Upcoming ARPA2 Projects

(October 29, 2017)

> *This document describes upcoming work that we intend to do as part of our second phase, IdentityHub. Services can prepare for ARPA2 integration by linking in these ARPA2 facilities. Through that, the benefits of control over identity would become available to those services.*

The IdentityHub is our perceived platform for identity management in our domain management platform. In our vision, the empowerment of the hosting industry is paramount in giving end users more control over their online identity, and regain a first-class citizen status.

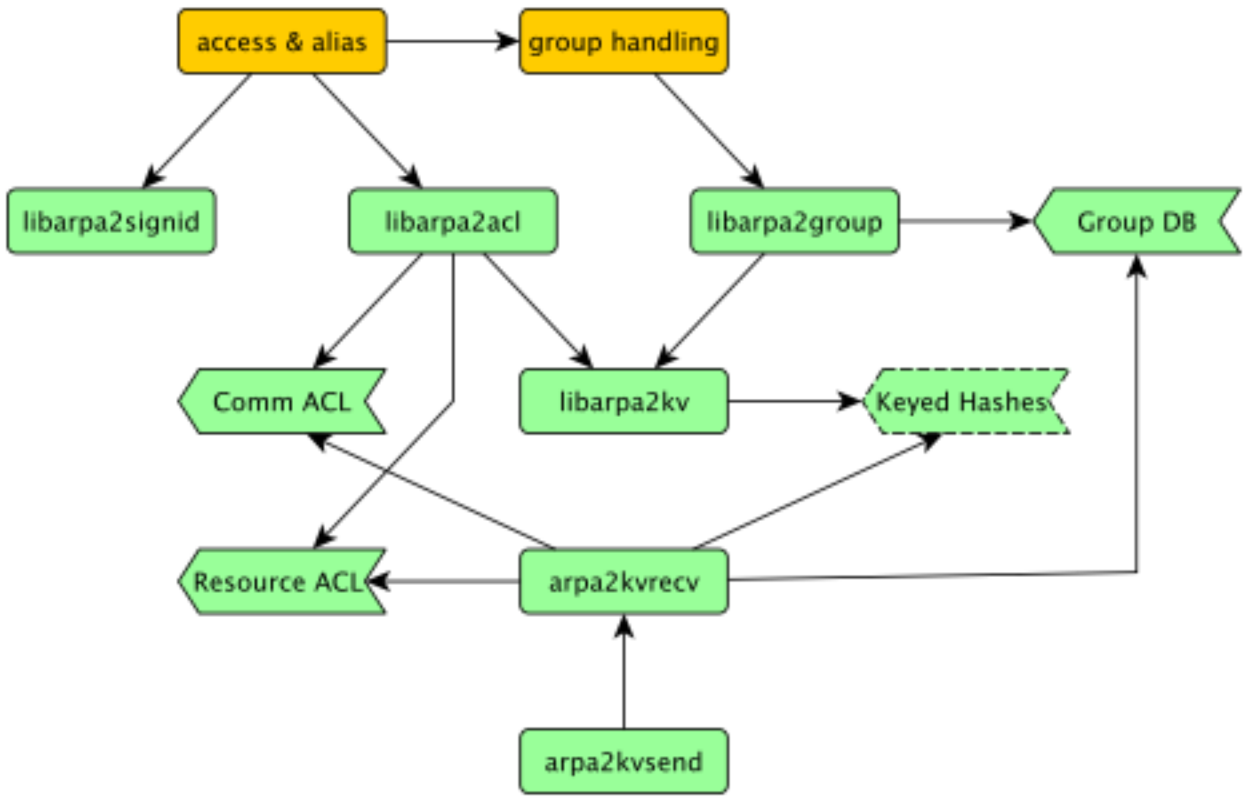
In our proposed architecture, hosting is conceptually split into two kinds, each of which can profile themselves through distinguishing factors as price, storage space and reliability of services and storage. One kind of hosting deals with identity, and being able to plugin a variety of services; the other kind of hosting provides specific plugin services. This enables users to choose open protocols which are hosted locally; massive service providers remain a valid choice to make for one's own domain, but there should be no put pressure on others that wish to communicate with it; open protocols provide a way out of that.

Mostly Generic Facilitation

Most of what is described below, is generic in nature. This is very good news; it means that services can share a broad basis of code. Indeed, the identity facilitation has always been designed without too much specificity of protocols in mind.

In the diagrams below, yellow indicates application-specific code portions, and green indicates support that is largely generic, with possibly a few parameters that vary behavioural classes for services.

- `libarpa2signid` deals with addresses that have built-in signatures to constrain message content
- `libarpa2acl` deals with access control lists, and finds aliases that match with it
- `libarpa2group` deals with group and role logic, including member/occupant addresses and their mapping to delivery addresses
- `libarpa2kv` deals with key/value databases with light-weight encryption, and `arpa2kvsend` with `arpa2kvrecv` deal with exchanging them
- `Comm ACL` lists ACL entries for Communication Services (such as email, chat, telephony)
- `Resource ACL` lists ACL entries for Resource Services (such as websites, or directories)
- `Group DB` lists group members / role occupants



Library: Encrypted Key/Value Lookup

We need a library `libarpa2kv` in several programming languages (at least C, Python and Go) for looking up key/value mappings in a database (at least, BerkeleyDB).

The key and value are encrypted, so as to make the information available on a need-to-know basis only. This is meant to stop invasions on privacy, as well as limit the use for crackers stealing these data sets. Imagine the damage done when contact data is stolen — spam could not only be sent to you, but also on behalf of someone you know!

This library fulfils a number of recurring tasks for our platform:

- Receive prepared keyed-hash contexts from another process
- Find an encrypted index key based on a prepared keyed-hash context
- Incorporate an application-specific string into the index key
- Incorporate lookup data into the index key
- Derive an encryption key from the keyed hash derivation of the index key
- Support iteration over variations of lookup keys that may occur (such as, on an ACL) using an efficient technique of cloned keyed-hash contexts
- Ignore initial 32 bits in retrieved database values
- Decrypt values retrieved from the database
- Validate message authentication codes on retrieved database values
- Tolerate mixed database content (implied by application-specific string and index key size)

Future versions can improve the scalability of the mechanisms dramatically through:

- Bloom filters to quickly see if an entry is present
- Clustered storage, where N out of M nodes hold a key/value mapping
- Possibly a demand-driven retrieval method using Diameter
- Distributed assignment of IdentityHub upstreams over nodes

Library: Access Control & Aliases

We need a library `libarpa2acl` in several programming languages (at least C, Python and Go) for manipulating identities, including aliases and access control lists for each of them.

This library fulfils a number of recurring tasks for our platform:

- Functionality for handling DoNAI and DoNAI Selector concepts, including matching and iteration.
- Matching entries against the ACLs for various aliases and selecting one while at it, based on `libarpa2kv`
- Checking if a considered / requested alias from external sources is welcomed by the ACLs

Library: Signed Identities

We need a library `libarpa2sigid` in several programming languages (at least C, Python and Go) for processing identities containing (light-weight) signatures.

These libraries fulfil a number of recurring tasks for our platform:

- Testing for full `+` and stripped `++` forms of signed identity in a DoNAI's local part
- Stripping of the signature and result in a static identity (with signing flags) that can be used in ACLs
- Generation of signed identities from a field holding flags and in interaction with the context of a particular service
- Verification of signed identities against a field holding flags and in interaction with the context of a particular service
- Some mechanism is needed for setting up a static key used to generate symmetric-key signatures in the signature of a signed identity
- This is a stand-alone mechanism that needs to be fed with protocol-specific information from the calling context (which usually is a service)

Library: Service Identities

TODO: We MAY need a library `libarpa2service` to deal with local parts that start with a `+` symbol. On the other hand, handling those may also be service-specific. Or is it something to route to a local handling party? This will probably not become clear before we definately entire into ServiceHub, as part of our third phase.

For now, we shall treat service identities as local to a service. In the future, we may decide to forward service requests over AMQP 1.0, to a central switchboard in the identity host, which may relay it to a subscribing service node. While registering a service with the ServiceHub, a "prefix" in the service identity space may then be configured. If anything, this library would wrap around generic message handling aspects (if that surpasses AMQP 1.0 at all) and make it easy to link to service-specific interfaces, such as LMTP delivery addresses.

Note that an identity may be both a service and a signed identity. The signed identity is then applied, as is the ACL, without realising the identity represents a service. Only when an accepted communication attempt is relayed to a handling place will it matter if it is a service.

Library: Groups and Roles

We need a library `libarpa2group` in several programming languages (at least C, Python and Go) for manipulating groups and roles.

This library fulfils a number of recurring tasks for our platform:

- Assumption to operate after the library for access control & aliases
- Iteration over members for communication services, based on `libarpa2kv`
- Shared access control for resource services, based on `libarpa2kv`
- Shielding off the identities of members/occupants

Tooling: Encrypted Key/Value Databases

We need tools `arpa2kvsend` and `arpa2kvrecv` in several programming languages (at least C) for encryption and for decryption of key/value databases, and reliably transporting them over SteamWorks.

The encryption tool fulfils the following recurring tasks for our platform:

- Support of SyncRepl for (authenticated) clients
- Encryption for key/value databases with public-key cryptography to share a session secret for a subset of the key/value mappings in the database
- Employing this encryption on-the-fly while traffic of certain attributes passes through over LDAP
- This will most likely be based on LillyDAP and sit in the traffic flow from SteamWorks Crank
- framework

The decryption tool fulfils the following recurring tasks for our platform:

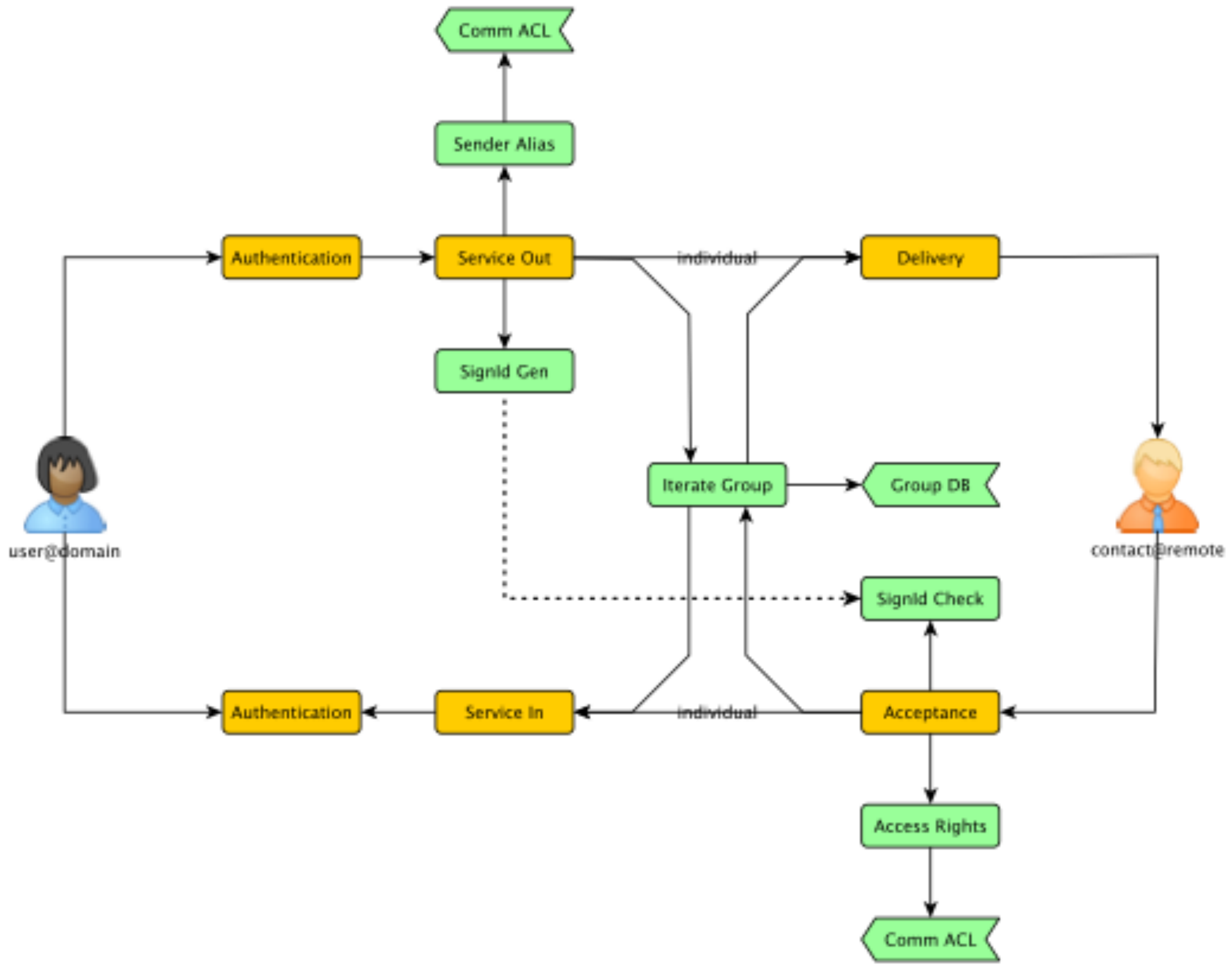
- Subscription to key/value databases via SteamWorks, notably a backend to a PulleyScript; subscription may be to an individual database or multiple that allow being combined

- Storage of subscribed, encrypted key/value mappings into local databases
- Decryption of private-key encrypted session keys, and their dissimination to server processes as prepared keyed-hashing context
- Decryption of key/value database entries with the keyed hashing context, based on service-presented contact information
- Insert a sender-specific value of 32 bits before each value inserted into the database
- Match this 32-bit sender-specific value before removing values from the database
- Offer a way to remove every database value that starts with a given 32-bit sender-specific value

Application Pattern: Communication Services

A communication service is an application pattern, where communication is passed into a service, and actively moved out over an open protocol to reach indicated recipients. Some of the traffic may be locally delivered, but that is usually just an optimisation pattern.

Communication services enforce the Communication ACL, where it is decided whether a remote party is welcomed to access a local user. Outward processing ensures using sender addresses that welcome the corresponding reply traffic.



Outward Processing

When a user passes data out through a communication service, the ACLs for its own identity are used to determine the best alias to use.

When no alias was originally provided, then it will now be added; when an alias was provided, then the Communication ACL serves as a verification that the recipient may actually respond. If not, interaction with the user may be needed, depending on the nature of the communication service.

Users may send addresses containing signatures, either using a previously used signature, or without a signature. The latter ends in `++` while the former only ends in a single `+` symbol. Pre-existing signatures are validated to ensure the possibility of responses; when provided, the signature is temporarily removed for ACL checking; an ACL can be used to match signed addresses ending in `++` but the ACL then remains static.

Users may pose as group members, either by presenting their own alias setup for use in a group, or by automatically selecting it as part of ACL processing. Usually, access to a group will be most concrete in the alias setup for group interaction.

When communicating data through a group, the sender identity may be replaced with the `group+member` address for the sender. Any descriptive user name and/or comments may also be replaced. All this is possible because the `group+member` address translates back to the user's own alias. For reasons of privacy it is vital to do this for both user-readable and machine-readable addresses, if those differ. Note that non-members may also be permitted access to a group; in this case, the sending user may not have a `group+member` address to use in protection of their identity. Also note that users may have multiple subscriptions to a group, each with a separate mapping, although that barely seems useful.

Outward processing of a group by a communication service involves iteration over member addresses, and matching each against the recipient address(es). When matched, traffic is forward to the member's delivery address, which is mentioned in the group iteration data. Each group member should receive at most one message through the group mechanism.

Outward processing of a role may be different from outward processing of a group, depending on the nature of a service. The idea is that any role occupant suffices to act on behalf of a role, whereas all members are usually involved as part of group communication.

Inward Processing

When an open protocol port receives data intended for a user, inward processing is enacted. Usually the intention is local delivery, but group and role processing may cause a loopback to the outward path. Note that we treat things like email forwarding are managed as group/role mechanisms. This ensures that the addresses are rewritten and appear to have come from the local environment, in support of symmetric paths for inward and outward traffic, and through that, better support for all sorts of traffic filters and origin validation.

As part of accepting inboud data, the Communication ACLs are queried for a matching entry. This entry should welcome the remote sender address for the intended address. Since an ACL expresses DoNAI Selector patterns, it can select specific senders or anyone at all. This allows us to detect how closely a sender is matched with the ACL, which may be helpful in limiting or redirecting access attempts; think of spam filtering.

When the inward receiving address ends in `+`, it is a signed address. This means that its signature is removed to make it end in `++` for matching with the Communication ACL. This removal is usually temporary, intended for ACL matching alone, though services may independently choose otherwise. It stands to reason that the signature must be validated as part of the access control process. Whether that happens before or after checking the Communication ACL depends on the service. Ideally, success or failure are reported without distinction between the two access requirements but, once more, this may vary for the pragmatics of a service protocol. **TODO:** We may want to put specific signed addresses on the black list in the Communication ACL, so as to block abusive patterns. Alternatively, we may add blockage in a specific blacklisting mechanism for signed address ingredients, where it might involve most of the signed information and therefore range over any signed messages supplied under those conditions. To be determined yet!

The remote sender may have specified a recipient with or without an alias. When no alias was provided, one is found through the Communication ACL, and silently added. Future return traffic is bound to mention the alias. When an alias was provided, but the Communication ACL does not match it, then another alias will be assigned instead. At this point however, most services should report back to the sender. Whether this is done through a notice or by blocking access to the service is an open matter. This may be a measure for fighting spam, for instance.

The recipient address may be a group or role address. Once accepted, inward data may be passed through group or role processing. This may involve setting up a `group+member` address under the group's domain name for the remote user, thereby granting group membership or role occupancy to remote users, along with the privacy towards other members or occupants. Note that remote users are potential abusers, and for that reason such aliasing tends to require registration as a group member or role occupant, so administrative staff can track abusers.

Concrete Application: Email

Email is an application of the application pattern for communication services. To keep it most portable, the recommended implementation would be in the form of a few milters and general routing protocols:

- One milter for outward access control & aliases
- One milter for inward access control & aliases
- One milter for groups and roles
- Services should be relayed over LMTP
- Outbound messages encryption, DKIM and SPF through SMTP filters

There can be various LMTP components to service ARPA2 components, and more specifically email; it should be noted that we will probably need to change this to a service that is forwarded over AMQP 1.0 in the future, where LMTP is just a service-specific connector for mail.

- Adding and removing, as well as inviting and rejecting group members / role occupants
- Storing MIME attachments in the Reservoir for those that lack AMQP support
- Setting up OpenPGP keys for automated encryption

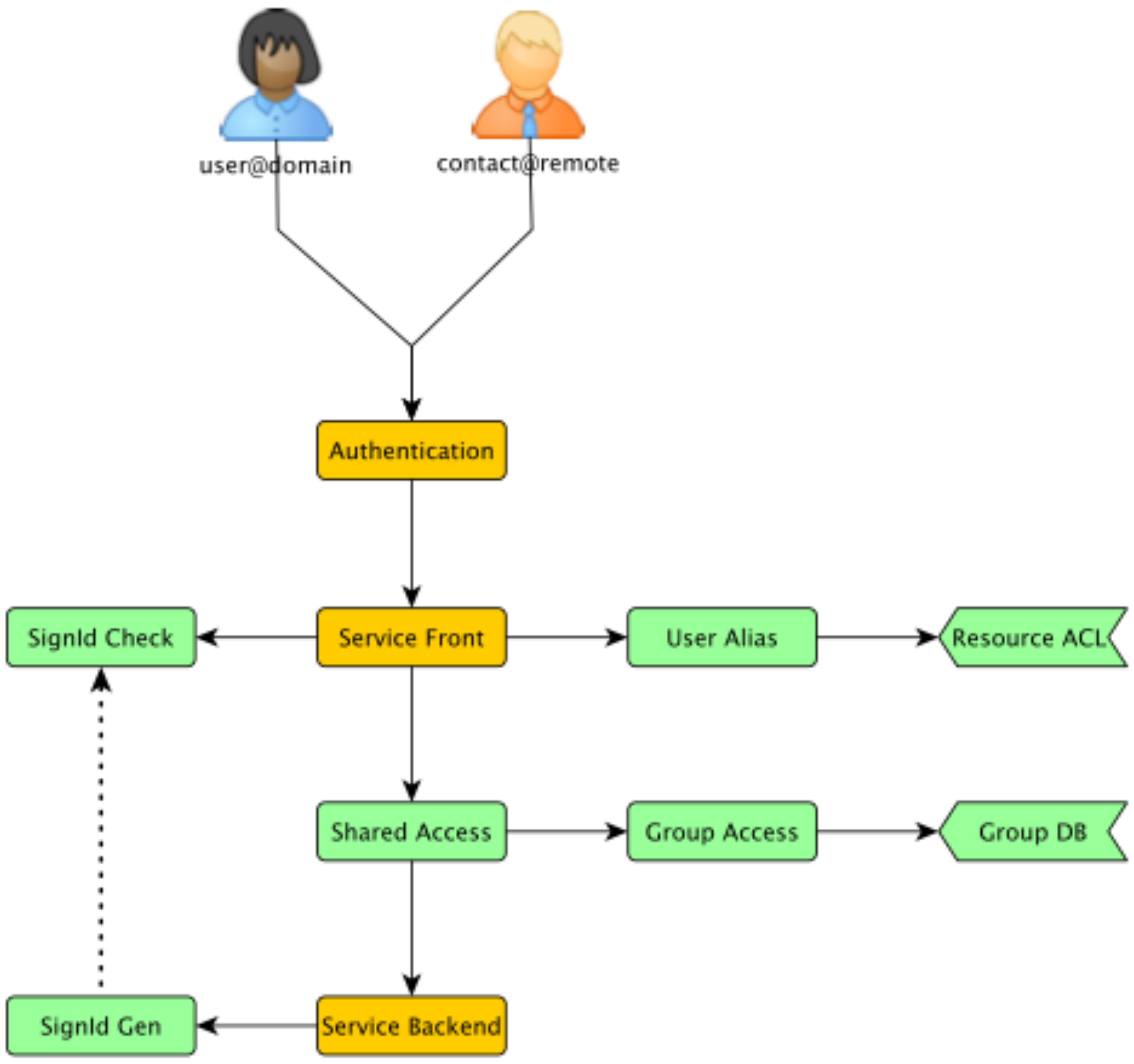
Concrete Application: Message Queueing

Message queueing is the communication service connected to the Reservoir service. It routes messages over AMQP 1.0, using a DoNAI to locate remote servers for pushing and pulling documents. The mechanism encrypts with TLS and authenticates with SASL, where SASL EXTERNAL is supported for clients that validated over TLS.

Application Pattern: Resource Services

Resource services are of much simpler structure than communication services. The method of passing data from one party to another is passive, by making it available and allowing the other party to read it from its resource location.

Resource services enforce a Resource ACL, expressing the rights to make changes to resources in a collection such as under the Reservoir.



Resource Access

Resource services access resources, for things like reading and writing data, creation and deletion, and so on. A generic one-letter encoding of access rights is used and interpreted within the context of a concrete service. To select a resource, a UUID for the application, and possibly an additional UUID for a resource instance, are concatenated and used against the user who wishes to connect to the resource. These are used to find a Resource ACL to apply.

Resource ACLs deliver a set of rights; so the requested access rights are not input to the ACL query, but rather their result. As in the Communication ACL, lookups also don't mention the ACL to use; these values are returned alongside the rights. The most powerful alias is selected for the requesting user, so that maximally potent access rights may be cached on their connection.

It is possible for a Resource ACL to grant access rights to remote users as well as to local users. Remote users may be selected with a very specific pattern or a very general, with access to all as the most general pattern possible. Given that the rights are assigned by an ACL too, the search for a match from the most concrete address form in the direction of more generic ones is helpful in finding the most concrete (and often the most empowering, but always the most applicable) access rights.

Shared access is used for groups and roles. Interestingly, the only aspect that is special for here is the use of the special alias `group+member` or `role+occupant` instead of a (local or remote) user's address. This means that the service aspects like storage are allocated under the group; furthermore, uploads, ownership and such are assigned to a group member instead of to a common `user` or `user+alias`. Only access-granted (local or remote) visitors who did not register as group members (or role occupants) will be shown with their full address; they are not granted the privacy of their delivery address.

Concrete Application: Reservoir

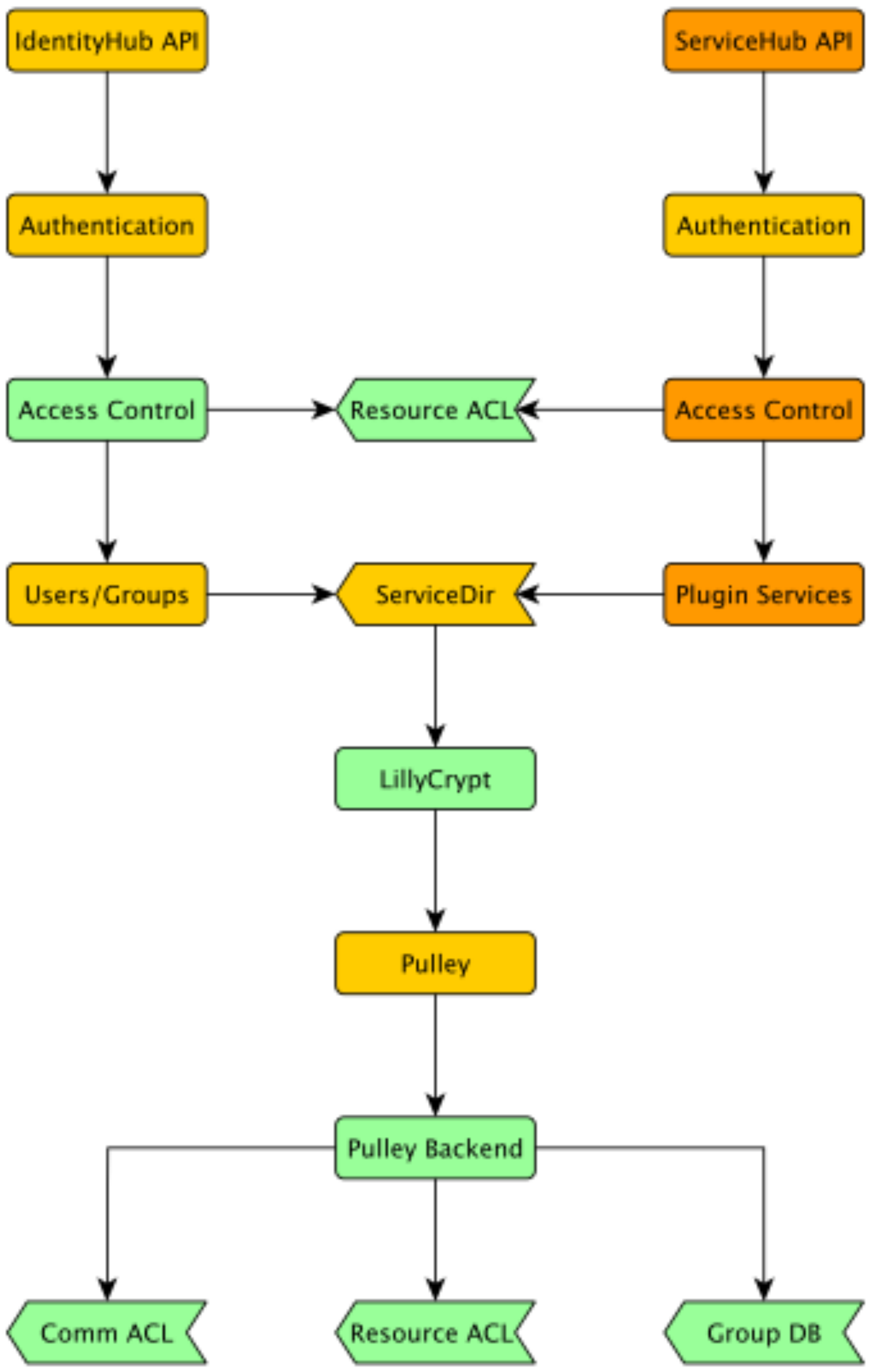
Reservoir is resource service that does not implement a new concept, but a technically different way of achieving an object store. It consists of the following components:

- Backend storage is the Riak KV object store
- Searchable index is an LDAP meta-data store
- Web frontend is made through WebDAV
- Input and output queues are made with AMQP 1.0

The result should allow standards-based searches, either the simplistic way through WebDAV or much more advanced through LDAP. The use of Riak KV lends us with replications facilities, one feature that hosting providers may use to improve the reliability of their service, a helpful factor for diversification of the market and price structures. AMQP 1.0 finally, is intended to be a secure manner of authenticated and encrypted object passing between users and domains. It may be used locally as an "upload" mechanism, or remotely as an "object push" mechanism. In general, it is meant as a mature replacement of MIME attachments in email, which are neither natural nor practical for these usage patterns.

Management Software

Lastly, we turn to the the hosting components that help to manage structures. In the IdentityHub, this mainly concerns the management of identities that may be incorporated into services as specified above.



Stuff put to Good Use

The proposed components that are silently assumed in the things described above; in other words, we are building on top of prior and current ARPA2 projects:

- TLS-KDH (such as via the TLS Pool)
- Kerberos Realm Crossover or `kxover`
- Remote PKCS #11 now known as `keehub`
- SteamWorks

- LillyDAP

The diagrams have a few blocks `Authentication` that includes a lot of our prior work. The various mechanisms explored there are available for the purpose of authentication. Where authorisation identities may be used, suggestions may be made about aliases (also for linked pseudonyms) including group aliases. Specific services may include yet more authorisation identity information, for example in resource paths or sender addresses. These may be subject to an authorisation inquiry, but simpler cases may be covered by an ACL or simply a comparison of the user part (both through `libarpa2acl`) may suffice. Note that there is a difference between the following:

- Authentication says who you are
- Authorisation says as who you may pose (in ARPA2)
- Access Control says what you may do

It is common for the last two to be combined, and we shall do it too; but it should be emphasised that in ARPA2 we have a part that evaluates whether an authenticated user may also act under another identity. It is the result of our rich identity model, which surpasses the most-occurring form (no identity changes).

IdentityHub

IdentityHub is the component through which domain administrators and users can add users, pseudonyms, groups, aliases, groups and roles.

Along with these identities there are Communication ACLs and Resource ACLs; these too will be managed under the framework.

As part of its responsibilities, the IdentityHub will make information about ACLs, groups and roles available to subscribing services.

The API to the IdentityHub can be viewed as a resource service; access to it is granted through the Resource ACL.

ServiceHub

ServiceHub is the component that manages services that are made available under the domain. Although it is destined for the next project phase, we already need some support for the facilitation of services through encrypted key/value databases.

The API to the ServiceHub can be viewed as a resource service; access to it is granted through the Resource ACL.

Not all the interfacing of ServiceHub will be provided during the IdentityHub project phase; orange marks things where we will show restraint until the next project phase.