Towards Distributed Multi-Signer

Proposal for an Initial Framework

Erik Bergström   Leon Fernandez   Johan Stenstam
(with help from a number of others)

Swedish Internet Foundation

March 19, 2025

~~Towards Distributed Multi-Signer~~

Proposal for an Initial Framework

Erik Bergström   Leon Fernandez   Johan Stenstam
(with help from a number of others)

Swedish Internet Foundation

March 19, 2025

~~Towards Distributed Multi-Signer~~

Signaling Zone Owner Intent Towards DNS Providers

Proposal for an Initial Framework

Erik Bergström Leon Fernandez Johan Stenstam
(with help from a number of others)

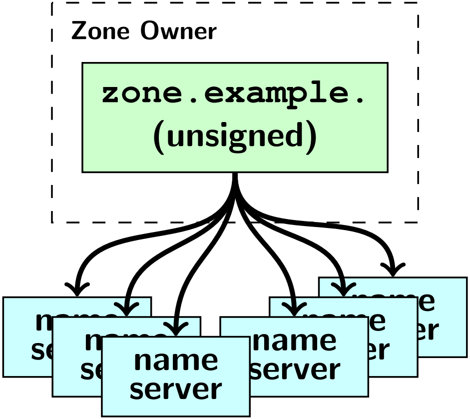
Swedish Internet Foundation

March 19, 2025

`draft-leon-dnsop-signaling-zone-owner-intent-00`

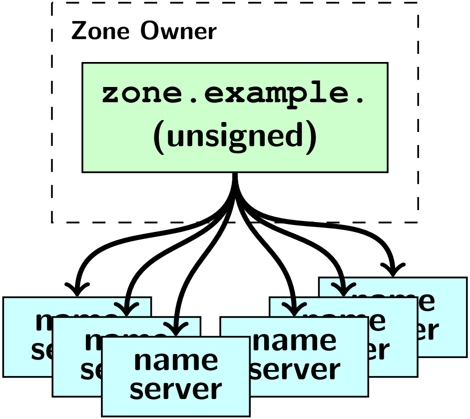
What is “Easy” When It Comes to DNS Provisioning?

Is This Easy?

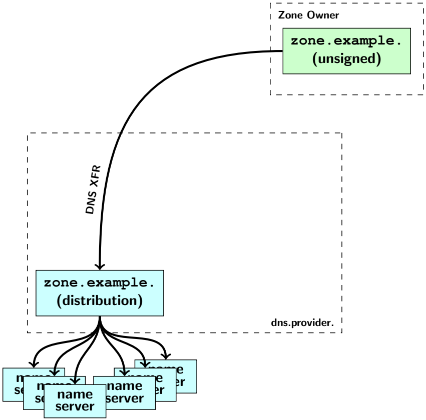


Is This Easy?

Yes.



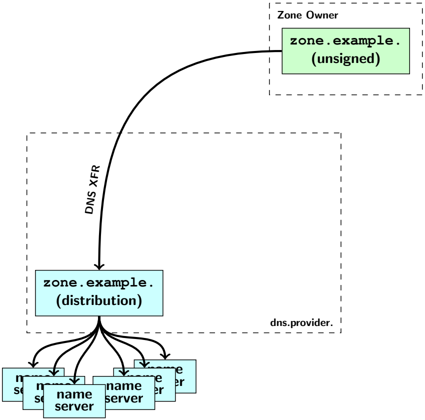
Also Easy?



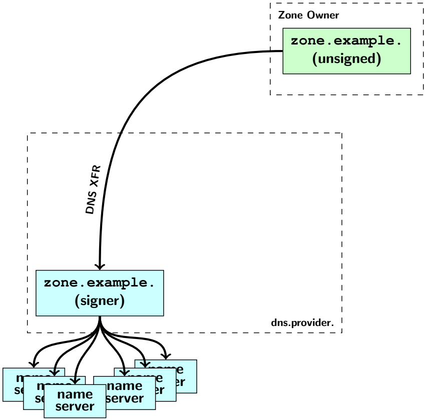
Zone Owner
zone.example.
(unsigned)
DNS XFR
zone.example.
(distribution)
dns.provider.
name
server
name
server

Also Easy?

Yes, still easy.

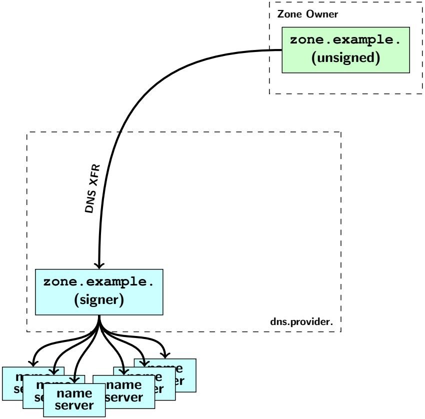


Provider Does DNSSEC Signing. Still easy?

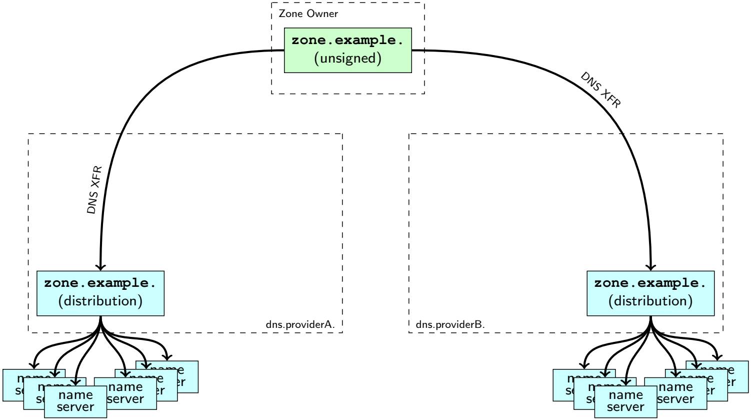


Provider Does DNSSEC Signing. Still easy?

Yes, provider doing DNSSEC signing is not a problem.

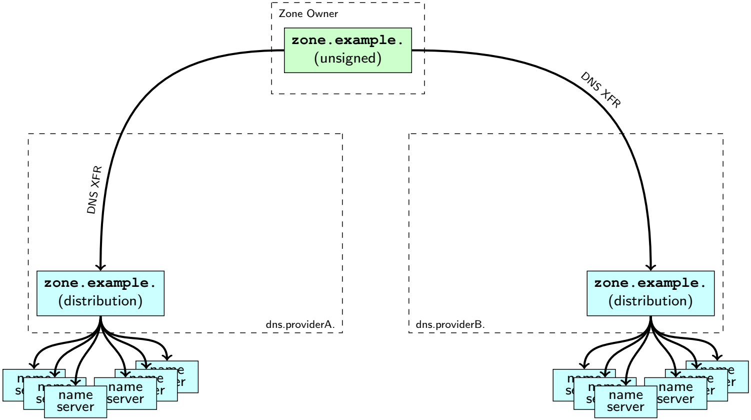


What About Multiple Providers?

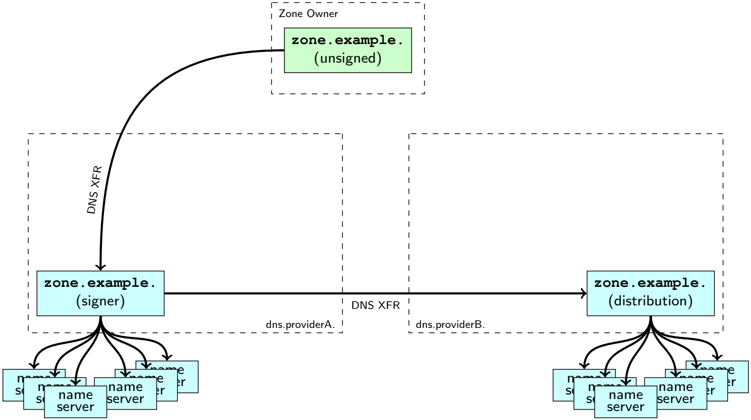


What About Multiple Providers?

Still doable (management of the `NS` RRset is the only issue).

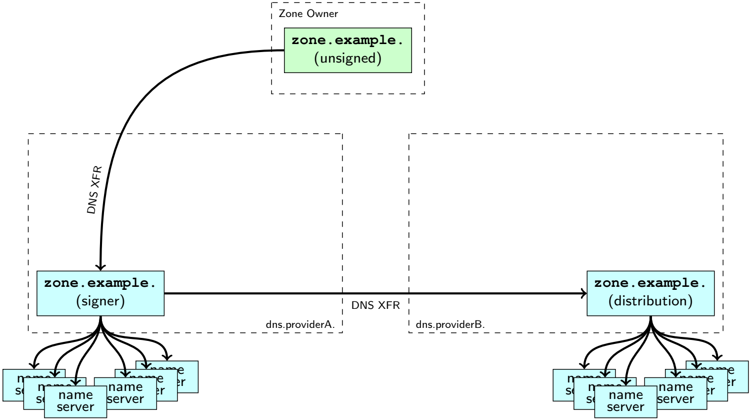


What About Multiple Providers When Using DNSSEC?



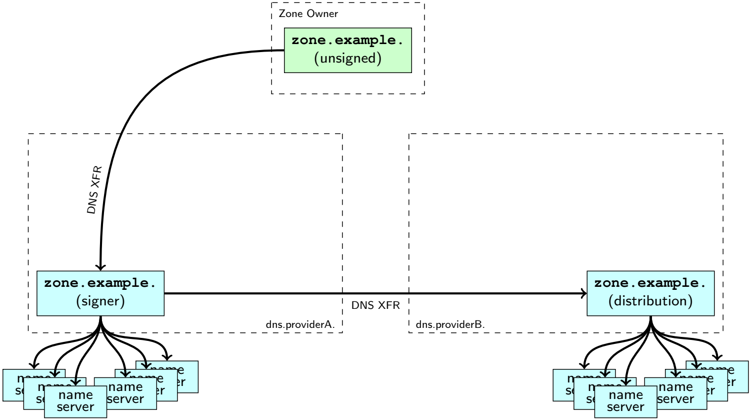
What About Multiple Providers When Using DNSSEC?

Management of the `NS` RRset is still an issue, but also arranging the zone transfer between the providers. Manual steps are needed.

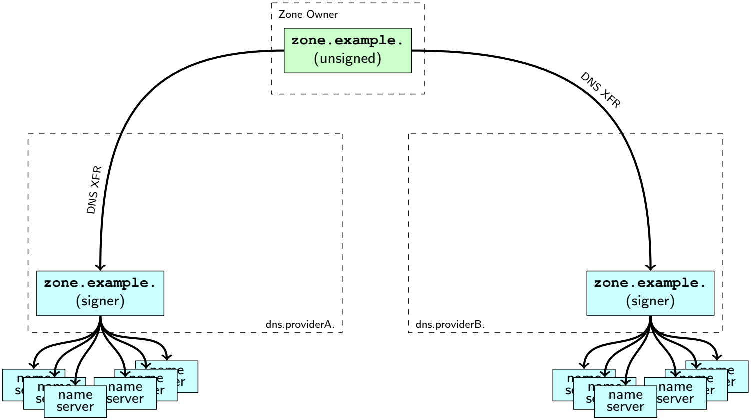


What About Multiple Providers When Using DNSSEC?

Management of the `NS` RRset is still an issue, but also arranging the zone transfer between the providers. Manual steps are needed. Not trivial, but doable.

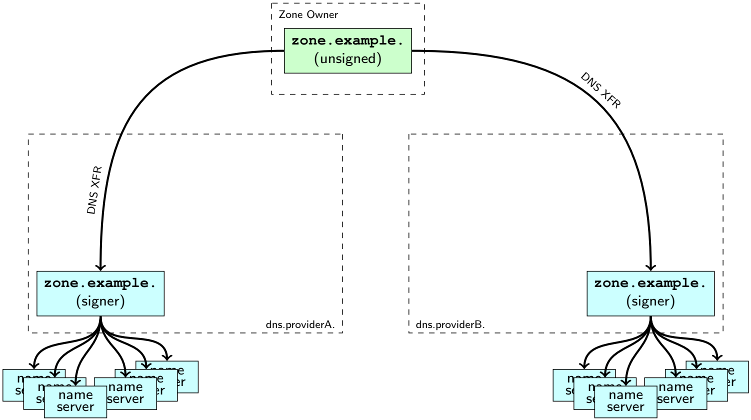


Multiple Providers That Sign The Same Zone?



Multiple Providers That Sign The Same Zone?

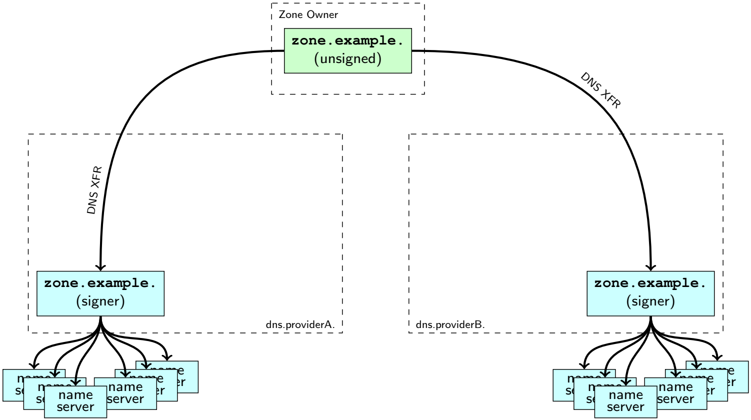
Management of the `NS` RRset is still an issue, but synchronization of the DNSKEYs is also needed. Key rollovers? Parent interaction? Doing this manually is not really an option.



Multiple Providers That Sign The Same Zone?

Management of the `NS` RRset is still an issue, but synchronization of the DNSKEYs is also needed. Key rollovers? Parent interaction? Doing this manually is not really an option.

This is (currently) hard.



Our Focus

We focus on **automation of specific, critical, DNS configuration** needed to enable production use of multi-signer.

In particular:

- automation **across multiple involved organizations**
- automation with a **minimum of knobs to tweak**

Both of these are crucially important. With too many knobs to tweak, the result will be too complicated to be usable. And automation inside a single organization. . . is really their problem, not an IETF issue.

Our Focus

We focus on **automation of specific, critical, DNS configuration** needed to enable production use of multi-signer.

In particular:

- automation **across multiple involved organizations**
- automation with a **minimum of knobs to tweak**

Both of these are crucially important. With too many knobs to tweak, the result will be too complicated to be usable. And automation inside a single organization. . . is really their problem, not an IETF issue.

The work on automatic delegation synchronization via **`DSYNC`** and DNS UPDATE is related

- that's another case where the core issue is **automation across organizations**

Introduction to Distributed Multi-Signer

Requirements:

- There must be no central controller that has the ability to modify customer zone data.
- Changes to the data that is signed must be automatically and immediately detected. In particular changes to the **`DNSKEY`** RRset.

This led to further development of the **"AGENT"** that we already use in the implementation of automatic delegation synchronization via DDNS (see **`draft-johani-dnsop-delegation-mgmt-via-ddns`**).

Introduction to Distributed Multi-Signer

Requirements:

- There must be no central controller that has the ability to modify customer zone data.
- Changes to the data that is signed must be automatically and immediately detected. In particular changes to the **`DNSKEY`** RRset.

This led to further development of the **"AGENT"** that we already use in the implementation of automatic delegation synchronization via DDNS (see **`draft-johani-dnsop-delegation-mgmt-via-ddns`**).

But there are also other previously known requirements:

- The **zone owner must be able to designate** which DNS providers should sign a zone.
- The DNS providers must be able to locate each other and establish secure communications.

We now propose a new record, **`HSYNC`**, as part of a solution.

Can We Make The Signer "Multi-Signer Agnostic"?

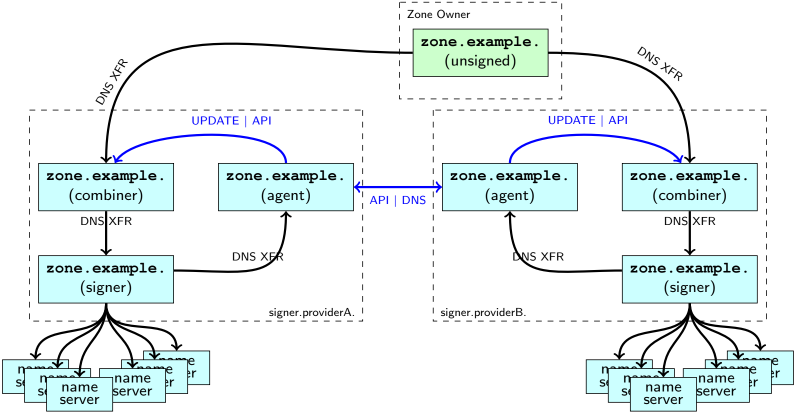
One major problem with earlier multi-signer attempts is that most signers act **either** as a secondary **or** as a recipient of DNS UPDATEs. Not both.

- Therefore needed changes to the zone (due to multi-signer synchronization) should arrive **upstream** of the signer.
- But to keep the zone owner outside of the multi-signer synchronization complexity, the changes should also arrive **downstream** of the zone owner.

Enter the COMBINER.

- The **combiner** is a new entity that essentially acts as a proxy for zone transfers between the zone owner and the signer.
- Additionally, it is able to modify the crucial apex RRsets (**`DNSKEY`**, **`CDS`**, **`CSYNC`** and **`NS`**). Only those four RRsets, and only on request from the **agent**.

Distributed Multi-Signer Architecture, Evolved



Source of Truth

The **combiner** changes the source of truth for the zone.

In an architecture with multiple providers that contribute data for synchronization it is necessary to have **three distinct "sources of truth"**:

- the zone owner for "normal zone content" (obviously)
- any signers for "DNSSEC data" (quite obviously)
- a third entity (the **agent**) that manages data synchronized via multi-signer processes in collaboration with other agents. This data is added to the zone via the **combiner**.

The Proposed **HSYNC** RRset

The **HSYNC** record is used by the zone owner to signal how DNS providers should handle the zone.

It has five fields in RDATA: **State**, **NSmgmt**, **Sign**, **Provider**, (a domain name that identifies the DNS provider) and **Upstream** (a domain name that identifies the upstream provider, or "**.**", if manual or no upstream).

```
zone.example. IN HSYNC {State} {NSmgmt} {Sign}  identity.providerA. {Upstream}
zone.example. IN HSYNC {State} {NSmgmt} {Sign}  identity.providerB. {Upstream}
zone.example. IN HSYNC {State} {NSmgmt} {Sign}  identity.providerC. {Upstream}
```

Example:

```
zone.example. IN HSYNC ON   OWNER  SIGN  identity.providerA.  agent.zone.owner.
zone.example. IN HSYNC ON   OWNER  SIGN  identity.providerB.  agent.zone.owner.
zone.example. IN HSYNC OFF  OWNER  SIGN  identity.providerC.  agent.zone.owner.
```

The Proposed **HSYNC** RRset

The **HSYNC** record is used by the zone owner to signal how DNS providers should handle the zone.

It has five fields in RDATA: **State**, **NSmgmt**, **Sign**, **Provider**, (a domain name that identifies the DNS provider) and **Upstream** (a domain name that identifies the upstream provider, or "." , if manual or no upstream).

OWNER|AGENT
NS management by owner or MSAs

```
zone.example. IN HSYNC {State} {NSmgmt} {Sign}  identity.providerA. {Upstream}
zone.example. IN HSYNC {State} {NSmgmt} {Sign}  identity.providerB. {Upstream}
zone.example. IN HSYNC {State} {NSmgmt} {Sign}  identity.providerC. {Upstream}
```

ON|OFF
Onboarded, or being offboarded

SIGN|NOSIGN
DNS Provider should sign zone

Example:

```
zone.example. IN HSYNC ON   OWNER  SIGN  identity.providerA.  agent.zone.owner.
zone.example. IN HSYNC ON   OWNER  SIGN  identity.providerB.  agent.zone.owner.
zone.example. IN HSYNC OFF  OWNER  SIGN  identity.providerC.  agent.zone.owner.
```

Confusing. Please Show Some Possible **HSYNC** Setups.

Yes, this is a bit confusing. Here are some setups that we would like to be able to describe via appropriate **HSYNC** RRsets:

- **Case A: "Normal multi-signer" setup:** multiple DNS providers, all of which sign the customer zone.
- **Case B: "Normal single signer" setup:** multiple DNS providers, only one of which sign the customer zone. All other providers fetch the zone from the provider that signs.
- **Case C: Owner signs:** Multiple DNS providers for a zone signed by the zone owner.
- **Case D: Multi-provider setup for unsigned zone:** multiple DNS providers for an unsigned zone.

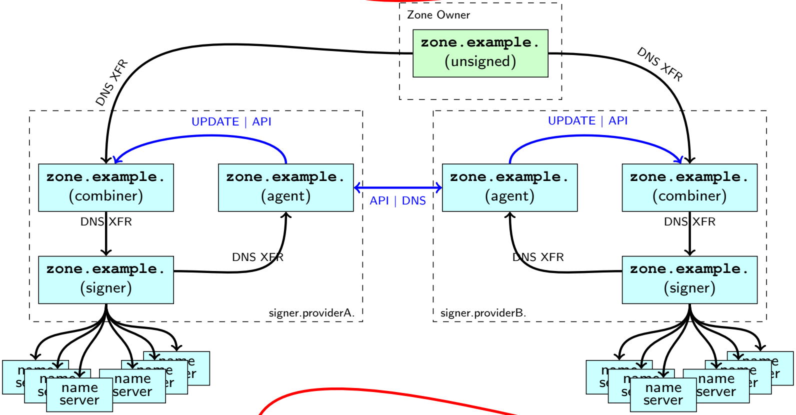
In each case the management of the zone **NS** RRset is either with the zone owner ("traditional model") or managed by the **agents** ("automated").

Case A: Multiple Signers

```
zone.example.    IN HSYNC ON OWNER SIGN   agent.signer.providerA.   .
zone.example.    IN HSYNC ON OWNER SIGN   agent.signer.providerB.   .
```

Manual upstream

Both providers sign the zone




```
zone.example.    IN HSYNC ON AGENT SIGN   agent.signer.providerA.   .
zone.example.    IN HSYNC ON AGENT SIGN   agent.signer.providerB.   .
```

Automatic NS management

Oooh, That's Complicated! Lots of Shiny Boxes!

Well, everything is relative.

Any "DNS provider" that is of nontrivial size has lots of "complexity" inside, of which this is just a small part.

More relevant here is the complexity from the "zone owner" perspective. And there the "change" is essentially from transferring the zone to a DNS provider that does **two things**:

- Sign the zone.
- Distribute the signed zone to a set of downstream nameservers.

To transfer the zone to a DNS provider that does **three things**:

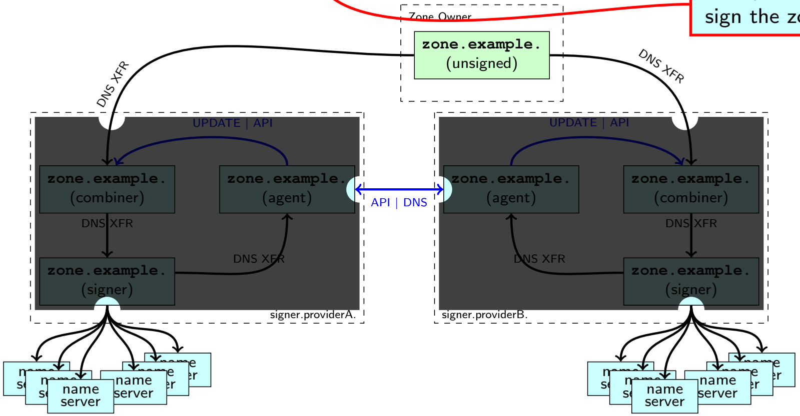
- Sign the zone.
- **Communicate with other DNS providers according to instructions in the `HSYNC` RRset.**
- Distribute the zone to a set of downstream nameservers.

This!

Case A: Multiple Signers, Second Look

```
zone.example.   IN HSYNC ON OWNER SIGN agent.signer.providerA.  .
zone.example.   IN HSYNC ON OWNER SIGN agent.signer.providerB.  .
```

Both providers sign the zone



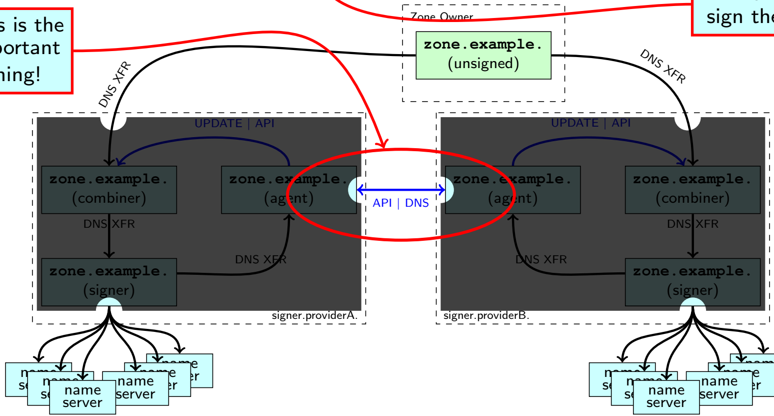

```
zone.example.   IN HSYNC ON AGENT SIGN agent.signer.providerA.  .
zone.example.   IN HSYNC ON AGENT SIGN agent.signer.providerB.  .
```

Case A: Multiple Signers, Second Look

```
zone.example.   IN HSYNC ON OWNER SIGN  agent.signer.providerA.  .
zone.example.   IN HSYNC ON OWNER SIGN  agent.signer.providerB.  .
```

Both providers sign the zone

This is the important thing!




```
zone.example.   IN HSYNC ON AGENT SIGN  agent.signer.providerA.  .
zone.example.   IN HSYNC ON AGENT SIGN  agent.signer.providerB.  .
```

Applicability of the **HSYNC** Record Beyond Multi-signer?

The **HSYNC** record is designed to be a general mechanism for introducing different DNS providers for a zone to each other.

- The intent was to enable them to synchronize the different multi-signer "processes".

However, once introduced, they can talk about other things than multi-signer synchronization.

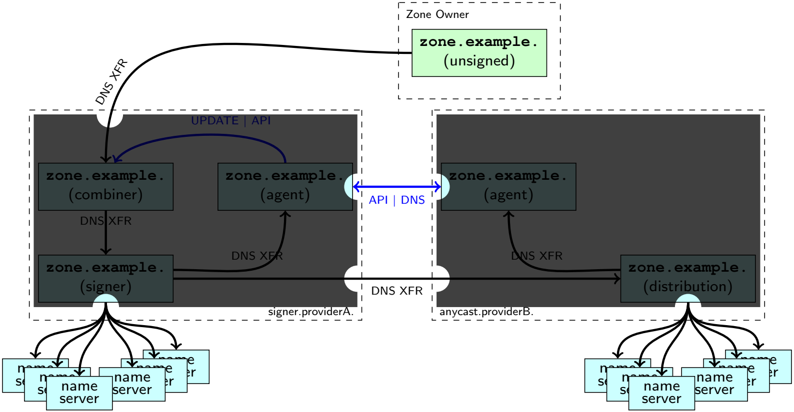
- The **HSYNC NSmgmt** field is used to signal how the zone owner wants DNS providers to deal with the **NS** RRset.
- The **HSYNC Upstream** field is an instruction for how the zone transfer chain should be built (or not).

Both of these are useful outside the multi-signer context.

- In fact, they may be useful even for **unsigned zones**.

Case B: Single Signer, Multiple DNS Providers

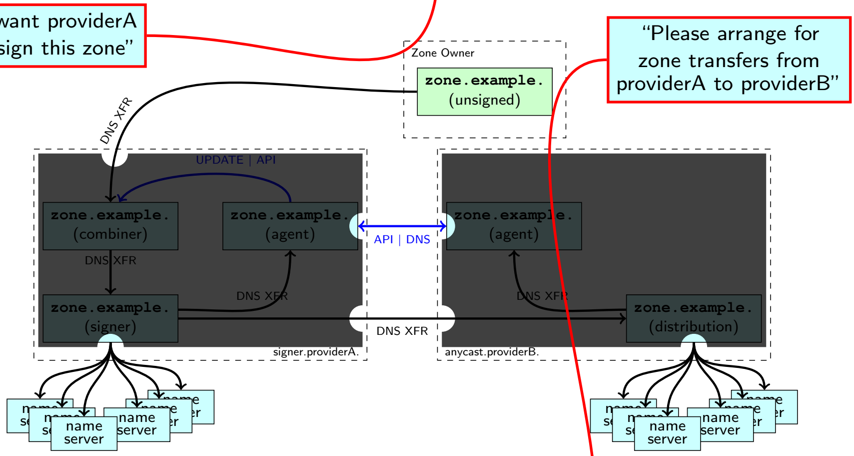
```
zone.example.   IN HSYNC ON OWNER   SIGN     agent.signer.providerA.   .
zone.example.   IN HSYNC ON OWNER   NOSIGN   agent.anycast.providerB.  agent.signer.providerA.
```




```
zone.example.   IN HSYNC ON AGENT   SIGN     agent.signer.providerA.    .
zone.example.   IN HSYNC ON AGENT   NOSIGN   agent.anycast.providerB.   agent.signer.providerA.
```

Case B: Single Signer, Multiple DNS Providers

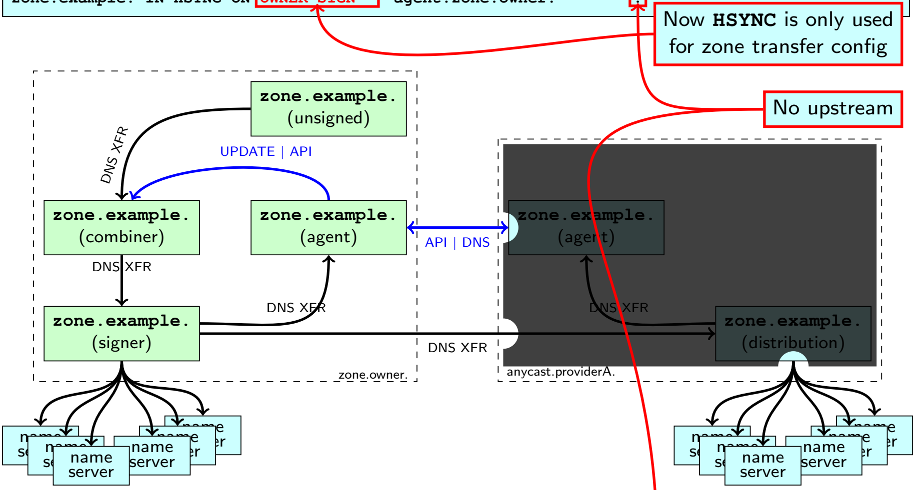
```
zone.example.   IN HSYNC ON OWNER  SIGN     agent.signer.providerA.   .
zone.example.   IN HSYNC ON OWNER  NOSIGN   agent.anycast.providerB. agent.signer.providerA.
```




```
zone.example.   IN HSYNC ON AGENT  SIGN     agent.signer.providerA.   .
zone.example.   IN HSYNC ON AGENT  NOSIGN   agent.anycast.providerB.  agent.signer.providerA.
```

Case C: Zone Signed by Owner, Multiple DNS Providers

```
zone.example. IN HSYNC ON OWNER NOSIGN agent.anycast.providerA. agent.zone.owner.
zone.example. IN HSYNC ON OWNER SIGN   agent.zone.owner.          .
```

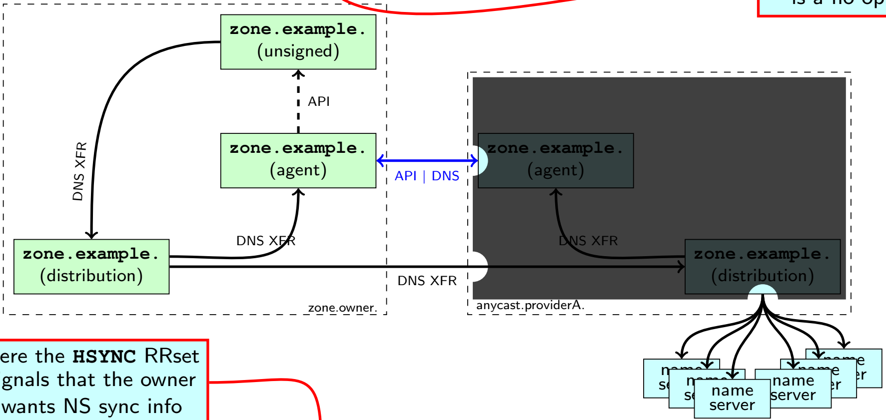



```
zone.example. IN HSYNC ON AGENT  NOSIGN  agent.anycast.providerA.  agent.zone.owner.
zone.example. IN HSYNC ON AGENT  SIGN    agent.zone.owner.          .
```

Case D: Unsigned Zone, Multiple DNS Providers

```
zone.example. IN HSYNC ON OWNER NOSIGN  agent.anycast.providerA.  agent.zone.owner.
zone.example. IN HSYNC ON OWNER NOSIGN  agent.zone.owner.         .
```

In this case the **HSYNC** RRset is a no-op



Here the **HSYNC** RRset signals that the owner wants NS sync info

```
zone.example. IN HSYNC ON AGENT NOSIGN  agent.anycast.providerA.  agent.zone.owner.
zone.example. IN HSYNC ON AGENT NOSIGN  agent.zone.owner.         .
```

Security Model

Before trusting an RRset (the **HSYNC** RRset) in an unsigned zone as the source to major configuration changes it is important to ask some questions.

Three immediate questions are:

- How can the DNS providers trust the correctness of the **HSYNC** RRset?
- How can the **agents** establish secure communication with each other?
- Does the information in the **HSYNC** RRset aid a potential attacker?

The answer to the first question is that the integrity of the unsigned zone is as secure as the security of the zone transfer.

- There are all sorts of alternatives in this space, including everything from TSIG, via VPNs and locked IP-addresses to XFR-over-TLS.

Also note that no part of the actual zone transfer chain is disclosed via the **HSYNC** RRset.

Security Model, cont'd

When it comes to the establishment of secure communication between **agents**, there is **a fundamental requirement on DNSSEC** for AGENTs, both when publishing data and when looking up data.

The **agents** identify each other from the **`HSYNC`** RRset and then use the "identity" field to look up the different information depending on the transport used.

- **DNS transport:** look up **`_dns._tcp.{identity} URI`**. Then look up the corresponding **`SVCB`** and finally the **`KEY`** record.
- **API transport:** look up **`_https._tcp.{identity} URI`**. Then look up the corresponding **`SVCB`** and finally the **`TLSA`** record.

When communicating over DNS, the **`KEY`** record is used to verify the messages from the other **agent**. Likewise, when using the API, the **`TLSA`** record is used to verify the certificate of the other **agent**.

Summary

We propose a new DNS record type, "**`HSYNC`**" (for "**H**orizontal **SYNC**hronization").

- **`HSYNC`** is used by the zone owner to distribute **instructions** to DNS providers, in a transparent way.
- The instructions describe the common choices for DNS providers:
  - Who should sign the zone?
  - How should the NS RRset for the zone be managed?
  - How should the zone transfers from the owner to the DNS providers be set up?

`HSYNC` started in a multi-signer context. But it was quickly realized that the applicability was much broader.

While **`HSYNC`** works well in a single provider context, the real benefits are when there are multiple providers involved (which is not uncommon).

Next Steps

We have a working implementation of the **HSYNC** record in the TDNS framework

- the **tdns-agent** has been enhanced to detect changes to the **HSYNC** RRset, and establish secure communication with remote agents
- there is a prototype implementation of the **tdns-combiner**
- there is messaging between providers for an initial sync process (we choose synchronization of the NS RRset)
- the **tdns-agent** has a new module for evaluation of data from remote agents against local policy.
- the details of this communication will be the topic of an upcoming internet-draft.

We are now looking for feedback on the current draft from the IETF community and hopefully WG adoption.

Status Update `delegation-mgmt-via-ddns`
IETF122 Bangkok

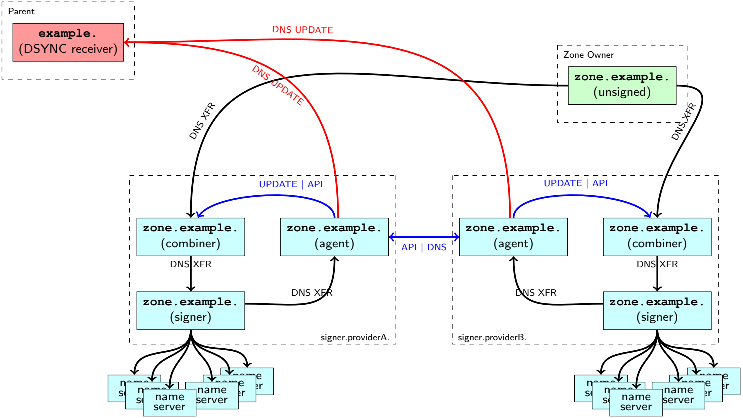
Erik Bergström Leon Fernandez Johan Stenstam

Swedish Internet Foundation

March 19, 2025

Delegation Synchronization via DDNS

Example scenario, based on the previous examples:



draft-...-delegation-mgmt-via-ddns-05

- The TDNS reference implementation is complete.
  - It now supports automatic bootstrapping of SIG(0) keys according to the same model as for CDS described in RFC8078, section 4.2.
  - It works the same via the TDNS authoritative server or via **tdns-agent** (to be able to use a “real” auth server, like BIND9, NSD, Knot-DNS, etc).
- Based on that experience, a couple of issues had to be addressed.
- In particular, a mechanism was needed for both child and parent to express requirements for SIG(0) key trust bootstrap. Examples include:
  - The parent needs to be able to signal that it does (or does not) support automatic bootstrapping of the child public SIG(0) key.
  - The child needs to be able to signal that it doesn’t want the parent to do automatic bootstrapping.

...-delegation-mgmt-via-ddns-05, cont'd

Secondly,

- The parent is no longer only consuming information from the child, but also publishing data in response.
- For that reason it was not sufficient to only sign the UPDATE from the child and we now also sign the response from the parent.

The draft has been updated to address these issues and now refers to a new draft (see next slide) for the key state reporting mechanism.

draft-berra-dnsop-keystate-01

- Another example is that the child needs to be able to inquire about the current state of a SIG(0) key in the parent, and the parent must be able to report back the details. Examples include:
  - Key is known and trusted (i.e. all is good)
  - Key is known but not yet trusted (bootstrapping in progress)
  - Key is refused (example: algorithm not accepted by the parent)
  - Key is invalid (bits are bad)
  - Key is known, but validation failed (for some reason)
  - etc.

To address these requirements the new draft, **`draft-dnsop-berra-keystate-01`**, describes a mechanism for both child and parent to report key state information to each other.

- This is implemented as a new EDNS(0) option.
- It is sort of a "bi-directional" version of EDE and not restricted to "errors".

Consequences.

As we now sign both the initial message (from the child) and the response (from the parent) and verify then in the recipient we have ended up with what is essentially a "DNS Transaction Machine".

- We have used this experience in the design of the **HSYNC** agent-to-agent communication, as an alternative to using a REST API.
- At this stage it is not clear that there would be any huge advantages doing the agent-to-agent communication over DNS.
- But it is quite clear that it is possible.

Questions?

Thanks & Contact Information

erik.bergstrom **leon.fernandez** **johan.stenstam**	$\left.\vphantom{\begin{matrix}a\\b\\c\end{matrix}}\right\}$ **@internetstiftelsen.se**

Code **https://github.com/johanix/tdns**