

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 18, 2019

C. Huitema
Private Octopus Inc.
D. Kaiser
University of Konstanz
October 15, 2018

**Privacy Extensions for DNS-SD
draft-ietf-dnssd-privacy-05**

Abstract

DNS-SD (DNS Service Discovery) normally discloses information about both the devices offering services and the devices requesting services. This information includes host names, network parameters, and possibly a further description of the corresponding service instance. Especially when mobile devices engage in DNS Service Discovery over Multicast DNS at a public hotspot, a serious privacy problem arises.

We propose to solve this problem by a two-stage approach. In the first stage, hosts discover Private Discovery Service Instances via DNS-SD using special formats to protect their privacy. These service instances correspond to Private Discovery Servers running on peers. In the second stage, hosts directly query these Private Discovery Servers via DNS-SD over TLS. A pairwise shared secret necessary to establish these connections is only known to hosts authorized by a pairing system.

Revisions of this draft are currently considered in the DNSSD working group.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of [BCP 78](#) and [BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 18, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](https://trustee.ietf.org/license-info) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

- [1. Introduction](#) [3](#)
- [1.1. Requirements](#) [4](#)
- [2. Design of the Private DNS-SD Discovery Service](#) [4](#)
- [2.1. Device Pairing](#) [5](#)
- [2.2. Discovery of the Private Discovery Service](#) [5](#)
- [2.2.1. Obfuscated Instance Names](#) [5](#)
- [2.2.2. Using a Predictable Nonce](#) [6](#)
- [2.2.3. Using a Short Proof](#) [7](#)
- [2.2.4. Direct Queries](#) [8](#)
- [2.3. Private Discovery Service](#) [9](#)
- [2.3.1. A Note on Private DNS Services](#) [10](#)
- [2.4. Randomized Host Names](#) [11](#)
- [2.5. Timing of Obfuscation and Randomization](#) [11](#)
- [3. Private Discovery Service Specification](#) [11](#)
- [3.1. Host Name Randomization](#) [12](#)
- [3.2. Device Pairing](#) [12](#)
- [3.3. Private Discovery Server](#) [12](#)
- [3.3.1. Establishing TLS Connections](#) [12](#)
- [3.4. Publishing Private Discovery Service Instances](#) [13](#)
- [3.5. Discovering Private Discovery Service Instances](#) [14](#)
- 3.6. Direct Discovery of Private Discovery Service Instances . 15
- [3.7. Using the Private Discovery Service](#) [16](#)
- [4. Security Considerations](#) [16](#)
- [4.1. Attacks Against the Pairing System](#) [16](#)
- [4.2. Denial of Discovery of the Private Discovery Service . .](#) [16](#)
- 4.3. Replay Attacks Against Discovery of the Private Discovery Service [17](#)
- [4.4. Denial of Private Discovery Service](#) [17](#)
- [4.5. Replay Attacks against the Private Discovery Service . .](#) [17](#)
- [4.6. Replay attacks and clock synchronization](#) [18](#)
- [4.7. Fingerprinting the number of published instances](#) [18](#)

5.	IANA Considerations	18
6.	Acknowledgments	19
7.	References	19
7.1.	Normative References	19
7.2.	Informative References	20
	Authors' Addresses	21

[1.](#) Introduction

DNS-SD [[RFC6763](#)] over mDNS [[RFC6762](#)] enables configurationless service discovery in local networks. It is very convenient for users, but it requires the public exposure of the offering and requesting identities along with information about the offered and requested services. Parts of the published information can seriously breach the user's privacy. These privacy issues and potential solutions are discussed in [[KW14a](#)] and [[KW14b](#)].

There are cases when nodes connected to a network want to provide or consume services without exposing their identity to the other parties connected to the same network. Consider for example a traveler wanting to upload pictures from a phone to a laptop when connected to the Wi-Fi network of an Internet cafe, or two travelers who want to share files between their laptops when waiting for their plane in an airport lounge.

We expect that these exchanges will start with a discovery procedure using DNS-SD [[RFC6763](#)] over mDNS [[RFC6762](#)]. One of the devices will publish the availability of a service, such as a picture library or a file store in our examples. The user of the other device will discover this service, and then connect to it.

When analyzing these scenarios in [[I-D.ietf-dnssd-prireq](#)], we find that the DNS-SD messages leak identifying information such as the instance name, the host name or service properties. We review the design constraint of a solution in [Section 2](#), and describe the proposed solution in [Section 3](#).

While we focus on a mDNS-based distribution of the DNS-SD resource records, our solution is agnostic about the distribution method and also works with other distribution methods, e.g. the classical hierarchical DNS.

The solution presented here relies on 1-1 pairings between clients and servers. Discussions during the IETF 101 in London showed that this requirement of a full mesh of pairings poses some scalability issues, as explained in [[I-D.ietf-dnssd-privacyscaling](#)]. The next revision of this draft may propose a different mechanism.

1.1. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [[RFC2119](#)].

2. Design of the Private DNS-SD Discovery Service

In this section, we present the design of a two-stage solution that enables private use of DNS-SD, without affecting existing users. The solution is largely based on the architecture proposed in [[Kw14b](#)] and [[K17](#)], which separates the general private discovery problem in three components. The first component is an offline pairing mechanism, which is performed only once per pair of users. It establishes a shared secret over an authenticated channel, allowing devices to authenticate using this secret without user interaction at any later point in time. We use the pairing system proposed in [[I-D.ietf-dnssd-pairing](#)].

The further two components are online (in contrast to pairing they are performed anew each time joining a network) and compose the two service discovery stages, namely

- o Discovery of the Private Discovery Service -- the first stage -- in which hosts discover the Private Discovery Service (PDS), a special service offered by every host supporting our extension. After the discovery, hosts connect to the PSD offered by paired peers.
- o Actual Service Discovery -- the second stage -- is performed through the Private Discovery Service, which only accepts encrypted messages associated with an authenticated session; thus not compromising privacy.

In other words, the hosts first discover paired peers and then directly engage in privacy preserving service discovery.

The stages are independent with respect to means used for transmitting the necessary data. While in our extension the messages for the first stage are transmitted using IP multicast, the messages for the second stage are transmitted via unicast. One could also imagine using a Distributed Hash Table for the first stage, being completely independent of multicast.

2.1. Device Pairing

Any private discovery solution needs to differentiate between authorized devices, which are allowed to get information about discoverable entities, and other devices, which should not be aware of the availability of private entities. The commonly used solution to this problem is establishing a "device pairing".

Device pairing has to be performed only once per pair of users. This is important for user-friendliness, as it is the only step that demands user-interaction. After this single pairing, privacy preserving service discovery works fully automatically. In this document, we utilize [[I-D.ietf-dnssd-pairing](#)] as the pairing mechanism.

The pairing yields a mutually authenticated shared secret, and optionally mutually authenticated public keys or certificates added to a local web of trust. Public key technology has many advantages, but shared secrets are typically easier to handle on small devices.

2.2. Discovery of the Private Discovery Service

The first stage of service discovery is to check whether instances of compatible Private Discovery Services are available in the local scope. The goal of that stage is to identify devices that share a pairing with the querier, and are available locally. The service instances can be browsed using regular DNS-SD procedures, and then filtered so that only instances offered by paired devices are retained.

2.2.1. Obfuscated Instance Names

The instance names for the Private Discovery Service are obfuscated, so that authorized peers can associate the instance with its publisher, but unauthorized peers can only observe what looks like a random name. To achieve this, the names are composed as the concatenation of a nonce and a proof, which is composed by hashing the nonce with a pairing key:

```
PrivateInstanceName = <nonce>|<proof>  
proof = hash(<nonce>|<key>)
```

The publisher will publish as many instances as it has established pairings.

The discovering party that looks for instances of the service will receive lists of advertisements from nodes present on the network. For each advertisement, it will parse the instance name, and then,

for each available pairing key, compares the proof to the hash of the nonce concatenated with this pairing key. If there is no match, it discards the instance name. If there is a match, it has discovered a peer.

2.2.2. Using a Predictable Nonce

Assume that there are N nodes on the local scope, and that each node has on average M pairings. Each node will publish on average M records, and the node engaging in discovery may have to process on average $N*M$ instance names. The discovering node will have to compute on average M potential hashes for each nonce. The number of hash computations would scale as $O(N*M*M)$, which means that it could cause a significant drain of resource in large networks.

In order to minimize the amount of computing resource, we suggest that the nonce be derived from the current time, for example set to a representation of the current time rounded to some period. With this convention, receivers can predict the nonces that will appear in the published instances.

The publishers will have to create new records at the end of each rounding period. If the rounding period is set too short, they will have to repeat that very often, which is inefficient. On the other hand, if the rounding period is too long, the system may be exposed to replay attacks. We initially proposed a value of about 5 minutes, which would work well for the mDNS variant of DNS-SD. However, this may cause an excessive number of updates for the DNS server based version of DNS-SD. We propose to set a value of about 30 minutes, which seems to be a reasonable compromise.

Receivers can pre-calculate all the M relevant proofs once per time interval and then establish a mapping from the corresponding instance names to the pairing data in form of a hash table. These M relevant proofs are the proofs resulting from hashing a host's M pairing keys alongside the current nonce. Each time they receive an instance name, they can test in $O(1)$ time if the received service information is relevant or not.

Unix defines a 32 bit time stamp as the number of seconds elapsed since January 1st, 1970 not counting leap seconds. The most significant 20 bits of this 32 bit number represent the number of 2048 seconds intervals since the epoch. 2048 seconds correspond to 34 minutes and 8 seconds, which is close enough to our design goal of 30 minutes. We will thus use this 20 bit number as nonce, which for simplicity will be padded zeroes to 24 bits and encoded in 3 octets.

For coping with time skew, receivers pre-calculate proofs for the respective next time interval and store hash tables for the last, the current, and the next time interval. When receiving a service instance name, receivers first check whether the nonce corresponds to the current, the last or the next time interval, and if so, check whether the instance name is in the corresponding hash table. For (approximately) meeting our design goal of 5 min validity, the last time interval may only be considered if the current one is less than half way over and the next time interval may only be considered if the current time interval is more than half way over.

Publishers will need to compute $O(M)$ hashes at most once per time stamp interval. If records can be created "on the fly", publishers will only need to perform that computation upon receipt of the first query during a given interval, and cache the computed results for the remainder of the interval. There are however scenarios in which records have to be produced in advance, for example when records are published within a scope defined by a domain name and managed by a "classic" DNS server. In such scenarios, publishers will need to perform the computations and publication exactly once per time stamp interval.

2.2.3. Using a Short Proof

Devices will have to publish as many instance names as they have peers. The instance names will have to be represented via a text string, which means that the binary concatenation of nonce and proof will have to be encoded using a binary-to-text conversion such as BASE64 ([\[RFC2045\] section 6.8](#)) or BASE32 ([\[RFC4648\] section 6](#)).

Using long proofs, such as the full output of SHA256 [[RFC4055](#)], would generate fairly long instance names: 48 characters using BASE64, or 56 using BASE32. These long names would inflate the network traffic required when discovering the privacy service. They would also limit the number of DNS-SD PTR records that could be packed in a single 1500 octet sized packet, to 23 or fewer with BASE64, or 20 or fewer with BASE32.

Shorter proofs lead to shorter messages, which is more efficient as long as we do not encounter too many collisions. A collision will happen if the proof computed by the publisher using one key matches a proof computed by a receiver using another key. If a receiver mistakenly believes that a proof fits one of its peers, it will attempt to connect to the service as explained in [Section 3.5](#) but in the absence of the proper pairwise shared key, the connection will fail. This will not create an actual error, but the probability of such events should be kept low.

The following table provides the probability that a discovery agent maintaining 100 pairings will observe a collision after receiving 100000 advertisement records. It also provides the number of characters required for the encoding of the corresponding instance name in BASE64 or BASE32, assuming 24 bit nonces.

Proof	Collisions	BASE64	BASE32
24	5.96046%	8	16
32	0.02328%	11	16
40	0.00009%	12	16
48	3.6E-09	12	16
56	1.4E-11	15	16

Table 1

The table shows that for a proof, 24 bits would be too short. 32 bits might be long enough, but the BASE64 encoding requires padding if the input is not an even multiple of 24 bits, and BASE32 requires padding if the input is not a multiple of 40 bits. Given that, the desirable proof lengths are thus 48 bits if using BASE64, or 56 bits if using BASE32. The resulting instance name will be either 12 characters long with BASE64, allowing 54 advertisements in an 1500 byte mDNS message, or 16 characters long with BASE32, allowing 47 advertisements per message.

In the specification section, we will assume BASE64, and 48 bit proofs composed of the first 6 bytes of a SHA256 hash.

2.2.4. Direct Queries

The preceding sections assume that the discovery is performed using the classic DNS-SD process, in which a query for all available "instance names" of a service provides a list of PTR records. The discoverer will then select the instance names that correspond to its peers, and request the SRV and TXT records corresponding to the service instance, and then obtain the relevant A or AAAA records. This is generally required in DNS-SD because the instance names are not known in advance, but for the Private Discovery Service the instance names can be predicted, and a more efficient Direct Query method can be used.

At a given time, the node engaged in discovery can predict the nonce that its peer will use, since that nonce is composed by rounding the current time. The node can also compute the proofs that its peers might use, since it knows the nonce and the keys. The node can thus

build a list of instance names, and directly query the SRV records corresponding to these names. If peers are present, they will answer directly.

This "direct query" process will result in fewer network messages than the regular DNS-SD query process in some circumstances, depending on the number of peers per node and the number of nodes publishing the presence discovery service in the desired scope.

When using mDNS, it is possible to pack multiple queries in a single broadcast message. Using name compression and 12 characters per instance name, it is possible to pack 70 queries in a 1500 octet mDNS multicast message. It is also possible to request unicast replies to the queries, resulting in significant efficiency gains in wireless networks.

2.3. Private Discovery Service

The Private Discovery Service discovery allows discovering a list of available paired devices, and verifying that either party knows the corresponding shared secret. At that point, the querier can engage in a series of directed discoveries.

We have considered defining an ad-hoc protocol for the private discovery service, but found that just using TLS would be much simpler. The directed Private Discovery Service is just a regular DNS-SD service, accessed over TLS, using the encapsulation of DNS over TLS defined in [[RFC7858](#)]. The main difference with plain DNS over TLS is the need for an authentication based on pre-shared keys.

We assume that the pairing process has provided each pair of authorized client and server with a shared secret. We can use that shared secret to provide mutual authentication of clients and servers using "Pre-Shared Key" authentication, as defined in [[RFC4279](#)] and incorporated in the latest version of TLS [[I-D.ietf-tls-tls13](#)].

One difficulty is the reliance on a key identifier in the protocol. For example, in TLS 1.3 the PSK extension is defined as:


```
opaque psk_identity<0..2^16-1>;

struct {
    select (Role) {
        case client:
            psk_identity identities<2..2^16-1>;

        case server:
            uint16 selected_identity;
    }
} PreSharedKeyExtension
```

According to the protocol, the PSK identity is passed in clear text at the beginning of the key exchange. This is logical, since server and clients need to identify the secret that will be used to protect the connection. But if we used a static identifier for the key, adversaries could use that identifier to track server and clients. The solution is to use a time-varying identifier, constructed exactly like the "proof" described in [Section 2.2](#), by concatenating a nonce and the hash of the nonce with the shared secret.

2.3.1. A Note on Private DNS Services

Our solution uses a variant of the DNS over TLS protocol [[RFC7858](#)] defined by the DNS Private Exchange working group (DPRIVE). DPRIVE further published an UDP variant, DNS over DTLS [[RFC8094](#)], which would also be a candidate.

DPRIVE and Private Discovery, however, solve two somewhat different problems. While DPRIVE is concerned with the confidentiality of DNS transactions addressing the problems outlined in [[RFC7626](#)], DPRIVE does not address the confidentiality or privacy issues with publication of services, and is not a direct solution to DNS-SD privacy:

- o Discovery queries are scoped by the domain name within which services are published. As nodes move and visit arbitrary networks, there is no guarantee that the domain services for these networks will be accessible using DNS over TLS or DNS over DTLS.
- o Information placed in the DNS is considered public. Even if the server does support DNS over TLS, third parties will still be able to discover the content of PTR, SRV and TXT records.
- o Neither DNS over TLS nor DNS over DTLS applies to mDNS.

In contrast, we propose using mutual authentication of the client and server as part of the TLS solution, to ensure that only authorized parties learn the presence of a service.

2.4. Randomized Host Names

Instead of publishing their actual host names in the SRV records, nodes could publish randomized host names. That is the solution argued for in [[RFC8117](#)].

Randomized host names will prevent some of the tracking. Host names are typically not visible by the users, and randomizing host names will probably not cause much usability issues.

2.5. Timing of Obfuscation and Randomization

It is important that the obfuscation of instance names is performed at the right time, and that the obfuscated names change in synchrony with other identifiers, such as MAC Addresses, IP Addresses or host names. If the randomized host name changed but the instance name remained constant, an adversary would have no difficulty linking the old and new host names. Similarly, if IP or MAC addresses changed but host names remained constant, the adversary could link the new addresses to the old ones using the published name.

The problem is handled in [[RFC8117](#)], which recommends to pick a new random host name at the time of connecting to a new network. New instance names for the Private Discovery Services should be composed at the same time.

3. Private Discovery Service Specification

The proposed solution uses the following components:

- o Host name randomization to prevent tracking.
- o Device pairing yielding pairwise shared secrets.
- o A Private Discovery Server (PDS) running on each host.
- o Discovery of the PDS instances using DNS-SD.

These components are detailed in the following subsections.

3.1. Host Name Randomization

Nodes publishing services with DNS-SD and concerned about their privacy MUST use a randomized host name. The randomized name MUST be changed when network connectivity changes, to avoid the correlation issues described in [Section 2.5](#). The randomized host name MUST be used in the SRV records describing the service instance, and the corresponding A or AAAA records MUST be made available through DNS or mDNS, within the same scope as the PTR, SRV and TXT records used by DNS-SD.

If the link-layer address of the network connection is properly obfuscated (e.g. using MAC Address Randomization), the Randomized Host Name MAY be computed using the algorithm described in [section 3.7 of \[RFC7844\]](#). If this is not possible, the randomized host name SHOULD be constructed by simply picking a 48 bit random number meeting the Randomness Requirements for Security expressed in [\[RFC4075\]](#), and then use the hexadecimal representation of this number as the obfuscated host name.

3.2. Device Pairing

Nodes that want to leverage the Private Directory Service for private service discovery among peers MUST share a secret with each of these peers. Each shared secret MUST be a 256 bit randomly chosen number. We RECOMMEND using the pairing mechanism proposed in [\[I-D.ietf-dnssd-pairing\]](#) to establish these secrets.

3.3. Private Discovery Server

A Private Discovery Server (PDS) is a minimal DNS server running on each host. Its task is to offer resource records corresponding to private services only to authorized peers. These peers MUST share a secret with the host (see [Section 3.2](#)). To ensure privacy of the requests, the service is only available over TLS [\[RFC5246\]](#), and the shared secrets are used to mutually authenticate peers and servers.

The Private Name Server SHOULD support DNS push notifications [\[I-D.ietf-dnssd-push\]](#), e.g. to facilitate an up-to-date contact list in a chat application without polling.

3.3.1. Establishing TLS Connections

The PDS MUST only answer queries via DNS over TLS [\[RFC7858\]](#) and MUST use a PSK authenticated TLS handshake [\[RFC4279\]](#). The client and server SHOULD negotiate a forward secure cipher suite such as DHE-PSK or ECDHE-PSK when available. The shared secret exchanged during pairing MUST be used as PSK. To guarantee interoperability,

implementations of the Private Name Server MUST support TLS_PSK_WITH_AES_256_GCM_SHA384.

When using the PSK based authentication, the "psk_identity" parameter identifying the pre-shared key MUST be identical to the "Instance Identifier" defined in [Section 3.4](#), i.e. 24 bit nonce and 48 bit proof encoded in BASE64 as 12 character string. The server will use the pairing key associated with this instance identifier.

3.4. Publishing Private Discovery Service Instances

Nodes that provide the Private Discovery Service SHOULD advertise their availability by publishing instances of the service through DNS-SD.

The DNS-SD service type for the Private Discovery Service is "_pds._tcp".

Each published instance describes one server and one pairing. In the case where a node manages more than one pairing, it should publish as many instances as necessary to advertise the PDS to all paired peers.

Each instance name is composed as follows:

pick a 24 bit nonce, set to the 20 most significant bits of the 32 bit Unix GMT time padded with 4 zeroes.

For example, on August 22, 2017 at 20h 4 min and 54 seconds international time, the Unix 32 bit time had the hexadecimal value 0x599C8E68. The corresponding nonce would be set to the 24 bits: 0x599C80.

compute a 48 bit proof:

proof = first 48 bits of HASH(<nonce>|<pairing key>)

set the 72 bit binary identifier as the concatenation of nonce and proof

set instance_name = BASE64(binary identifier)

In this formula, HASH SHOULD be the function SHA256 defined in [\[RFC4055\]](#), and BASE64 is defined in [section 6.8 of \[RFC2045\]](#). The concatenation of a 24 bit nonce and 48 bit proof result in a 72 bit string. The BASE64 conversion is 12 characters long per [\[RFC6763\]](#).

3.5. Discovering Private Discovery Service Instances

Nodes that wish to discover Private Discovery Service Instances SHOULD issue a DNS-SD discovery request for the service type "_pds._tcp". They MAY, as an alternative, use the Direct Discovery procedure defined in [Section 3.6](#). When using the Direct Discovery procedure over mDNS, nodes SHOULD always set the QU-bit (unicast response requested, see [\[RFC6762\] Section 5.4](#)) because responses related to a "_pds._tcp" instance are only relevant for the querying node itself.

When nodes send a DNS-SD discovery request, they will receive in response a series of PTR records, each providing the name of one of the instances present in the scope.

For each time interval, the querier SHOULD pre-calculate a hash table mapping instance names to pairings according to the following conceptual algorithm:

```
nonce = 20 bit rounded time stamp of the \  
  respective next time interval padded to \  
  24 bits with four zeroes  
for each available pairing  
  retrieve the key Xj of pairing number j  
  compute F = first 48 bits of hash(nonce, Xj)  
  construct the binary instance_name as described \  
  in the previous section  
  instance_names[nonce][instance_name] = Xj;
```

The querier SHOULD store the hash tables for the previous, the current, and the next time interval.

The querier SHOULD examine each instance to see whether it corresponds to one of its available pairings, according to the following conceptual algorithm:


```
for each received instance_name:
    convert the instance name to binary using BASE64
    if the conversion fails,
        discard the instance.
    if the binary instance length is not 72 bits,
        discard the instance.

nonce = first 24 bits of binary.

Check that the 4 least significant bits of the nonce
have the value 0, and that the 20 most significant
bits of the nonce match the first 20 bits of
the current time, or the previous interval (20 bit number
minus 1) if the current interval is less than half over,
or the next interval (20 bit number plus 1) if the
current interval is more than half over. If the
nonce does not match an acceptable value, discard
the instance.

if ((Xj = instance_names[nonce][instance_name]) != null)
    mark the pairing number j as available
```

The check of the current time is meant to mitigate replay attacks, while not mandating a time synchronization precision better than 15 minutes.

Once a pairing has been marked available, the querier SHOULD try connecting to the corresponding instance, using the selected key. The connection is likely to succeed, but it MAY fail for a variety of reasons. One of these reasons is the probabilistic nature of the proof, which entails a small chance of "false positive" match. This will occur if the hash of the nonce with two different keys produces the same result. In that case, the TLS connection will fail with an authentication error or a decryption error.

3.6. Direct Discovery of Private Discovery Service Instances

Nodes that wish to discover Private Discovery Service Instances MAY use the following Direct Discovery procedure instead of the regular DNS-SD Discovery explained in [Section 3.5](#).

To perform Direct Discovery, nodes should compose a list of Private Discovery Service Instances Names. There will be one name for each pairing available to the node. The Instance name for each name will be composed of a nonce and a proof, using the algorithm specified in [Section 3.4](#).

The querier will issue SRV record queries for each of these names. The queries will only succeed if the corresponding instance is present, in which case a pairing is discovered. After that, the querier SHOULD try connecting to the corresponding instance, as explained in [Section 3.4](#).

[3.7. Using the Private Discovery Service](#)

Once instances of the Private Discovery Service have been discovered, peers can establish TLS connections and send DNS requests over these connections, as specified in DNS-SD.

[4. Security Considerations](#)

This document specifies a method for protecting the privacy of nodes that offer and query for services. This is especially useful when operating in a public space. Hiding the identity of the publishing nodes prevents some forms of "targeting" of high value nodes. However, adversaries can attempt various attacks to break the anonymity of the service, or to deny it. A list of these attacks and their mitigations are described in the following sections.

[4.1. Attacks Against the Pairing System](#)

There are a variety of attacks against pairing systems, which may result in compromised pairing secrets. If an adversary manages to acquire a compromised key, the adversary will be able to perform private service discovery according to [Section 3.5](#). This will allow tracking of the service. The adversary will also be able to discover which private services are available for the compromised pairing.

Attacks on pairing systems are detailed in [[I-D.ietf-dnssd-pairing](#)].

[4.2. Denial of Discovery of the Private Discovery Service](#)

The algorithm described in [Section 3.5](#) scales as $O(M*N)$, where M is the number of pairings per node and N is the number of nodes in the local scope. Adversaries can attack this service by publishing "fake" instances, effectively increasing the number N in that scaling equation.

Similar attacks can be mounted against DNS-SD: creating fake instances will generally increase the noise in the system and make discovery less usable. Private Discovery Service discovery SHOULD use the same mitigations as DNS-SD.

The attack could be amplified if the clients needed to compute proofs for all the nonces presented in Private Discovery Service Instance

names. This is mitigated by the specification of nonces as rounded time stamps in [Section 3.5](#). If we assume that timestamps must not be too old, there will be a finite number of valid rounded timestamps at any time. Even if there are many instances present, they would all pick their nonces from this small number of rounded timestamps, and a smart client will make sure that proofs are only computed once per valid time stamp.

4.3. Replay Attacks Against Discovery of the Private Discovery Service

Adversaries can record the service instance names published by Private Discovery Service instances, and replay them later in different contexts. Peers engaging in discovery can be misled into believing that a paired server is present. They will attempt to connect to the absent peer, and in doing so will disclose their presence in a monitored scope.

The binary instance identifiers defined in [Section 3.4](#) start with 24 bits encoding the most significant bits of the "UNIX" time. In order to protect against replay attacks, clients SHOULD verify that this time is reasonably recent, as specified in [Section 3.5](#).

4.4. Denial of Private Discovery Service

The Private Discovery Service is only available through a mutually authenticated TLS connection, which provides state-of-the-art protection mechanisms. However, adversaries can mount a denial of service attack against the service. In the absence of shared secrets, the connections will fail, but the servers will expend some CPU cycles defending against them.

To mitigate such attacks, nodes SHOULD restrict the range of network addresses from which they accept connections, matching the expected scope of the service.

This mitigation will not prevent denial of service attacks performed by locally connected adversaries; but protecting against local denial of service attacks is generally very difficult. For example, local attackers can also attack mDNS and DNS-SD by generating a large number of multicast requests.

4.5. Replay Attacks against the Private Discovery Service

Adversaries may record the PSK Key Identifiers used in successful connections to a private discovery service. They could attempt to replay them later against nodes advertising the private service at other times or at other locations. If the PSK identifier is still valid, the server will accept the TLS connection, and in doing so

will reveal being the same server observed at a previous time or location.

The PSK identifiers defined in [Section 3.3.1](#) start with the 24 most significant bits of the "UNIX" time. In order to mitigate replay attacks, servers SHOULD verify that this time is reasonably recent, and fail the connection if it is too old, or if it occurs too far in the future.

The processing of timestamps is however affected by the accuracy of computer clocks. If the check is too strict, reasonable connections could fail. To further mitigate replay attacks, servers MAY record the list of valid PSK identifiers received in a recent past, and fail connections if one of these identifiers is replayed.

[4.6.](#) Replay attacks and clock synchronization

The mitigation of replay attacks relies on verification of the time encoded in the nonce. This verification assumes that the hosts engaged in discovery have a reasonably accurate sense of the current time.

[4.7.](#) Fingerprinting the number of published instances

Adversaries could monitor the number of instances published by a particular device, which in the absence of mitigations will reflect the number of pairings established by that device. This number will probably vary between 1 and maybe 100, providing the adversary with maybe 6 or 7 bits of input in a fingerprinting algorithm.

Devices MAY protect against this fingerprinting by publishing a number of "fake" instances in addition to the real ones. The fake instance identifiers will contain the same nonce as the genuine instance identifiers, and random bits instead of the proof. Peers should be able to quickly discard these fake instances, as the proof will not match any of the values that they expect. One plausible padding strategy is to ensure that the total number of published instances, either fake or genuine, matches one of a few values such as 16, 32, 64, or higher powers of 2.

[5.](#) IANA Considerations

This draft does not require any IANA action.

6. Acknowledgments

This draft results from initial discussions with Dave Thaler, and encouragements from the DNS-SD working group members. We would like to thank Stephane Bortzmeyer and Ted Lemon for their detailed reviews of the working draft.

7. References

7.1. Normative References

- [RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", [RFC 2045](#), DOI 10.17487/RFC2045, November 1996, <<https://www.rfc-editor.org/info/rfc2045>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4055] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", [RFC 4055](#), DOI 10.17487/RFC4055, June 2005, <<https://www.rfc-editor.org/info/rfc4055>>.
- [RFC4075] Kalusivalingam, V., "Simple Network Time Protocol (SNTP) Configuration Option for DHCPv6", [RFC 4075](#), DOI 10.17487/RFC4075, May 2005, <<https://www.rfc-editor.org/info/rfc4075>>.
- [RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", [RFC 4279](#), DOI 10.17487/RFC4279, December 2005, <<https://www.rfc-editor.org/info/rfc4279>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", [RFC 5246](#), DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", [RFC 6763](#), DOI 10.17487/RFC6763, February 2013, <<https://www.rfc-editor.org/info/rfc6763>>.

7.2. Informative References

- [I-D.ietf-dnssd-pairing]
Huitema, C. and D. Kaiser, "Device Pairing Using Short Authentication Strings", [draft-ietf-dnssd-pairing-04](#) (work in progress), April 2018.
- [I-D.ietf-dnssd-prireq]
Huitema, C., "DNS-SD Privacy and Security Requirements", [draft-ietf-dnssd-prireq-00](#) (work in progress), September 2018.
- [I-D.ietf-dnssd-privacyscaling]
Huitema, C., "DNS-SD Privacy Scaling Tradeoffs", [draft-ietf-dnssd-privacyscaling-00](#) (work in progress), September 2018.
- [I-D.ietf-dnssd-push]
Pusateri, T. and S. Cheshire, "DNS Push Notifications", [draft-ietf-dnssd-push-15](#) (work in progress), September 2018.
- [I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", [draft-ietf-tls-tls13-28](#) (work in progress), March 2018.
- [K17] Kaiser, D., "Efficient Privacy-Preserving Configurationless Service Discovery Supporting Multi-Link Networks", 2017, [<http://nbn-resolving.de/urn:nbn:de:bsz:352-0-422757>](http://nbn-resolving.de/urn:nbn:de:bsz:352-0-422757).
- [KW14a] Kaiser, D. and M. Waldvogel, "Adding Privacy to Multicast DNS Service Discovery", DOI 10.1109/TrustCom.2014.107, 2014, [<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7011331>](http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7011331).
- [KW14b] Kaiser, D. and M. Waldvogel, "Efficient Privacy Preserving Multicast DNS Service Discovery", DOI 10.1109/HPCC.2014.141, 2014, [<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7056899>](http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7056899).
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", [RFC 4648](#), DOI 10.17487/RFC4648, October 2006, [<https://www.rfc-editor.org/info/rfc4648>](https://www.rfc-editor.org/info/rfc4648).

- [RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", [RFC 6762](#), DOI 10.17487/RFC6762, February 2013, <<https://www.rfc-editor.org/info/rfc6762>>.
- [RFC7626] Bortzmeyer, S., "DNS Privacy Considerations", [RFC 7626](#), DOI 10.17487/RFC7626, August 2015, <<https://www.rfc-editor.org/info/rfc7626>>.
- [RFC7844] Huitema, C., Mrugalski, T., and S. Krishnan, "Anonymity Profiles for DHCP Clients", [RFC 7844](#), DOI 10.17487/RFC7844, May 2016, <<https://www.rfc-editor.org/info/rfc7844>>.
- [RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", [RFC 7858](#), DOI 10.17487/RFC7858, May 2016, <<https://www.rfc-editor.org/info/rfc7858>>.
- [RFC8094] Reddy, T., Wing, D., and P. Patil, "DNS over Datagram Transport Layer Security (DTLS)", [RFC 8094](#), DOI 10.17487/RFC8094, February 2017, <<https://www.rfc-editor.org/info/rfc8094>>.
- [RFC8117] Huitema, C., Thaler, D., and R. Winter, "Current Hostname Practice Considered Harmful", [RFC 8117](#), DOI 10.17487/RFC8117, March 2017, <<https://www.rfc-editor.org/info/rfc8117>>.

Authors' Addresses

Christian Huitema
Private Octopus Inc.
Friday Harbor, WA 98250
U.S.A.

Email: huitema@huitema.net
URI: <http://privateoctopus.com/>

Daniel Kaiser
University of Konstanz
Konstanz 78457
Germany

Email: daniel.kaiser@uni-konstanz.de

