Towards Distributed Multi-Signer:

A Road Trip with Detours

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

Swedish Internet Foundation

February 17, 2025

Agenda

This is our plan.

2pp Multi-signer definition and problem statement

5pp Short review of how we got here; where we started, detours, etc.

5pp Introduction to Distributed Multi-Signer

4pp New EDNS(0) OPT opcodes to improve signalling between entities. The KeyState and Multi-Signer opcodes.

7pp The proposed **`HSYNC`** RRset.

2pp Security models.

3pp Open questions. Stuff we're working on.

2pp If time permits: a few words on the Zone-Pipeline draft.

- Wrapup and general discussion

Agenda

This is our plan. But if you don't like it, we have others :-)

2pp Multi-signer definition and problem statement

5pp Short review of how we got here; where we started, detours, etc.

5pp Introduction to Distributed Multi-Signer

4pp New EDNS(0) OPT opcodes to improve signalling between entities. The KeyState and Multi-Signer opcodes.

7pp The proposed **`HSYNC`** RRset.

2pp Security models.

3pp Open questions. Stuff we're working on.

2pp If time permits: a few words on the Zone-Pipeline draft.

- Wrapup and general discussion

Definition of "Multi-Signer"

For the purpose of this work we use the following definition:

> *Multi-signer is an architecture where it is technically easy to add and remove signers from a zone in order to remove the "single point of failure" caused by having only a single signer.*
> *Furthermore, at least some of the signers must be managed by different organizations.*

Problem Statement

This has some implications. In particular the "technically easy" part.

- Once there are multiple signers, they **must be able to collaborate in the shared management** of the zone.
- This collaboration must be sufficiently robust to **deal with on- and offboarding of signers** (by the zone owner).
- The collaboration must also include **automated management of shared data** (like **`DNSKEY`**s and **`NS`** records.)
- Finally the collaboration must be able to **manage the delegation information in the parent** (if wanted by the zone owner).

All of these are doable, especially in a single organization. However, the last point ("different organizations") is the most problematic.

- The complexity of the multi-signer "algorithms" **forces automation across multiple organizations**.

Short Review of the Last Three Years Work

"Multi-signer" has been talked about for a long time and there are several open source or proprietary implementations. There are commercial requests for such functionality. However, multi-signer has not really "taken off". There are likely multiple reasons for this. Among those are:

- It is difficult (from a business risk POV) for a "signer" to accept a design where an external third-party (the multi-signer controller) has write access to the customer zone.

This has forced a rethink and a redesign and there is now at least an "architecture" for a distributed version of a multi-signer system.

- Implementation of the distributed version of multi-signer is still work in progress.

The **DSYNC** Detour

Apart from the business risk perspective, the following quickly became clear:

- "Multi-signer" really, really, needs a mechanism to signal the introduction of new DNSKEYs and the removal of old ones.
  - DNSSEC-signers (like BIND9) will add a new ZSK without telling anyone (as they consider ZSK rollovers to be a local issue).
- We wanted something like a **NOTIFY(DNSKEY)**.

This led to the generalized-notify draft (together with Peter Thomassen and John Levine).

- . . . although in that draft we choose to focus on **NOTIFY(CDS)** and **NOTIFY(CSYNC)**, as they had more immediate applicability
- Eventually **this became the DSYNC record**.

The Delegation Mgmt via DNS UPDATE Detour

Then we realized (during a lunch with Mark Andrews during an IETF meeting) that the **DSYNC** record opened up the door for a rather clean solution to the problem of:

How to do child-to-parent delegation updates via DDNS?

- This is a well-known rathole with at least three previous attempts. . . (that all failed)

On review of the previous attempts it still seemed quite clear that they failed exactly because of issues that the **DSYNC** record is designed to solve.

- To open up for a new attempt at solving this problem the **DSYNC**-based approach has been **documented in the delegation-mgmt-via-ddns** draft.
- It has also been fully implemented.

The KeyState EDNS(0) OPT Code Detour

Delegation Management via DNS UPDATE and SIG(0) took some effort, but it worked out quite nicely.

- However, feedback from others and own experience showed that could become a bit complicated over time.
- This prompted the next question to be (in the context of only DNS UPDATEs):

> How to manage "transactions" and maintain "state" over time between different parties?

- I.e. between the child and the parent in this case.

Eventually it was solved by the introduction of the KeyState EDNS(0) OPT opcode, modelled after Extended DNS Errors, and **documented via the KeyState draft**.

- **KeyState** has also been implemented.

Finally Back To Multi-Signer (three years later. . .)

It was obvious that in the absence of the centralized controller we needed another mechanism to "designate" who should sign a zone, and the only reasonable answer was the **zone owner**.
With that decided the next question was:

How should the different signers be introduced to each other?

- Our current proposal is to use the **`HSYNC`** RRset (for identification)
- . . . plus various DNS lookups for **`URI`**, **`SVCB`**, **`TLSA`** and **`KEY`** records to establish secure communications.

Then we ran into problems similar to what we had earlier encountered with the DNS UPDATEs between organisations and signalling state over time.
But now we knew the path forward for such problems.

- Enter the Multi-Signer EDNS(0) OPT opcode (specified, but not yet implemented).

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 the development of the **"Multi-Signer Agent"**, or **MSA**.

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.

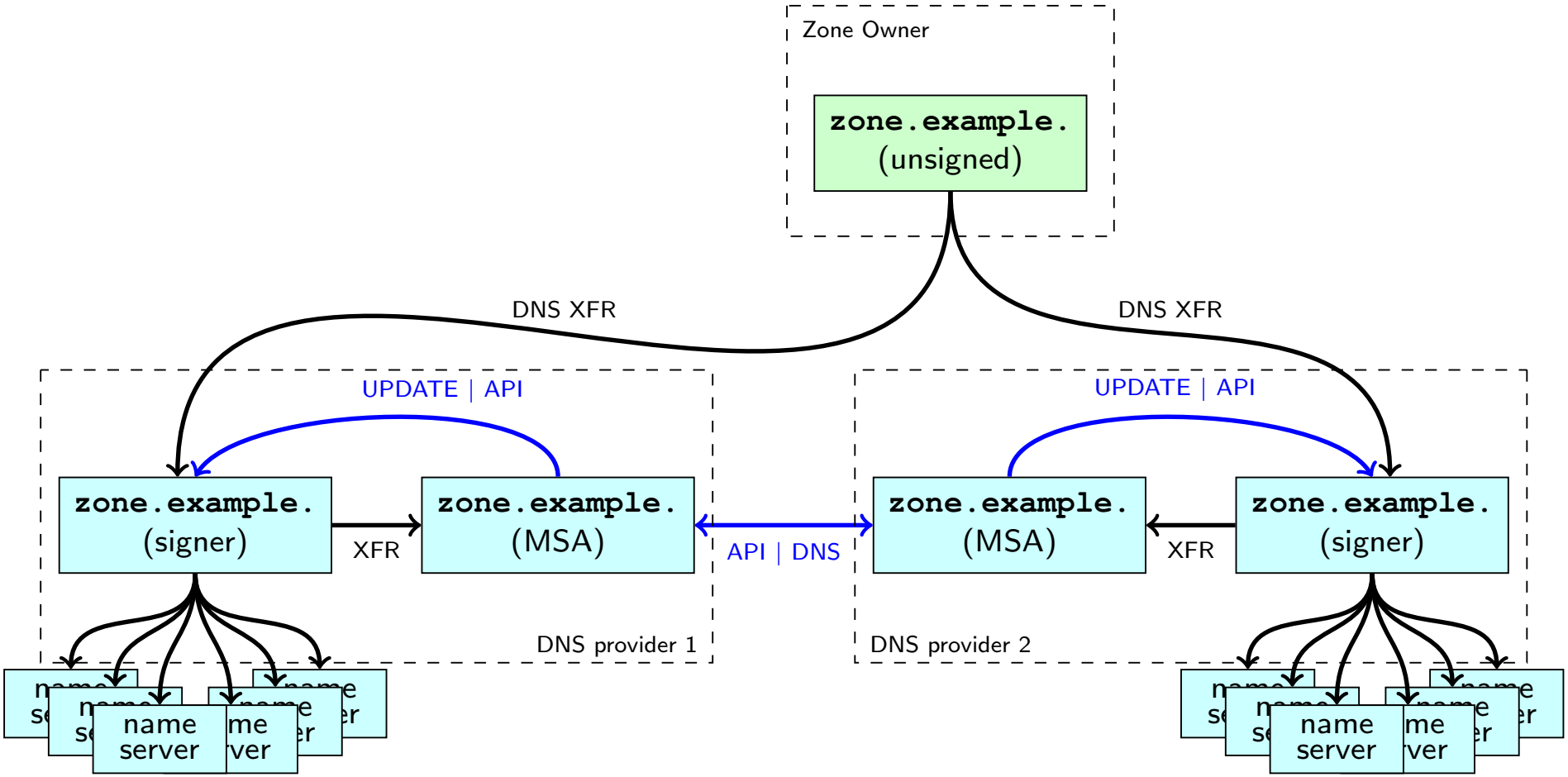
This led to the development of the **"Multi-Signer Agent"**, or **MSA**.

But there are also the 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.

That led to the current proposal for the **`HSYNC`** record.

Initial Distributed Multi-Signer Architecture



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

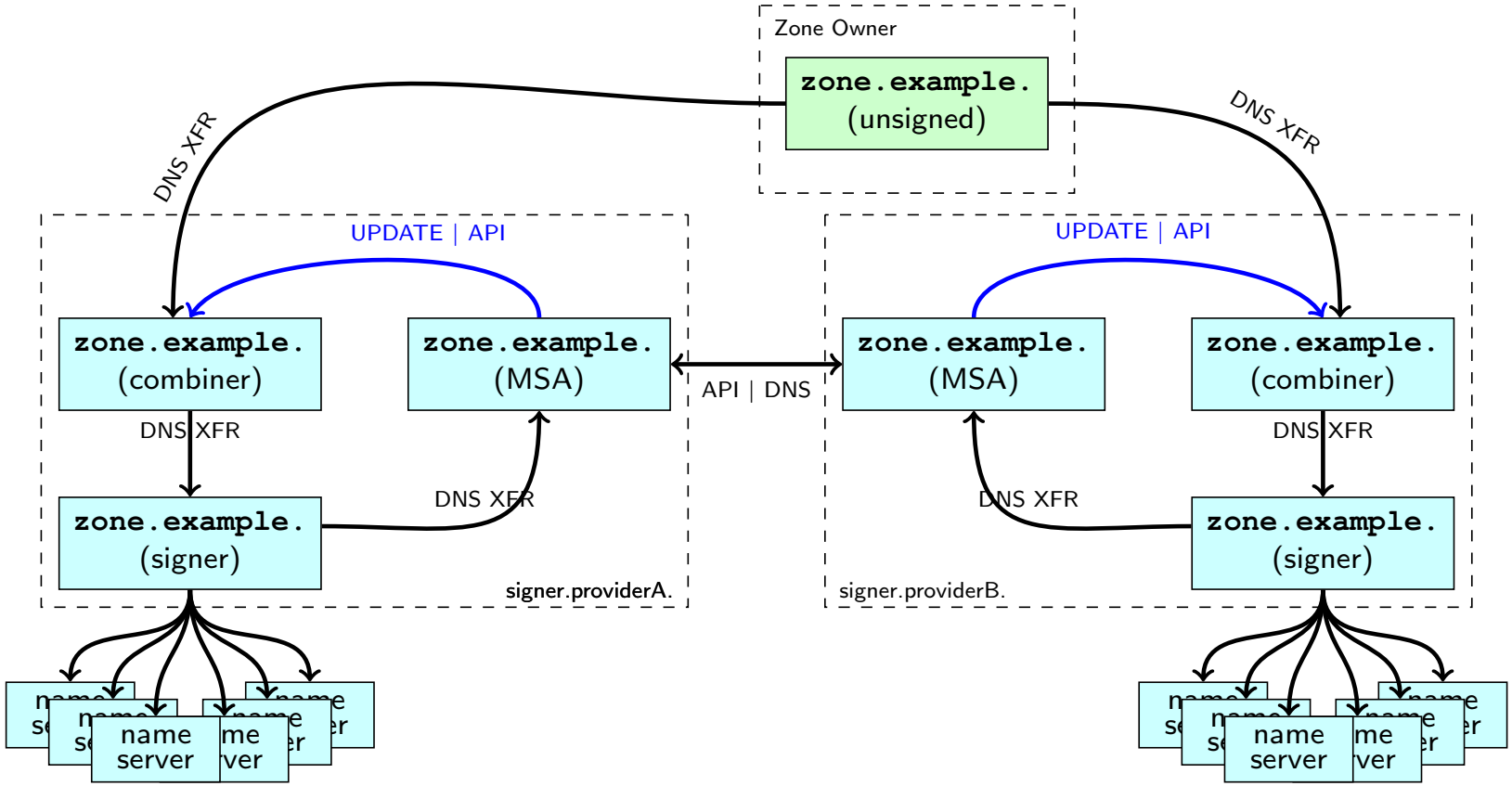
There were some problems with the initial architecture.

- The most serious one was 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.
- It is able to modify the crucial apex RRsets (**`DNSKEY`**, **`CDS`**, **`CSYNC`** and **`NS`**). Only those four RRsets, and only on request from the MSA.

Distributed Multi-Signer Architecture, Evolved



Source of Truth

The COMBINER changes the source of truth for the zone.

In a multi-signer architecture it seems to be necessary to have **three distinct "sources of truth"**:

- the zone owner for "normal zone conent" (obviously)
- any signers for "DNSSEC data" (quite obviously)
- a third entity (the MSA) that manages data synchronized via multi-signer processes in collaboration with other MSAs. This data is added to the zone via the "COMBINER".

Before the KeyState EDNS(0) Option

The original proposal in **`draft-johani-dnsop-delegation-mgmt-via-ddns`** was based on the parent acquiring and eventually trusting the child SIG(0) key.

Identified problems:

- Risk of the child operator and the parent receiver getting out of sync.
- No way to signal parent policies to the child (or vice-versa).
- No way for the child to request information from the parent other than as the result of an UPDATE transaction.

After the KeyState EDNS(0) Option

Why a new EDNS(0) Option?

Because new functionality was needed:

- EDNS(0) allows extended functionality.
- Allows signaling information both ways.
- Allows signaling parent policies. E.g. "Automatic bootstrapping allowed" or "Manual bootstrapping required"
- Allows child to request information, from parent, about current key state. E.g., if SIG(0) key is known and trusted.

The KeyState EDNS(0) Option modelled on RFC 8914, "Extended DNS Errors", but is extended to be bi-directional and allow other types of signalling than just error reporting.

Multi-Signer EDNS(0) Option

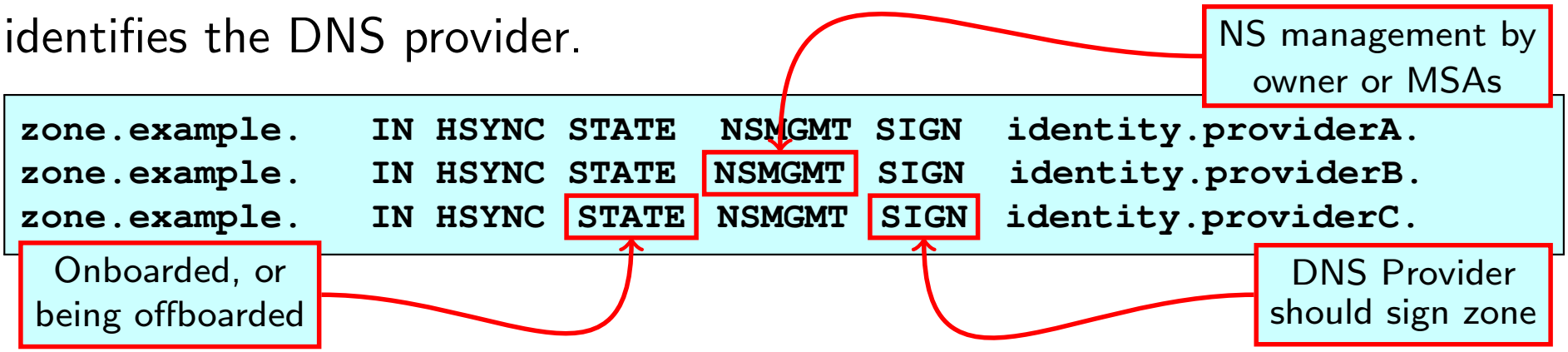
This is also a new EDNS(0) Option. This is based on the functionality requirements in **`draft-leon-distributed-multi-signer`**.

- Used for synchronization and signaling in a similar way as in KeyState EDNS(0) Option.
- Signaling transport mode and synchronization model
- Used to signal **OPERATION**, eg. **`hello`**, **`heartbeat`**, etc, messages

The Proposed **HSYNC** RRset

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

It has four fields: **STATE**, **NSMGMT**, **SIGN** and a domain name that identifies the DNS provider.



Example:

```
zone.example.   IN HSYNC ON    OWNER  YES  identity.providerA.
zone.example.   IN HSYNC ON    OWNER  YES  identity.providerB.
zone.example.   IN HSYNC OFF   OWNER  YES  identity.providerC.
```

Confusing. Please show the possible **HSYNC** setups.

Yes, this is a bit confusing. Let's try to untangle this. These are the setups that we would like to be able to describe via appropriate **HSYNC** RRsets:

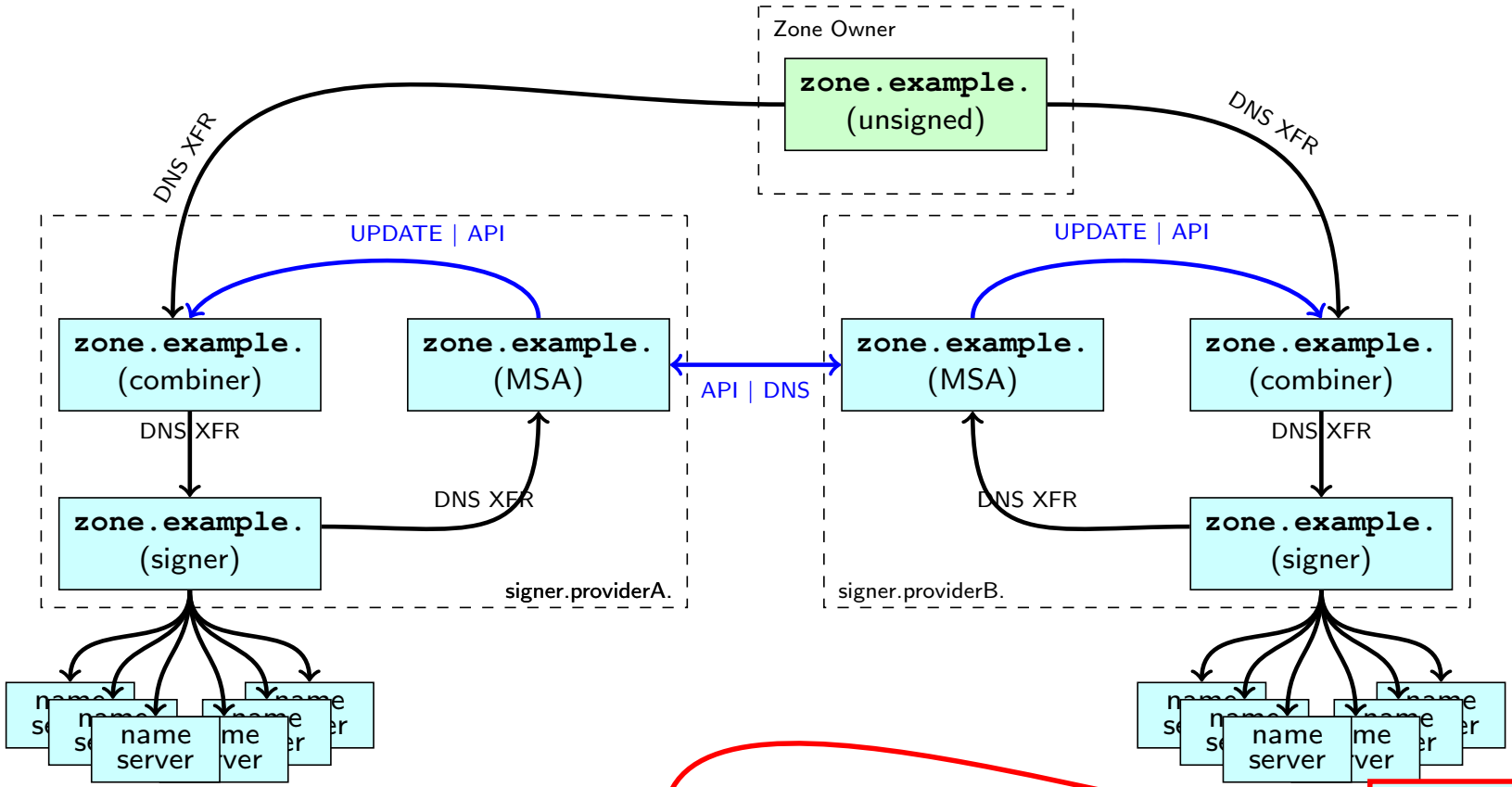
- **Case A: "Normal multi-signer" setup:** multiple DNS providers, all of which sign the customer zone.
- **Case B: "Normal single signer" setup:** multiple DNS providers, only one of which sign the customer zone. All other providers fetch the zone from the provider that signs.
- **Case C: Owner signs:** 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 MSAs ("automated").

Case A: Multiple Signers

```
zone.example.    IN HSYNC ON OWNER YES   msa.signer.providerA.
zone.example.    IN HSYNC ON OWNER YES   msa.signer.providerB.
```

Manual NS management

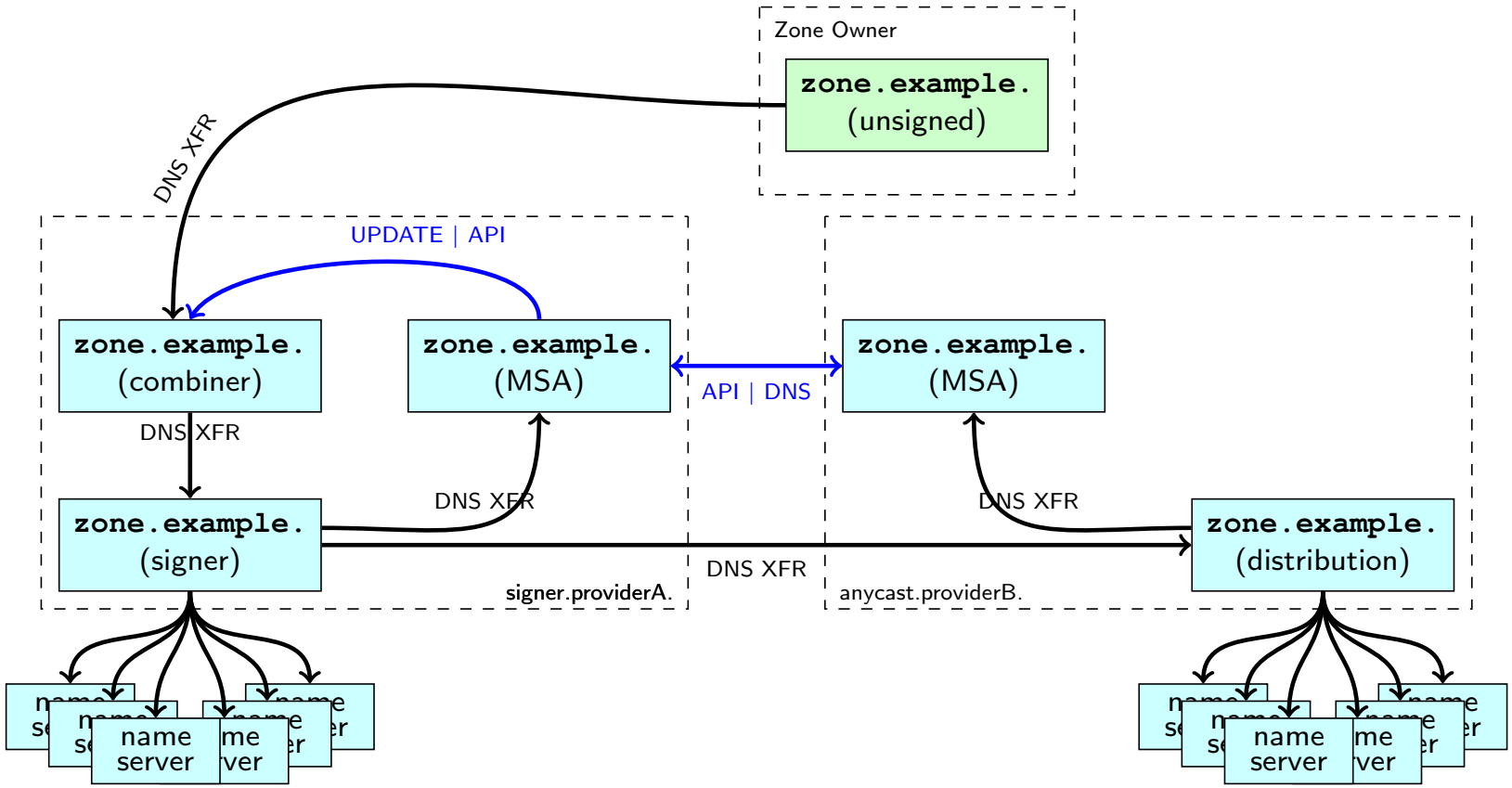



```
zone.example.    IN HSYNC ON AGENT YES   msa.signer.providerA
zone.example.    IN HSYNC ON AGENT YES   msa.signer.providerB.
```

Automatic NS management

Case B: Single Signer, Multiple DNS Providers

```
zone.example.    IN HSYNC ON OWNER   YES   msa.signer.providerA.
zone.example.    IN HSYNC ON OWNER   NO    msa.anycast.providerB.
```

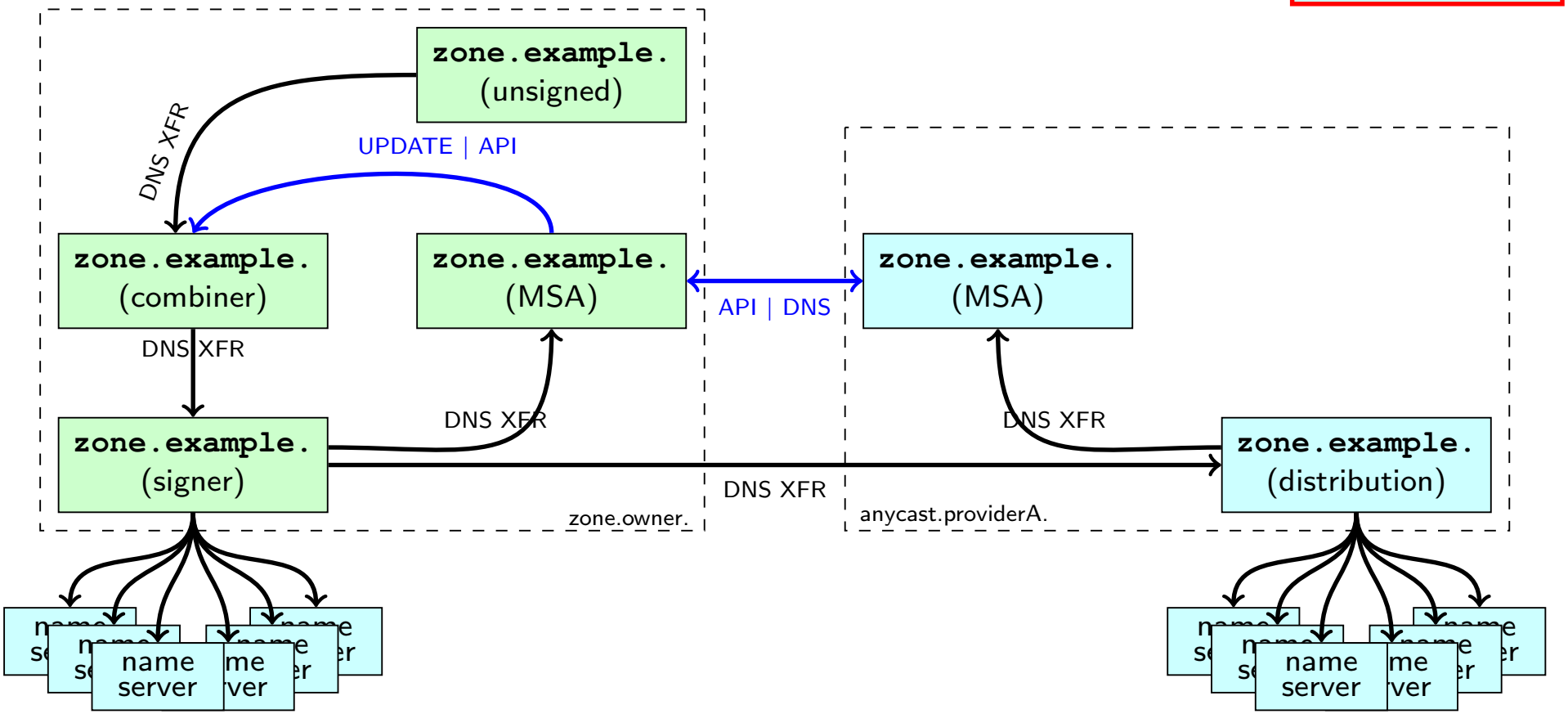



```
zone.example.    IN HSYNC ON AGENT   YES   msa.signer.providerA.
zone.example.    IN HSYNC ON AGENT   NO    msa.anycast.providerB.
```

Case C: Zone Signed by Owner, Multiple DNS Providers

```
zone.example.    IN HSYNC ON OWNER NO  msa.anycast.providerA.
zone.example.    IN HSYNC ON OWNER YES msa.zone.owner.
```

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

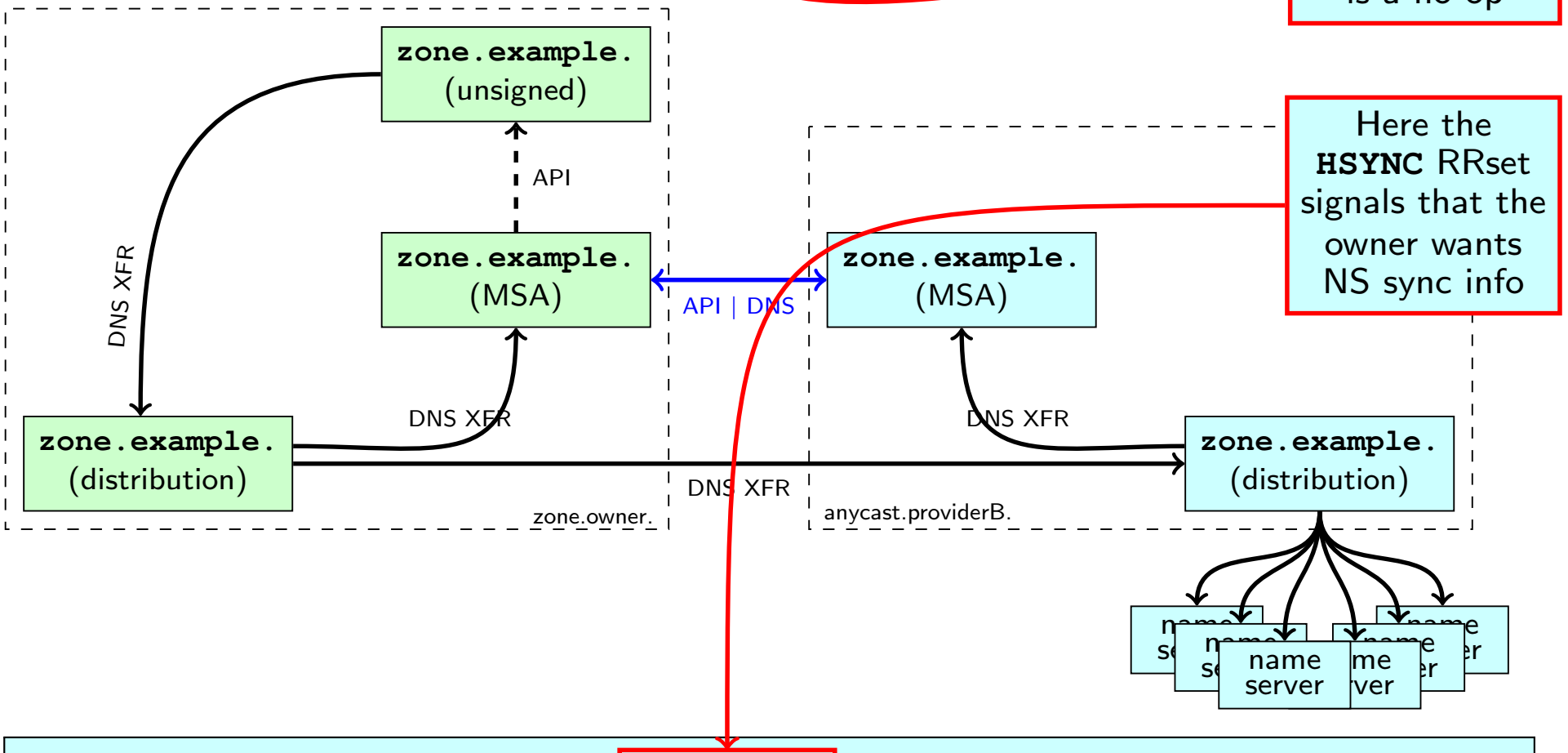



```
zone.example.    IN HSYNC ON AGENT NO  msa.signer.providerA.
zone.example.    IN HSYNC ON AGENT YES msa.anycast.providerB.
```

Case D: Unsigned Zone, Multiple DNS Providers

```
zone.example.    IN HSYNC ON OWNER NO   msa.anycast.providerA.
zone.example.    IN HSYNC ON OWNER NO   msa.zone.owner.
```

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




```
zone.example.    IN HSYNC ON AGENT NO   msa.anycast.providerA.
zone.example.    IN HSYNC ON AGENT NO   msa.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 MSAs 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 VPNs, via locked IP-addresses to XFR-over-TLS.

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

Security Model, cont'd

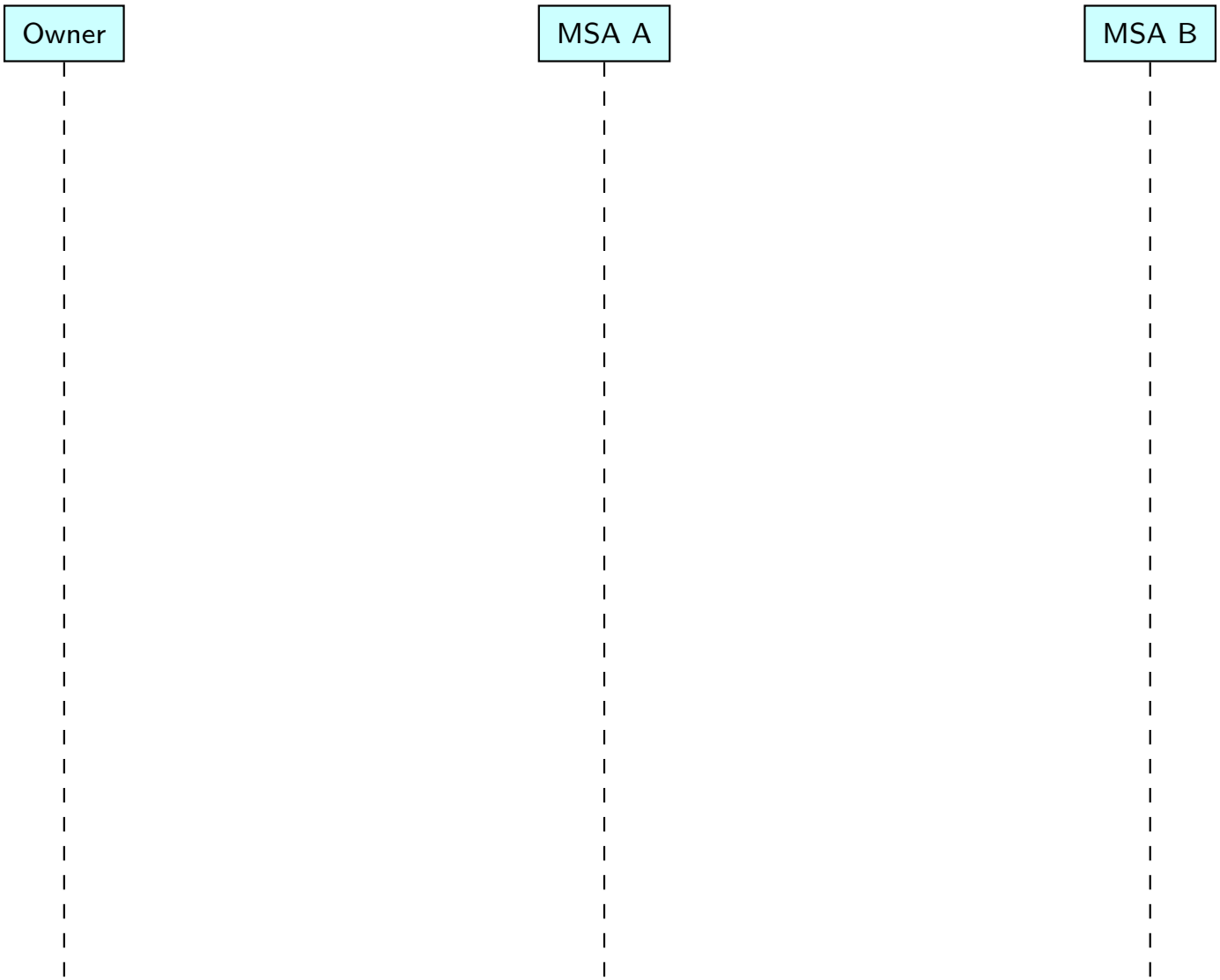
When it comes to the establishment of secure communication between MSAs, there is a fundamental requirement on DNSSEC for MSAs, both when publishing data and when looking up data.

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

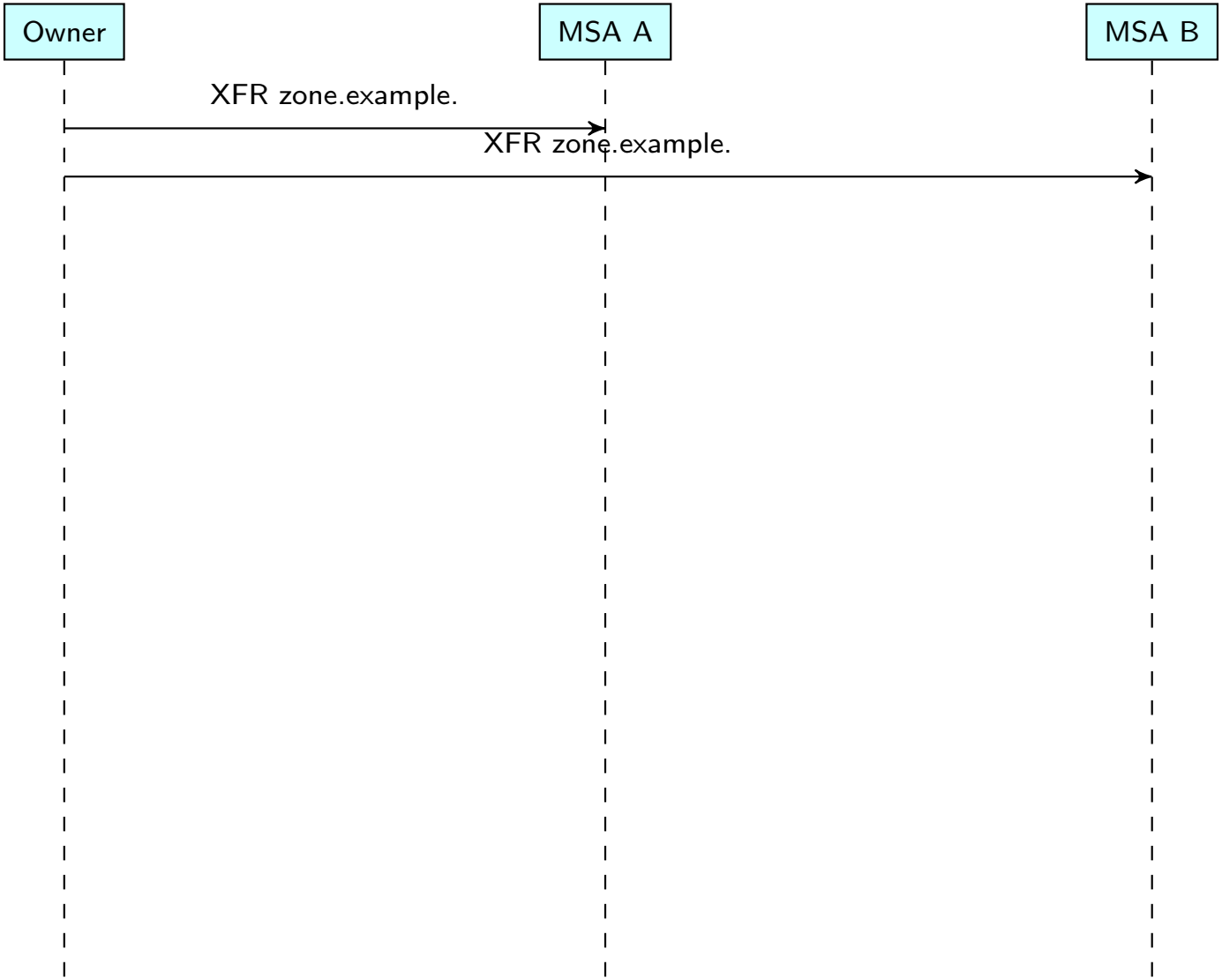
- **DNS transport:** look up **`_dns._tcp.{identity} URI`**. Then look up the corresponding **SVCB** and finally the **KEY** record.
- **API transport:** look up **`_https._tcp.{identity} URI`**. Then look up the corresponding **SVCB** and finally the **TLSA** record.

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

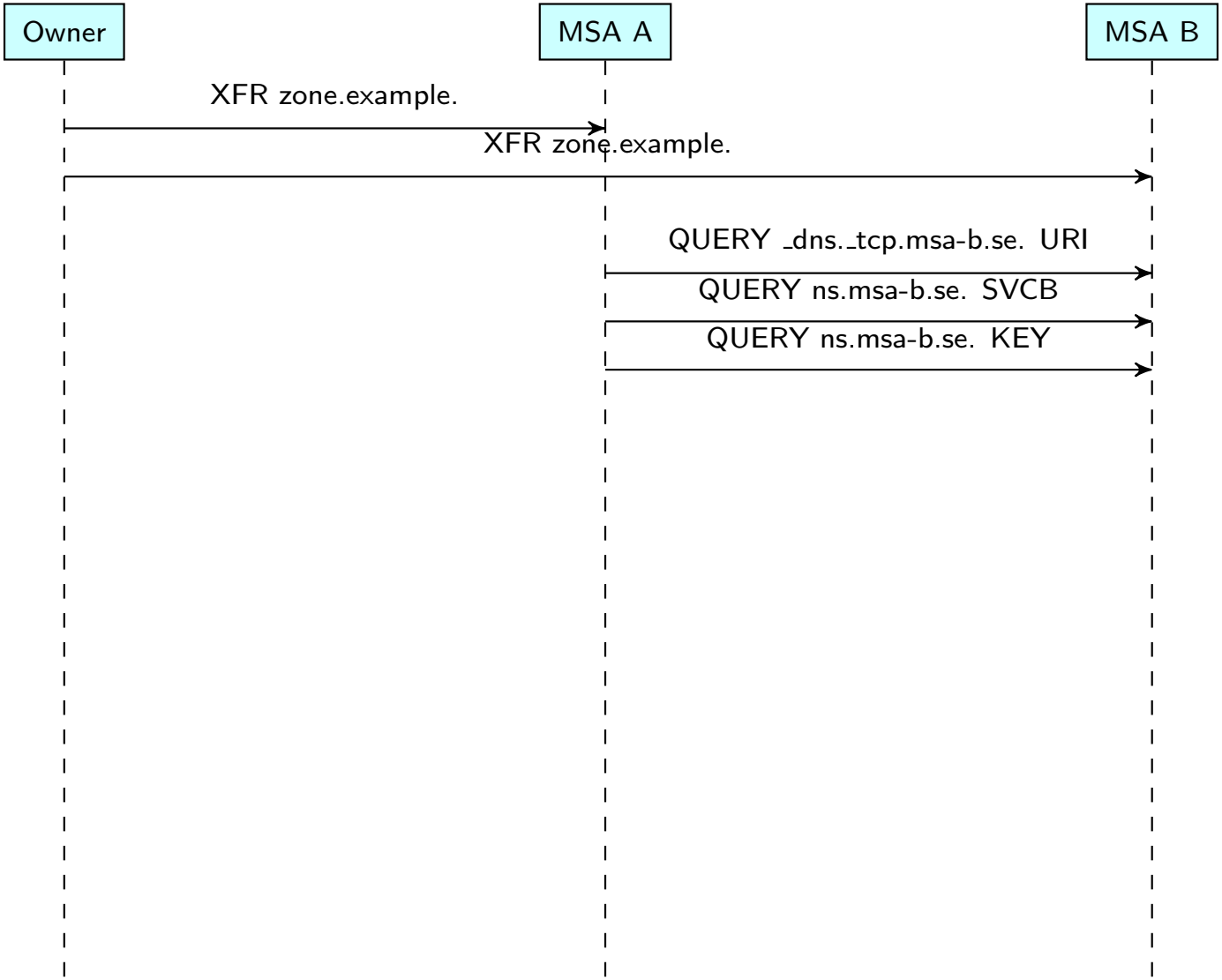
Establishing MSA Comms, Basic



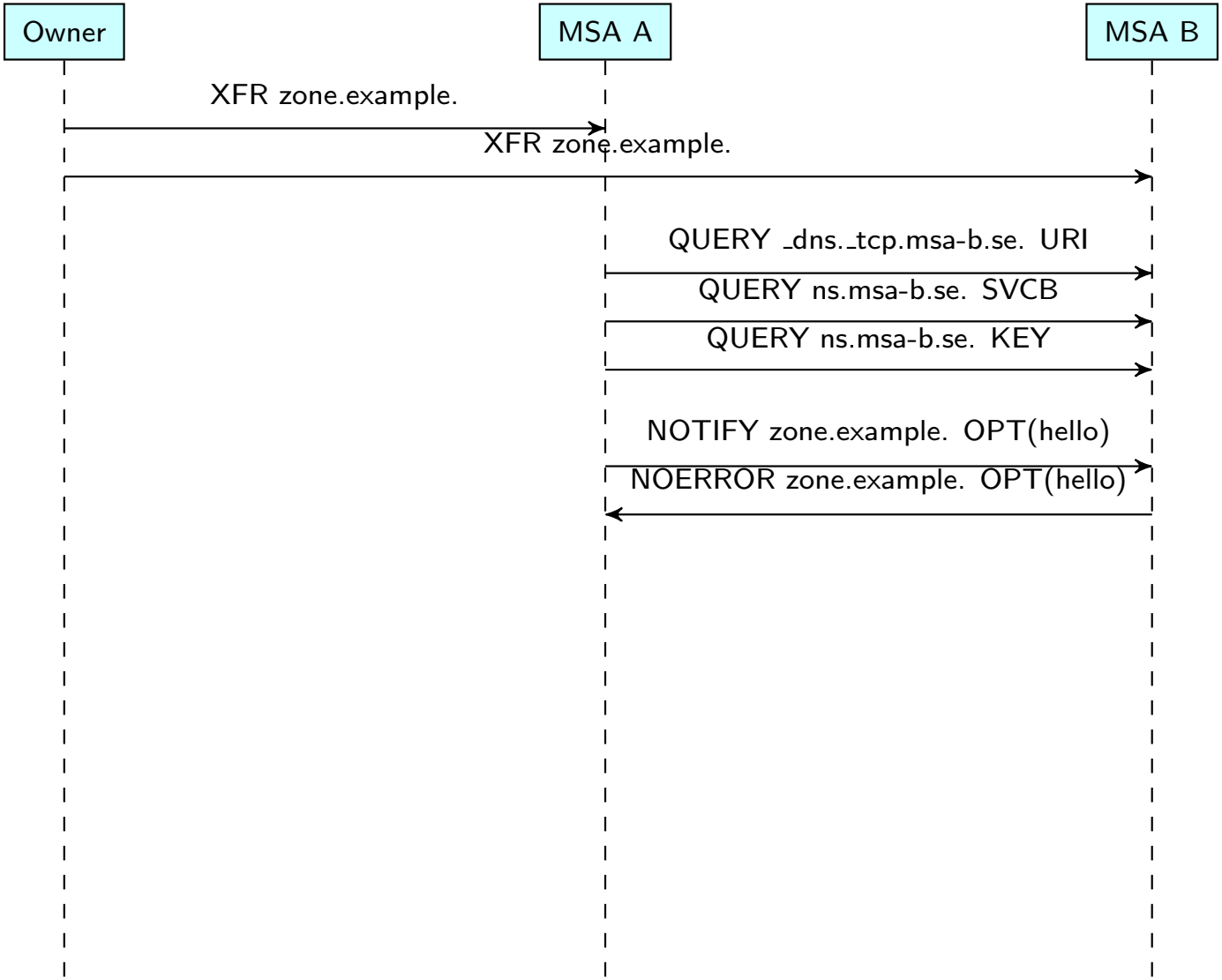
Establishing MSA Comms, Basic



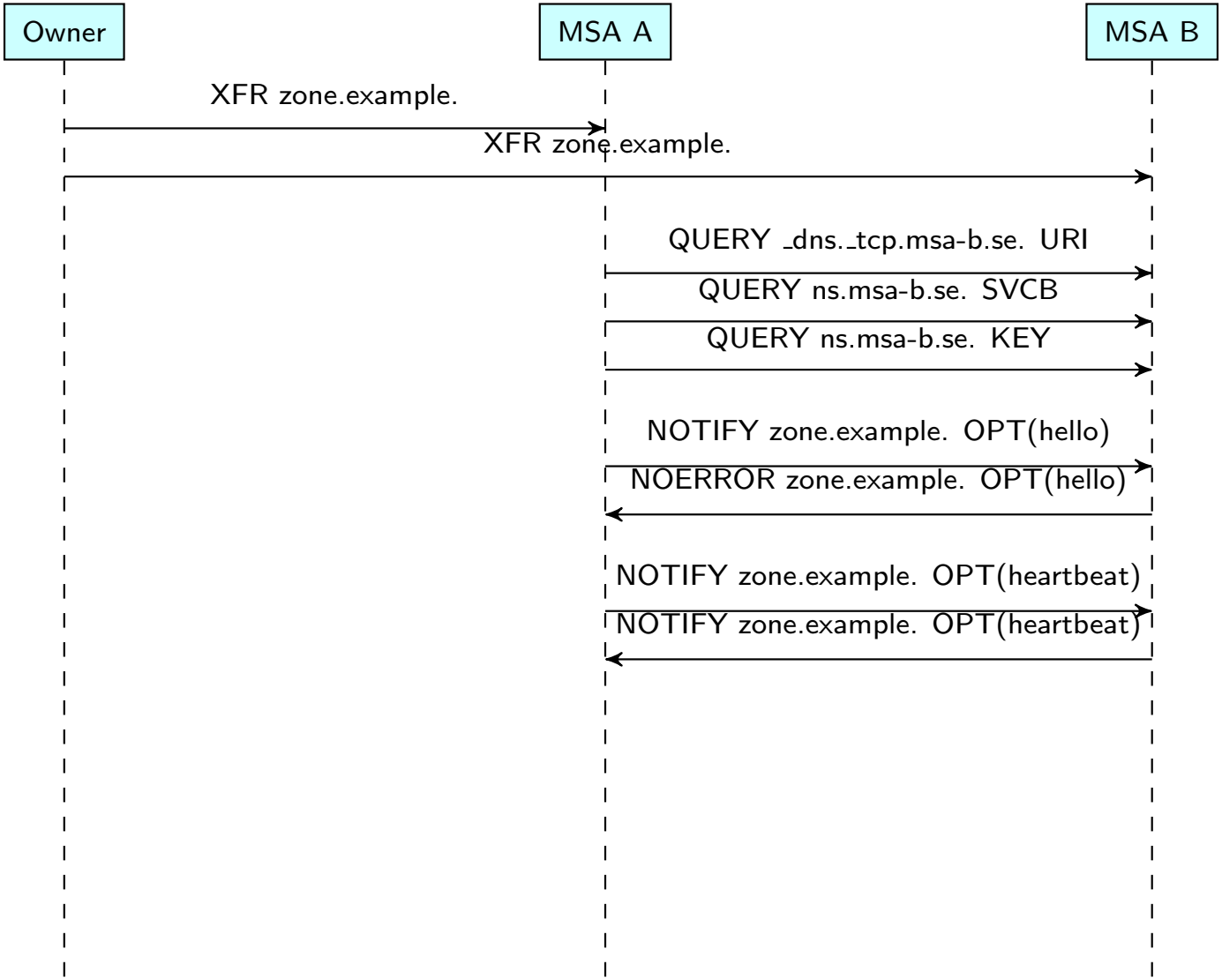
Establishing MSA Comms, Basic



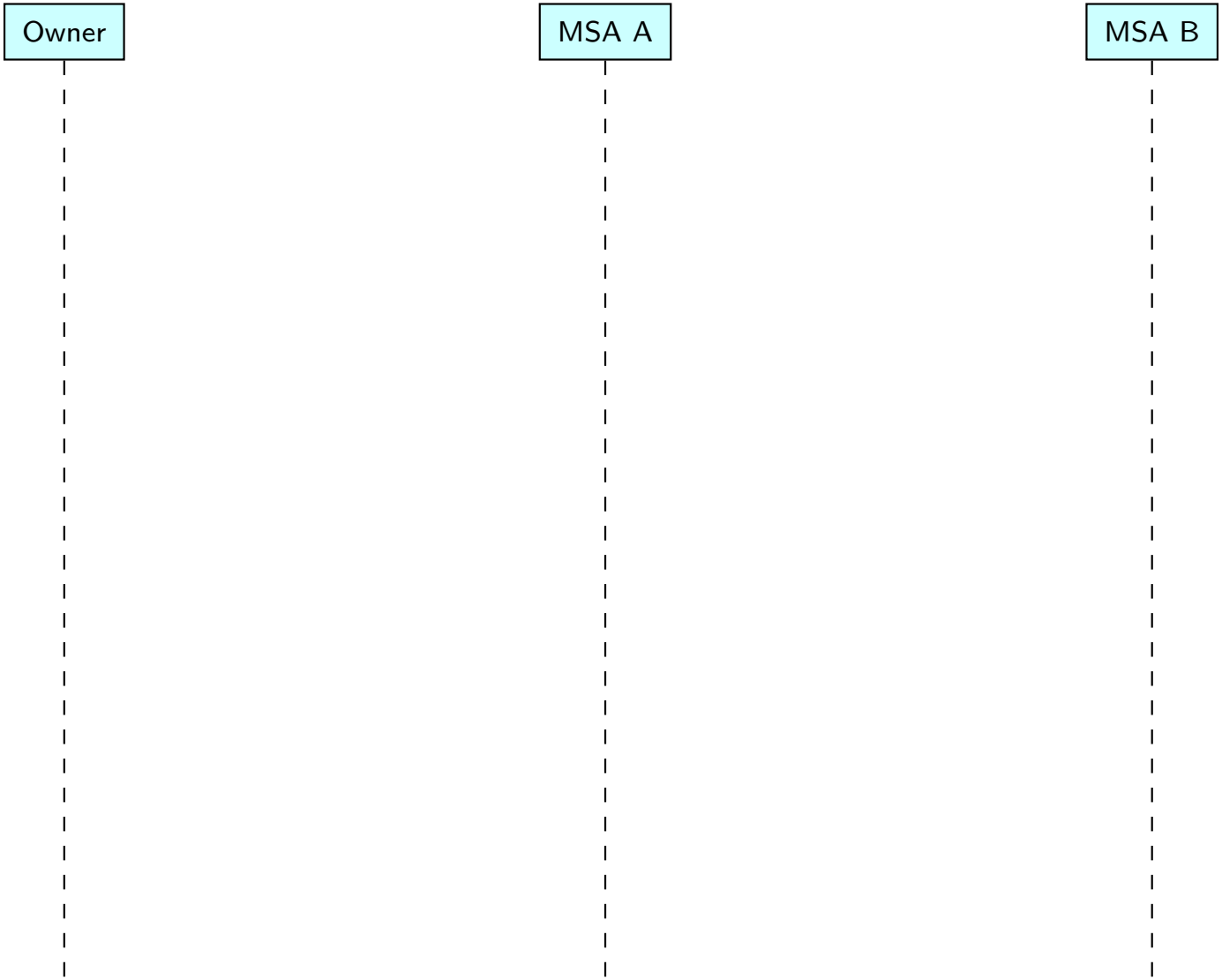
Establishing MSA Comms, Basic



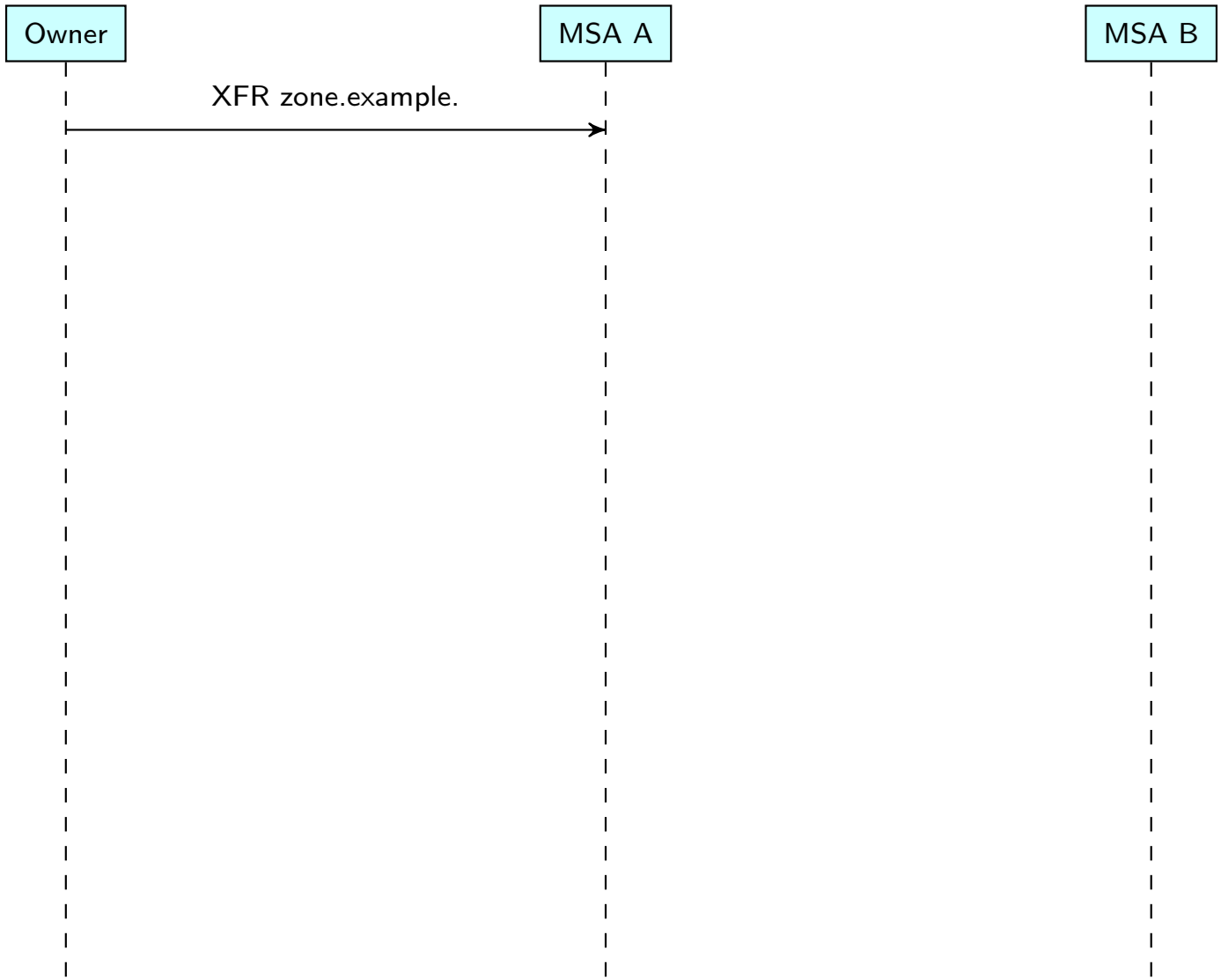
Establishing MSA Comms, Basic



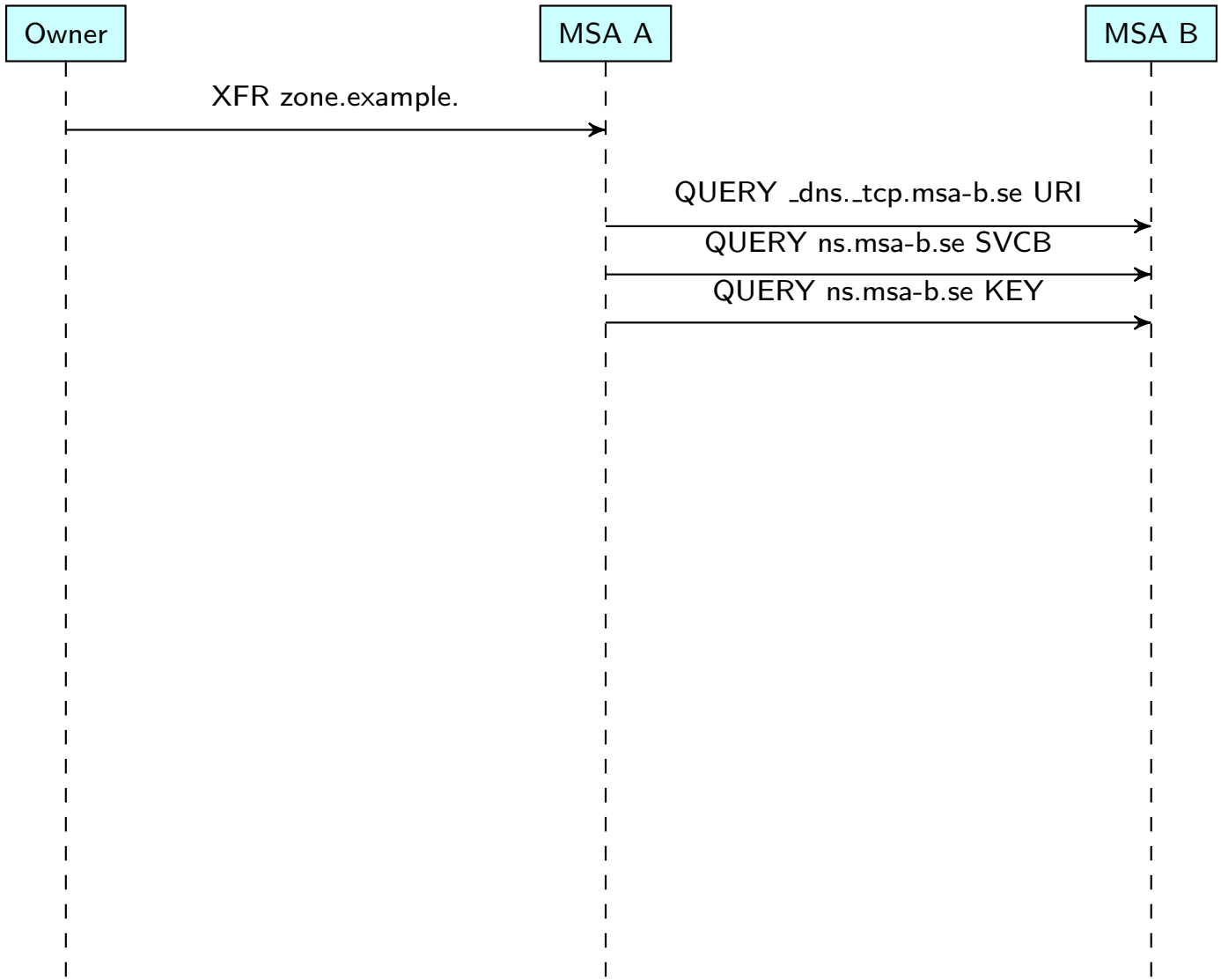
Establishing MSA Comms, XFR Delay



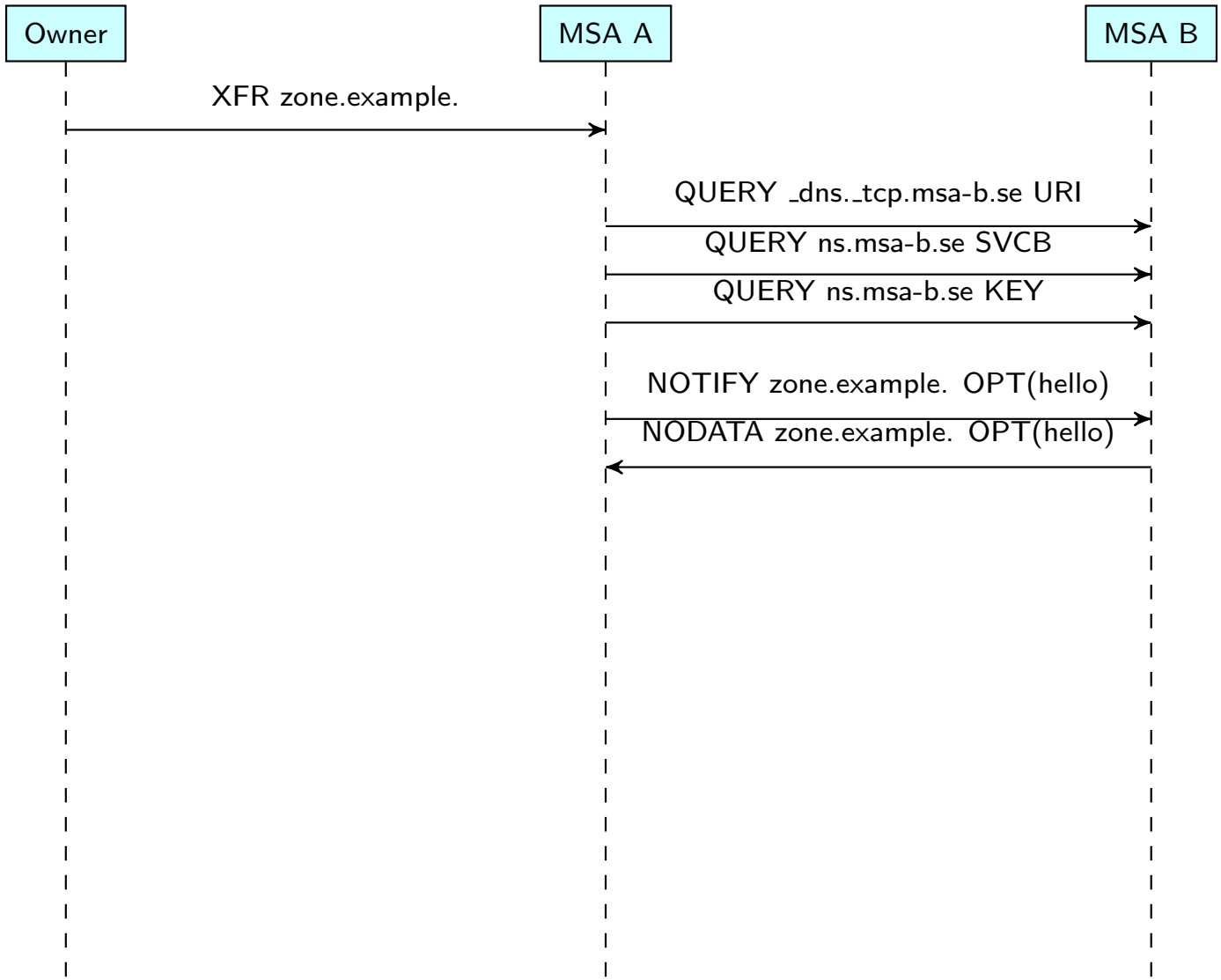
Establishing MSA Comms, XFR Delay



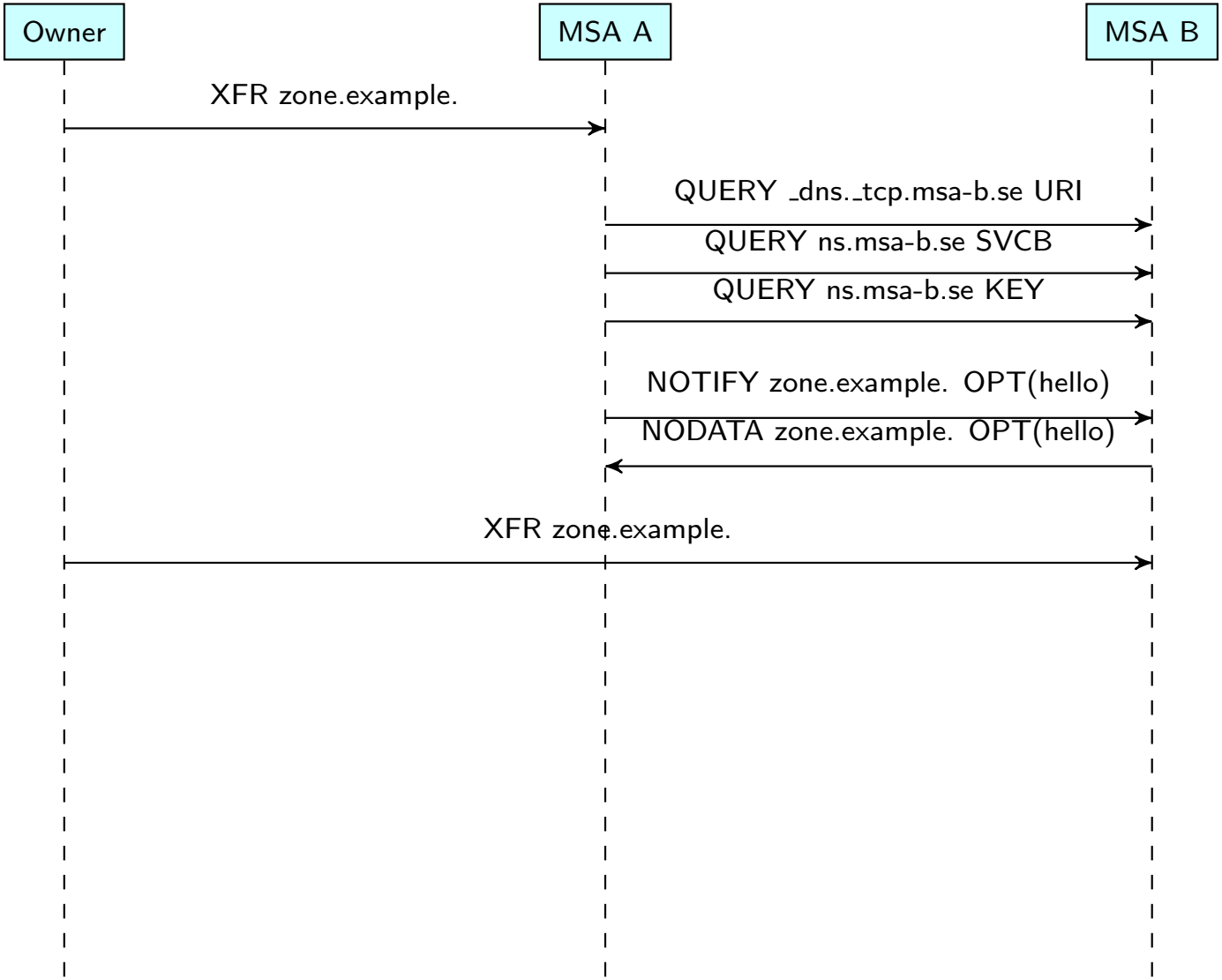
Establishing MSA Comms, XFR Delay



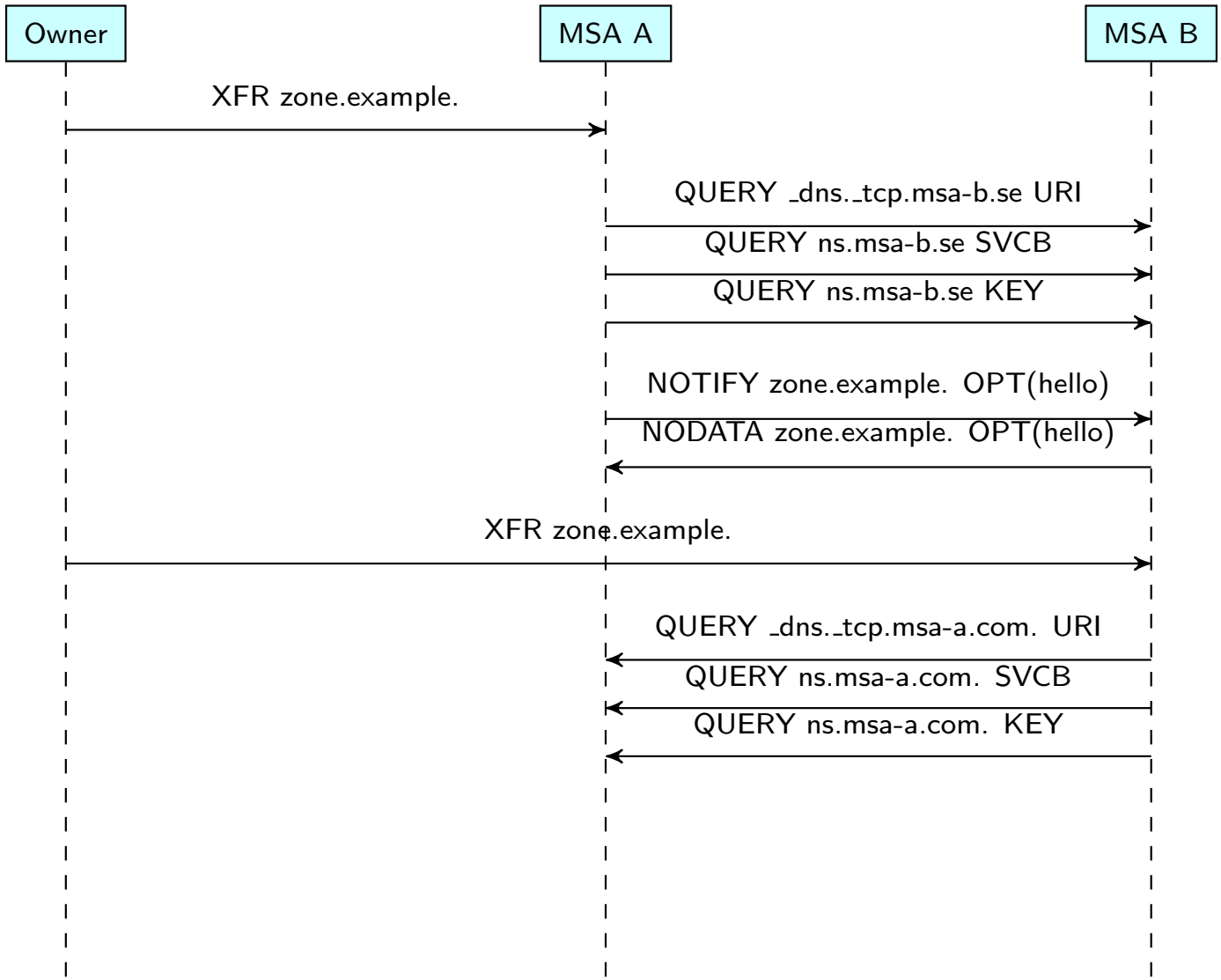
Establishing MSA Comms, XFR Delay



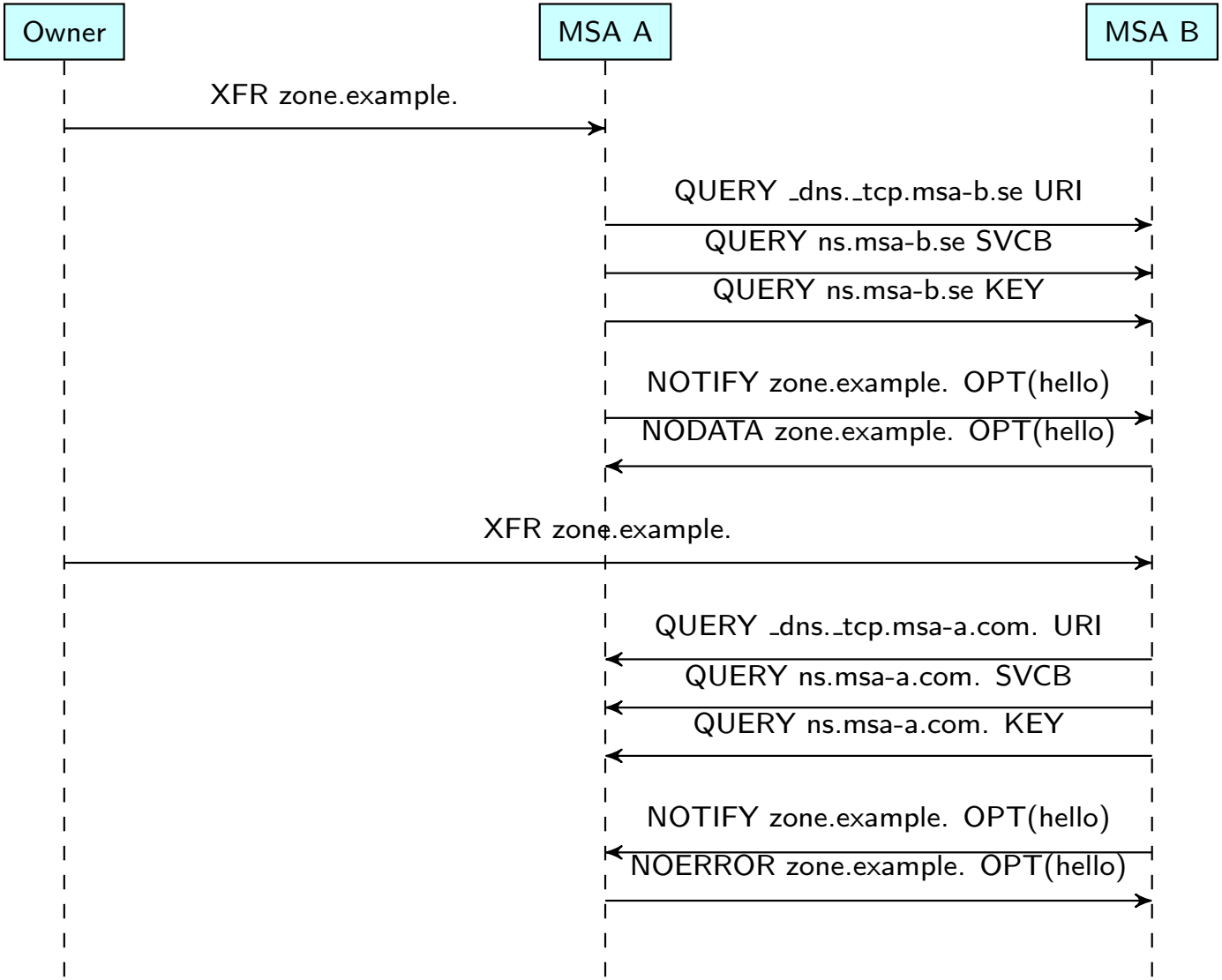
Establishing MSA Comms, XFR Delay



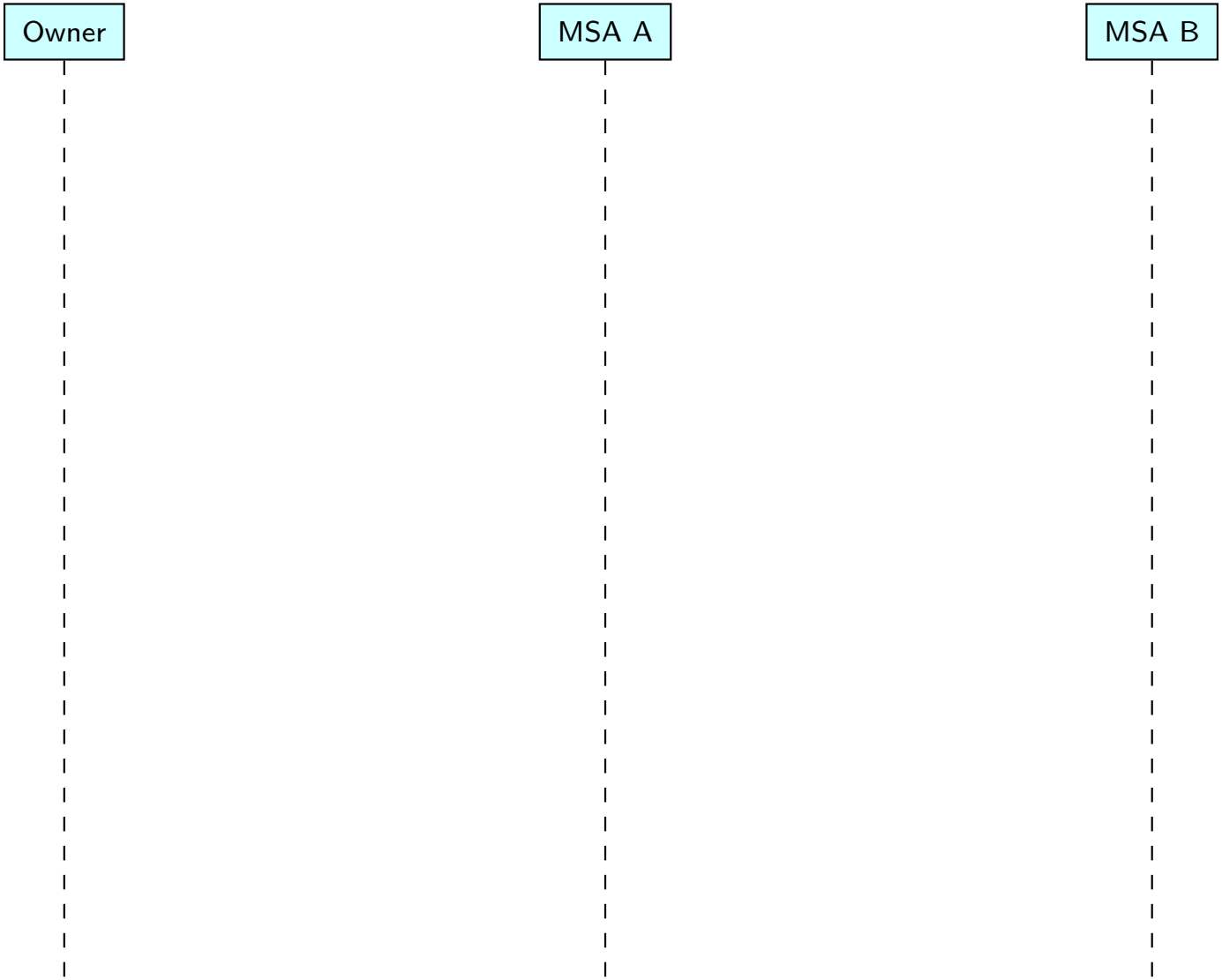
Establishing MSA Comms, XFR Delay



Establishing MSA Comms, XFR Delay

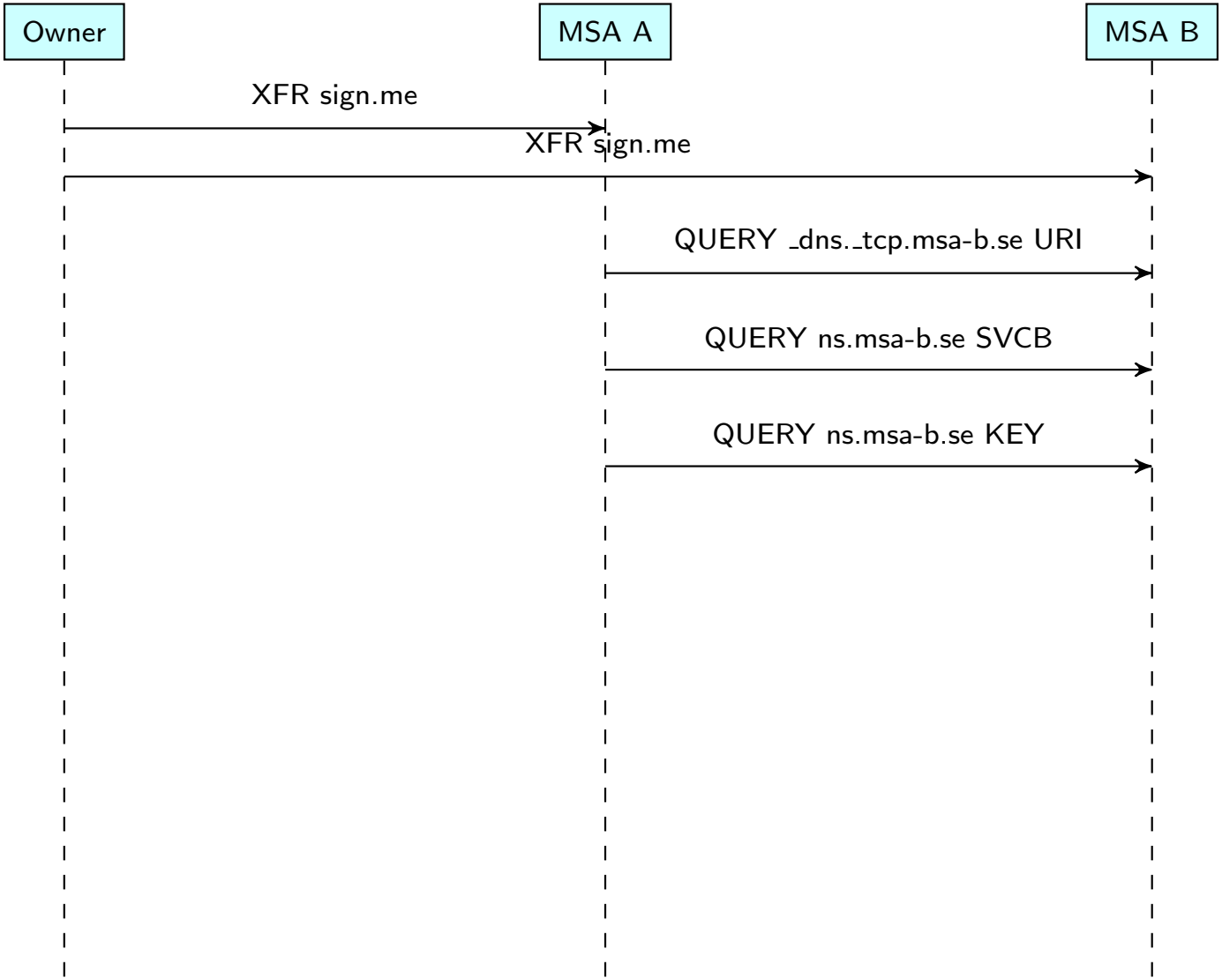


Establishing MSA Comms, Race

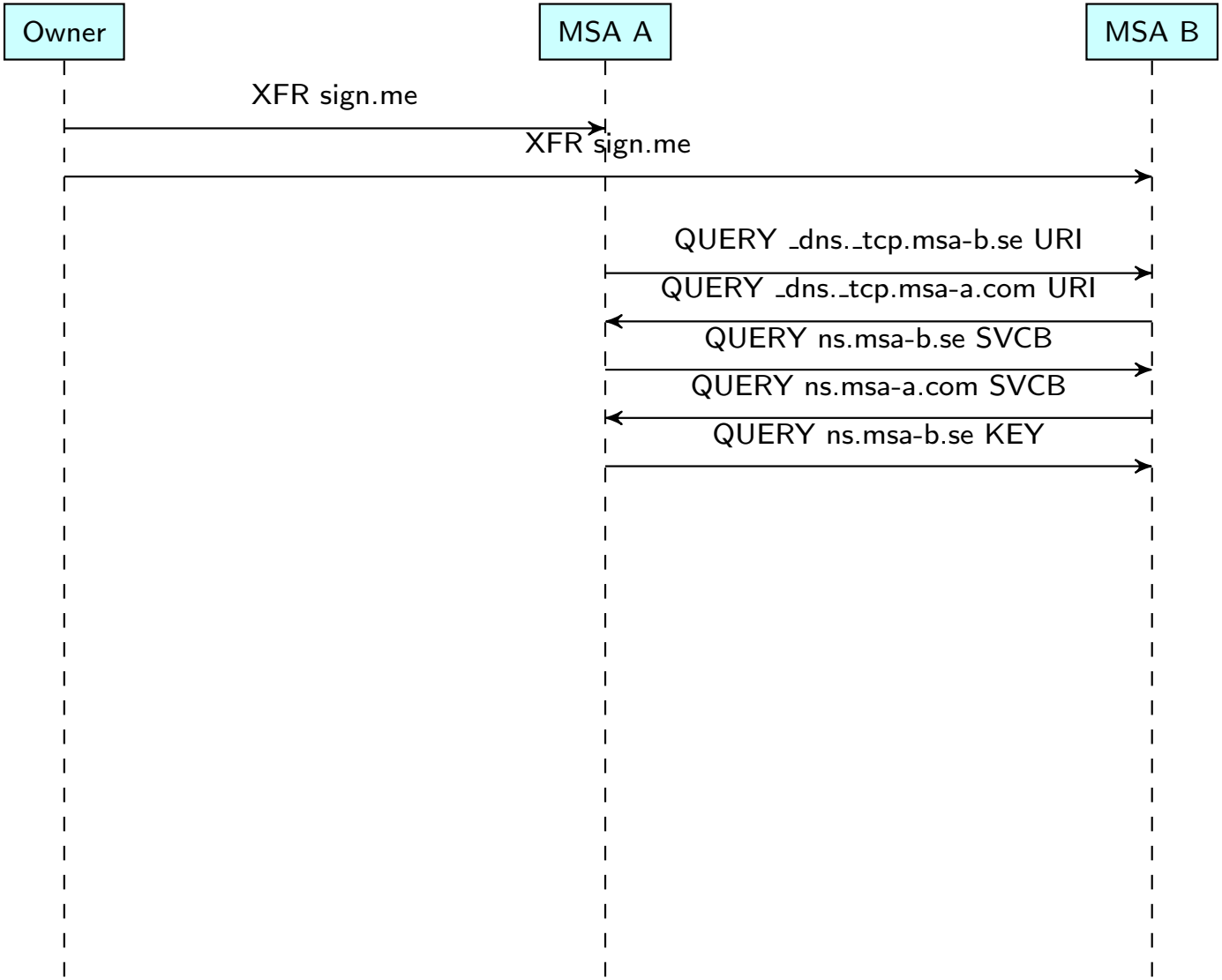


Establishing MSA Comms, Race

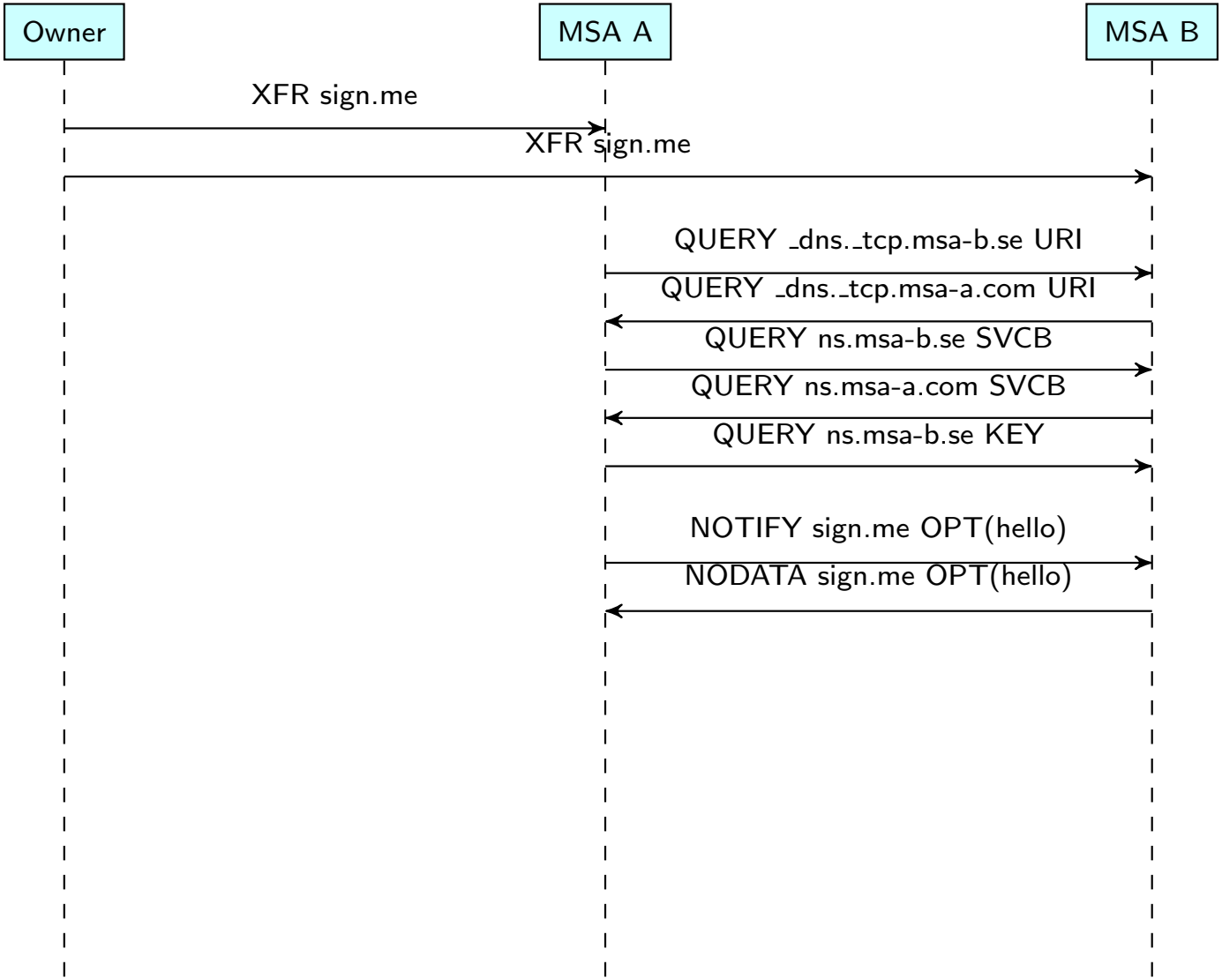
Establishing MSA Comms, Race



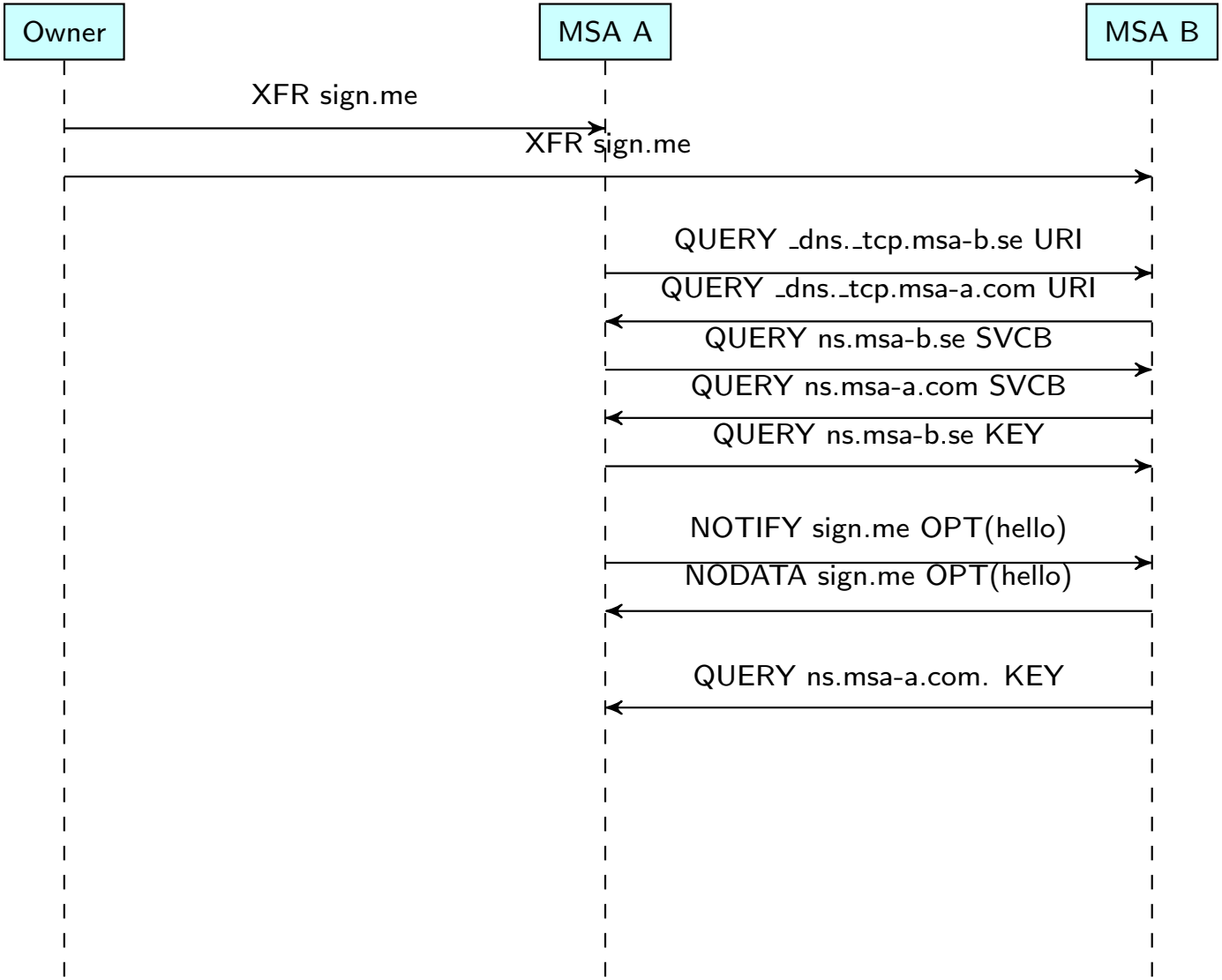
Establishing MSA Comms, Race



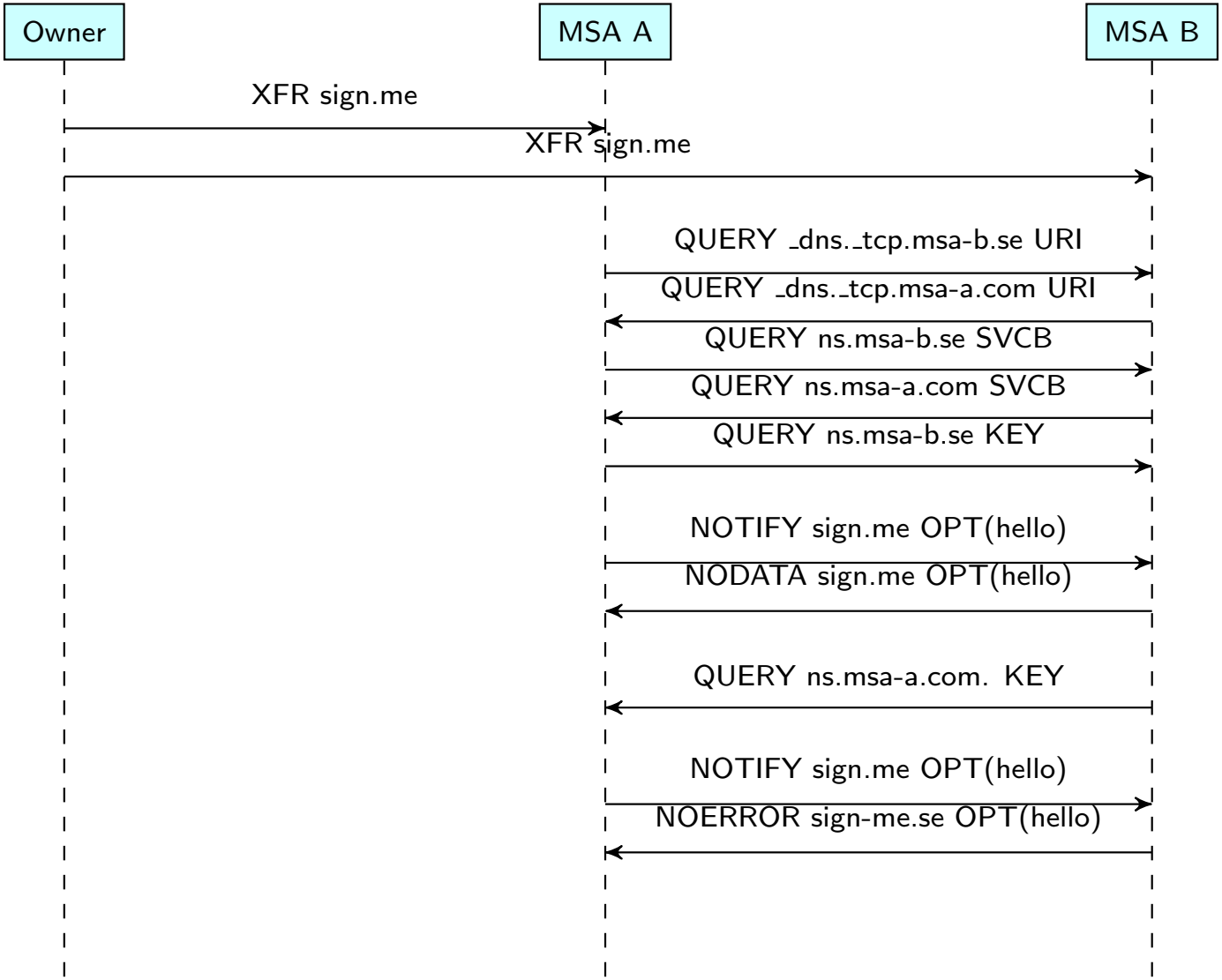
Establishing MSA Comms, Race



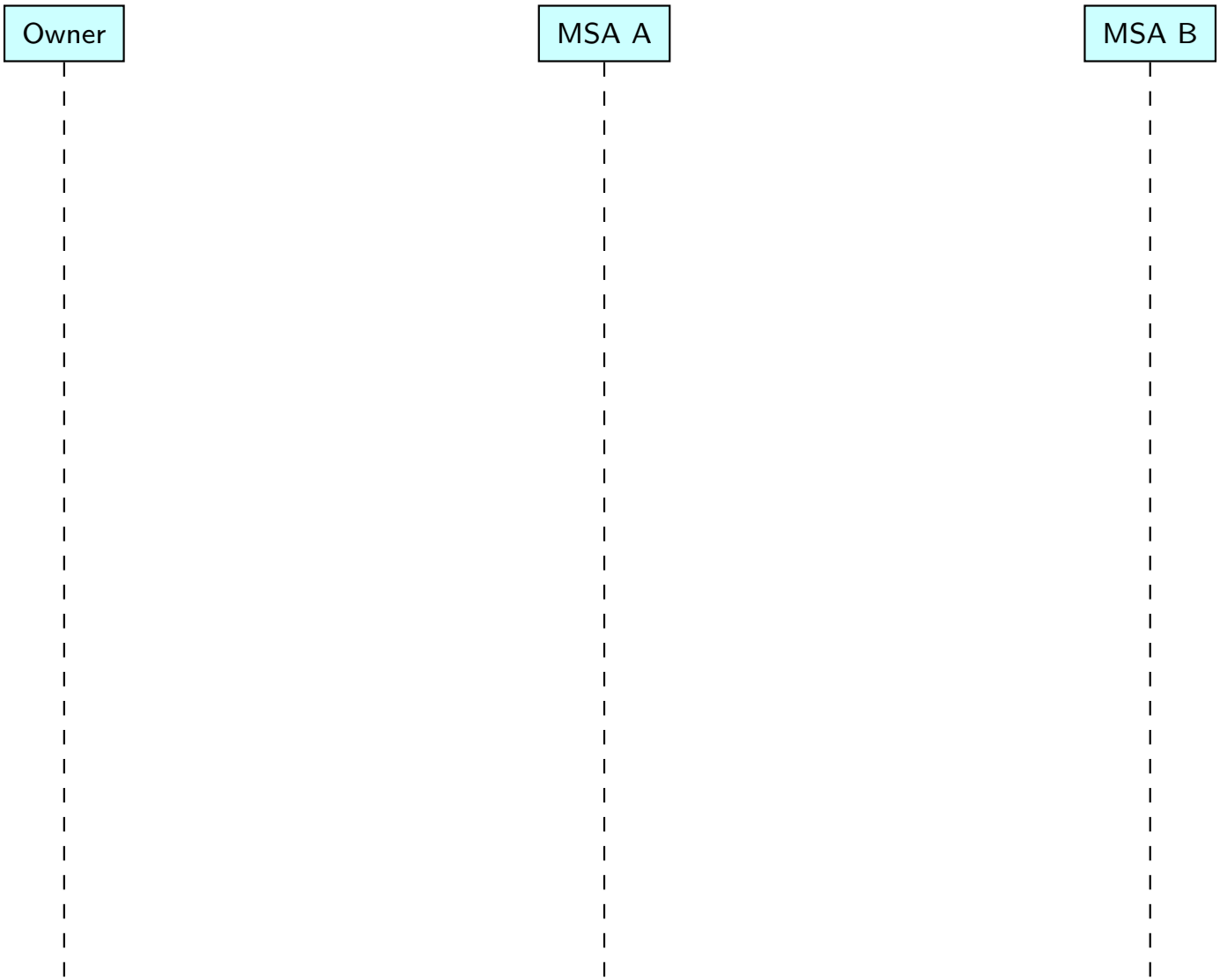
Establishing MSA Comms, Race



Establishing MSA Comms, Race

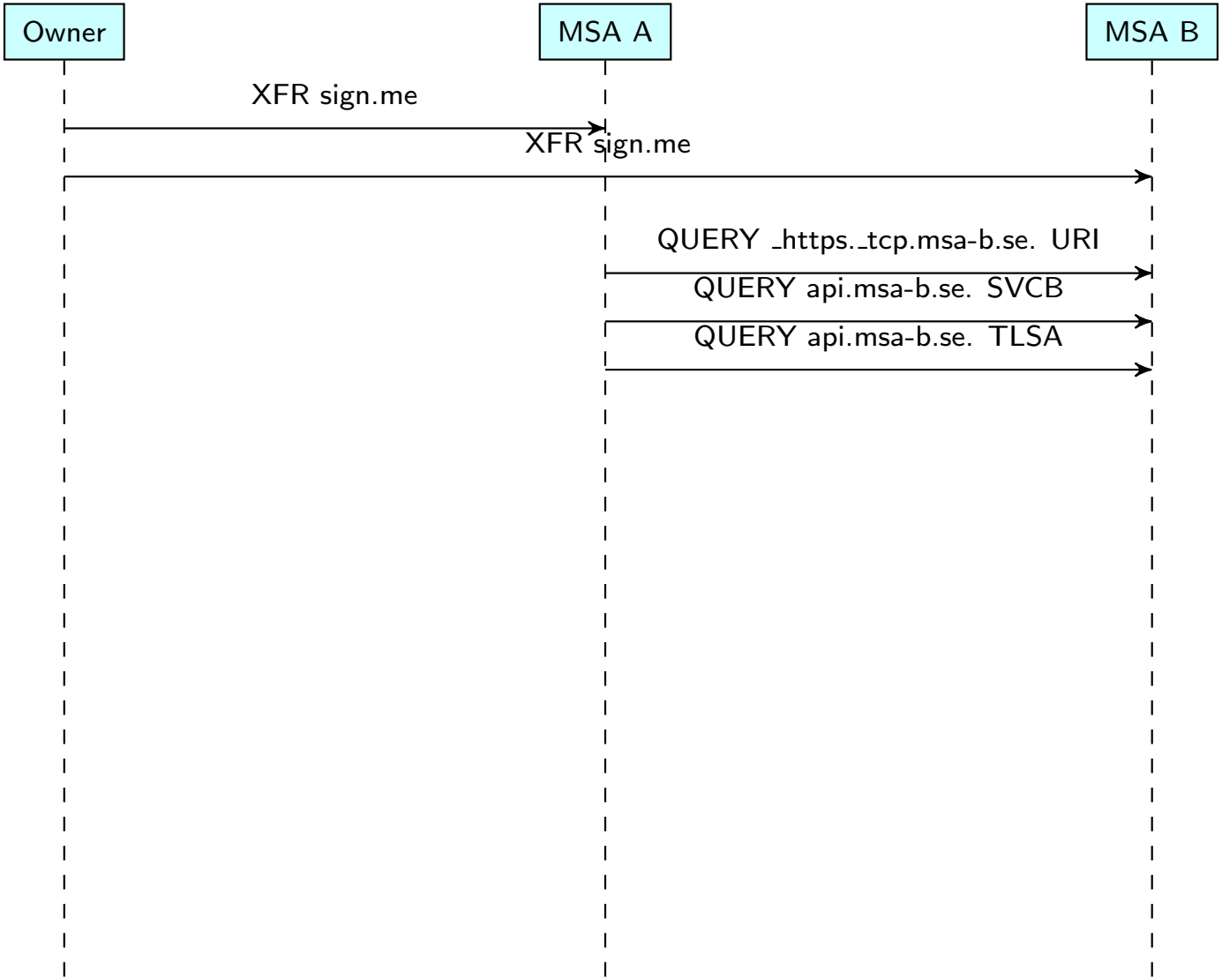


Establishing MSA Comms, Basic HTTPS

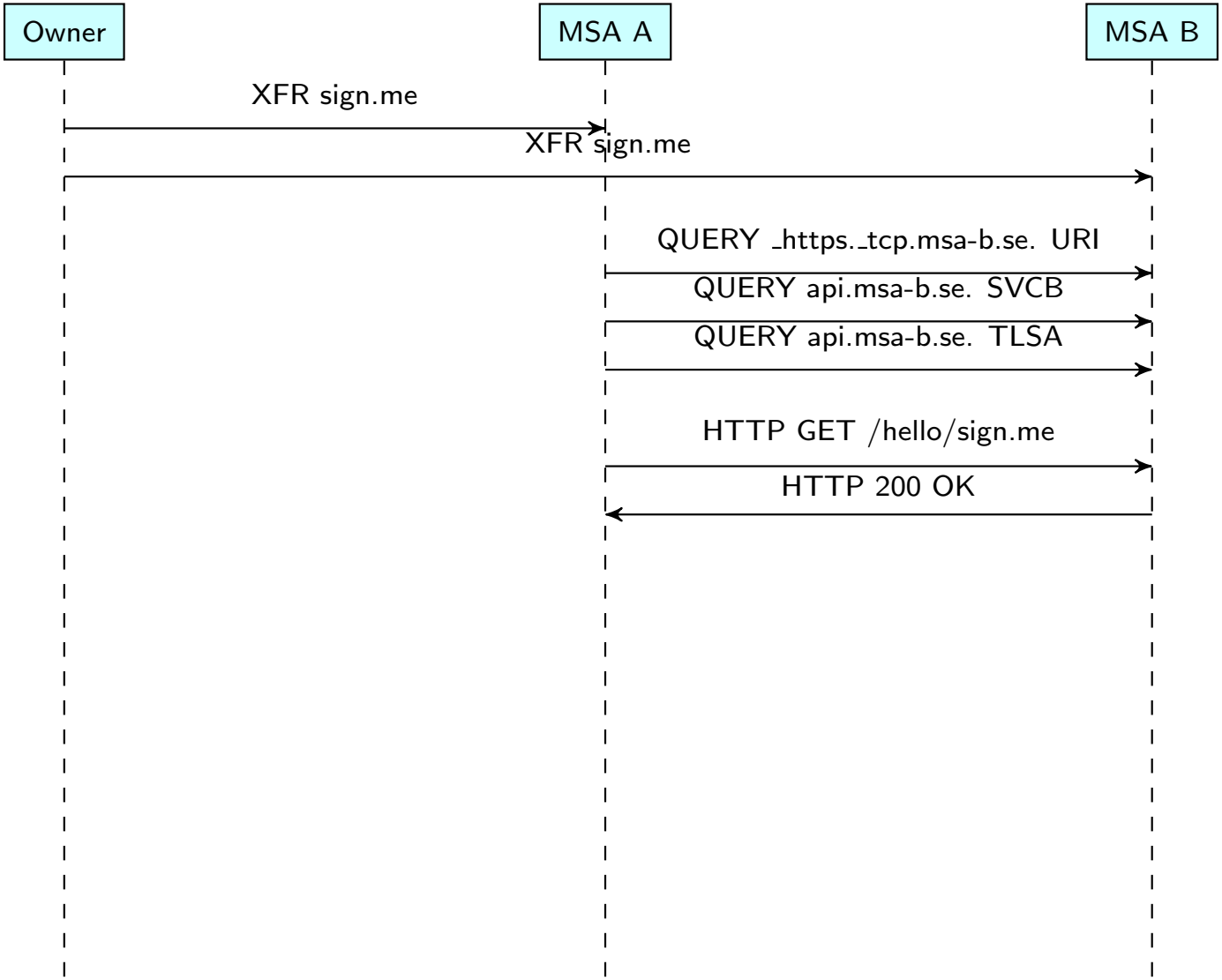


Establishing MSA Comms, Basic HTTPS

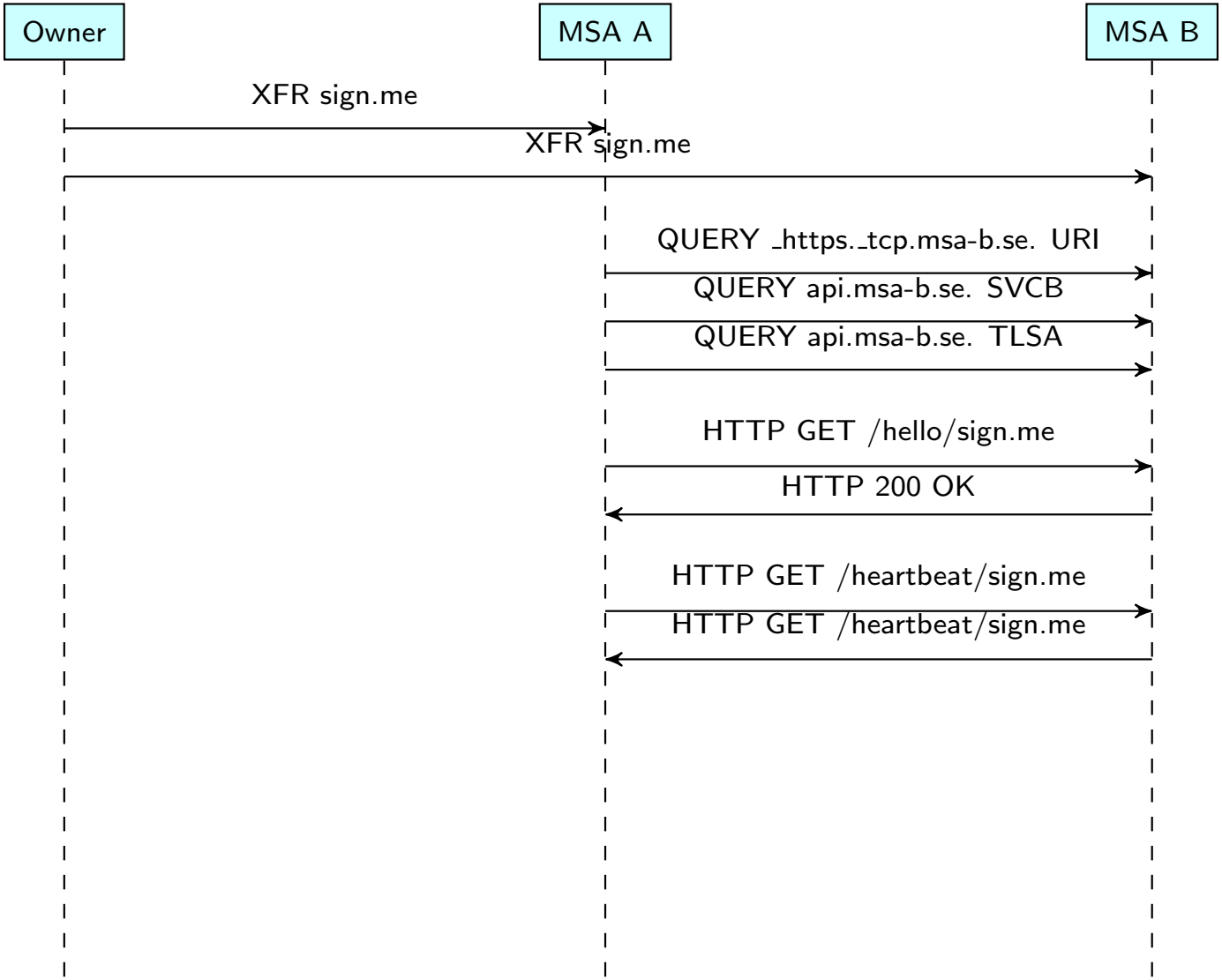
Establishing MSA Comms, Basic HTTPS



Establishing MSA Comms, Basic HTTPS



Establishing MSA Comms, Basic HTTPS



Is There a Recurring Theme in These Different Projects?

Possibly.

Automation of specific, critical, DNS configuration is a theme. In particular:

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

- Automatic delegation management is an issue only when child and parent zones are managed by different organizations.
  - The common case, and the reason that we don't really have such automation yet.
- Multi-signer setups where all signers are managed inside a single organization (like inside the .SE or .NZ registries. . .) is not really an issue.
  - But when the zone owner and the multiple signers are in different organizations the complexity increase rapidly.

Is There a Recurring Solution?

That's less clear. However...

We have approached these problems in a DNS-centric way, sometimes by defining additions to the DNS protocol:

- The new **DSYNC** record, perhaps also a new **HSYNC** record.
- Use of new EDNS(0) OPTs to implement new types of communication within the DNS protocol.

But, when possible, also by using existing mechanisms, perhaps in a new way:

- Use of SIG(0) for authentication outside of DNS UPDATE.

Is There a Recurring Solution?

That's less clear. However. . .

We have approached these problems in a DNS-centric way, sometimes by defining additions to the DNS protocol:

- The new **DSYNC** record, perhaps also a new **HSYNC** record.
- Use of new EDNS(0) OPTs to implement new types of communication within the DNS protocol.

But, when possible, also by using existing mechanisms, perhaps in a new way:

- Use of SIG(0) for authentication outside of DNS UPDATE.

Sometimes, however, good judgement may have been lacking:

- A painful tendency to solve problems by inventing yet another new DNS RRtype ending in **-SYNC**.

Is There a Recurring Solution?

That's less clear. However...

We have approached these problems in a DNS-centric way, sometimes by defining additions to the DNS protocol:

- The new **DSYNC** record, perhaps also a new **HSYNC** record.
- Use of new EDNS(0) OPTs to implement new types of communication within the DNS protocol.

But, when possible, also by using existing mechanisms, perhaps in a new way:

- Use of SIG(0) for authentication outside of DNS UPDATE.

Sometimes, however, good judgement may have been lacking:

- A painful tendency to solve problems by inventing yet another new DNS RRtype ending in **-SYNC**.
- Just wait until we publish the draft for the **KITCHEN-SYNC** record.

Open Questions

Among the things we haven't sorted out yet:

- Why use DNS? Why not just use an API like everyone else?
- Are the two requirements on the **SIGNER** sufficient?
- Using the MSA SIG(0) keys for authenticating DNS UPDATEs to the parent.
- Should authorization be needed for DNS UPDATEs to the parent?

Why Insist On Using DNS For All This?

Configuration can be used. APIs can be used. Etc.

Perhaps we should. But we don't know yet.

There are multiple reasons for at least trying to use DNS. Among them:

- Most importantly: transparency and verifiability.
  - For the use case "automation across multiple orgs", "hidden" configs always run the risk of getting out of sync without being noticed.
- We only need loose coupling, not tight sync and lockstep.
- Distribution via DNS is easy to make very redundant and robust.
- DNS has a strong track record of "getting the message through". Other choices of solutions may in some cases run into connectivity issues (firewalls, port filters, etc).

As we don't know yet, we aim to test both.

- Both the delegation sync (via **`DSYNC`**) and multi-signer sync (triggered by **`HSYNC`**) have hooks for "API" as an alternative.

Requirements on the **SIGNER**

One of the cornerstones of this architecture is to try to get as close as possible to being able to run a “normal” signer, that should not have to know about all the multi-signer details.

But we have been unable to relax the following two requirements:

- The SIGNER **MUST** be able to keep incoming **`DNSKEY`**s (that originate from other DNS providers) in the RRset together with the DNSKEYs that it is managing.
- The SIGNER **MUST** be able do key rollovers based on an ”external clock”
  - like the MSA controlling that the rollover proceeds in sync with other providers.

Are there any other corner cases lurking out of sight?

Using MSA SIG(0) Keys for Delegation Management

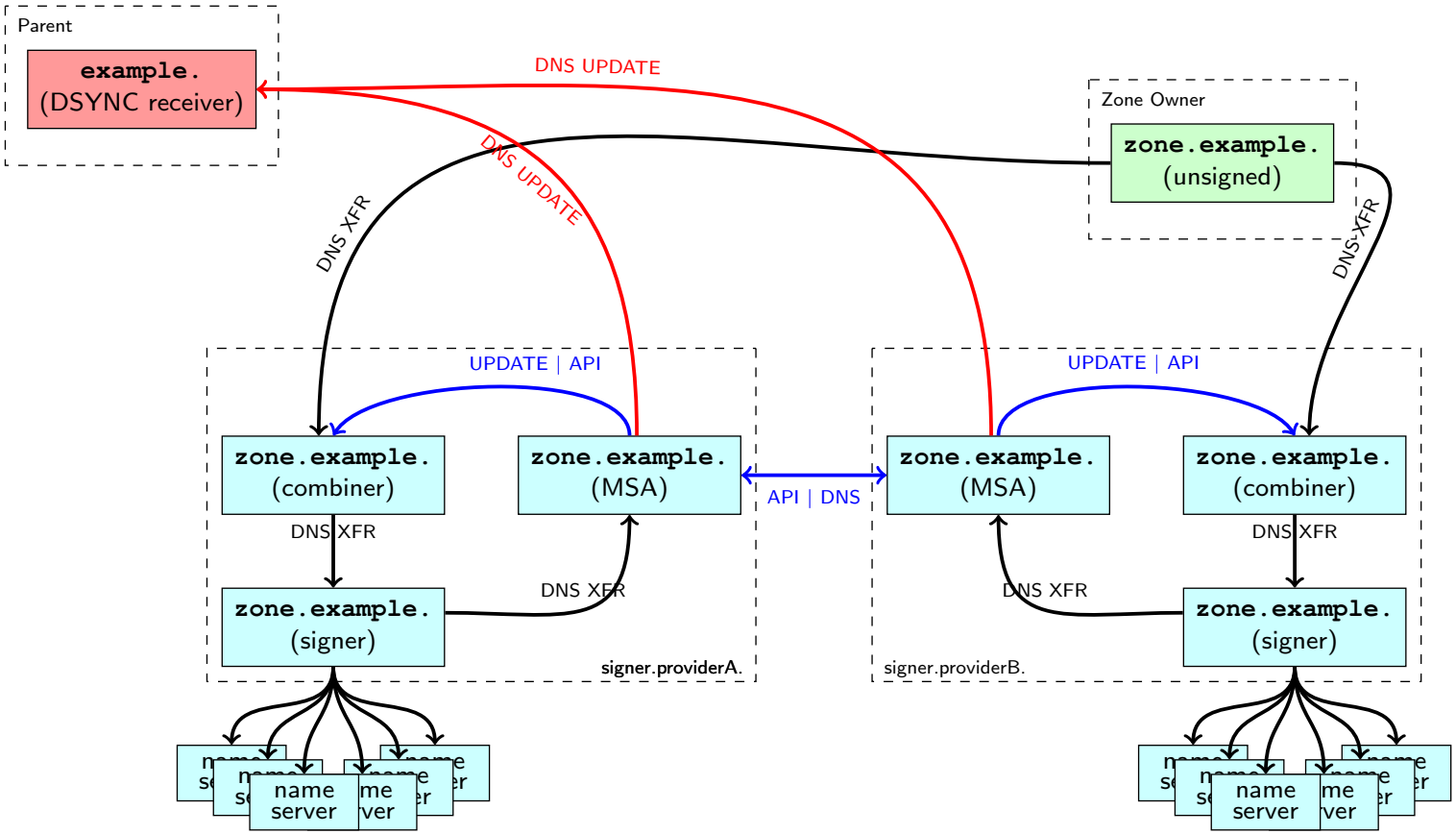
In **`draft-...-delegation-mgmt-via-ddns`** there is presently a limitation to using a single SIG(0) key (at a time, rollovers are supported). And the key should have the name of the child zone.

In the multi-signer use case of this it would be great to be able to use the MSAs for this.

- The MSAs already have SIG(0) keys, including a published **`KEY`** record.
- Their raison d'être is to provide service to the zone owner.

However, while it is possible to enhance the delegation-mgmt to allow multiple keys, it is difficult to figure out how to bootstrap SIG(0) keys with "arbitrary names" as authoritative for signing updates to delegation data.

Multi-Signer Setup, Including DDNS To Parent



Authorization needed from the owner for DNS UPDATEs to the parent?

"Normal" multi-signer already interacts with the parent.

- By publishing CDS and CSYNC RRsets (and hoping for parent-side scanners).

In fact, in a multi-signer context, **not** allowing the DNS providers to interact with the parent at all would most likely not work well.

Does delegation synchronization via DNS UPDATE require some additional authorization by the zone owner?

- If not, what is the rationale?

This issue is not "how" (it would be easy to extend the **`HSYNC`** record if needed), but only "if".

- In particular, as a stated goal is to avoid unnecessary knobs and buttons.

Questions?

Thanks & Contact Information

erik.bergstrom
leon.fernandez
johan.stenstam
@internetstiftelsen.se

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