Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 12, 2016

C. Huitema
Microsoft
D. Kaiser
University of Konstanz

June 10, 2016

Privacy Extensions for DNS-SD
draft-huitema-dnssd-privacy-01.txt

Abstract

DNS-SD allows discovery of services published in DNS or MDNS. The publication normally discloses information about the device publishing the services. There are use cases where devices want to communicate without disclosing their identity, for example two mobile devices visiting the same hotspot.

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.

Status of This Memo

This Internet-Draft will expire on December 12, 2016.

Copyright Notice

Table of Contents

1. Introduction

DNS-SD [RFC6763] enables distribution and discovery in local networks without configuration. 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. Some of the information published by the announcements can be very revealing. 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]. 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 Section 2, we find that the DNS-SD messages leak identifying information such as instance name, host name or service properties. We review the design constraint of a solution in Section 4, and describe the proposed solution in Section 5.

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. Privacy Implications of DNS-SD

DNS-Based Service Discovery (DNS-SD) is defined in [RFC6763]. It allows nodes to publish the availability of an instance of a service by inserting specific records in the DNS ([RFC1033], [RFC1034], [RFC1035]) or by publishing these records locally using multicast DNS (mDNS) [RFC6762]. Available services are described using three types of records:

PTR Record: Associates a service type in the domain with an "instance" name of this service type.

SRV Record: Provides the node name, port number, priority and weight associated with the service instance, in conformance with [RFC2782].

TXT Record: Provides a set of attribute-value pairs describing specific properties of the service instance.

In the remaining subsections, we will review the privacy issues related to publishing instance names, node names, service attributes and other data, as well as review the implications of using the discovery service as a client.

2.1. Privacy Implication of Publishing Service Instance Names

In the first phase of discovery, the client obtains all the PTR records associated with a service type in a given naming domain. Each PTR record contains a Service Instance Name defined in Section 4 of [RFC6763]:

```
Service Instance Name = <Instance> . <Service> . <Domain>
```

The <Instance> portion of the Service Instance Name is meant to convey enough information for users of discovery clients to easily select the desired service instance. Nodes that use DNS-SD over mDNS [RFC6762] in a mobile environment will rely on the specificity of the instance name to identify the desired service instance. In our example of users wanting to upload pictures to a laptop in an Internet Cafe, the list of available service instances may look like:

```
Alice's Images         . _imageStore._tcp . local
Alice's Mobile Phone   . _presence._tcp   . local
Alice's Notebook       . _presence._tcp   . local
Bob's Notebook         . _presence._tcp   . local
Carol's Notebook       . _presence._tcp   . local
```

Alice will see the list on her phone and understand intuitively that she should pick the first item. The discovery will "just work".

However, DNS-SD/mDNS will reveal to anybody that Alice is currently visiting the Internet Cafe. It further discloses the fact that she uses two devices, shares an image store, and uses a chat application supporting the _presence protocol on both of her devices. She might currently chat with Bob or Carol, as they are also using a _presence supporting chat application. This information is not just available to devices actively browsing for and offering services, but to anybody passively listing to the network traffic.

2.2. Privacy Implication of Publishing Node Names

The SRV records contain the DNS name of the node publishing the service. Typical implementations construct this DNS name by concatenating the "host name" of the node with the name of the local domain. The privacy implications of this practice are reviewed in [I-D.ietf-intarea-hostname-practice]. Depending on naming practices, the host name is either a strong identifier of the device, or at a minimum a partial identifier. It enables tracking of the device, and by extension of the device's owner.

2.3. Privacy Implication of Publishing Service Attributes

The TXT record's attribute and value pairs contain information on the characteristics of the corresponding service instance. This in turn reveals information about the devices that publish services. The amount of information varies widely with the particular service and its implementation:

- Some attributes like the paper size available in a printer, are the same on many devices, and thus only provide limited information to a tracker.

- Attributes that have freeform values, such as the name of a directory, may reveal much more information.

Combinations of attributes have more information power than specific attributes, and can potentially be used for "fingerprinting" a specific device.

Information contained in TXT records does not only breach privacy by making devices trackable, but might directly contain private information about a device user. For instance the _presence service reveals the "chat status" to everyone in the same network. Users might not be aware of that.

Further, TXT records often contain version information about services allowing potential attackers to identify devices running exploit-prone versions of a certain service.

2.4. Device Fingerprinting

The combination of information published in DNS-SD has the potential to provide a "fingerprint" of a specific device. Such information includes:

- The list of services published by the device, which can be retrieved because the SRV records will point to the same host name.
- The specific attributes describing these services.
- The port numbers used by the services.
- The values of the priority and weight attributes in the SRV records.

This combination of services and attributes will often be sufficient to identify the version of the software running on a device. If a device publishes many services with rich sets of attributes, the combination may be sufficient to identify the specific device.

There is however an argument that devices providing services can be discovered by observing the local traffic, because different services have different traffic patterns. The observation could in many cases also reveal some specificities of the service's implementation. Even if the traffic is encrypted, the size and the timing of packets may be sufficient to reveal that information. This argument can be used to assess the priority of, for example, protecting the fact that a device publishes a particular service. However, we may assume that the developers of sensitive services will use counter-measures to defeat such traffic analysis.

2.5. Privacy Implication of Discovering Services

The consumers of services engage in discovery, and in doing so reveal some information such as the list of services they are interested in and the domains in which they are looking for the services. When the clients select specific instances of services, they reveal their preference for these instances. This can be benign if the service type is very common, but it could be more problematic for sensitive services, such as for example some private messaging services.

One way to protect clients would be to somehow encrypt the requested service types. Of course, just as we noted in Section 2.4, traffic analysis can often reveal the service.

3. Limits of a Simple Design

We first tried a simple design for mitigating the issues outlined in Section 2. The basic idea was to advertise obfuscated names, so as to not reveal the particularities of the service providers. This design is tempting, because it only requires minimal changes in the DNS-SD processing. However, as we will see in the following subsections, it has two important drawbacks:

- The simple design leads to UI issues, because users of unmodified DNS-SD agents will see a mix of clear text names and obfuscated names, which is unpleasant.

- With this simple design, there is no good way to hide the type of services provided or consumed by a specific node.

- The simple design either requires having a shared key between all "authorized users" of a service, which implies substandard key management practices, or publishing as many instances of a service as there are authorized users, which leads to the scaling issues discussed in Section 3.3.

Both issues are mitigated by the two-stage design presented in Section 4. The following subsections detail the simple design, and its drawbacks.

3.1. Obfuscated Instance Names

The privacy issues described in Section 2.1 could be solved by obfuscating the instance names. Instead of a user friendly description of the instance, the nodes would publish a random looking string of characters. To prevent tracking over time and location, different string values would be used at different locations, or at different times.

Authorized parties have to be able to "de-obfuscate" the names, while non-authorized third parties will not be. For example, if both Alice's notebook and Bob's laptop use an obfuscation process, the list of available services should appear differently to them and to third parties. Alice's phone will be able to de-obfuscate the name of Alice's notebook, but not that of Bob's laptop. Bob's phone will do the opposite. Carol will do neither.

Alice will see something like:

```
   QwertyUiopAsdfghjk (Alice's Images)       . _imageStore._tcp . local
   GobbeldygookBlaBla (Alice's Mobile Phone) . _presence._tcp   . local
   MNbvCxzLkjhGfdEdhg (Alice's Notebook)     . _presence._tcp   . local
   Abracadabragooklybok (Bob's Notebook)     . _presence._tcp   . local
   Carol's Notebook                          . _presence._tcp   . local
```

Bob will see:

```
   QwertyUiopAsdfghjk                    . _imageStore._tcp . local
   GobbeldygookBlaBla                    . _presence._tcp   . local
   MNbvCxzLkjhGfdEdhg                    . _presence._tcp   . local
   Abracadabragooklybok (Bob's Notebook) . _presence._tcp   . local
   Carol's Notebook                      . _presence._tcp   . local
```

Carol will see:

```
   QwertyUiopAsdfghjk   . _imageStore._tcp . local
   GobbeldygookBlaBla   . _presence._tcp   . local
   MNbvCxzLkjhGfdEdhg   . _presence._tcp   . local
   Abracadabragooklybok . _presence._tcp   . local
   Carol's Notebook     . _presence._tcp   . local
```

In that example, Alice, Bob and Carol will be able to select the appropriate instance. It would probably be preferable to filter out the obfuscated instance names, to avoid confusing the user. In our example, Alice and Bob have updated their software to understand obfuscation, and they could easily filter out the obfuscated strings that they do not like. But Carol is not using this system, and we could argue that her experience is suboptimal.

3.2. Names of Obfuscated Services

Instead of publishing the actual service name in the SRV records, nodes could publish a randomized name. There are two plausible reasons for doing that:

- Having a different service name for privacy enhanced services will ensure that hosts that are not privacy aware are not puzzled by obfuscated service names.

- Using obfuscated service names prevents third parties from discovering which service a particular host is providing or consuming.

The first requirement can be met with a simple modification of an existing name. For example, instead of publishing:

```
QwertyUiopAsdfghjk . _imageStore._tcp . local
GobbeldygookBlaBla . _presence._tcp   . local
```

Alice could publish some kind of "translation" of the service name, such as:

```
QwertyUiopAsdfghjk . _vzntrFgber._tcp . local
GobbeldygookBlaBla . _cerfrapr._tcp   . local
```

The previous examples use rot13 translation. It does not provide any particular privacy, but it does ensure that obfuscated services are named differently from clear text services.

Making the service name actually private would require some actual encryption. The main problem with such solutions is that the client needs to know the service name in order to compose the DNS-SD query for services. There are several options:

- The service name is chosen by the client. For example, the client could encrypt the original service name and a nonce with a key shared between client and server. Upon receiving the queries, the server would attempt to decrypt the service name. If that succeeds, the server would respond with PTR records created on the fly for the new service name.

- The service name is chosen by the server and cannot be predicted in advance by the client. For example, the server could encrypt a nonce and the original service name. The client retrieves such services by doing a wild card query, then attempting to decrypt the received responses.

- The service name is chosen by the server in a way that can be predicted in advance by the client. For example, the server could encrypt some version of the data and time and the original service name. The data and time are encoded with a coarse precision, enabling the client to predict the value that the server is using, and to send the corresponding queries.

None of these solutions is very attractive. Creating records on the fly is a burden for the server. If clients must use wildcard queries, they will need to process lots of irrelevant data. If clients need to predict different instance names for each potential server, they will end up sending batches of queries with many different names. All of these solutions appear like big departures from the simplicity and robustness of the DNS-SD design.

3.3. Scaling Issues with Obfuscation

In Section 3.1, we assumed that each advertised record contains a name obfuscated with a shared key. This approach is easy to understand, but it contains hidden assumptions. Let's look at one of our examples:

```
Abracadabragooklybok (Bob's Notebook) . _presence._tcp   . local
```

We only see one record for Bob's Notebook, obfuscated using the unique shared secret associated with Bob's Notebook. That means that every device paired with Bob's Notebook will have a copy of that shared secret. This is a possible solution, but there are known issues with having a secret shared with multiple entities:

- If for some reason the secret needs to be changed, every paired device will need a copy of the new secret before it can participate again in discovery.
- If one of the previous pairings becomes invalid, the only way to block the corresponding devices from discovery is to change the secret for all other devices.

Key management becomes much easier if it is strictly pair-wise. Two paired devices, or to pairs of users, can simply renew their pairing and get a new secret. If a device ceases to be trusted, the pairing data and the corresponding secret can just be deleted and forgotten. But using strictly pair-wise keys yields a scaling issue. Let's assume that:

- Each device maintains an average of N pairings.
- There are on average M devices present during discovery.

In the single key scenario, after issuing a broadcast query, the querier will receive a series of responses, each of which may well be obfuscated with a different key. If the receiver has N pre-existing pairings and receives M obfuscated responses, the cost will scale as O(M*N), i.e. try all N pairing keys for each of the M responses to see what matches. But if the keys are specific to each pair of devices, the obfuscation becomes complicated. When receiving a request, the publisher does not know which of its N keys the querier can decrypt. One simple solution would be to send N responses, but then the load on the querier will scale as O(M*N^2). That can go out of hand very quickly.

To solve the scaling issue, we consider a two-stage solution that uses an optimized discovery procedure to discover privacy-compatible

devices; and uses point to point encrypted exchanges to privately discover the available services.

4. 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, and with better scalability than the simple solution presented in Section 3. The solution is largely based on the architecture proposed in [KW14b], which separates the general private discovery problems in three components: Pairing, discovery of a private discovery service, and actual service discovery through this private service. Pairing has to provide the private discovery servers with means for mutual authentication, e.g. with an authenticated shared secret. The private discovery servers provide actual service discovery with an authenticated connection. Our solution applies this architecture in the context of DNS-SD. It is based on the following components:

- Adding a pairing system to DNS-SD, described in Section 4.1, through which authorized peers can establish shared secrets;

- Defining the Private Discovery Service through which other services can be advertised in a private manner;

- And, publishing availability of the Private Discovery Service using DNS-SD, so that peers can discover their services without compromising their privacy.

These are independent with respect to means used for transmitting the necessary data.

4.1. 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". In our discovery scenarios, we envisage two kinds of pairings:

1. Inter-user pairing is a pairing between devices of "friends". Since it has to be performed manually, e.g. by the means described above, it is important to limit it to once per pair of friends.

2. Intra-user pairing is a pairing of devices of the same user. It can be performed without any configuration by a meta-service

(pairing data synchronization service) in a trusted (home) network.

The result of the pairing will be a shared secret, and optionally mutually authenticated public keys added to a local web of trust. Public key technology has many advantages, but shared secrets are typically easier to handle on small devices. We offer both a simple pairing just exchanging a shared secret, and an authenticated pairing using public key technology.

4.1.1. Shared Secret

Goal of the pairing process is establishing pairwise shared secrets. If two users can leverage a secure private off-channel, it suffices for one user to generate the shared secret and transmit it over this off-channel. It would be possible for the users to meet and orally agree on a password that both users enter in their devices. This has the disadvantage of user-chosen passwords to have low entropy and the inconvenience of having to type the password. Leveraging QR-codes can overcome these disadvantages: one user generates a shared secret, displays it in form of a QR-code, and the other user scans this code. Strictly speaking, displaying and scanning QR-codes does not establish a secure private channel, as others could also photograph this code; but it is reasonable secure for the application area of private service discovery. Using Bluetooth LE might also be considered satisfactory as a compromise between convenience and security.

4.1.2. Secure Authenticated Pairing Channel

Optionally, various versions of authenticated DH can be used to exchange a mutually authenticated shared secret (which among other possibilities can leverage QR-codes for key fingerprint verification). Using DH gives the benefit of provable security and the possibility to perform a pairing when not being able to meet in person. Further, using DH to generate the shared secret has the advantage of both parties contributing to the shared secret (multiparty computation).

4.1.3. Public Authentication Keys

The public/private key pair - if at all - is just used for the aforementioned authenticated DH to grant a mutually authenticated shared secret. Obtaining and verifying a friend's public key can be achieved by different means. For obtaining the keys, we can either leverage an existing PKI, e.g. the PGP web of trust, or generate our own key pairs (and exchange them right before verifying). For authenticating the keys, which boils down to comparing fingerprints

on an off-channel, we distinguish between means that demand users to be in close proximity of each other, and means where users do not have to meet in person. The former can e.g. be realized by verifying a fingerprint leveraging QR-Codes, the latter by reading a fingerprint during a phone call or using the socialist millionaires protocol.

4.2. 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 discovered using regular DNS-SD procedures, but the list of discovered services will have to be filtered so only paired devices are retained.

We have demonstrated in Section 3.3 that simple obfuscation would require publishing as many records per publisher as there are pairings, which ends up scaling as O(M*N^2) in which M is the number of devices present and N is the number of pairings per device. We can mitigate this problem by using a special encoding of the instance name. Suppose that the publisher manages N pairings with the associated keys K1, K2, ... Kn. The instance name will be set to an encoding of N "proofs" of the N keys, where each proof is computed as function of the key and a nonce:

instance name = <nonce><F1><F2>..<Fn>

Fi = hash (nonce, Ki), where hash is a cryptographic hash function.

The querier can test the instance name by computing the same "proof" for each of its own keys. Suppose that the receiver manages P pairings, with the corresponding keys X1, X2, .. Xp. The receiver verification procedure will be:

```
for each received instance name:
   retrieve nonce from instance name
   for (j = 1 to P)
      retrieve the key Xj of pairing number j
      compute F = hash(nonce, Xj)
      for (i=1 to N)
         retrieve the proof Fi
         if F is equal to Fi
            mark the pairing number j as available
```

The procedure presented here requires on average O(M*N) iterations of the hash function, which is the same scaling as the "shared secret" variant. It requires O(M*N^2) comparison operations, but these are less onerous than cryptographic operations. Further, when setting the nonce to a timestamp, the Fi have to be calculated only once per time interval.

The number of pairing proofs that can be encoded in a single record is limited by the maximum size of a DNS label, which is 63 bytes. Since this are characters and not pure binary values, nonce and proofs will have to be encoded using BASE64 ([RFC2045] section 6.8), resulting in at most 378 bits. The nonce should not be repeated, and the simplest way to achieve that is to set the nonce to a 32 bit timestamp value. The remaining 346 bits could encode up to 10 proofs of 32 bits each, which would be sufficient for many practical scenarios.

In practice, a 32 bit proof should be sufficient to distinguish between available devices. However, there is clearly a risk of collision. The Private Discovery Service as described here will find the available pairings, but it might also find a spurious number of "false positives." The chances of that happening are however quite small: less than 0.02% for a device managing 10 pairings and processing 10000 responses.

4.3. 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 simple DNS over TLS is the need for authentication.

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 "hint" described in Section 4.2, by concatenating a nonce and the hash of the nonce with the shared secret.

4.3.1. 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 is also working on an UDP variant, DNS over DTLS [I-D.ietf-dprive-dnsodtls], which would also be a candidate.

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

- 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.
- 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.
- 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.

4.4. Randomized Host Names

Instead of publishing their actual name in the SRV records, nodes could publish a randomized name. That is the solution argued for in [I-D.ietf-intarea-hostname-practice].

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.

4.5. 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 [I-D.ietf-intarea-hostname-practice], 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.

5. Private Discovery Service Specification

The proposed solution uses the following components:

- Host name randomization to prevent tracking.

- Device pairing yielding pairwise shared secrets.

- A Private Discovery Server (PDS) running on each host.

- Discovery of the PDS instances using DNS-SD.

These components are detailed in the following subsections.

5.1. 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 4.5. 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.

5.2. 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. The shared secret MUST be a 256 bit randomly chosen number. The secret SHOULD be exchanged via device Pairing. The pairing process SHALL establish a mutually authenticated secure channel to perform the shared secret exchange. It is RECOMMENDED for both parties to contribute to the shared secret, e.g. by using a Diffie-Hellman key exchange.

TODO: need to define the pairing service, or API. The API approach assumes that pairing is outside our scope, and is done using BT-LE, or any other existing mechanism. This is a bit of a cope-out. We could also define a pairing system that just sets the pairing with equivalent security as the "push button" or "PIN" solutions used for BT or Wi-Fi. And we could at this stage leverage a pre-existing security association, e.g. PGP identities or other certificates. If we do that, we should probably dedicate a top level section to specifying the minimal pairing service. Using a pre-existing asymmetric security association, we can use a key exchange similar to IKEv2 (RFC 7296). IKEv2 leverages the SIGMA protocols, which provide various methods of authenticated DH. It would also be possible to authenticate DH using symmetric passwords (e.g. Bellovin-Merritt).

5.3. 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 5.2). 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.

5.3.1. 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 cypher suite such as DHE-PSK or ECDHE-PSK when available. The shared secret exchanged during pairing MUST be used as PSK.

When using the PSK based authentication, the "psk_identity" parameter identifying the pre-shared key MUST be composed as follow, using the conventions of TLS [RFC7858]:

```
struct {

          uint32 gmt_unix_time;

          opaque random_bytes[4];

} nonce;

long_proof = HASH(nonce  | pairing_key )
proof = first 12 bytes of long_proof
psk_identity = BASE64(nonce) "." BASE64(proof)
```

In this formula, HASH SHOULD be the function SHA256 defined in [RFC4055]. Implementers MAY eventually replace SHA256 with a stronger algorithm, in which cases both clients and servers will have to agree on that algorithm during the pairing process. The first 32 bits of the nonce are set to the current time and date in standard UNIX 32-bit format (seconds since the midnight starting Jan 1, 1970, UTC, ignoring leap seconds) according to the client's internal clock. The next 32 bits of the nonce are set to a value generated by a secure random generator.

In this formula, the identity is finally set to a character string, using BASE64 ([RFC2045] section 6.8). This transformation is meant to comply with the PSK identity encoding rules specified in section 5.1 of [RFC4279].

The server will check the received key identity, trying the key against the valid keys established through pairing. If one of the key matches, the TLS connection is accepted, otherwise it is declined.

5.4. 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._tls".

Each published instance describes one server and up to 10 pairings. In the case where a node manages more than 10 pairings, it should publish as many instances as necessary to advertise all available pairings.

Each instance name is composed as follows:

```
pick a 32 bit nonce, e.g. using the Unix GMT time.
set the binary identifier to the nonce.

for each of up to 10 pairings
   hint = first 32 bits of HASH(<nonce>|<pairing key>)
   concatenate the hint to the binary identifier

set instance-ID = 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 32 bit nonce and up to 10 pairing hints result a bit string at most 332 bit long. The BASE64 conversion will produce a string that is up to 59 characters long, which fits within the 63 characters limit defined in [RFC6763].

5.5. Discovering Private Discovery Service Instances

Nodes that wish to discover Private Discovery Service Instances will issue a DNS-SD discovery request for the service type. These request will return a series of PTR records, providing the names of the instances present in the scope.

The querier SHOULD examine each instance to see whether it hints at 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 a not multiple of 32 bits,
      discard the instance.

   nonce = first 32 bits of binary.
   for each 32 bit hint after the nonce
      for each available pairing
         retrieve the key Xj of pairing number j
         compute F = hash(nonce, Xj)
         if F is equal to the 32 bit hint
            mark the pairing number j as available
```

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 hint, 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.

5.6. 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.

6. Security Considerations

This document specifies a method to protect the privacy of service publishing nodes. 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.

6.1. Attacks Against the Pairing System

There are a variety of attacks against pairing systems. They may result in compromised pairing keys. If an adversary manages to acquire a compromised key, the adversary will be able to perform private service discovery according to Section 5.5. This will allow tracking of the service. The adversary will also be able to discover which private services are available for the compromised pairing.

To mitigate such attacks, nodes MUST be able to quickly revoke a compromised pairing. This is however not sufficient, as the compromise of the pairing key could remain undetected for a long time. For further safety, nodes SHOULD assign a time limit to the validity of pairings, discard the corresponding keys when the time has passed, and establish new pairings.

This later requirement of limiting the Time-To-Live can raise doubts about the usability of the protocol. The usability issues would be mitigated if the initial pairing provided both a shared secret and the means to renew that secret over time, e.g. using authenticated public keys.

6.2. Denial of Discovery of the Private Discovery Service

The algorithm described in Section 5.5 scales as O(M*N), where M is the number of pairing per nodes 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 is amplified because the clients need to compute proofs for all the nonces presented in Private Discovery Service Instance names. One possible mitigation would be to require that such nonces correspond to rounded timestamps. 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 could make sure that proofs are only computed once per valid time stamp.

6.3. 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 5.4 start with 32 bits encoding the "UNIX" time. In order to protect against replay attacks, clients MAY verify that this time is reasonably recent.

TODO: should we somehow encode the scope in the identifier? Having both scope and time would really mitigate that attack.

6.4. Denial of Private Discovery Service

The Private Discovery Service is only available through a mutually authenticated TLS connection, which provides good protections. 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.

6.5. 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 5.3.1 start with 32 bits encoding 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 mitigated 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.

7. IANA Considerations

This draft does not require any IANA action. (Or does it? What about the _pds tag?)

8. Acknowledgments

This draft results from initial discussions with Dave Thaler, and encouragements from the DNS-SD working group members.

9. References

9.1. 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, <http://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, <http://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, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc5246>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <http://www.rfc-editor.org/info/rfc6763>.

9.2. Informative References

[I-D.ietf-dnssd-push] Pusateri, T. and S. Cheshire, "DNS Push Notifications", draft-ietf-dnssd-push-07 (work in progress), April 2016.

[I-D.ietf-dprive-dnsodtls] Reddy, T., Wing, D., and P. Patil, "DNS over DTLS (DNSoD)", draft-ietf-dprive-dnsodtls-06 (work in progress), April 2016.

[I-D.ietf-intarea-hostname-practice] Huitema, C., Thaler, D., and R. Winter, "Current Hostname Practice Considered Harmful", draft-ietf-intarea-hostname-practice-02 (work in progress), May 2016.

[I-D.ietf-tls-tls13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-13 (work in progress), May 2016.

[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>.

[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>.

[RFC1033] Lottor, M., "Domain Administrators Operations Guide", RFC 1033, DOI 10.17487/RFC1033, November 1987, <http://www.rfc-editor.org/info/rfc1033>.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <http://www.rfc-editor.org/info/rfc1034>.

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, DOI 10.17487/RFC1035, November 1987, <http://www.rfc-editor.org/info/rfc1035>.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, DOI 10.17487/RFC2782, February 2000, <http://www.rfc-editor.org/info/rfc2782>.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <http://www.rfc-editor.org/info/rfc6762>.

[RFC7626] Bortzmeyer, S., "DNS Privacy Considerations", RFC 7626, DOI 10.17487/RFC7626, August 2015, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc7858>.

Authors' Addresses

Christian Huitema
Microsoft
Redmond, WA 98052
U.S.A.

Email: huitema@microsoft.com

Daniel Kaiser
University of Konstanz
Konstanz 78457
Germany

Email: daniel.kaiser@uni-konstanz.de