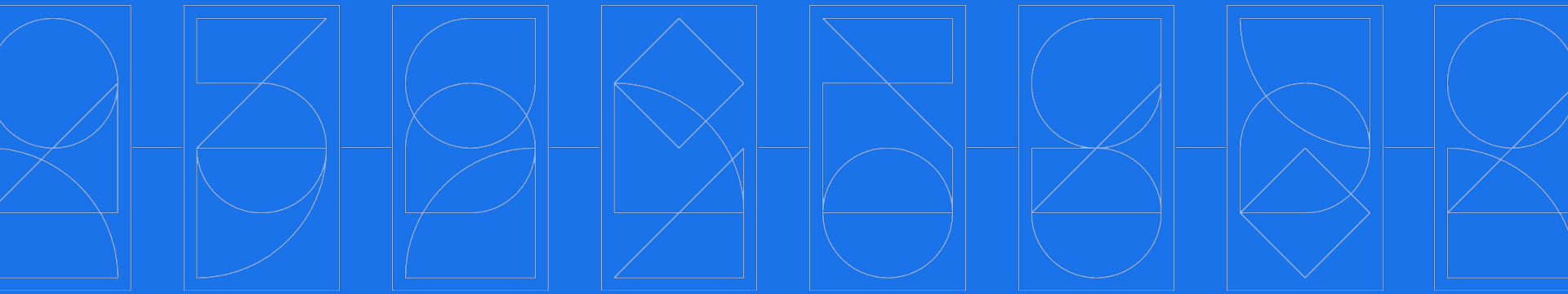
TrustFabric

A guide to applying the Claimant Model



The **Claimant Model** is a formal framework to describe what's important to log and who will verify its contents.

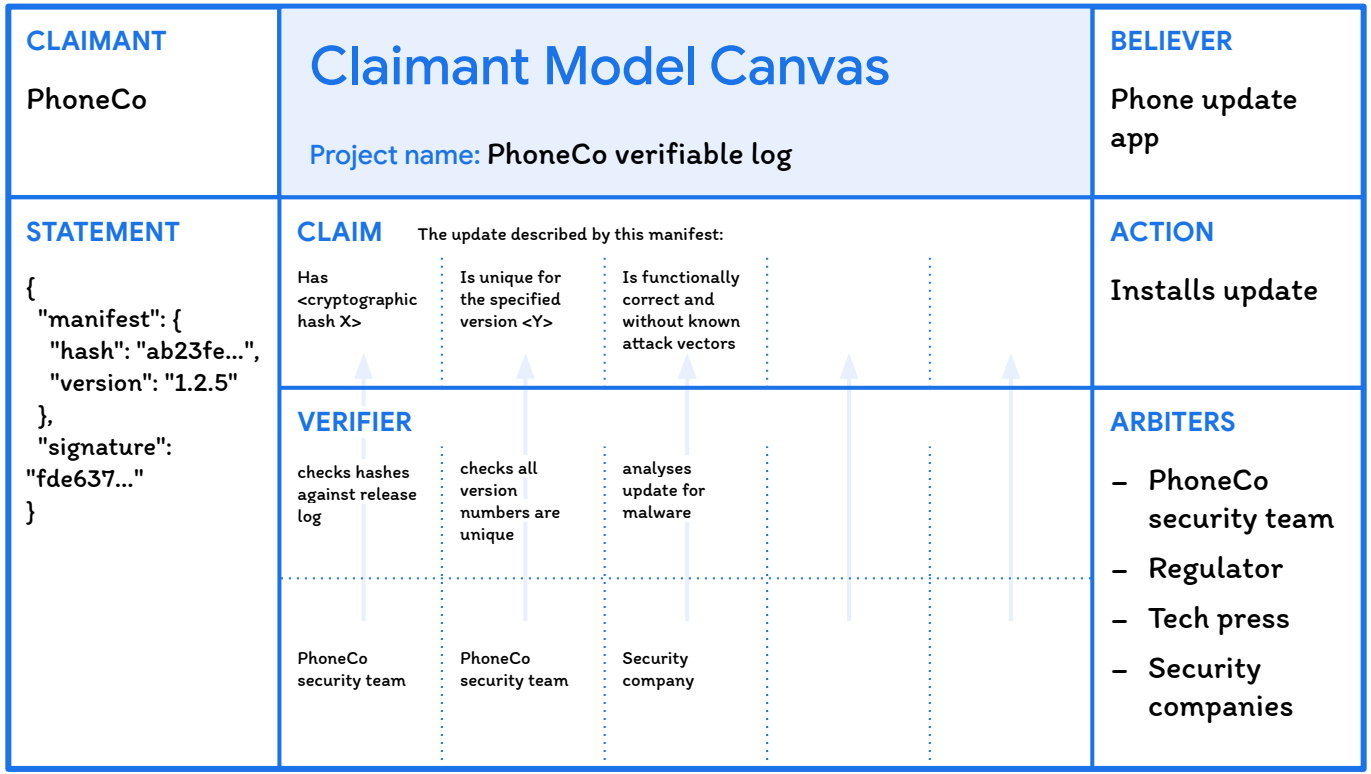
This presentation uses a fictional scenario to show how to fill in the canvas, step-by-step.

At the end is a template canvas you can use to complete a Claimant Model for your domain.

Hopefully you have tried answering these questions first to help you examine and clarify the design of your verifiable system.

A filled in canvas looks like this:

Claimant Model Canvas template



THE SCENARIO

- PhoneCo makes smartphones. They release software updates every month.
- When they publish software updates, they sign them with a private key.
- When their customers' update app gets a new software update, it checks the signature came from PhoneCo.
- The phone updater app doesn't install an update without a valid signature.

THE SCENARIO

- One of PhoneCo's competitors private key gets stolen and used to release malicious updates.
- PhoneCo want to protect themselves against that scenario and have designed a way for a verifiable log to do that.

We'll document PhoneCo's verifiable system using the **Claimant Model canvas.**

PhoneCo decide to start logging a manifest every time they release a new genuine software update.

It contains the version number and a cryptographic hash of the update:

```
{
  "manifest": {
    "hash": "ab23fe...",
    "version": "1.2.5"
  },
  "signature": "fde637..."
}
```

The phone updater app will look in the log for a matching manifest before installing any updates.

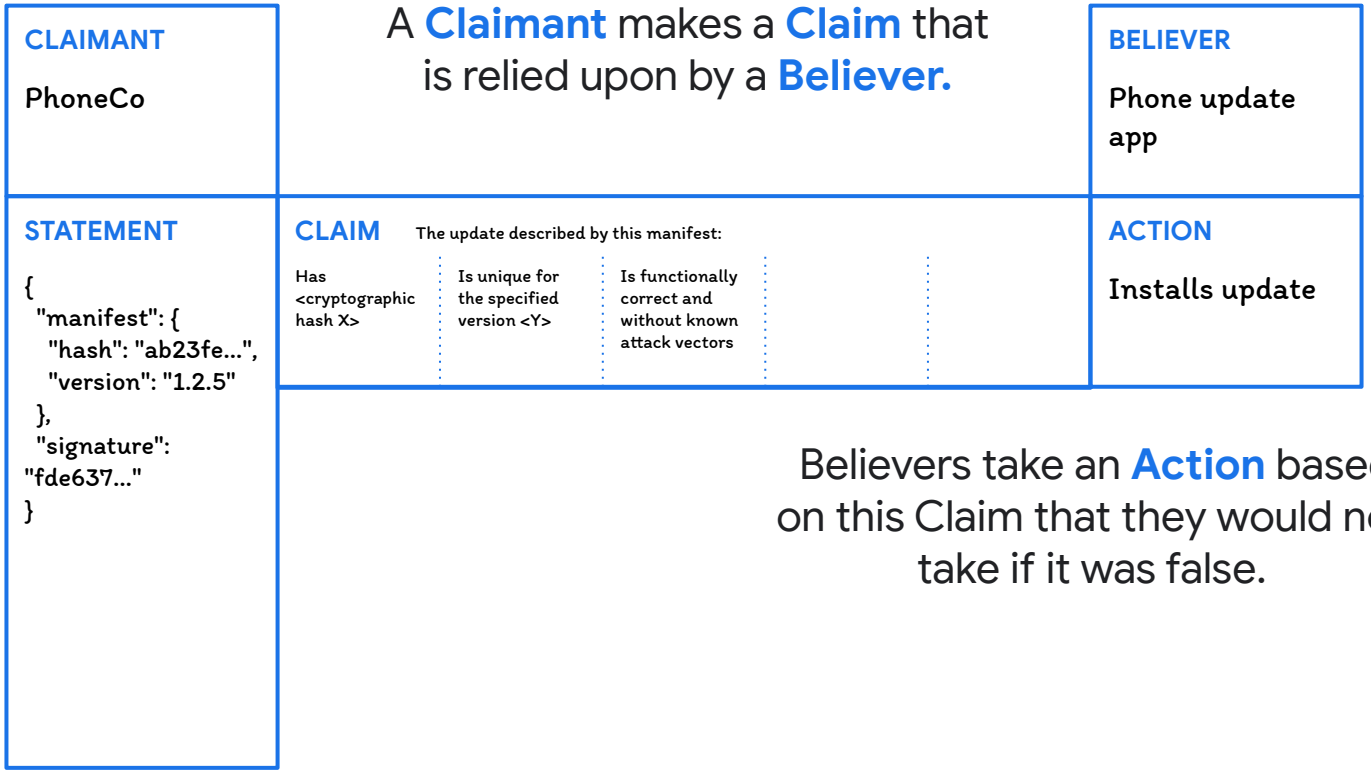
By signing and putting this data in a log, PhoneCo is making a **claim**...

The update described by this manifest:

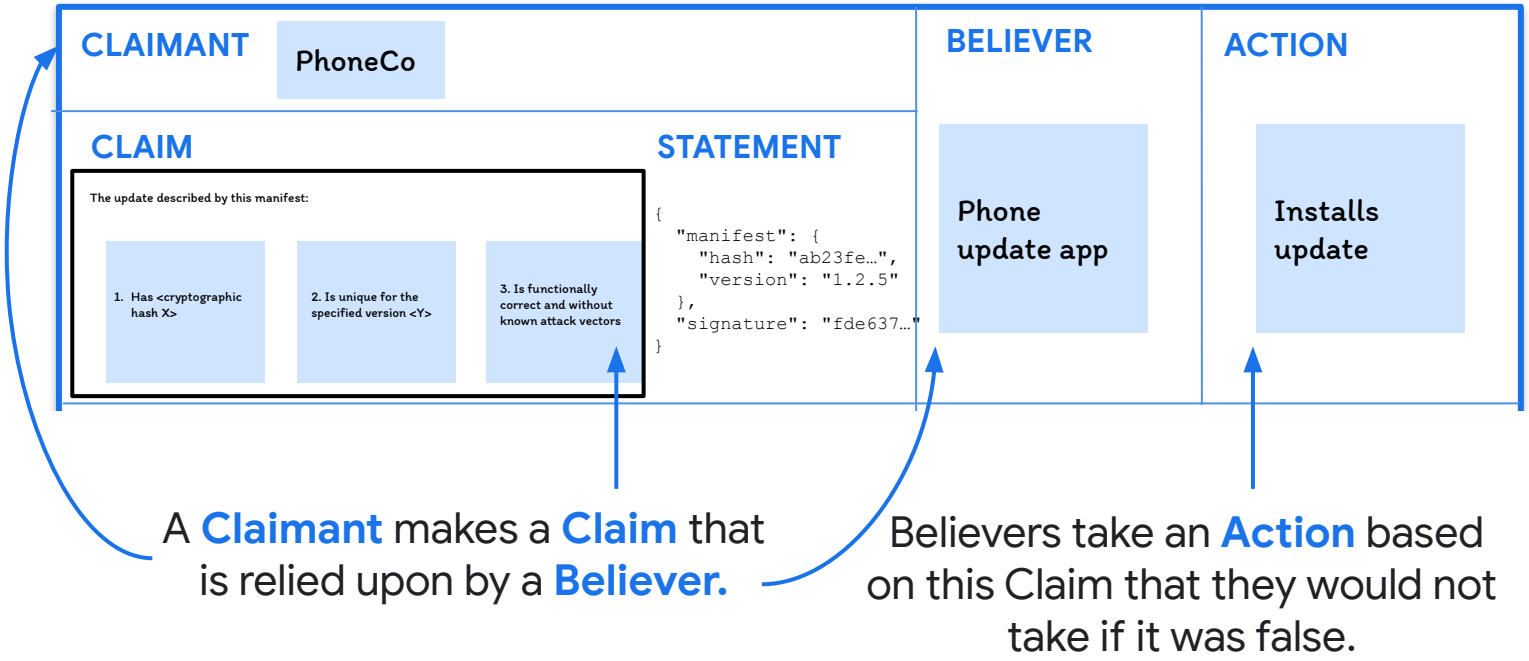
1. Has <cryptographic hash X>
2. Is unique for the specified version <Y>
3. Is functionally correct and without known attack vectors

... that the phone update app **believes** is true in order to perform an **action**, install an update.

Now we can fill in some of the canvas:



Now we can fill in some of the canvas:

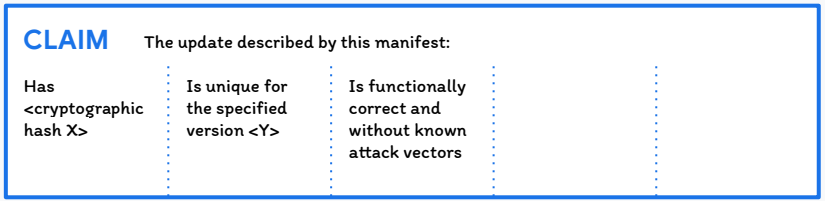


At this point, PhoneCo are putting things in a log, but nothing else.

This isn’t enough to solve PhoneCo’s problem. If their private keys were stolen, a malicious actor could still release a bad update and publish a fake manifest to the log.

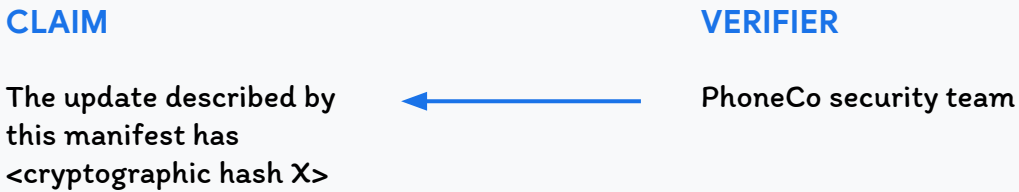
In order to rely on the data in the log, everything must be verified. We need to consider who can verify the claims in the log.

In this case, there are several parts to each claim. Each part needs to be **verified**.



For each part of the claim, PhoneCo asked themselves who is in an authoritative position to verify that part.

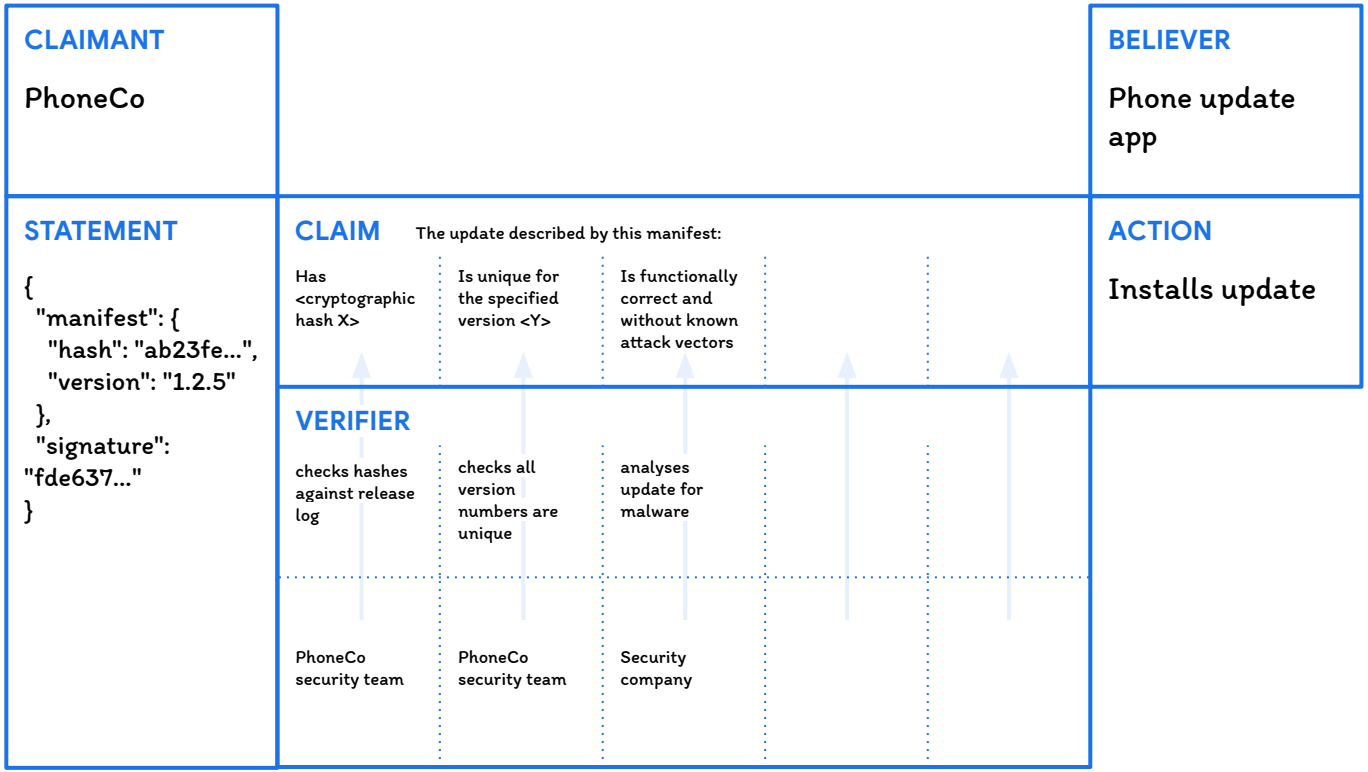
Only PhoneCo can authoritatively say if a particular hash in a log entry was a genuine release.



Anyone could check whether a version number appeared in multiple log entries. PhoneCo's security team takes on this responsibility.

A third-party security company could analyse each software update for malicious code and attack vectors.

We can now add the verifiers to the canvas:



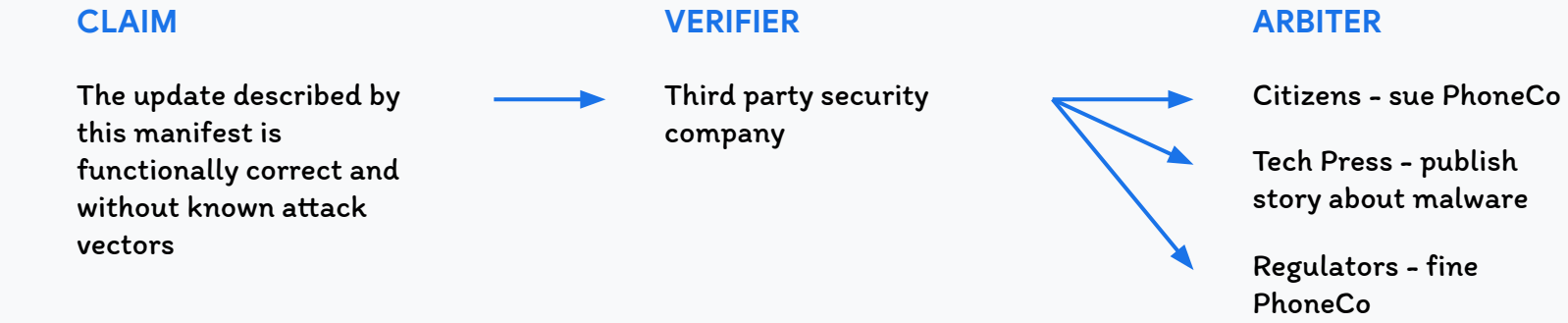
Finally, PhoneCo thinks about what action should be taken if verifiers discover a malicious entry.

Without this, there are no consequences of malicious actions. It still doesn't solve PhoneCo’s problem as a malicious manifest may be detected but would still be installed.

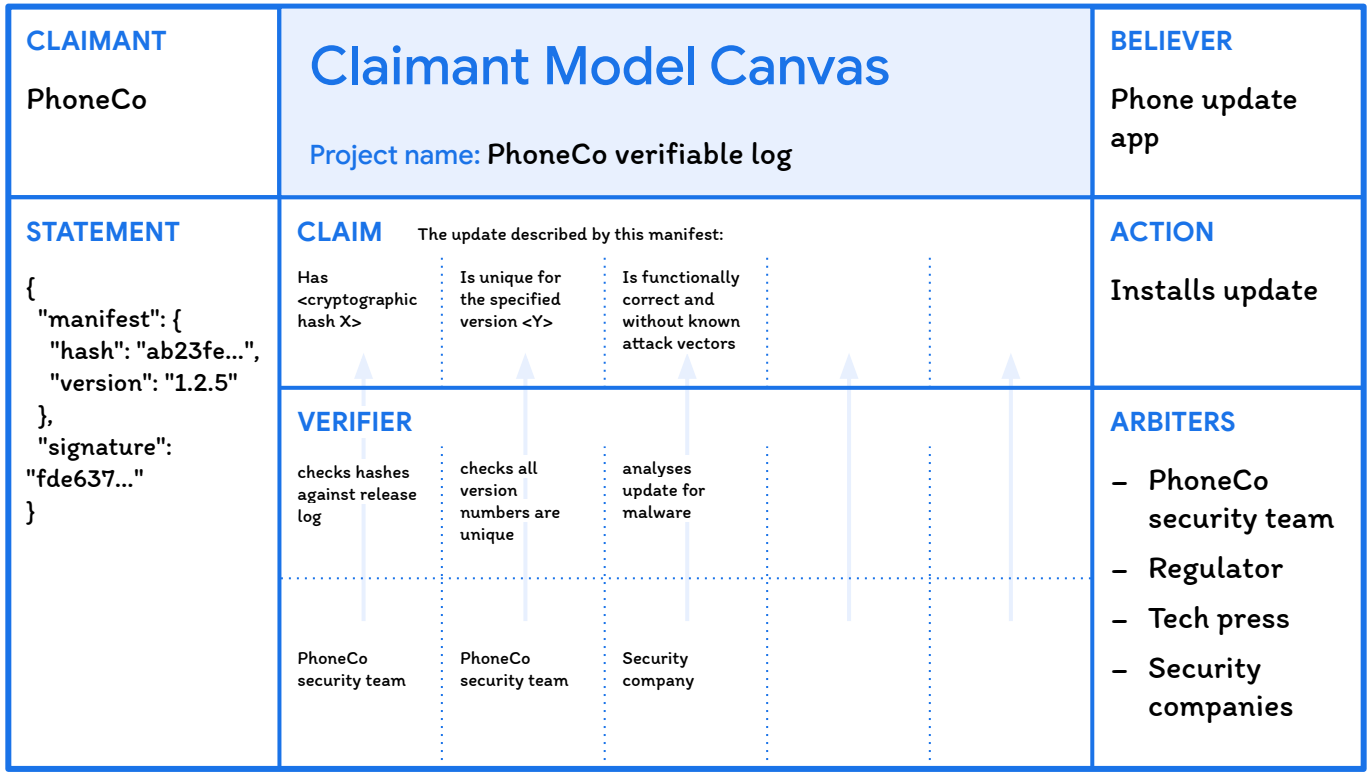
If PhoneCo's security team spots a manifest with an unrecognised hash, they assume their private key has been stolen, and rotate their keys.

If PhoneCo's security team spots a duplicated version number, they assume their private key has been stolen, and rotate their keys.

If a third-party security company found malware in PhoneCo's update, they could tell the tech press and regulators.



The canvas is now complete



All software updates are logged, every manifest is verified continually, and there’s a plan for verifiers to take action if they discover a malicious manifest.

Now, if PhoneCo’s private key was stolen, a malicious update would be detected since all manifests are continually verified.

Use this process to document your own system using the **Claimant Model Canvas.**

TrustFabric

Claimant Model Canvas

Project name:

CLAIMANT

BELIEVER

STATEMENT

CLAIM

ACTION

VERIFIER

ARBITERS