Nate's Dogma

A. Introduction

I'm collectively referring to this set of ideas as my "dogma". The word was chosen carefully, mostly from molecular biology(e.g. central dogma, Anfinsen's, etc.), but also as strong, controversial opinions intended to spur discussion.

This is presented because I feel our incrementalist approach is leading to less functionality and greater complexity than new, highly centralized identity systems offer. Most of the ideas could be pursued independently, but the dogma encapsulates what I believe must be fulfilled in order for distributed identity systems to compete successfully with highly centralized systems.

This would not require any organization to use a distributed model; similarly, it would not force a centralized model. Instead, it specifies a distributed identity *data* model. This leaves the actual flows and processing to happen in diverse ways, including through a centralized system that is itself a consumer of the distributed data. Done right, this will also give deployers the flexibility to pick and choose which pieces of their identity infrastructure they would manage on a granularity finer than "IdP" or "SP", a transition foreshadowed by the attribute authority and external identity sources.

This approach depending on resolution of many things risks introducing complexity and fragility, making efficient protocol design and caching crucial, as well as reinforcing the value of aggregators of data, such as federations.

These are the same ideas that underpinned the identity vector concept, but that munged a proposed solution with a position statement.

B. Nate's Starter Dogma

1. Provisioning and SSO

Provisioning and SSO are special cases of the same general problem, but we have solved them independently and accidentally introduced artificial distinctions. This is evidenced by overwhelming commonalities, such as "front-channel provisioning" being part of SSO, privacy and security considerations being invariant, the attributes being sent through provisioning needing to match somehow whatever the user shows up with, and deprovisioning being a shared challenge.

I want one name for each attribute, and I want one system where I can write rules about where my attributes go and audit them. "Bulk" provisioning and "authenticated user as delivery vector" are the two special cases.

Problem statement: We don't really have a provisioning service implemented yet anyway, and maintaining a separate provisioning implementation is arguably far more work for both implementers and deployers than retrofitting our SSO implementation into a multi-purpose implementation.

Problem statement: I don't think we can cleanly address common problems like deprovisioning and privacy and security rules when identity data release can happen in many ways.

Problem statement: Other solutions are beginning to offer unified provisioning and SSO solutions, and this has been the business model for some of the more successful identity startups since time immemorial. We are already behind in functionality.

2. Redesigning Metadata

Metadata is intended as absolute truth, but it works only for a subset of our problems. The most common approach to discovery by large sites(`domain` -> `IdP`) is poorly compatible with the way metadata is structured (`IdP` -> `domains`). Attribute requirements are notoriously difficult to express, making out of band communication necessary anyway. Representing provisioning requirements in metadata hasn't been considered.

Dynamic metadata has been difficult to get done, and that's because we're trying to build `DNSSEC` + `IPsec` rather than `TLS`. Don't paint the entire universe: paint specific, known truths. Rather than making metadata dynamic, make our identity systems more dynamic than metadata. Every entity name must be a URL that is resolveable into a description of the entity and how to talk with it.

The content of metadata needs to be replaced or amended to be both more informative(provisioning, attribute, and consent stuff) and less

informative(deprecate lightly used data fields). It also needs the right primary index(probably domain, because that's the more common bootstrap for discovery) and secondary indices.

Rather than trying to solve the distributed signature problem, I want the replacement for metadata to include pointers to attesting authorities. A provider could proceed to query a mutually trusted authority to verify the good standing of a recipient. This also allows a single entity to point to multiple authorities, something that is a challenge with signatures over files. I also don't have to magically know your trusted authorities: you tell me who they are. Mutual TLS authentication to both the recipient and the authority gives cryptographic security to this chain of relays.

This is a provider-centric alternative to the federation-centric being pursued today.

Problem statement: Distributed signature is one of the most challenging problems I know of. Do you hand out keys with no control over file contents? Do you sign files and give them to people, basically reproducing the CA model at a second layer? I'm not convinced it even offers a superior solution. Pointers and queries are vastly easier in practice. They don't fix the problem, as the authority in the response still has to be meaningful to the asker, but it does add the ability to build chains of authority and removes the need for a single MDA to act as an oracle.

Problem statement: Rekeying with your key scattered all over the place is brutally painful. Federations are supposed to alleviate this, but wouldn't distributed rekeying be alleviated by just querying the server and then validating the credentials?

Problem statement: Being able to definitively say that an entity doesn't exist is effectively impossible, while reconciliation of different answers from different authorities is effectively unspecified.

Problem statement: Most large SP's that are in heavy use by academia (Office 365, Google Apps, and Box being the elephants in the room) do discovery by domain, and so almost definitionally can't use our metadata approach for perhaps the most important function.

Problem statement: Metadata gives no cues about why attributes are requested, why they're needed in terms both administrators and consent users could interpret, what would happen if they are not received, what provisioning requirements exist, and basically everything else I had to write into the service-specific guidances for NET+.

Problem statement: We have to agree on authorities that we trust out of band. There is no way to discover from an entity which authorities vouch for it. This constraint has led to everything getting herded into one gigantic aggregation service where we there are trust and compatibility issues.

Problem statement: I have to ensure that all my metadata everywhere is the same. This means I either force all my partners through a single federation, or I have to keep all representations of me consistent, or I risk incoherent provider behavior. Allowing multiple directly expressed relationships relieves these restraints somewhat, and it allows a provider to seek simply a connection that meets attribute and trust requirements rather than ensuring that all metadata everywhere is the same.

3. Modularity and Composability

We have bundled functionality into reasonably natural categories we call "identity provider" and "service provider", or less commonly, "relying party". Other identity systems are blurring these boundaries, particularly by becoming larger and more centralized. An entity will typically act as either an identity provider or a service provider in a transaction today, though some entities play both roles at times, demonstrating that composition of functionality is feasible. Further decomposition of functionality, by contrast, is generally poorly specified.

I want to move with deliberacy towards distribution: rather than smelting everything into monoliths, I want to make each step of a federated flow and the data it needs as portable and composable as possible. Striking the right balance between flexibility and complexity is crucial to the success of distributed systems over central systems, and if it's done right, a monolith becomes a special case of a portable system.

Perhaps it'd be natural to first cleave authentication and attributes completely, or reduce the need for the heavy lifting of federated identity to be done by each end SP, a goal shared by more and more services and reflected in new implementations in other sectors. I don't know.

I don't want to build an opaque system or gateway; I want to build a distributed platform. The rules and the data upon which any transformation engine operates must be published and reusable where possible, giving maximal flexibility and visibility to the processing pipe. Tools should be able to parse such rules so that it becomes possible to evaluate, end to end, whether the identity connection necessary for use of a given service can be made using attributes and authorities that are already known. Missing pieces can be flagged and options suggested.

Problem statement: We shouldn't attempt to compete with large corporate entities when it comes to building centralized platforms or products. That's what they do, and they do it very well. We need to design to our strengths.

Problem statement: Much identity data is opaque or in random formats and there are tens of protocols in use, making it borderline impossible to build one suite that handles all of it, much less does so elegantly.

Problem statement: We've talked for a long time about external authentication sources and how to build SP shims or gateways. It's time to get serious about making that a viable architectural approach, because people apparently really want to use it in deployments no matter our advice.

Problem statement: Our IdP/SP architecture is increasingly dated and it is being used less and less by other deployments, which has already led to interoperability issues.

Problem statement: Large integrated services that do everything are hard to customize and are generally opaque, each of which can tempt deployers to build hacks which eventually lead to a less stable architecture than one designed for flexibility at the outset.

4. Attributes

Attributes are given different names and values in different protocols, but they are intended to be the same data with the same semantics. The protocol can't take precedence over the data. Every attribute must be named with a URL that is resolveable into a schematic description of the attribute and how it should be treated, and this definition must be as protocol-neutral as possible.

We need a way to express equivalencies without universal consensus. URL-based attribute names plus a schema/equivalency language would make nuanced processing possible upon receipt of a mysterious attribute. This ability would be useful for "attribute families", such as `first_name`, `givenName`, `cn`, `firstname`; it would also be useful for internationalization, or for defining attributes as compositions of other attributes. It could also allow, for example, an organization to flag that its edition of `eduPersonPrincipalName` is non-reassignable, or that a particular attribute is informative rather than trusted data.

Attributes, like entities, are "owned" by URL. Standards organizations should host their own attribute definitions, while para-standards organizations should host umbrella definitions and comparisons.

Consensus is paramount but not strictly necessary as long as a chain of transformations and references can be walked to arrive at the data you need from the data you have. This is not introducing a problem, but instead recognizing a truth: it is a running battle to get "your" attributes used by others, and static standards have not solved the problem alone. This is a degree of freedom to accommodate that reality. Anyone choosing to use an attribute in its own little world faces the same ramifications of the decision.

Problem statement: If I get a random attribute named with a URN or a string, I have no idea where to even begin interpreting it other than asking the IdP out of band.

Problem statement: There are many roughly equivalent attributes and an SP has to ask for all variations on those attributes right now and the IdP must guess the intent of the question rather than asking for “something in the given name category as defined by REFEDS”.

Problem statement: An SP has no way to evaluate the quality of data received. For example, dealing with mail as an attribute: sometimes used as an identifier, sometimes institutionally populated, and sometimes user populated. That makes it very difficult for an SP to know how to use or trust the received data. A schema language would allow for nuanced interpretation and communication of attributes, preferably paired with simplistic flags for specific attributes to indicate how to interpret a commonly named attribute with varied semantics, such as "this mail attribute value is informative at best, even though I have some verified mail attribute values too".

Problem statement: Attributes, even those defined by a singular standards body, are named differently in different protocols. Only someone familiar with identity will be able to understand this and trace this.

Problem statement: Attributes are effectively non-extensible right now. Allowing organizations to host “editions” of attributes with additional bits and pointers back to the authoritative reference would permit this. Whether this is good or not is debatable, but I think history has shown that it tends to happen no matter what, and doing it by design rather than by circumstance is better.

5. Security Model

I like the use of HTTP and HTTP error codes for communication such as proposed by MDQ and CIFER. However, the tandem use of XML signature for success and a completely different mechanism like HTTP error codes for failure leads to challenging and ambiguous implementation and interpretation. Error conditions and success conditions must use the same trust mechanism. That mechanism should be TLS whenever possible because it’s computationally cheap and attackers would fail the handshake at the outset, which considerably reduces the threat surface for MITM, denial of service, cache poisoning, and other attacks. It would also enable unambiguous response, fail, succeed, or other conditions.

Problem statement: Our implementations typically depend on a heavily layered set of protocols, while the interpretation of these protocols is much less clear. Attempts have been made at tying things together with concepts like channel binding, but those address small segments of the problem and they’re hard. It seems much easier to flatten our stack.

Problem statement: Exposing run-time services to the world is a dangerous proposition and I want a layer of protection that is very hard for deployers to screw up. XML signature is particularly disproportionately computationally expensive, and thus risky to expose.

Problem statement: The browser CA trust model stinks, and there have been challenges with a TLS implementation that rhymes with broken. However, the specification itself is sound, and even a failed handshake should indicate the full contents of the certificate, which would allow the CA to be identified. If we think hard about it, I suspect we can make it into a more flexible model and leverage it more heavily.

Problem statement: Metadata aggregates contain an implicit “if they ain’t listed, we don’t know ‘em”. Dynamic metadata doesn’t have this implicit interpretation. What does it mean when an authority says “no”? How does a provider distinguish between an unrecognized provider, one that doesn’t even exist, one that changed authorities, and one that has been compromised? Does the provider need to be able to do this?

Problem statement: XML signature, as well as crypto in general, is hard, and implementers botch it all the time. We need to make implementation easier because we’re apparently not going to stop people from rolling their own code bases or using lightly maintained releases.

6. `scopedUserKey`

There must be a way for an application to refer to a specific federated user record at an identity source that has been used with that application. This is necessary in order to reasonably handle deprovisioning, merges, status checks and data updates — especially for native clients that perform logins sporadically — identity portability, and so forth. An identity source must be able to dereference this key for a predefined, very extended period of time. A basic scoped key called `scopedUserKey` modeled after `eduPersonTargetedID` of the form `opacity@idp` is proposed with some rudimentary protocol-neutral examples as a way to do this below.

The scope of the recipients of the key and the structure of the key are left deliberately undefined. The only entity that should care about the value of the `opacity` is the IdP itself. This lets the IdP select how broadly each key is used(one service, one set of services, whole world, or even a unique value for every transaction like a transient identifier so long as the IdP was willing to persist it), whether to encrypt and imbed values or use serial numbers or just use UUID's, etc., independently and on a case-by-case basis.

Problem statement: An SP has no guaranteed way to persistently refer back to a user at an IdP. This was originally intentional for privacy

reasons, but it has turned deprovisioning into a nightmare, which is arguably far worse privacy in and of itself. If an SP can expect to query an IdP persistently about the same key, then the problem becomes vastly more tractable. An IdP that still wishes to provide the privacy of a truly one-off, `transientId` style key can still do so, but at the cost of being able to dereference all of those keys for a long time.

Problem statement: SP's have real challenges dealing with the huge variety in identifiers and semantics associated with those identifiers. I don't think we can unify the world under a single identifier, so instead I want to have a key that creates portability and flexibility in identifier choice.

Problem statement: An SP that wants to repeatedly query an IdP about user status has no consistent way to do so. This has come up repeatedly in contexts from authorization through native client support.

Problem statement: There is no really wide-scale, sophisticated approach to identity portability, attribute aggregation, or delegation available. They have been use cases since day 2. It's easy to come up with extremely ornate solutions here that do a lot more than this, but this is intended to be the most flexible and braindead approach possible.

C. scopedUserKey examples

I'll use the provisional attribute name `https://idm.standard/attributes/scopedUserKey` and won't bother with real URL encoding or valid HTTP or anything.

There are three separate pieces that I would build:

> **A)** A definition for the attribute: `scopedUserKey`, multivalued, each value of the form `key@idp`. Each would mean "this user, known to you by this key, has a record persisted here, and you can query me about it using that key". This would leave a lot unspecified, but would definitionally exclude reassignment of any form and the `idp` would be expected to always be able to pull a current or deleted record from the `key`. This is basically a clone of `persistentId` with slashed NameQualifiers and semantics, or `eduPersonTargetedId` with scopes.
>
> **B)** An endpoint services could query for updated user data, answering to the form `scopedUserKey@idp`.
>
> **C)** An endpoint to allow services to request temporary, centrally brokered access tokens to use in direct queries between them.

1. Identity Persistence

> A service provider issues a query of the `@idp` using this key. The IdP is obliged to respond with a referral, fresh user data, or hypothetically never but probably occasionally, "error".
>
> The `scopedUserKey` could be used on an ongoing basis by the recipient to poll for updates, changes, or renamings, or to move a user from one provider to another. There are probably other creative uses. Responses are basically the moral equivalents of HTTP 200, 301, 410, and 404 over mutually authenticated TLS.
>
> > `200`: "Yes, I know that user, and here's what I know about them."
> > `301`: "Yes, I knew that user, but go look here for a new key because they were merged/the `scopedUserKey` was compromised/they moved to this other IdP."
> > `410`: "That user did exist, but that user is gone. You should deprovision them."
> > `404`: "You asked about a `scopedUserKey` that, as far as I can tell, never existed. Call me."

```
GET
https://idm.ndk.local/idp/profile/AttributeQuery?https://idm.standard/attributes/scope
dUserKey=ServiceAKnowsUserByThisKey@https://idm.ndk.local/idp

200 OK
https://idm.standard/attributes/eduPersonPrincipalName=ndk@ndk.local
https://idm.standard/attributes/eduPersonScopedAffiliation=student@ndk.local


GET
https://idm.ndk.local/idp/profile/AttributeQuery?https://idm.standard/attributes/scope
dUserKey=ServiceAKnowsUserByThisKey@https://idm.ndk.local/idp

301 Moved Permanently
https://idm.standard/attributes/scopedUserKey=ButTheUsersRealNewKeyNameIsShirley@https
://idm.foobar.overthere/idp
```

2. Aggregation

A very lightly pondered way to use the `scopedUserKey` for consentful attribute aggregation between two services:

1) User goes to service provider A, logs into service provider A using their IdP. Their IdP sends attributes. One of those attributes is:

> https://idm.standard/attributes/scopedUserKey = ThisIsRandomOrMaybeNotAndTargetedOrMaybeNotAndStructuredOrMaybeNot@https://idp.ndk.local/idp

2) At some point, the user indicates to service provider A that they want to grab information from service provider B using `ndk.local`'s linking service. To do this, we can leverage the user's existing SSO sessions to make a quick and mostly invisible hop. This hop would be a redirect to `ndk.local`'s identifier linking service.

> 302 Redirect
> https://idm.ndk.local/idp/profile/expressKey?aa=https://service.provider.B/entityID&https://idm.standard/attributes/scopedUserKey=ServiceAKnowsUserByThisKey@https://idm.ndk.local/idp

3) The IdP presents the user with the details of the requested connection to service provider B and asks for the user's consent
4) Upon consent, a `scopedUserKey` token is generated by the IdP of the form

> https://idm.standard/attributes/scopedUserKey = EncryptedBlob@https://service.provider.B/entityID

The encrypted blob contains the user's key shared between the IdP and service provider B and a current timestamp. It would be encrypted with service provider B's public key and something like AES, but the algorithm should be signaled in metadata. The user gets sent back to `service.provider.A`.

> 302 Redirect
> https://service.provider.A/Shibboleth.sso/SAML2/POST?https://idm.standard/attributes/scopedUserKey=EncryptedBlob@https://service.provider.B/entityID

5) The user is redirected to service provider A with their one-time correlation token in hand, and they deliver it for use. Service provider A makes a query of service provider B using client TLS to authenticate both service providers and the one-time correlation token to authorize this request. Service provider B would decrypt the encrypted element in the token, check the timestamp for recentness, and recover the pairwise identifier between service provider B and the IdP.

> GET
> https://service.provider.A/idp/profile/AttributeQuery?https://idm.standard/attributes/scopedUserKey=EncryptedBlob@https://service.provider.B/entityID

6) Service provider B dereferences the pairwise identifier into who they know the user to be. Data is gathered and returned to service provider A.

> 200 OK
> https://lalala.org/attributes/studentrecord/data=whatever

So, from user's perspective:

1) I'm logged in through my IdP at service provider A and I want to get my data from service provider B, so I click the link button
2) I'm presented with a consent dialogue asking if I want to permit this linkage.
3) I click yes, and I get redirected to service provider A, which then magically gets the information from service provider B

From service provider A's perspective:

1) My authenticated user wants to link records with service provider B, so I'll craft a linking request for the user to deliver to the IdP, and redirect the user off
2) Yay, my user is back, and this time with a one-time use linking token
3) I issue a query to service provider B, presenting my certificate as proof of my identity, and validating their certificate as well. I present the one-time use linking token in this query.
4) Service provider B returns me a data set. Yay.
5) User, back over to you...

From service provider B's perspective:

1) Service provider A just connected to me in a trusted way and delivered a one-time linking token
2) I'll decrypt the token, check the timestamp to make sure it's reasonably recent, and map the received `scopedUserKey` to a principal
3) Looks legit. I'll send a response back to service provider A with the data they requested.

D. Nate's Gambit

We either go fully distributed and try to build something that the world has not yet seen, or we try to recapitulate the monolithic model that has begun to emerge in the corporate sector. I believe our intermediate path will lead to us getting pigeonholed into a small segment of identity management, and that the distributed solution is inherently better, so we should at least try.

Accepting the gambit means going unabashedly distributed. Declining means building a more centralized platform. Doing neither risks defaulting to becoming an increasingly niche player.