Signalling Zone Owner Intention

Proposal for a Complete Framework

Johan Stenstam[1] Steve Crocker[2]
(with help from a number of others)

[1]Swedish Internet Foundation

[2]Edgemoor Research Institute

June 6, 2025

~~Signalling Zone Owner Intention~~

Proposal for a Complete Framework

Johan Stenstam[1] Steve Crocker[2]
(with help from a number of others)

[1]Swedish Internet Foundation

[2]Edgemoor Research Institute

June 6, 2025

~~Signalling Zone Owner Intention~~

A Unified Architecture for Multi-Provider DNS

Proposal for a Complete Framework

Johan Stenstam[1] Steve Crocker[2]
(with help from a number of others)

[1]Swedish Internet Foundation

[2]Edgemoor Research Institute

June 6, 2025

The Multi-Signer Story So Far

- RFC8901 is the first IETF standard for multi-signer.
- In 2022 a colleague and I designed and implemented "M^{U}S$_{I}$C ", aka "Multi-Signer Controller", with logic based on RFC8901.
- M^{U}S$_{I}$C worked fine technically, but fell flat on its face, as we had not understood the business aspects of the multi-signer problem.
- Another problem with M^{U}S$_{I}$C was that it was designed as a special case solution of "the multi-signer problem", not as a general framework for the much larger multi-provider problem.

The Multi-Signer Story So Far

- RFC8901 is the first IETF standard for multi-signer.
- In 2022 a colleague and I designed and implemented "M^{U}S$_{I}$C ", aka "Multi-Signer Controller", with logic based on RFC8901.
- M^{U}S$_{I}$C worked fine technically, but fell flat on its face, as we had not understood the business aspects of the multi-signer problem.
  - **This time we're trying to get also that part right.**
- Another problem with M^{U}S$_{I}$C was that it was designed as a special case solution of "the multi-signer problem", not as a general framework for the much larger multi-provider problem.
  - **This issue is also addressed.**

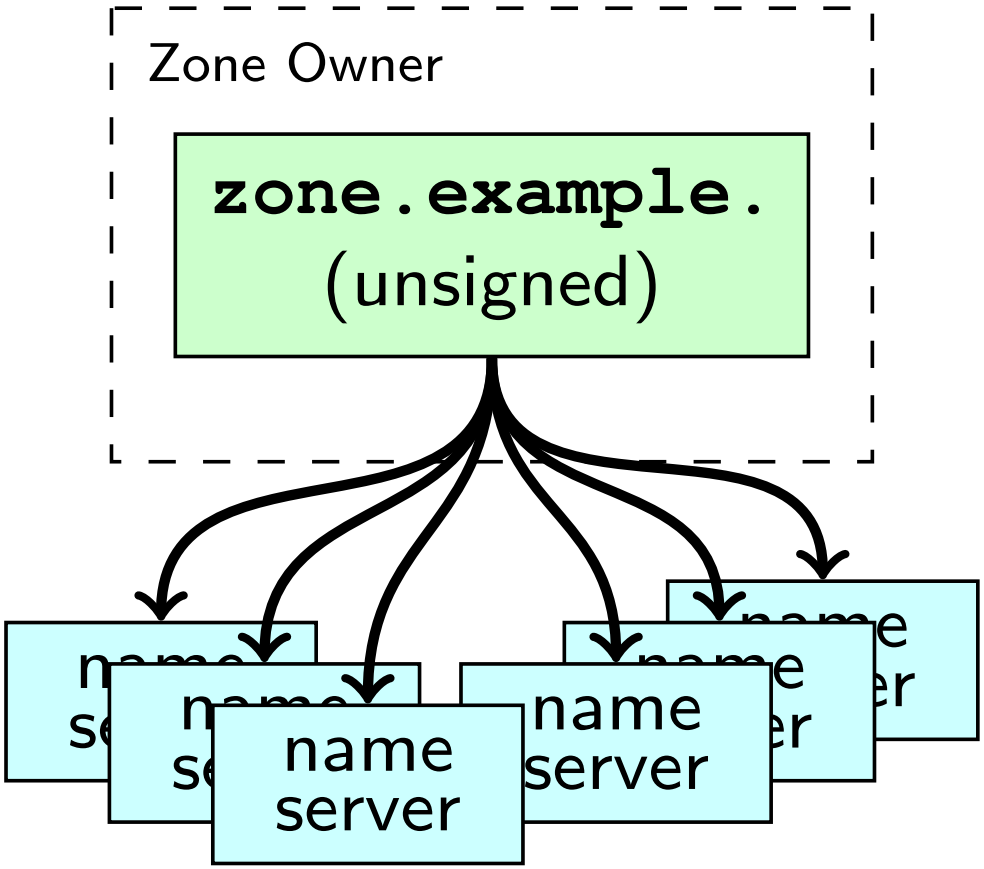
Multi-Provider Scenarios Are Interesting Because...

- They are much more common than multi-signer scenarios
  - Multi-provider is the "best current practice"
  - There are likely already millions of zones using multiple providers
- Enen more importantly, multi-provider is the norm for the Internet's **most important zones**.
- Multi-provider has its own set of requirements and challenges that require a thought-out architecture.
- Given a general architecture for multi-provider DNS, multi-signer suddenly becomes just one more aspect of the same architecture.

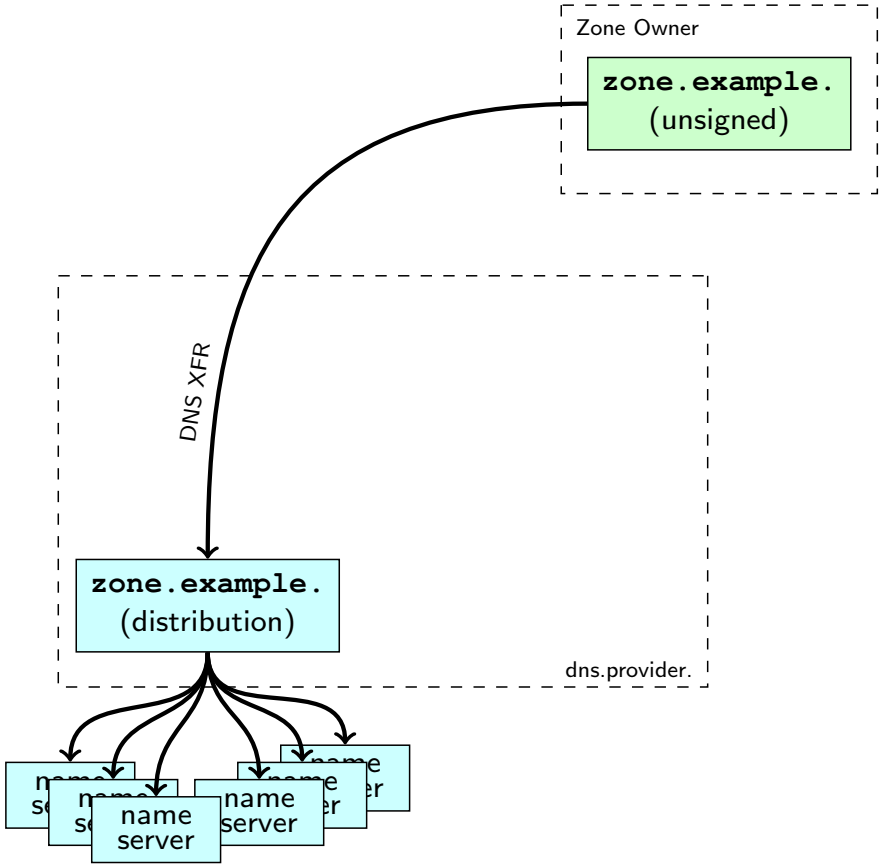
So Let's Take A Look At Some Scenarios

Here is a set of possible scenarios of increasing complexity.

Scenario 1: As Simple As It Gets...

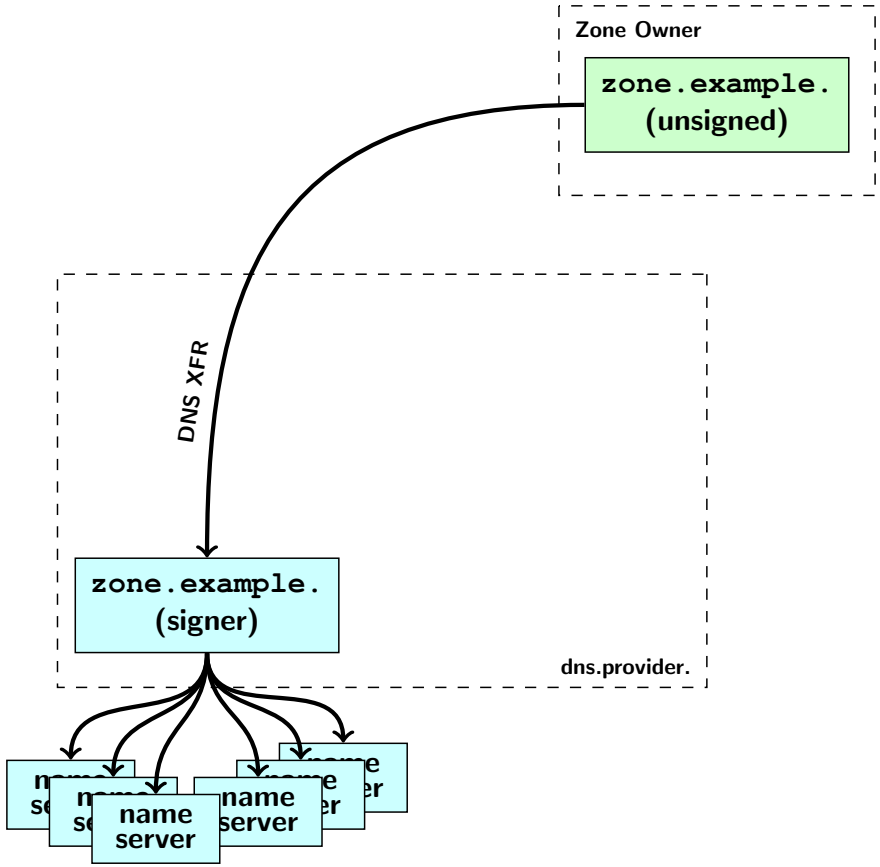


Scenario 2: Provider Only Publishes the Zone



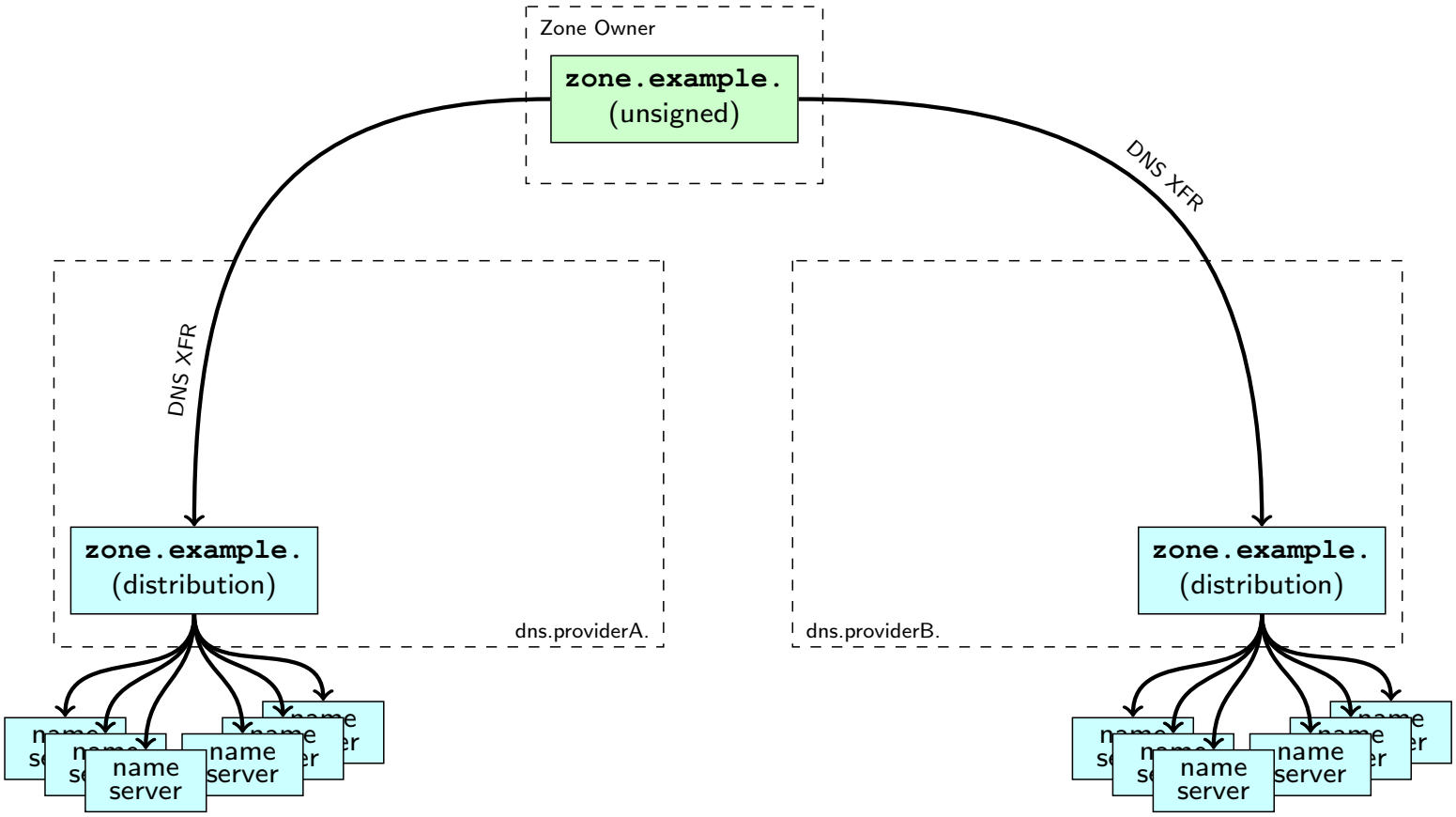
Scenario 3: Provider Does DNSSEC Signing

This is still simple, provider doing DNSSEC signing is not a problem.



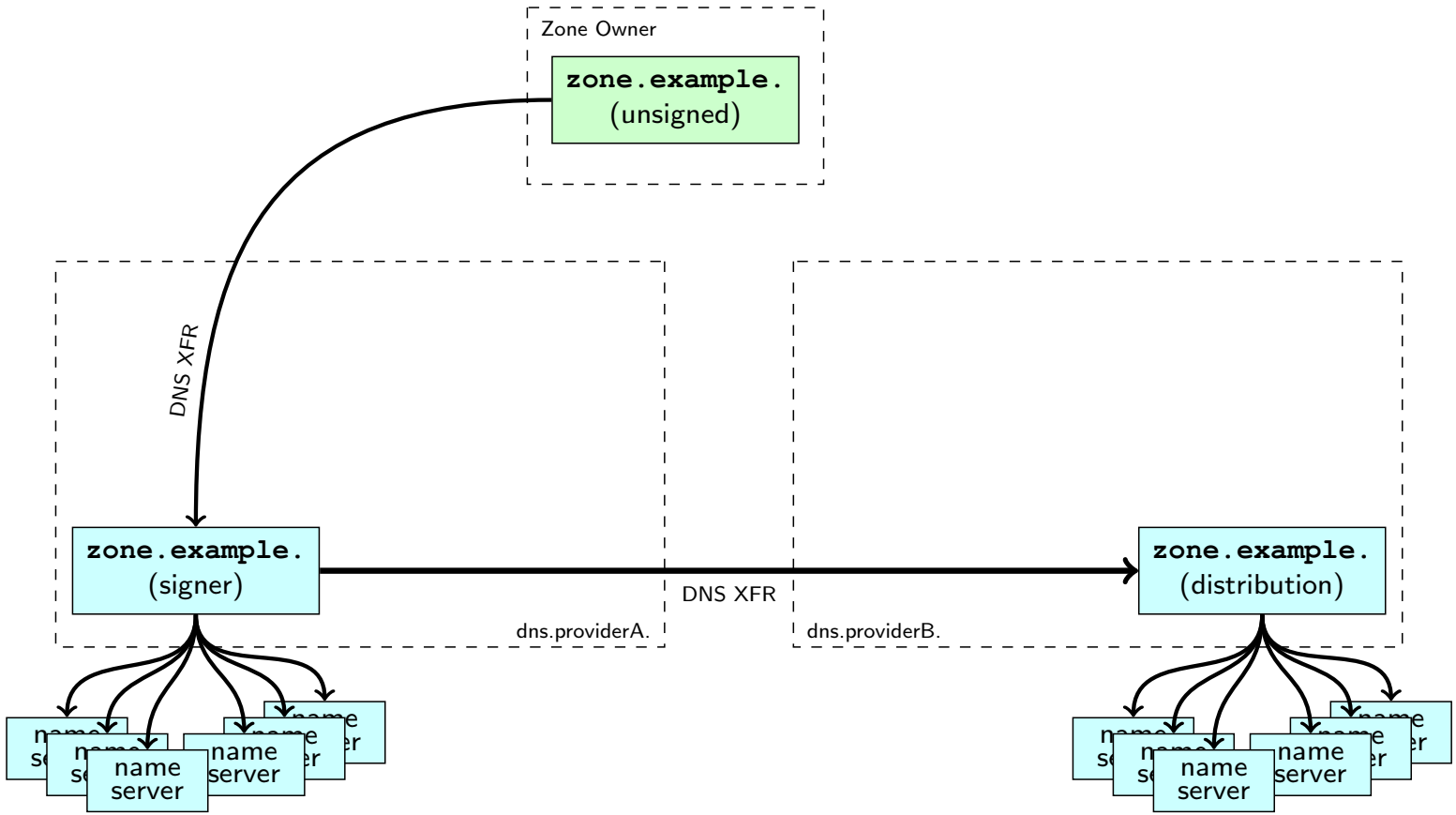
Scenario 4: Multiple Providers?

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



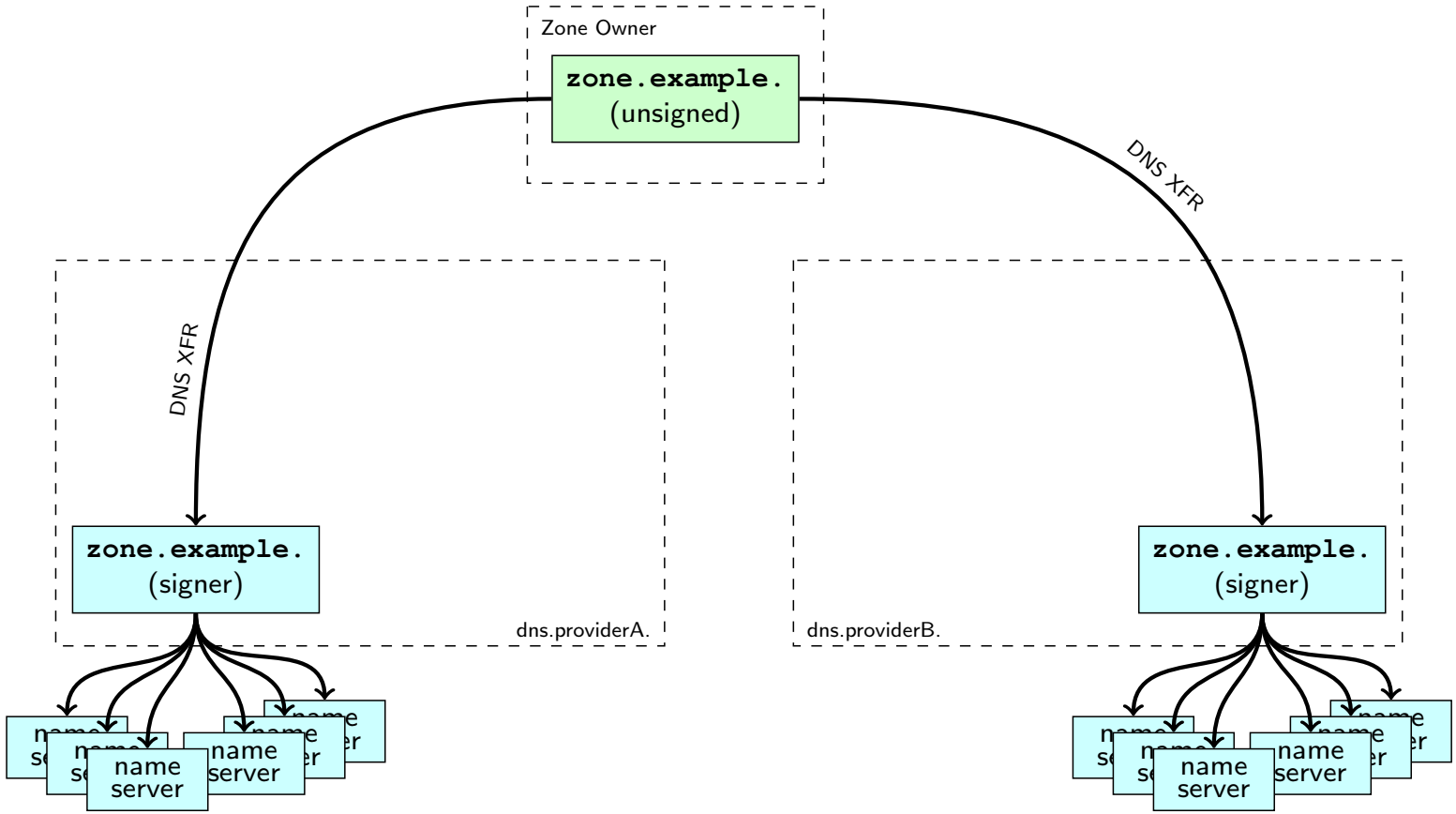
Scenario 5: Multiple Providers When Using DNSSEC

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.



Scenario 6: Multiple Providers That Sign The Same Zone

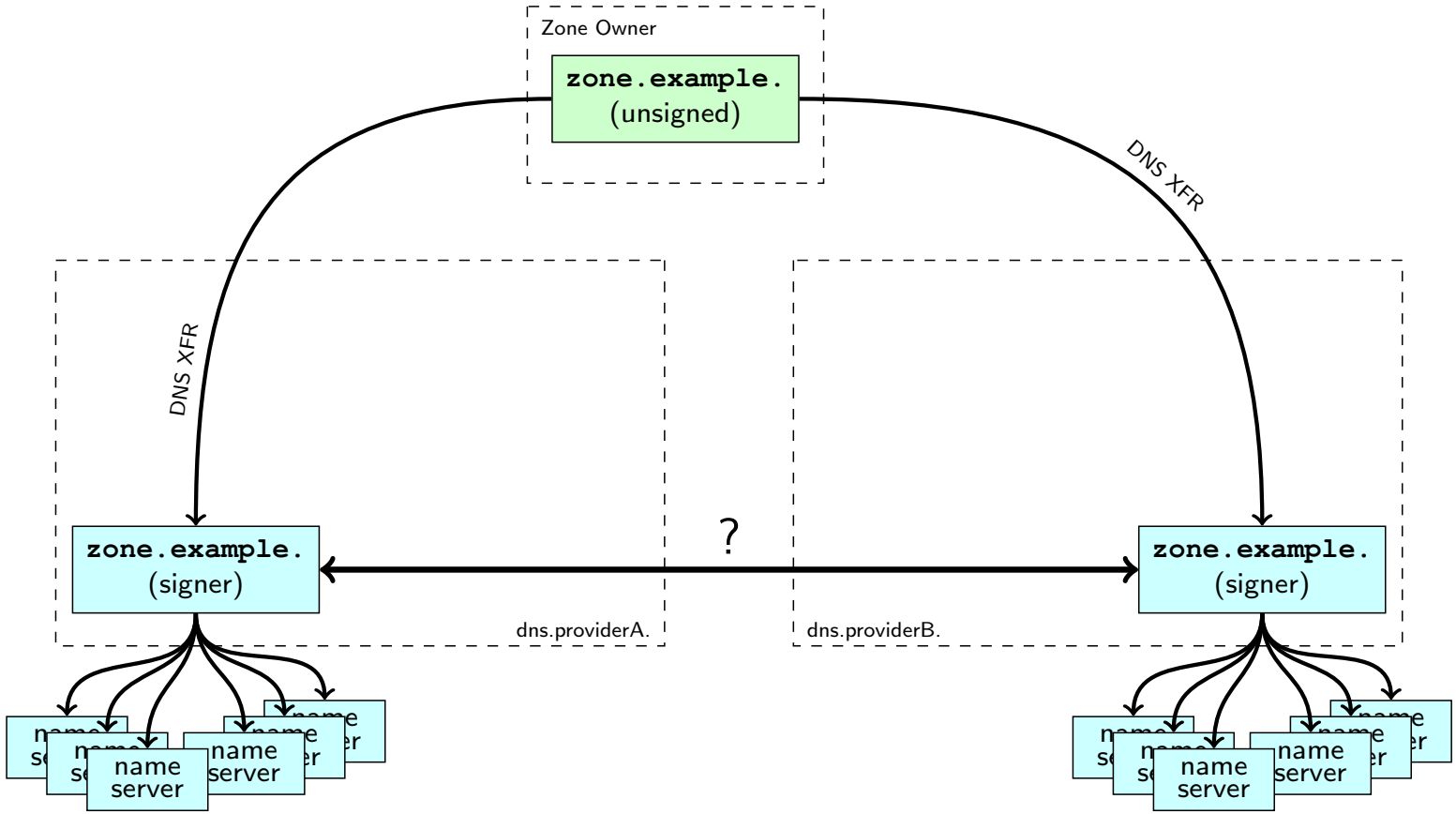
Management of the `NS` RRset is still an issue, but synchronization of the DNSKEYs is also needed. Key rollovers? Parent interaction? Doing this manually is not really an option.



Scenario 6: 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 "multi-signer", and it is (currently) hard.



What is the Take-Away From the Scenarios?

They may be summarized as a set of requirements for **what** (is needed):

- A robust mechanism for joint management of the **`NS`** and **`DNSKEY`** RRsets (and in some cases also the **`CDS`** and **`CSYNC`** RRsets).
- Support for DNSSEC key rollovers in sync between providers
  - As one provider rolls keys, it must wait for the other providers to publish the new keys before continuing with the rollover
- Support for management of the delegation information in the parent.
- Support for on- and offboarding of providers.
- Support for provider-to-provider zone transfers.

Last, but not least:

- A robust mechanism for expressing who is responsible for each of the above requirements.

The Current "Research Question"

How to **design robust automation** for **multi-provider scenarios in general** and **multi-signer scenarios in particular**

Specifically:

- 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.

The Current "Research Question"

How to **design robust automation** for **multi-provider scenarios in general** and **multi-signer scenarios in particular**

Specifically:

- 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.

- Arguably, a major reason for the slow adoption of DNSSEC is that we **exposed too many options, buttons, alternatives, etc**, to the operator.

The Current "Research Question"

How to **design robust automation** for **multi-provider scenarios in general** and **multi-signer scenarios in particular**

Specifically:

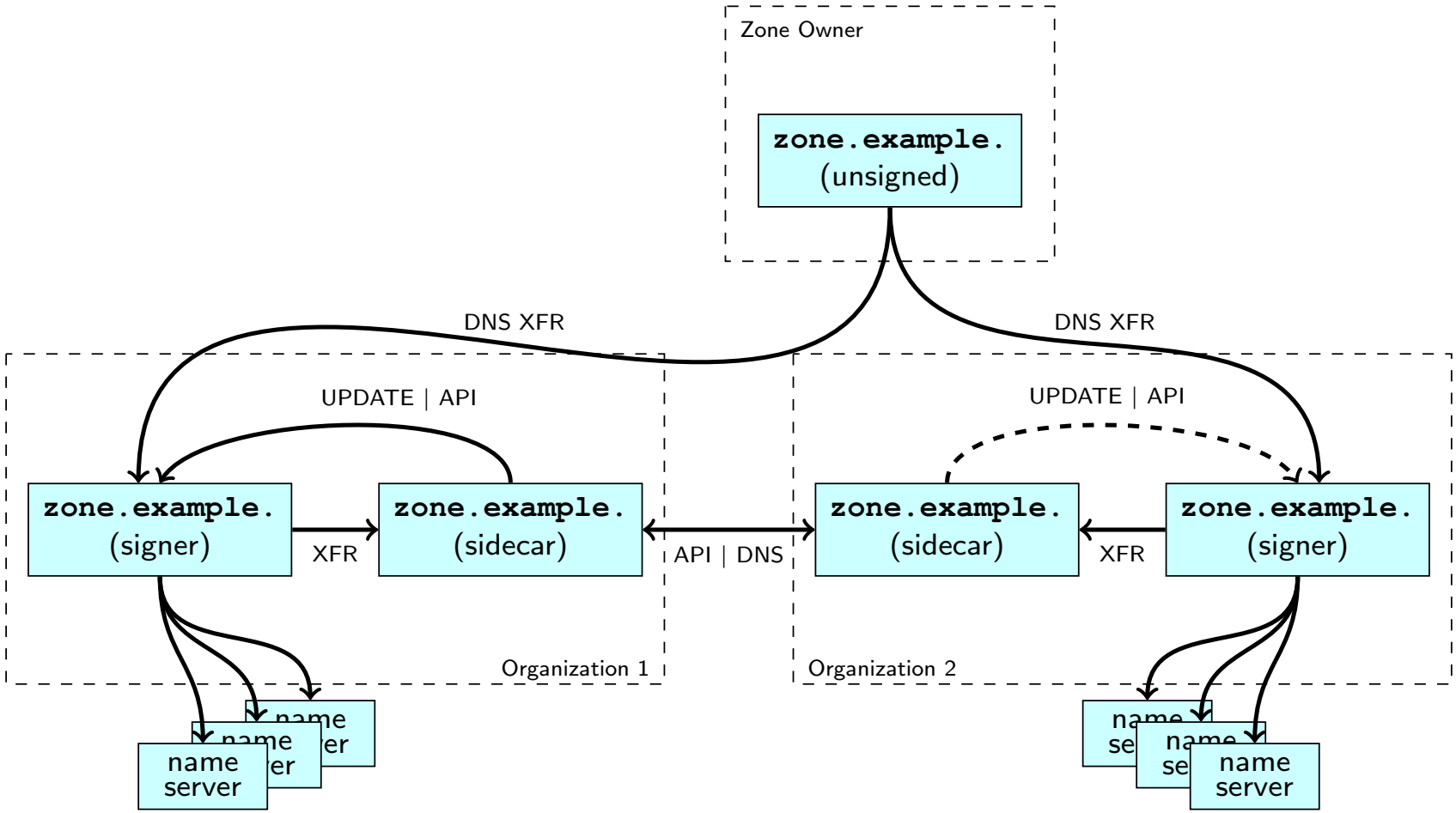
- automation **across multiple involved organizations**
- automation with a **minimum of knobs to tweak**

Both of these are crucially important. With too many knobs to tweak, the result will be too complicated to be usable.

- Arguably, a major reason for the slow adoption of DNSSEC is that we **exposed too many options, buttons, alternatives, etc**, to the operator.

This resulted in an architecture where the DNS providers communicated with each other via a new component that we refer to as the "**agent**" (previously called "sidecar"). The reason for the **agent** is to avoid adding multi-provider complexity to the **signer**.

Distributed Multi-Signer Architecture (Autumn 2024)



More Architecture Evolution

This architecture got several things right, but at least one thing is missing:

- It didn't resolve the problem of how to combine the RRsets that need to be changed as a result of multi-provider DNS with the rest of the zone data.

The problem is that for multi-signer to work without involving the zone owner in the operations, it is necessary for the DNS providers (in particular DNS providers that provide signing services) to be able to make certain changes to the zone.

- Eg., the DNS providers must be able to modify the apex RRsets **`DNSKEY`** and **`NS`** (and possibly also **`CDS`** and **`CSYNC`**).

More Architecture Evolution

This architecture got several things right, but at least one thing is missing:

- It didn't resolve the problem of how to combine the RRsets that need to be changed as a result of multi-provider DNS with the rest of the zone data.

The problem is that for multi-signer to work without involving the zone owner in the operations, it is necessary for the DNS providers (in particular DNS providers that provide signing services) to be able to make certain changes to the zone.

- Eg., the DNS providers must be able to modify the apex RRsets **`DNSKEY`** and **`NS`** (and possibly also **`CDS`** and **`CSYNC`**).

- But where should these changes be made?

Allowing The Signer To Be "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. (Yes, it's a bad name, suggestions welcome.)

- 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 "**How**" Requirements

In addition to the requirements on **what** is needed, we also need to fulfill some requirements on **how** to do it:

- 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 announced (without polling). In particular changes to the **`DNSKEY`** RRset.

This led to further development of the **"agent"**.

Distributed Multi-Signer "**How**" Requirements

In addition to the requirements on **what** is needed, we also need to fulfill some requirements on **how** to do it:

- 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 announced (without polling). In particular changes to the **`DNSKEY`** RRset.

This led to further development of the **"agent"**.

But there is also the previously known requirement that:

- The DNS providers must be able to locate each other and establish secure communications.

Distributed Multi-Signer "**How**" Requirements

In addition to the requirements on **what** is needed, we also need to fulfill some requirements on **how** to do it:

- 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 announced (without polling). In particular changes to the **`DNSKEY`** RRset.

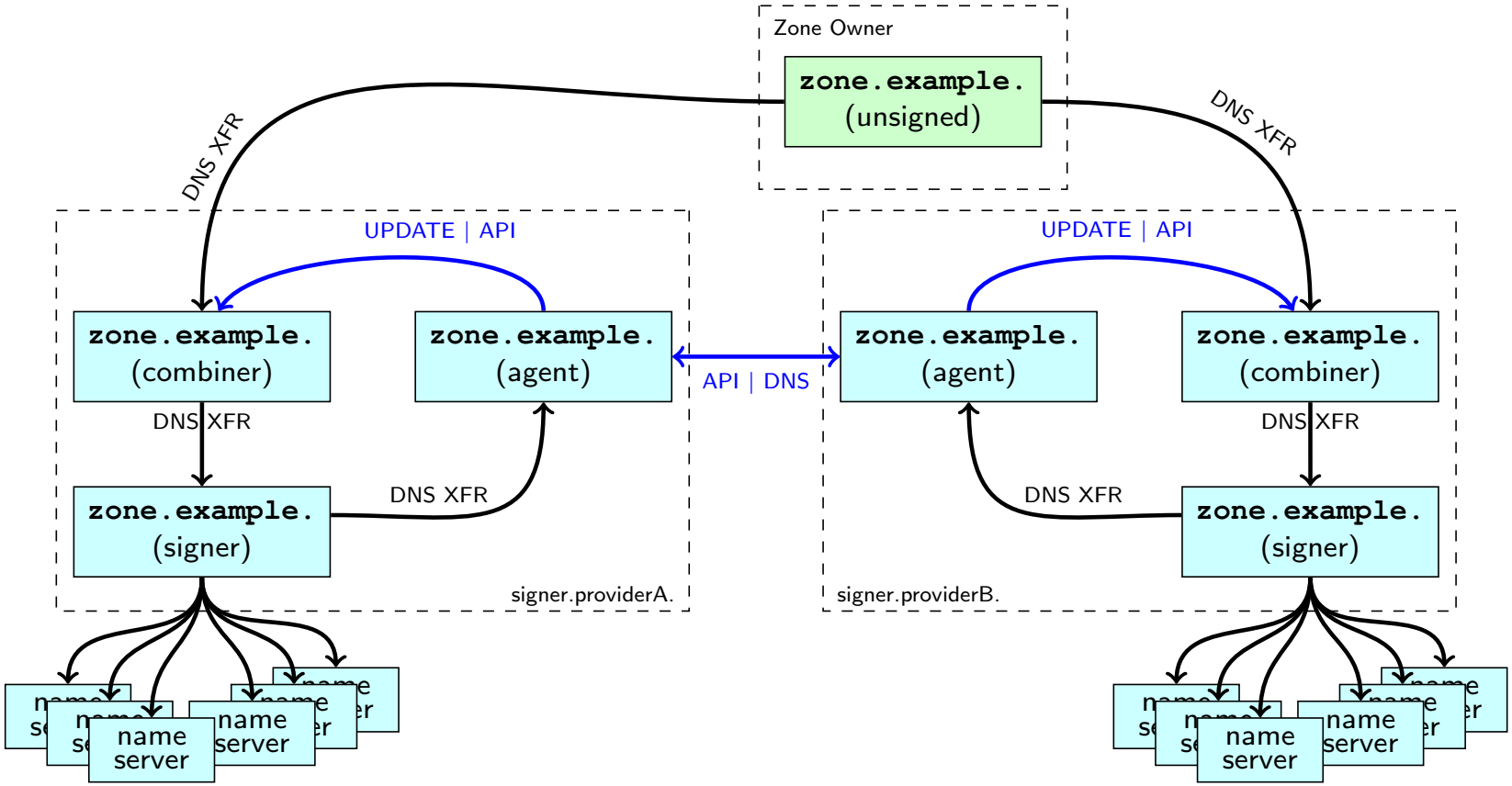
This led to further development of the **"agent"**.

But there is also the previously known requirement that:

- The DNS providers must be able to locate each other and establish secure communications.

The result of this was an updated architecture, where a DNS provider now has three distinct components: a **signer** and an **agent** like before, but now also a **combiner**.

Distributed Multi-Signer Architecture, Evolved



Ok, So Where Do We Go From Here?

We have an architecture that satisfies the **What** and **How** requirements.

- What is missing is a mechanism for specifying who the DNS providers for a zone are.
- The responsibilities for each DNS provider must also be specified. Eg. should ProviderB sign the zone or not? Should ProviderA provide the zone to ProviderB or not?

All such specification is the **responsibility of the zone owner**. But it is important to ensure that the specifications to each provider is consistent with each other.

The current proposal is to use a new DNS record, the **`HSYNC`** record, for this purpose.

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}
```

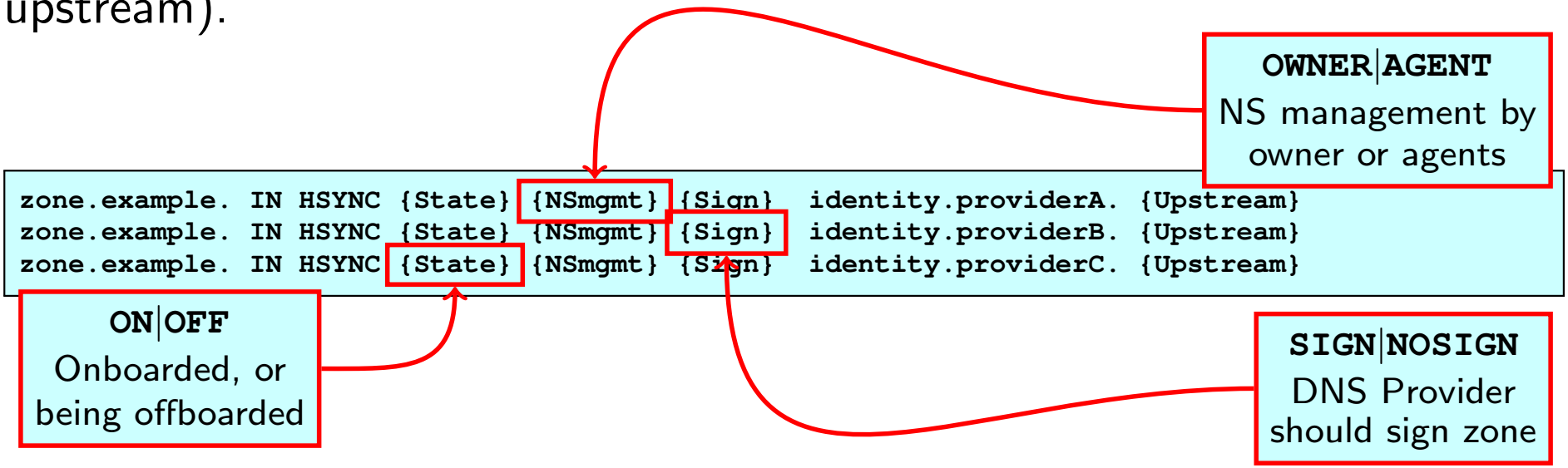
Example:

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

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).



Example:

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

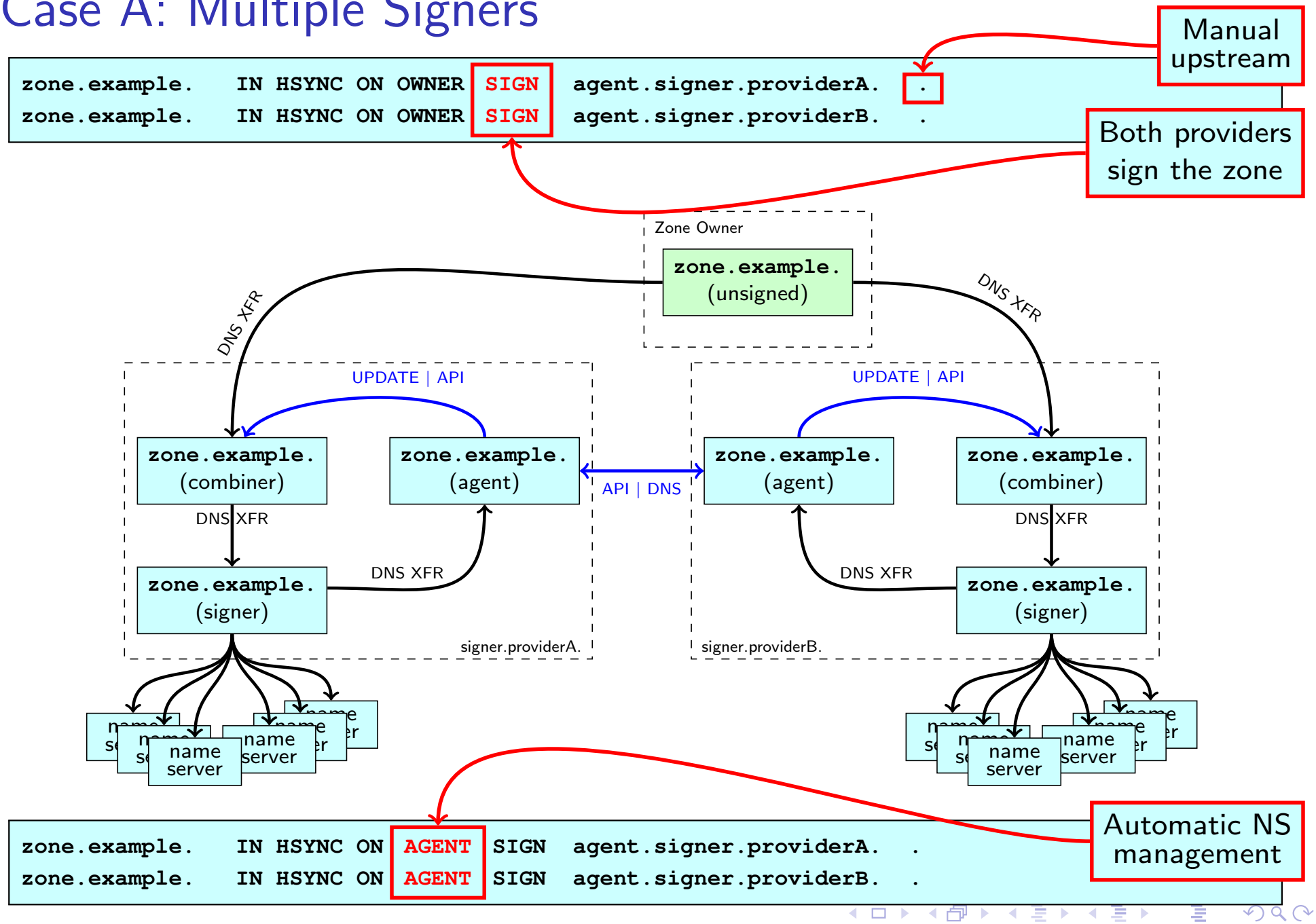
Let's Look at **HSYNC** Setups for Some Scenarios

Here are some examples of scenarios (i.e. **zone owner intent**), expressed 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").

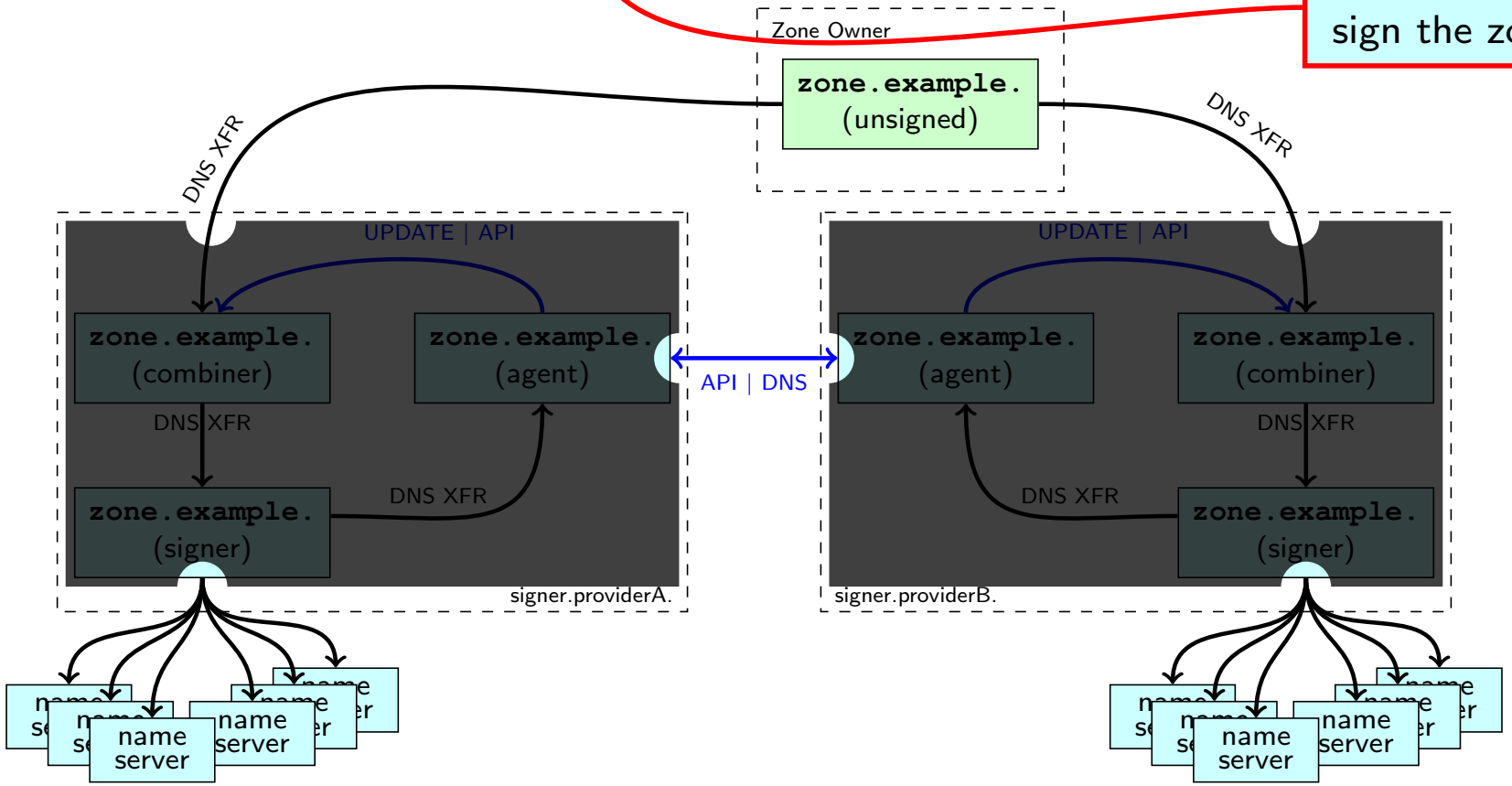
Case A: Multiple Signers



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



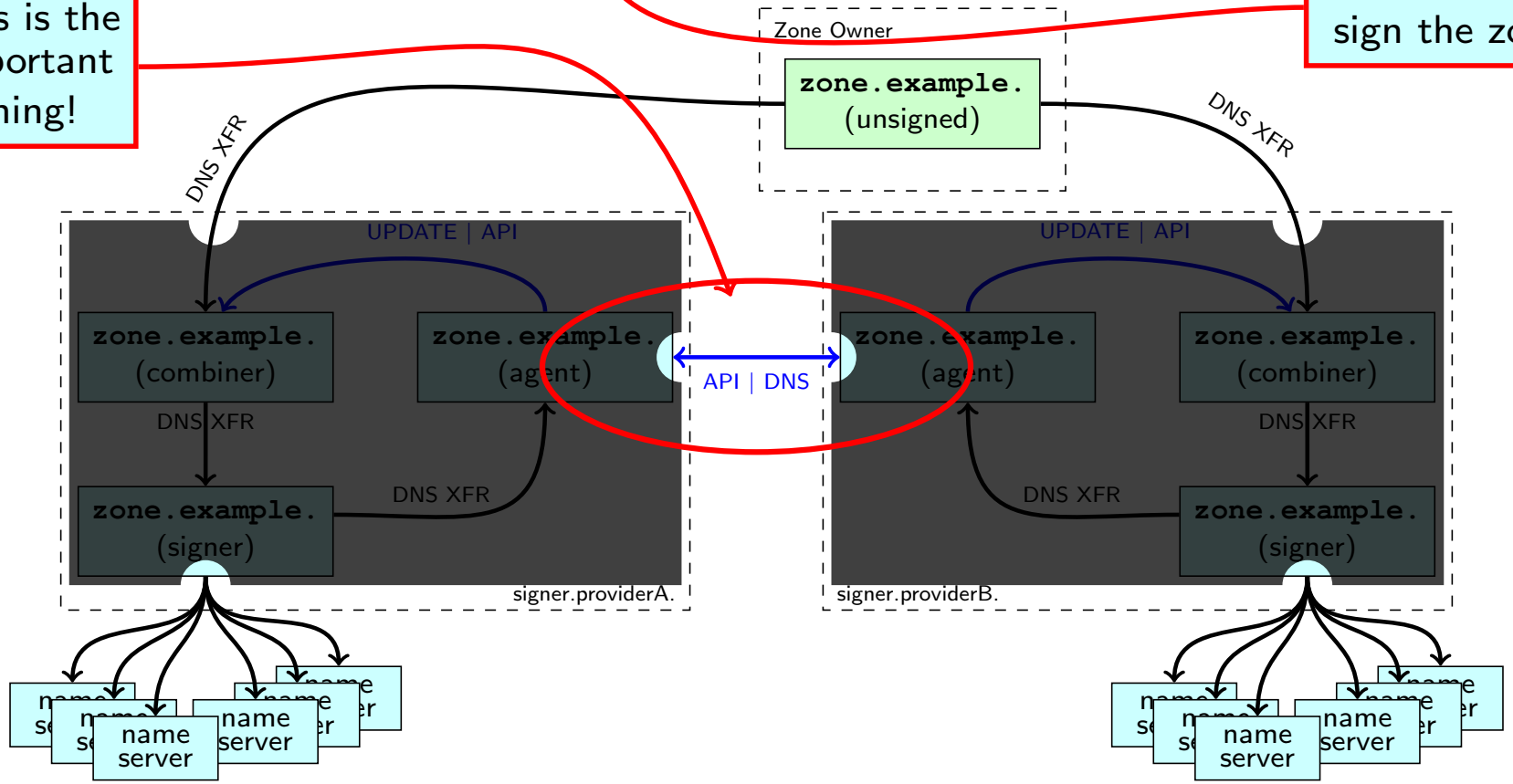

```
zone.example.   IN HSYNC ON  AGENT  SIGN   agent.signer.providerA.   .
zone.example.   IN HSYNC ON  AGENT  SIGN   agent.signer.providerB.   .
```

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.   .
```

This is the important thing!

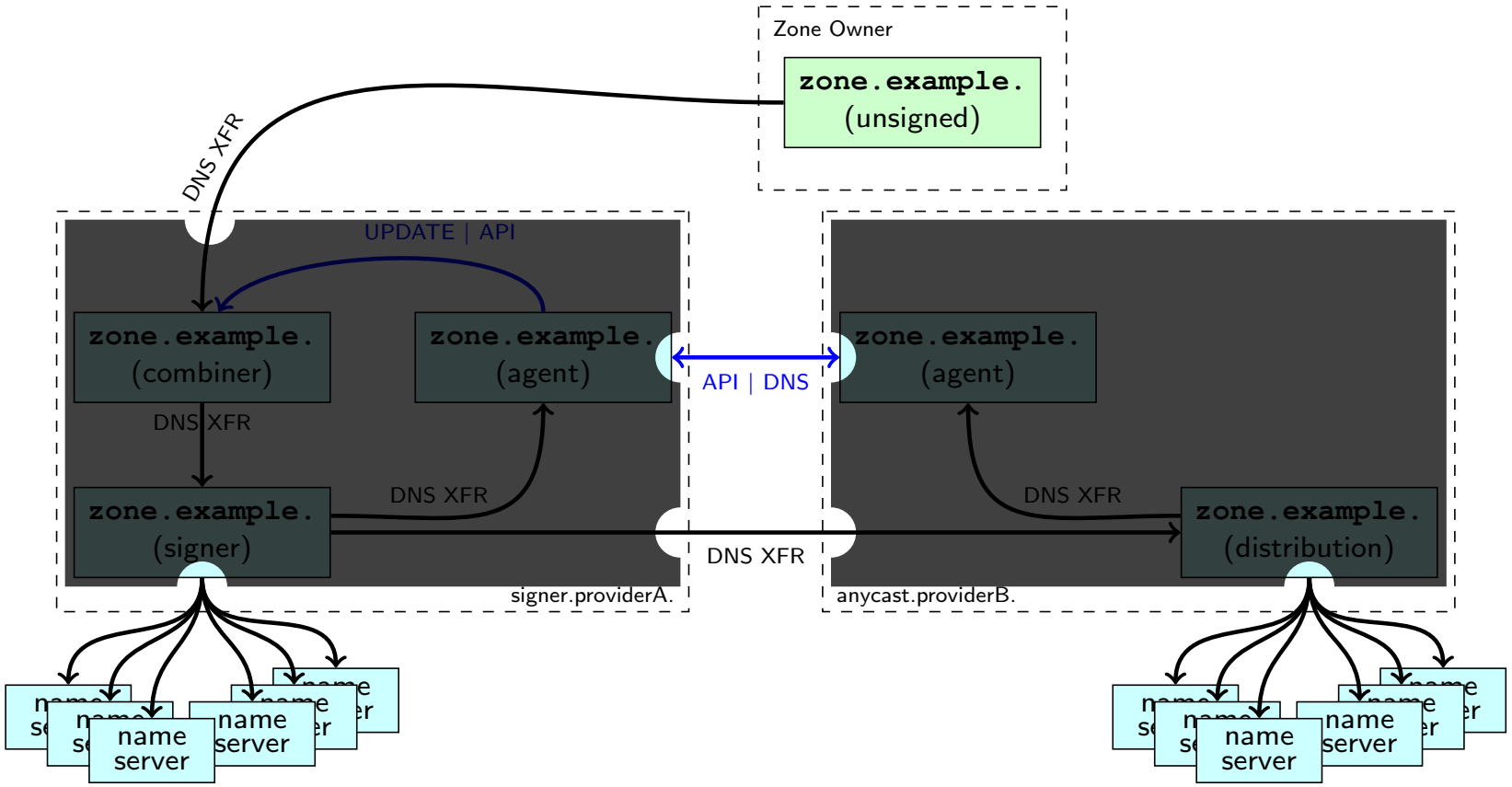
Both providers sign the zone




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

Case B: Single Signer, Multiple DNS Providers

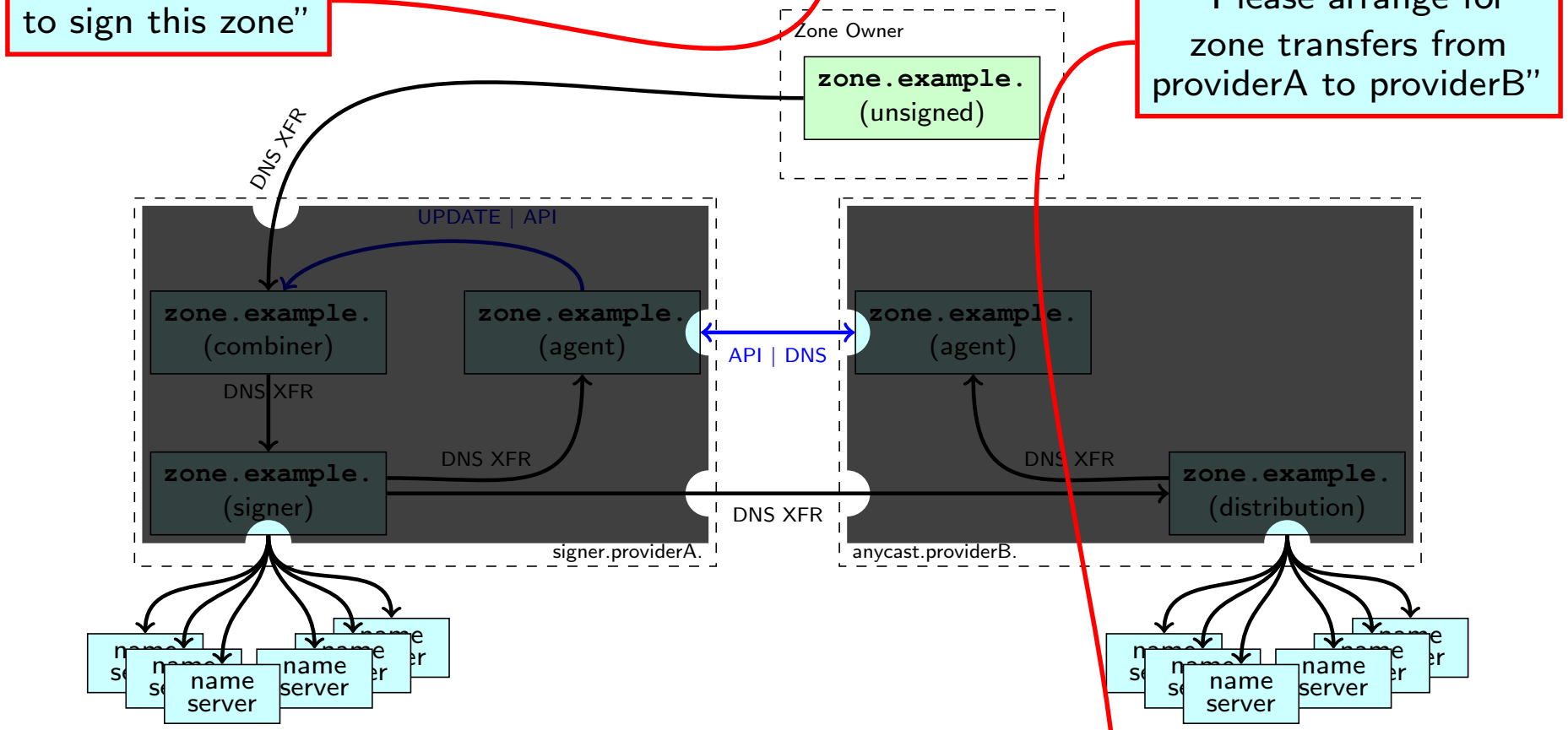
```
zone.example.   IN HSYNC ON OWNER   SIGN     agent.signer.providerA.   .
zone.example.   IN HSYNC ON OWNER   NOSIGN   agent.anycast.providerB.  agent.signer.providerA.
```




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

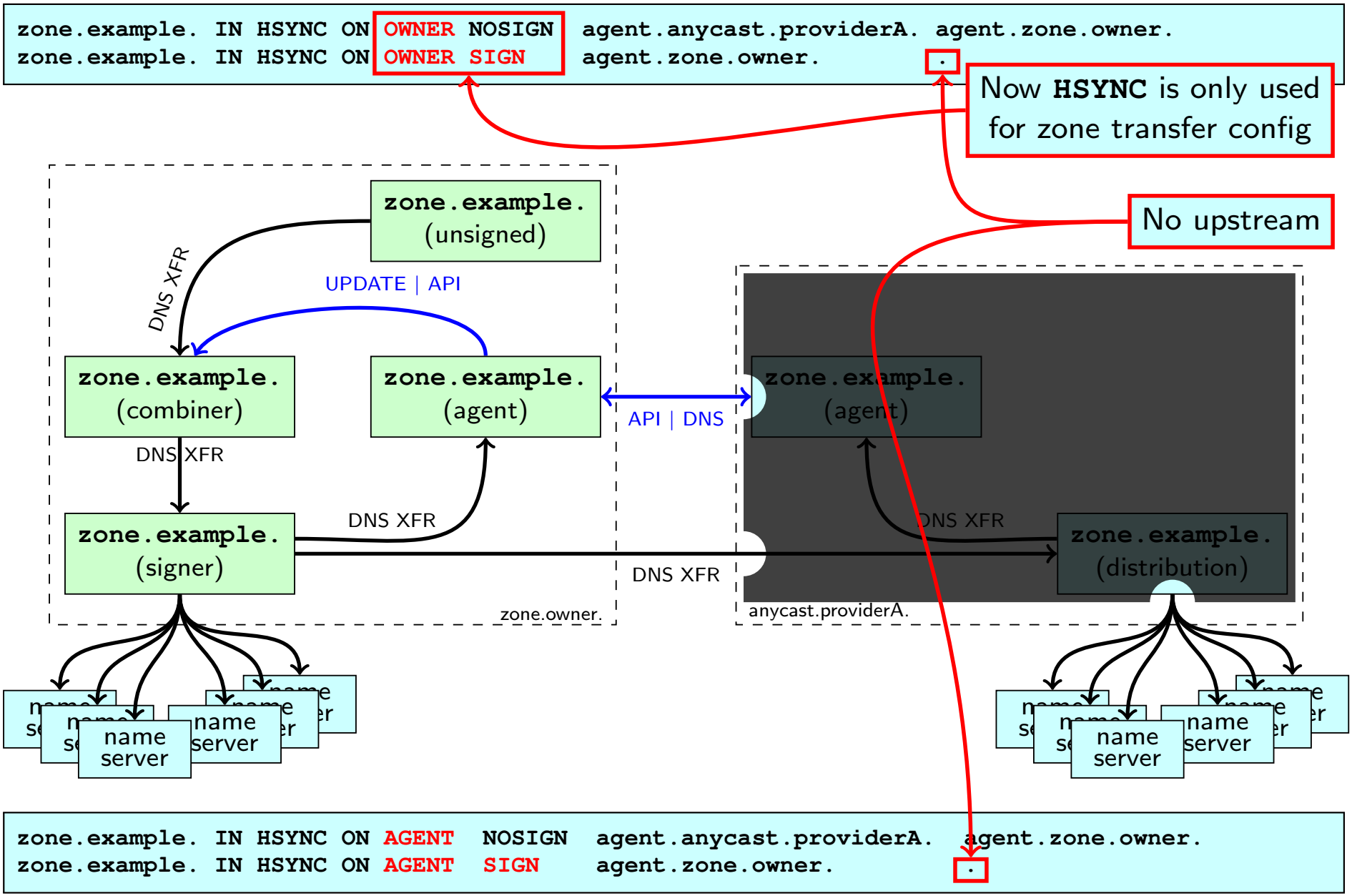
Case B: Single Signer, Multiple DNS Providers

```
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.
```

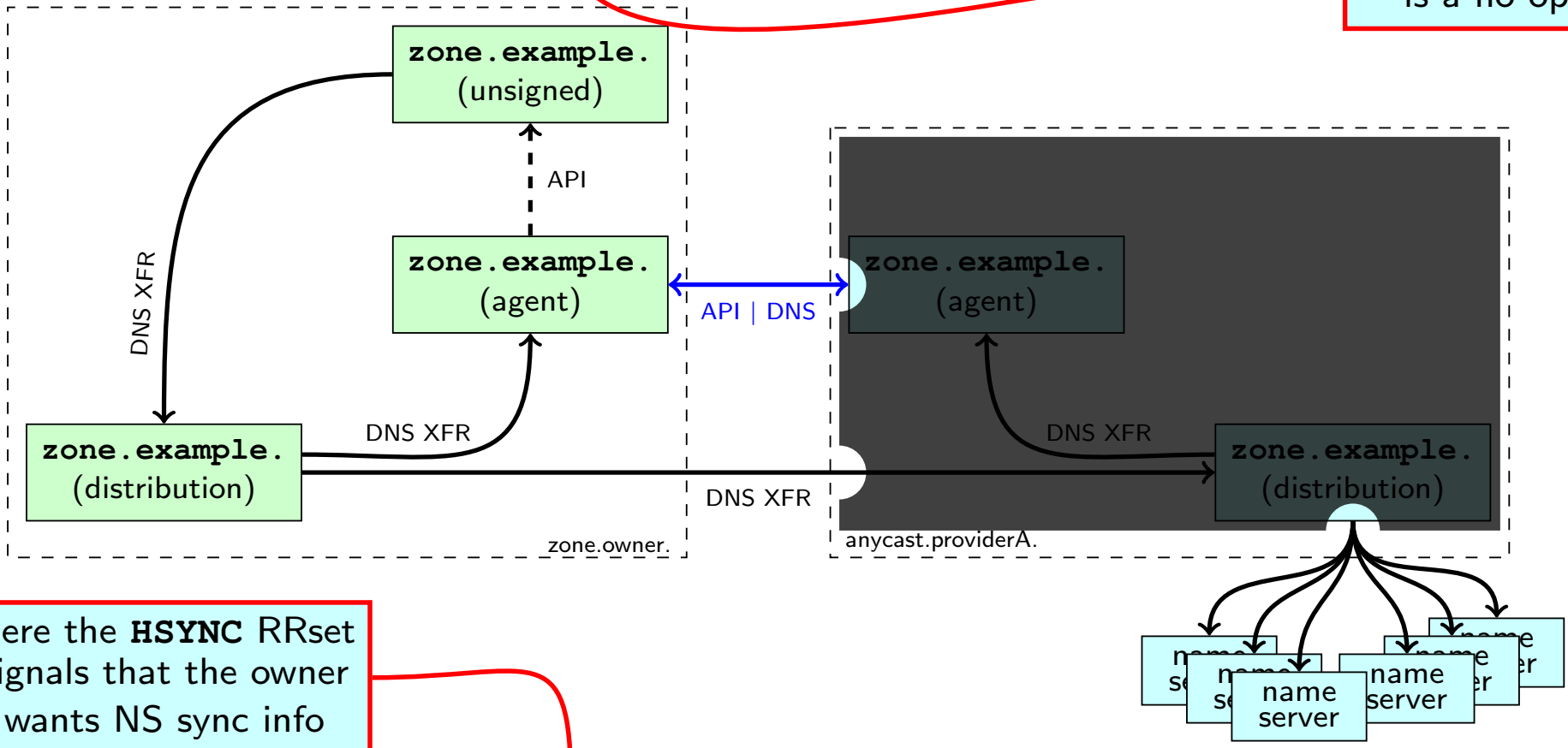
Case C: Zone Signed by Owner, Multiple DNS Providers



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 a possibly 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**.

Protecting the **agents**

The **HSYNC** RRset does expose the identity, and thereby indirectly the location of the **agents**. That does make them vulnerable to potential attacks.

- On the other hand, should an agent be unable to communicate for some time (be it due to an attack or some other reason), the various synchronization processes will simply pause and continue when the agent is back online.
- Hence, the risk is limited to potential delays, not integrity of data or service.

On the other hand, if this exposure is considered to be a problem, then the solution is quite simple:

- We can extend the protocol to include a **HANDOVER** transaction which restarts the initial handshake, but with an agent not exposed via the **HSYNC** RRset.

Summary

- We propose a unified architecture for all forms of DNS multi-provider configurations, both with and without multiple signers.
- We recommend that the specification of the desired configuration is 100% explicit, with no assumptions.
  - The specification is entirely the responsibility of the zone owner
  - Hence, the specification must be provided by the zone owner.
- We propose a new DNS record type, "**HSYNC**" (for "**H**orizontal **SYNC**hronization") to be used for this specification.
  - **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?

Implementation Status and Next Steps

We have a working implementation of most of the architecture (including full support for 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 a **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 the **agent-to-agent communication** will be the topic of an upcoming internet-draft.

Your Questions Here :-)

Thanks & Contact Information

johan.stenstam@internetstiftelsen.se

steve@shinkuro.com

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