Exploring the discussion about what it is we can sign. These options are not necessarily exclusive. This is very rough. The relation to TUF and in-toto are quite hand wavey as they omit all the details of the other things those protocols do.

TODO (jc) read the literature on semantics of digital signatures. Well, maybe not all of it.

"Git model"

I call this the git model as it corresponds to signing on a git repo.

- Signature on a blob. This is equivalent to signing a git commit. This does not say what the blob is, just that the signer (by convention) authored it, or did some checking of it when adding it. Obviously you have to check which signatures you accept (out of band), for example with git check is this person a project maintainer. This could be done in a registry by having a signature object pointing at the blob, but we might want to optimise the storage.
- Signature on a tag. This is equivalent to signed tags in git. This is currently not easy to do in a registry, as there is no way to attach labels or metadata to tags, but this can be changed (Derek has some ideas). Note that multiple tags pointing at the same content can be signed, and tags may be updated if the registry allows, but they have to be re-signed in this case.

In both these cases, you need to understand whose signature it is, and whether you trust it. In addition, there is no real information as to what the object is. For example you can do a substitution attack, in the same way as you could exchange two git repos with the same contributors, as the commit signatures would still be valid. Discussing these models with NYU as they have done a lot of work on git commit signing. Overall this model is quite simple to understand and relatively efficient.

This very roughly corresponds to what Notary v1 does in part, in that Notary really just attests names, although it has other features on top like delegation. It works best if you are in control of the signatures, and know what the root key should be. TUF has a mirroring model that Notary does not implement. TODO (jc) write up mirroring models and relate, as this is partly what is confusing about this model.

"Content model"

The idea is to make signatures about what the content actually is. So we sign something that has some sort of BOM in the metadata being signed. If the object describes that it is version 3.06 of MySQL, and it is, say, signed by the MySQL maintainers (as determined by some mechanism, eg further attestation or well known signature etc), then we know what we have got.

The advantage of this is that you know what you are getting, wherever it comes from. The disadvantage is that checking is much more complex. First you need to check the signature. Then you need to decide if the key is the right key for the data it validates, for example is this actually the valid key for MySQL. Then you have to check the data matches the version you expected. This makes for a more complex client.

"Claims model"

In this model a signature attests not just to the content but to a set of claims, so you sign the claims and the content. You may use multiple signatures with different claims. For example a signature might attest that this image has been through CI and passed its tests. It could also validate that someone looked at it and attest that it is MySQL version 3.1, the someone could be the MySQL maintainers or someone else.

Very roughly this is the kind of claim that in-toto uses signatures for, basically signatures over process steps. This is something that clients have asked me for, or even assumed Notary provides.

Example workflows

TODO. Scanning, checking SBOM etc. What could happen where.