# Towards Distributed Multi-Signer:

## Architecture Overview

Johan Stenstam

Swedish Internet Foundation

November 11, 2024

# Problem Statement

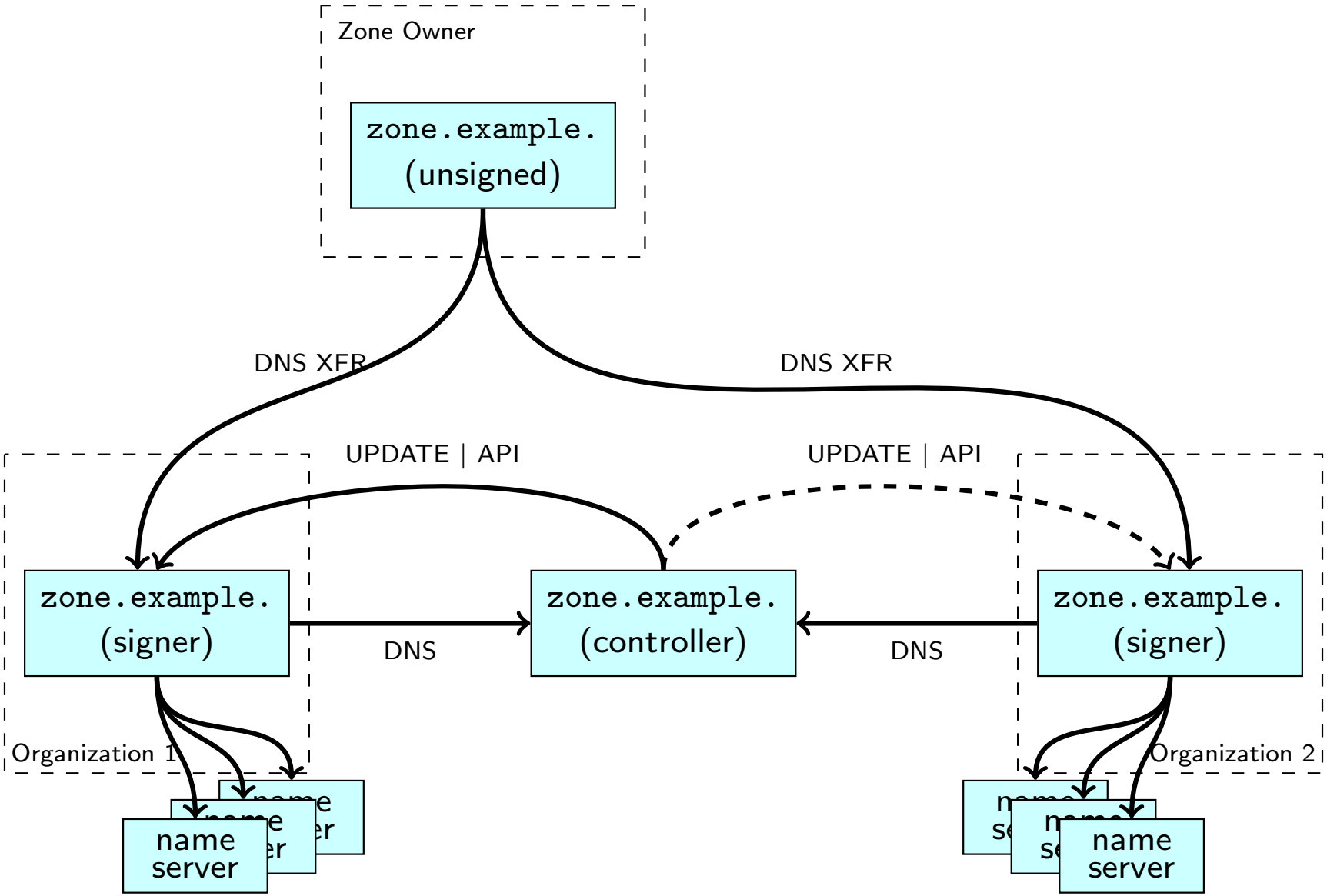
"Multi-signer" has been talked about for a long time and there are several open source or proprietary implementations. However, multi-signer has not really "taken off".

- There are likely multiple reasons for this.
- One major issue has been that 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 re-think 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.

# Original Monolithic Multi-Signer Architecture



# Who is in Control?

The original implementation of a multi-signer controller was "monolithic", i.e. it was a single, separate, controller that told the signers what to do.

In the distributed version there are multiple controllers, each next to a signer (in a so-called "sidecar" design).

# Why Separate Multi-Signer Controllers (aka "Sidecars")?

It is absolutely possible to have a distributed multi-signer setup without any other components than the signers.

However, there are advantages with having the multi-signer controller as a separate component next to the signer:

- Multi-signer is a complex system, there is a lot of logic to each "process".
- The available signers (typically open-source name servers that provide inline-signing) do not have this logic today.
- By using a "sidecar" design the signer can be left unmodified.
- The only requirement is that the signer sends the signed zone to the sidecar and that the sidecar has a mechanism to apply changes to the zone.

# What is a Multi-Signer "Process"?

A "process" is a sequence of state transitions that will cause a zone to move from the current state to a desired end state.

A typical example is the addition of a new **Signer** for a zone. This automatically causes the zone to enter the state `SIGNERS-UNSYNCHED`. To return to the desired state `SIGNERS-SYNCHED`, the zone undergoes the **ADD-SIGNER** process:

`SIGNERS-UNSYNCHED` → `ZSK-SYNCHED` → `CDS-KNOWN` → `CDS-SYNCHED` →
`DS-SYNCHED` → `CDS-REMOVED` → `NS-KNOWN` → `NS-SYNCHED` →
`CSYNC-PUBLISHED` → `PARENT-SYNCHED` → `SIGNERS-SYNCHED`

While the process contains many steps, each step is straightforward.

# What is a Multi-Signer "Process", cont'd

Multi-signer processes are normally linear, but they do involve a number of "steps".

- Processes are implemented as "Finite State Machines".
- Each "step" is a state transition, from one stable state to another.
- Each state transition has a defined **Pre-condition** that must be true.
- Each state transition may have an **Action** to take.
- Each state transition has a defined **Post-condition** that must be true (otherwise the transition is rolled back).

- In the monolithic version of multi-signer the controller is responsible for checking the pre-conditions and for taking the action.
- In the distributed version this responsibility must still be taken, either by one of the sidecars or by all of them.

# Alternatives for Controlling the Multi-Signer Process

Once there are multiple controllers, there are two alternatives for controlling the multi-signer process:

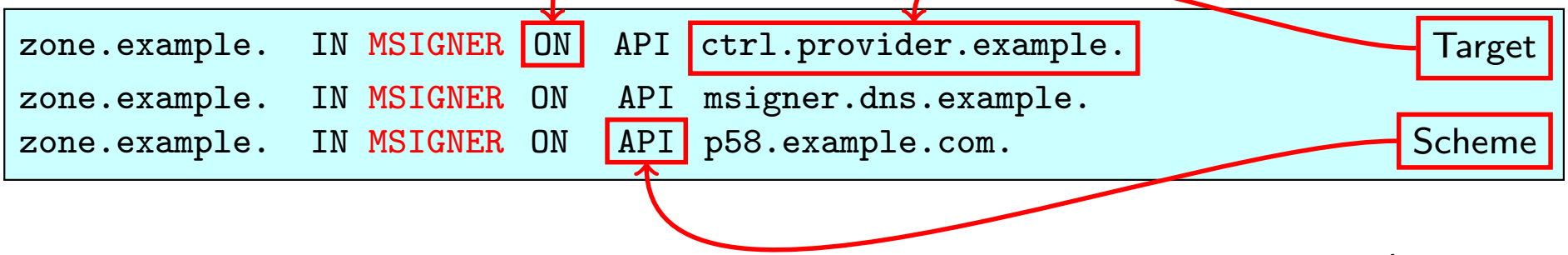
- One controller acts as the "leader" for the process. Before starting the process, the leader is elected.
  - **Pro:** Simple to implement. Current approach is to use a REST API.
  - **Con:** Requires a non-DNS based mechanism to control the process (pure-DNS may be possible, not yet clear).
- The controllers are fully independent.
  - **Pro:** Pure-DNS based control.
  - **Con:** More complex to implement, as this requires a more fine-grained coordination mechanism.

In both cases the controllers must be able to locate each other and establish secure communication.

- The zone owner uses the `MSIGNER` RRset, published in the zone, to designate the signers that should sign the zone.

# Designating the Signers: The `MSIGNER` RRset

The `MSIGNER` RRset is published in the unsigned zone by the zone owner and contains identifiers for the designated controllers.




- The **State** indicates whether this signer is being on-boarded (or is already in sync, `ON`), or is being off-boarded (`OFF`).
- The **Scheme** indicates the mechanism to use for communication. `API` means "use the REST API mechanism", while `DNS` means "use the DNS based mechanism".
- The **Target** is an identifier for the controller.

**Note that all controllers for a zone MUST use the same Scheme.**

# API-based Synchronization

When a zone designates a set of signers, the signers will locate each other using the `MSIGNER` RRset.

- To enable secure communication the sidecars look up and DNSSEC-validate the **TLSA** RRset for the `MSIGNER Target` name.
- Sidecars that do not previously know about each other will initiate contact ("**hello**") and then maintain a periodic "**heartbeat**".
- When an event that triggers a multi-signer process is detected, the sidecars will **elect a leader** for this process.
- The leader will step through the process in exactly the same way as in the old model.
- The **leader uses the API to coordinate** the other sidecars.

**The advantage with this approach is that once the leader is elected, the actual process is exactly the same as in the old model.**

# DNS-based Synchronization

When a zone designates a set of signers, the signers will locate each other using the `MSIGNER` RRset.

- For secure communication the sidecars use SIG(0)-signed DNS messages. To do this they will first need to look up and DNSSEC-validate the public SIG(0) keys for the other sidecars.
- When an event that triggers a multi-signer process is detected, no leader is elected.
- Instead, all sidecars are notified about the event and each sidecar will evaluate the event independently.
- Each sidecar will step through the process in the same way as in the old model, with the only difference that the sidecars will notify each other at each step.
  - Should, for some reason, two sidecars "disagree" then no further steps will be taken and while the current "state" is safe and stable, manual intervention will be needed.

# Implementing State Transitions: API vs DNS

Some state transitions do not change any zone data. Eg. when waiting until the parent zone has published a new `DS` RRset.
Other state transitions imply making some sort of modification to the zone: adding or removing `DNSKEY`s, publishing a `CDS` or `CSYNC`, etc.
In the REST API case, implementing actions that modify the zone data in each signer will be different depending on who is the "Leader":

- If the signer sits next to the Leader, then the Leader has a mechanism for updating the zone (usually via an API or a DNS UPDATE).
- If the signer is a "remote" signer (i.e. a signer next to one of the other agents) the Leader will send the change as a request to that agent to deal with locally.
- The local agent will analyze the change according to local policies, security verifications, etc. When satisfied it will make the change via its local mechanism to update the zone in the local signer.

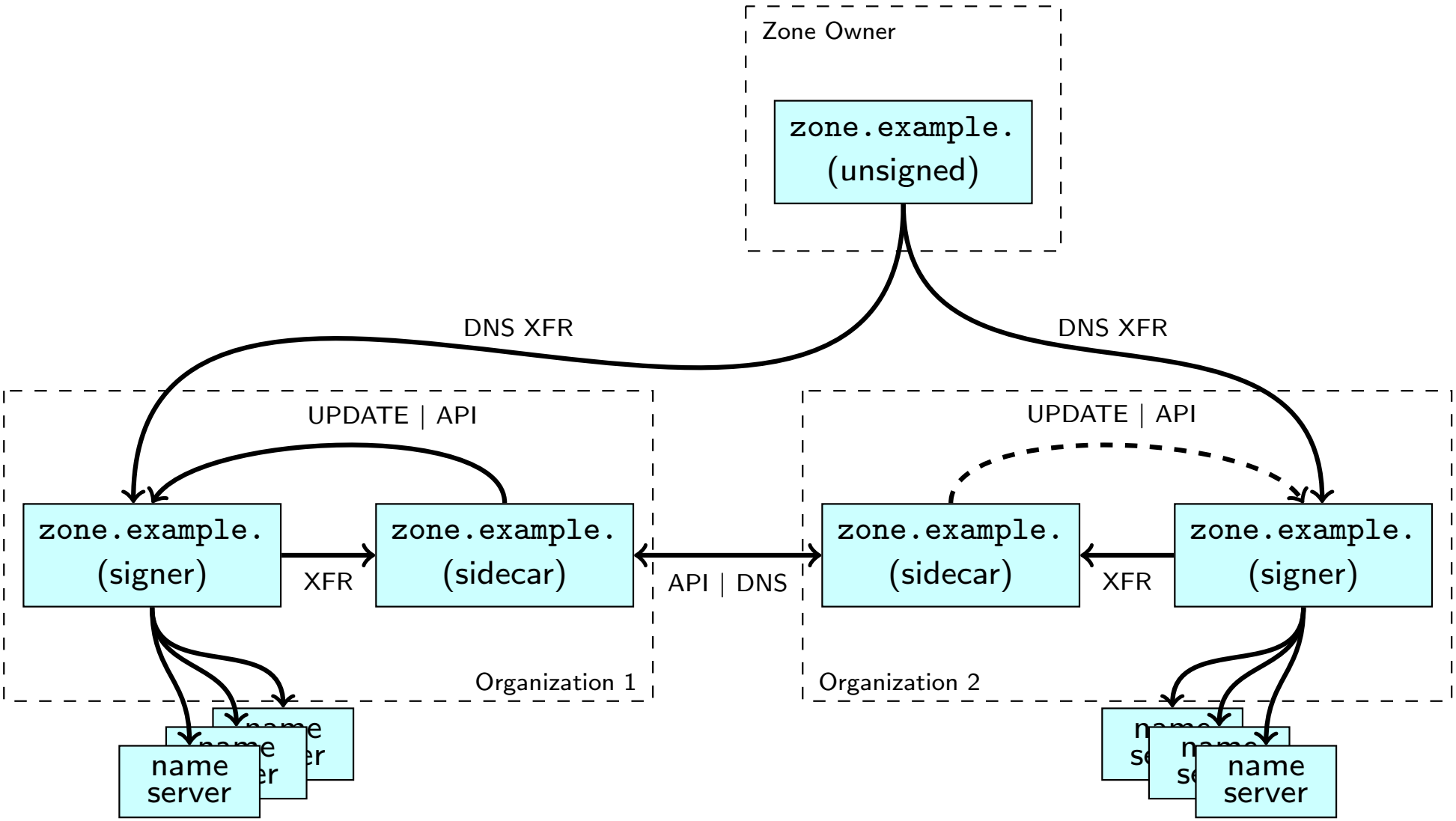
# Comparing the two Alternatives

Both mechanisms will work.

- The API-based mechanism is simpler to implement, as it is based on the existing controller mechanism.
- The DNS-based mechanism is perhaps a "cleaner" solution, but requires more changes to the existing controller logic.

**The current plan is to implement both alternatives.**

# Proposed Distributed Multi-Signer Architecture



# What About Key Rollovers?

Key rollovers (eg. one signer adding a Zone-Signing Key) are a complicating factor that can not be ignored.

- The reason it cannot be ignored is that modern signers do not ask permission before initiating a key rollover.
- For distributed multi-signer to be able to work with unmodified signers, key rollovers must work.

The good thing is that the sidecar design already has a mechanism for notifying the other sidecars about a key rollover:

- Sidecars are technically "secondaries" for the zone, downstream from the signer.
- Sidecars already check the incoming zone from the signer to analyze the `MSIGNER` RRset (to locate the other sidecars).
- Sidecars also analyze the incoming `DNSKEY` RRset. If a change is detected, the sidecar will notify the other sidecars.

# What About Key Rollovers, cont'd

What happens in the recipient end of a `DNSKEY` change notification? First, the recipient sidecar will check that the `DNSKEY` RRset actually changed.

Once that is established, there are two cases:

- If the zone is **NOT** in the midst of a multi-signer process, then the zone should enter the multi-signer process for synchronization of the `DNSKEY` RRset (which is simpler than adding a new signer).
- If, on the other hand, the zone **IS** in the midst of an ongoing multi-signer process, then the zone should just restart the process.
  - The reason is that synchronization of the `DNSKEY` RRset is a part of all the other multi-signer processes.

# Summary

- The `MSIGNER` RRset is used to designate the multi-signer "signers" for a zone in a way that enables dynamic discovery of the controllers.
- The sidecar design is specifically intended to be able to provide a distributed multi-signer service using unmodified signers.
- By analyzing the `MSIGNER` and `DNSKEY` RRsets, the sidecars can detect key rollovers and other changes to the zone that affect the multi-signer process.
- Implementation work is ongoing.

# Questions & Contact Information

| | |
|---|---|
| **Email** | johan.stenstam@internetstiftelsen.se |
| **Code** | https://github.com/johanix/tdns |
| | https://github.com/DNSSEC-Provisioning/music |

"Code:" This is the code we are working on, and it works. But it does not yet implement all the features discussed here.