determining future descriptor identifiers. This may only be done by the client as opposed to the directory that stores the descriptor. A hidden service generates descriptor cookies for each client and distributes pairs of client key identifiers and descriptor cookies to them in protocol step 5.

Time Period The time period TP makes sure that a descriptor identifier changes periodically, so that the descriptor is stored on changing directories. The time period is constructed in a way that transition from one period to the next is equally distributed over the whole time period depending on the permanent key identifier. For example, if the period has a length of one day, a certain descriptor's time period is always incremented at a specific time of the day.

Replica Index The replica index RI provides for multiple replicas of a descriptor to be stored on distinct directories. A directory that stores one descriptor, however, is not meant to determine storage locations of the other replicas.

These three components, together with the client key identifier, are composed to new descriptor identifier. Third parties need to be able to verify that a descriptor identifier belongs to the holder of the private client key CK_{AB}, but without knowing the secret descriptor cookie DC_{AB}. So, instead of concatenating these four components and applying a secure hash function, this process is separated into two steps: First, a secure hash function is applied to descriptor cookie, time period, and replica index to obtain the *secret identifier part*: $S = H(DC_{AB}, TP, RI)$. In a second step, a secure hash function determines the actual descriptor identifier from client key identifier and the secret identifier part: $H(H(CK_{AB}), S)$. The secret identifier part is included in the signed descriptor content as is the client key.

A directory node verifies that a descriptor is eligible for being stored under a claimed descriptor identifier in two steps: First, the directory node verifies the signature of the descriptor content with the included public client key. Next, it generates a descriptor identifier using the

client key identifier and the included secret identifier part and compares it with the claimed descriptor identifier.

4.4 Encryption of Introduction Points

As next step the introduction points that are contained in a hidden service descriptor can be encrypted for the requesting client. This prevents the directory nodes from attempting to access a service themselves. The symmetric descriptor cookie DC_{AB} is used to encrypt the introduction points part of a descriptor that is uploaded to a directory server in step 3. It is, however, not possible to encrypt the remaining parts of a descriptor, because a directory node still needs to be able to verify legitimacy of storage under the claimed descriptor identifier. After downloading a descriptor in step 7, the client decrypts this descriptor part to learn about the introduction points.

4.5 Delayed Descriptor Publication

With the changes so far it is *almost* impossible to link two or more hidden service descriptors for different clients to be issued by the same hidden service. The only problem that remains is simultaneous publishing of two or more descriptors to the same directory or to two collaborating directories. For one thing they contain the same or very close timestamps, for another thing the upload requests arrive in short order.

As a countermeasure descriptors for different clients that ought to be stored on the same directory are delayed, so that only one descriptor is uploaded to a directory at a time. The remaining descriptors are uploaded with a delay of, e.g., 30 seconds. Further, descriptors for different clients that are to be stored on different directories are delayed for a random time of up to, e.g., 30 seconds to hide relations from colluding directories.

Certainly, this does not prevent linking entirely, but it makes it somewhat harder. There is a conflict between hiding links between clients and making a service available in a timely manner.

4.6 Client Identification at Hidden Service

The last change to the hidden service protocol affects the ability of a hidden service to selectively remove authorization of a client. Clients need to identify themselves to a hidden service using the credentials they obtained before, so that the hidden service can attribute possible misuse to one of his clients.

For this purpose the client includes her descriptor cookie DC_{AB} in the encrypted introduction request that she sends to the introduction point in step 10 and that is forwarded to the hidden service in step 12. The hidden service checks whether a contained descriptor cookie is valid before extending a circuit to the client's rendezvous point.

There need to be two limitations to prevent attacks by introduction points and rogue authorized clients:

1. An introduction point could attempt to relay valid introduction requests to force the hidden service to repeatedly extend new circuits to the enclosed rendezvous point. As a

defense, the hidden service memorizes rendezvous cookies of valid requests and drops duplicates.

2. A rogue authorized client could go after the same goal and send a large number of valid introduction requests with different rendezvous cookies. As a countermeasure, the hidden service accepts only a limited number of requests containing the same descriptor cookie, e.g. 10 per hour. In repeated cases of misuse the service provider can decide to remove a client's authorization.

5 Analysis of Security Implications

At the beginning of this report it was found that two different approaches to utilize the unchanged hidden service protocol for virtual private services raise several security issues. A brief analysis shall evaluate whether the extended hidden service protocol can resist these security problems.

Unauthorized Access Attempts Only authorized clients can prompt a hidden service to extend circuits to rendezvous points and thereby establish a connection. Rogue authorized clients can only force a limited number of connections to be established in a given time before risking detection by the service. The service's introduction points cannot force the service to extend any circuit, because they lack a descriptor cookie to send a valid introduction request. An introduction point does not learn about other introduction points for launching an attack on them. Neither directory nodes nor unauthorized clients learn about the introduction points to make an attempt to access. The only thing a directory node could do to learn about a descriptor cookie is break the encryption of introduction points or reverse the secure hash function, which is why the descriptor cookie should be generated by Tor instead of using a possibly weak password.

Service Activity An adversary trying to track service activity needs to know the link between a client key identity and the service to be tracked and needs to observe current existence of a hidden service descriptor for that client key identity. This reduces the set of possible adversaries to former clients, because currently authorized clients know the service activity anyway. Hence, the adversary must have observed a link between her own and another client key identity at a time when she was still authorized. Further, the adversary needs to run or collaborate with a directory node that currently stores a descriptor for the client key identity of the still authorized client.

Anonymous Service Usage An attack to uncover anonymous service usage would be performed similarly to the attack on service activity.

Network Load Though not being a security problem, an approach that puts a high load on the network is not practical, either. The presented approach reuses the same set of introduction points for an arbitrary number of clients. However, it issues a number of hidden service descriptors that is proportional to the number of authorized clients. With a distributed directory as assumed for this approach this may be acceptable for the network. But it still imposes the task of publishing a large number of descriptors on the service.

Links Between Pseudonyms Directory node operators could try to conclude from multiple descriptor uploads on short notice that the descriptors were issued by the same hidden service. This knowledge can be accumulated over time. If one of the clients in the set of jointly observed client keys is removed at some time, the directory node operator can collaborate with the removed client to exploit existence of a descriptor that is still published to attack or profile the service.

6 Conclusion

This report motivated the use of virtual private services which are offered by private users and for a limited, possibly changing set of clients. An analysis of two common approaches to utilize Tor hidden services for this task has shown several security problems that might turn out to be privacy-relevant for the service provider. An extension of the hidden service protocol was presented that is designed to better support virtual private services. A subsequent security analysis has discussed possible attacks on the stated security properties.

The proposed protocol extensions are not meant to replace the existing hidden service design. The only overlapping of public and private hidden services is the distributed directory. This can be used by both types of services, which even makes sense in terms of scalability. The remaining changes of private services only affect the hidden service and the clients.

References

[1] Roger Dingledine and Nick Mathewson. *Tor Rendezvous Specification*, February 2008. https://tor-svn.freehaven.net/svn/tor/trunk/doc/spec/rend-spec.txt.

[2] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th USENIX Security Symposium*, pages 303–320, August 2004.

[3] Saikat Guha and Paul Francis. Identity trail: Covert surveillance using dns. In *Proceedings of the Seventh Workshop on Privacy Enhancing Technologies (PET 2007)*, pages 153–166, Ottawa, Canada, June 2007. Springer.

[4] Tobias Kamm, Thomas Lauterbach, Karsten Loesing, Ferdinand Rieger, and Christoph Weingarten. *Hidden Service Authentication*, September 2007. https://tor-svn.freehaven.net/svn/tor/trunk/doc/spec/proposals/121-hidden-service-authentication.txt.

[5] Karsten Loesing. *Distributed Storage for Tor Hidden Service Descriptors*, May 2007. https://tor-svn.freehaven.net/svn/tor/trunk/doc/spec/proposals/114-distributed-storage.txt.

[6] Karsten Loesing, Markus Dorsch, Martin Grote, Knut Hildebrandt, Maximilian Röglinger, Matthias Sehr, Christian Wilms, and Guido Wirtz. Privacy-aware presence management in

instant messaging systems. In *20th IEEE International Parallel and Distributed Processing Symposium*, April 2006.

[7] Karsten Loesing, Maximilian Röglinger, Christian Wilms, and Guido Wirtz. Implementation of an instant messaging system with focus on protection of user presence. In *Proceedings of the Second International Conference on Communication System Software and Middleware*. IEEE CS Press, January 2007.