# ARPA2 Handbook Documentation

***Release 0.0.1***

**Rick van Rein**

**Sep 27, 2017**

# Table of Contents

*This document details how you can get up to pace with the ARPA2 projects, and how these hang together to form what we have come to call the InternetWide Architecture. The intention being, to give you an idea of practical steps to take, and their relation to other work in this large project.*

The **ARPA2** project is an umbrella for open source projects that implement the open source and open networking ideals of a secure, private and decentral Internet. In addition, most projects aim to be accessible to less technically inclined users, for example through hosting providers that can install the ARPA2 software stack.

To achieve its ideals of control by users over their online presence, the individual projects adhere to an integral design, known as the **InternetWide Architecture**. It derives its name from InternetWide.org, the overall project frontal used to collect funds from which some vital ARPA2 projects are funded.

This is definately an ambitious project, not just for its goals but also for its size and intended impact. We are happy to see many people in the technical community to respond with great enthousiasm to our attempt at integrating the software that we all love to use, and to make it into a whole that is hopefully easy to install, but more importantly, that is simple enough to manage and use to be of value to everyday users who do not happen to have a technical background.

We integrate security and privacy as integral pieces of the design (the InternetWide.org architect is indeed a cryptographer) but we never loose sight of practical usability of what this brings. So expect great assets like single-signon, realm-crossover and bring-your-own-identity... and to see it combined with users, groups, roles, aliases, pseudonymity.

*There's no reason we can't have it all... all it takes is a concerted effort!*

# CHAPTER 1

## Introducing InternetWide / ARPA2

*This is an introduction to the ideology that underpins the InternetWide foundation and its handywork in the form of ARPA2 projects.*

The big idea behind ARPA2 is to create an environment where you are in control. Old-school hosting providers pioneered the idea by letting their users host their own websites, email addresses and such, all under their own domain name. This led to the so-called "LAMP stack", an acronym of the underlying technologies Linux-Apache-MySQL-PHP.

Since the introduction of this scheme, the Internet has seen a tremendous amount of innovation. But, as a result of cut-throat competition on the pricing of the relatively standard LAMP stack, most hosting providers have had a hard time keeping up. In their place, we now see specialised services based on centralised service hosting, much to the disadvantage of the privacy of individual users.

With ARPA2, we hope to regain the level of individual control that we had in the days of the LAMP stack, but without the limitations of just having web and email. This is not so strange in itself — chat and (video) telephony have long been standardised and can be run anywhere, anytime. All you need is an Internet connection — *in principle*.

Many advanced computer users actually run these services for themselves, and are successful at evading the centralised control from the few large silos that reign todays Internet (and the privacy of its users). Their level of understanding however, is not available to anyone. This is where hosting providers used to step in.

The competition between hosting providers however, makes it difficult for them to keep up with new developments. They might have picked up on some technologies, but there is no integratal adoption of things like chat and telephony — technologies that are pretty vital in the protection of individual freedom. To step in, we are developing

- An architecture for the Internet, based on loosely connected domains, either hosted privately or with a hosting provider. We coined the name InternetWide Architecture for this, and you will read a lot about it in this handbook.
- An open source software distribution that supports both the technical-savvy individual or company, and the hosting provider who simply wants to run a modern software stack that integrates with other hosting providers' domains, among others because they follow the same architecture. We coined the name ARPA2 as the umbrella project name for our individual development projects building towards the InternetWide Architecture.

The purpose is to achieve a lot of functionality, available to all, with choice of service provider and always supportive of do-it-yourself. A lot of excellent software is already available in the open source community; the trick is to identify

and build any extensions that these components may need in order to fit into the architecture. In almost all cases, this comes down to adding a plugin to which the applications are already pretty open, but for which nobody has identified the need yet. Our approach from an integral architecture helps us to find precisely the itches that need to be scratched if we are to get all these sublime components to work together as the concerted whole that leads to the perfect end-user experience. This is why we opted for the (admittedly grand) name "InternetWide Architecture" for this design — it simply describes best what we are aiming for.

The architecture is designed by a cryptographer, with serious attention for matters of security and privacy. The entire architecture is complex, because it touches upon many matters, but as we worked on it, a rather clear image has formed, fulfilling most or all of our hopes and dreams of getting to a mature, decentralised Internet, where individuals and companies regain their individuality with the security and privacy that they so much deserve to have.

As the project matures, we should see increasing numbers of users who feel less obliged to "do what everyone does" and accept the gradual tightening grip of silos on individual privacy. We are not here to put those silos out of business or disconnect them from users of the ARPA2 software distributio — we are only here to offer users a choice between a silo or individual responsibility for their online presence.

CHAPTER 2

# Phases of the InternetWide Project

The work on the InternetWide Architecture is split into four phases.

> **Status:** The architecture of SecureHub and IdentityHub is crystal clear. Coding of SecureHub is reaching a level where it is practically usable, the work on IdentityHub has started. Architecture of ServiceHub is relatively clear, but only a few work items have been tackled. SocialHub is developing as an architecture, but is not completely clear at this point, although we have a few resounding ideas standing by!

## ARPA2 phase 1: SecureHub

> *The SecureHub phase establishes a number of foundations for security. In spite of all the attention that has gone into our online security, much of it does not really come together.*

This is the first phase of ARPA2, and it is targeted at delivering a number of security premises on which the later phases can build. The end-user appeal of this layer is not very direct, but without this the IdentityHub phase would not be possible.

The main projects in SecureHub are:

- **TLS Pool** is a daemon that takes TLS out of applications. The reasoning is that application developers have another mind set (namely, a functional drive) than security experts (who want to lock out anything dangerous) and that these are best handled in different bits of software. The TLS Pool is a background program, with its own memory regions, in which it may juggle credentials without the application ever needing to worry about it. The interface between TLS Pool and application talks of identities, in the form of a domain name or a user under a domain name; integration with the TLS Pool from an application's perspective tends to take only an hour!
- **SteamWorks** distributes configuration information over LDAP. This means that central configuration (or, if you like the term, provisioning of configuration) is possible. This is useful in many situations where users are less qualified to do so, and prefer to leave technical matters to a more qualified person, be it a company's administrator or a security service provider. Although SteamWorks makes it possible to centralise control, it still is a matter of subscription from the client's end — and therefore, a choice made on the client machine.
- **Kerberos** is a very old security protocol, but it is also the best option when dealing with centralised authentication. So, if you are to centrally manage accounts for users, groups and roles, as well as secure connections

to services and machines, then Kerberos is ideal. This is the customary choice in companies of any reasonable scale (they might call it Active Directory, Samba, FreeIPA or Windows for Workgroups — but that still means they use Kerberos for authentication). In the IdentityHub phase, we will turn to extensions that make Kerberos suitable as a mechanism for the Internet as a whole and for card-swipe usage patterns; but in the SecureHub phase, our aim is to use it locally, under our own domain or realm.

- **TLS-KDH** introduces Kerberos in TLS. We do this in a modern way, where we always incorporate the mechanisms for Forward Secrecy. The integration of Kerberos with HTTP or HTTPS is surprisingly weak, even if it is in high demand. Adding a strong Kerberos mechanism to TLS means that we can use it in HTTPS, but also for STARTTLS-styled protocols like mail, chat and telephony — the list of uses is virtually endless. Given that the call for TLS is resounding these days, and given the inefficiency of doing this with X.509 certificates, it is good to know that our research has shown that authentication with TLS-KDH is about 5000 times as efficient as when using X.509 certificates.

After the SecureHub project, it is our hope to see a lot of spin-off work. It should not surprise anyone that TLS-KDH is integrated into the TLS Pool, so any application using that has a great opportunity of hooking into the InternetWide Architecture.

Proper integration of the TLS Pool does not only mean adding a generic TLS tunnel around it (though one is delivered with the TLS Pool), because that leaves the problem of communicating the authenticated identities of the two sides to the wrapped program. The best integration comes from a client program, be it a web server or client or proxy, or perhaps mail tools. When these start to communicate TLS, they may hand off their socket to the TLS Pool, indicate what is being expected about identities and tell the TLS Pool to start shaking hands with the other side. When done, the TLS Pool should respond with the authenticated identities of the local and remote end, or none if they could not be authenticated. The program then continues over a new socket, speaking the plaintext version of the protocol but resting assured that an external daemon handles connection privacy and integrity in just the way the user likes to have that handled. The authenticated identities can henceforth be used in the application program.

This use of the TLS Pool means that TLS-KDH can be used where it is supported — a great gain in efficiency, and proponents of short-lived credentials might argue that the security is also under much tighter control. Future extensions to Kerberos, such as realm crossover and pseudonymity will all be handled automatically. And when the IdentiyHub starts supporting centralised creation of new identities, the TLS Pool will follow suit — and the application can safely ignore any such thing.

We even intend to extend the TLS Pool idea at some point, to incorporate alternatives to TLS, such as SSH and GSS-API based protocols. This would lead to a slightly modified flow in applications, but only during their call for a handshake — after that they would once more have authenticated identities on each end, and reliance on another program to handle privacy and authenticity. This could give rise to alternative protocol elements such as `STARTSSH` and `STARTGSS` — thereby allowing us to switch from TLS to another security protocol if we feel so inclined. In security, it is always good to have one to throw away, just to help us instantly resolve security problems when the need arises.

## ARPA2 phase 2: IdentityHub

> *The IdentityHub phase provides a management console for users, over which they can create identities for users, groups, roles, pseudonyms.*

This is the second phase of ARPA2, and it is targeted at delivering an identity control cockpit, including every bit of cryptography that can be used to prove those identities. The IdentityHub supports Kerberos, X.509 certificates and OpenPGP keys.

But that's not all. The IdentityHub hinges on a Bring Your Own Identity idea, where you have an identity under your local domain that you should be able to bring anywhere you go online. Well, anywhere supportive of the InternetWide Architecture, of course. Still, there are other initiatives, and we seek to support those too, as well as possible and wherever it is not harmful to end user privacy and security.

This means that anyone could potentially see who you are, and link your behaviours throught that one identity. In protection of your identity, we therefore support the use of aliases, pseudonyms, roles and groups, all helpful to control how you appear to those parties with which you engage in online contact.

All this may sound straightforward enough, but let us assure you that there are quite a few technical problems to be resolved. Enough to making it quite a challenge. The reason we take this on is that we have set ourselves the goal of redesigning the whole thing, without limiting ourselves to the confines of an individual interest group; history has shown that individual interests make it difficult to come to something that benefits the Internet as a whole.

Are we arrogant for undertaking this? Perhaps. Do we stand a chance? Certainly... we see many organisations, some very large, gasp at the extent of what the IdentityHub wants to do, and fall in love with it. The general response comes down to *you are doing something that we hold dear... but if we did it, we would be forced to confine ourselves to just this little corner*. It does seem that adding up all those individual corners leads pretty much to what the IdentityHub does though, so we have good hopes for broad support.

*If anyone can do this, it will be the open source community!*

Parties preparing to adopt the IdentityHub are advised to do the following:

- Use Kerberos for authentication
- Use the TLS Pool for authentication where possible
- Use Diameter or, failing that, use RADIUS for authorisation

The general naming scheme for IdentityHub will be the DoNAI, meaning domain-or-user-at-domain. We have defined Selectors to be used as a kind of wildcard matches, and Access Control Lists to derive relationships.

In order to establish realm crossover, meaning the mechanism that an identity from one security realm can authenticate to another, we have found that the Kerberos variety is the most probable road to success, chiefly because it only requires changes in centrally controlled components. Having said that, there are no reasons to block things like OpenID Connect or even OAuth but they are each facing problems – such as that everyone wants to play the role of identity provider, while nobody wants to be on the trusting end of the game.

It is however vital for personal control to be in charge of the identity provider component. If anything should be hosted by ourselves, it is the cornerstone for deciding *this session may enter on behalf of Charles* – and that is quadrupally true on an Internet where identity providers have second thoughts about the privacy of their users.

## ARPA2 phase 3: ServiceHub

> *The ServiceHub phase establishes a plugin mechanism for online services. Given that one manages a domain name at a hosting provider of choice, there may already be a set of services; but ServiceHub enables plugging services provided by other, more specialised providers to appear under the domain name.*

This is the third phase of ARPA2, and it is targeted at making online services pluggable. The result is that a basic identity provider can host the IdentityHub processes, perhaps with a few basic services, and that anyone else on the Internet can offer plugins that can be added to these domain names. Doing so would enable the centrally managed users, groups, roles, aliases and pseudonyms to be used with those plugins, so there is an obvious need for some connection during such collaborations between hosting providers.

We believe that splitting the hosting market is helpful to achieve specialisation. This should be helpful in stopping the cut-throat competition between hosting providers, who now all offer more-or-less compatible packages. On an Internet where it pays to specialise, there should be more variety, more experiments and more joy for everyone concerned. It also leads to an Internet which is not so easily overtaken by one central party that attempts to block out other oferings.

## ARPA2 phase 4: SocialHub

> *The SocialHub phase turns the Internet as a whole into a social network. This is not a new idea in itself, but we have gotten used to thinking social services are web-based and, as a result, located with a more-or-less central provider. This model has shown to be detrimental to our control over online presence and privacy, so we turn it around. Nobody minds a web interface, but the heart of social networking should just be... networking. And by that, we mean at the level of the general Internet Protocol, where everyone is an equal.*

This is the first phase of ARPA2, and it is targeted at letting people build social networks, in which they can exchange snippets of information in a variety of ways, either over peer-to-peer communication or using more classical mechanisms, but always founded on open protocols and/or open standards.

# CHAPTER 3

# Getting Started

*How can you get on board? What are the steps to gradually incorporate the ARPA2 projects, and by that the InternetWide Architecture, into your systems?*

This chapter describes how you can get on board, and start to use the components that we built for this ever-expanding software stack. We do not target specific use cases yet. The steps described below are global, and reference more detailed instructions (or indicate that this software is still being worked on).

## Adopting PKCS #11 (standardised secret store)

In case you haven't heard of PKCS #11, it is an API that is designed to conceal secret and private keys from being observed, or exported in plain view. These objects are the cornerstones of cryptographic protection, and being unable to replicate them is helpful to your security and privacy.

**Status:** There is a wide array of mature software choices for PKCS #11, ranging from software, through simple USB keys, to high-end, hardened bastions that would rather destroy your keys than release them to a physical intruder.

- PKCS #11 is a foundation on which we built the TLS Pool. We realise it is an extra effort to install and get used to, but PKCS #11 adds great value in terms of flexible security deployments. Once it is setup and running, it should not give any more headaches, other than to attackers after your keys.
- We advise you to get started with SoftHSMv2, because it is a sophisticated and mature open source product. If the need for hardware protection arises later on, you merely swap the library that implements your PKCS #11 API (and juggle keys as deemed necessary).
- If you want to "play" with PKCS #11, you may have a look at your mail or web clients; they often support plugging in PKCS #11 libraries, on which you can then store your local key material and certificates. This is also what the TLS Pool does; it stores private keys behind a PKCS #11 API and links them to certificates that are stored in a local identity database.
- There is a standard for pkcs11: URIs which can describe either tokens or objects (such as keys, certificates, or data) on that token. This is not a *location* but an *identifier*, in the sense that it provides selection criteria for a token and, given extra parameters, objects on a token. You will need to supply your software with a library that implements the PKCS #11 API along with each URI.

- When you get started with the TLS Pool, we will talk you through the steps of using PKCS #11 for that purpose. These steps will assume SoftHSMv2 and GnuTLS tooling, but you can easily vary if you feel a need; PKCS #11 solutions are fairly exchangeable, other than that their implemented security level varies drastically.
- When we initiate the IdentityHub, we are going to manage PKCS #11 at a (paid to be trustworthy) hosting provider — or perhaps on your Raspberry Pi at home — and we will then introduce Remote PKCS #11. This will allow you to share your identities anywhere you are, including on your mobile device. And yes, we have concerned ourselves with the security of that design!

## Adopting TLS Pool (splitting security from applications)

The TLS Pool is the piece of software that takes security knowledge out of applications. It relies on PKCS #11 and, in part because of that, is not easy to get started. Once it is running however, you should find that it easily "clicks in" and grants you a lot of control over security, in one place for all the applications concerned. As a matter of fact, you can even centralise management.

**Status:** The TLS Pool works well on servers, but using it on client platforms is a bit awkward. This is mostly due to lacking multi-user support, but if your client is in practice a single-user platform you should be able to use the TLS Pool quite nicely.

- The first thing to do is perhaps to start using the TLS Pool, and use it from the applications that you rely on.
- If your server does not rely on authenticated client identities, than it may suffice to simply use the TLS Tunnel distributed with the TLS Pool. This tool can even use simple scripts to talk a remote server into `STARTTLS` mode, and then proceed as before.
- If your client wants to use the TLS Pool to authenticate to a server, it may also suffice to use the TLS Tunnel. It will authenticate the client, so this may even add value to non-TLS applications. For instance, you could direct your mail client to a locally hosted SMTP server, which is then redirected by the TLS Tunnel, authenticating with `STARTTLS` and using credentials kept in the TLS Pool. Dependent on the server side, this may be helpful or confusing to client authentication; the ideal situation being to rely on `SASL EXTERNAL` authentication, referring back to an identity that was authenticated over TLS.
- Use of the TLS Pool for authentaction by applications is not something that the ARPA2 project can directly influence, but it may help if you ask the producers of your favourite pieces of software to look into it — or if you organise the needed patches. It has been shown repeatedly that the addition of the TLS Pool to an existing application takes about an hour for someone well-versed in the application's inner structure!
- If you have an application that has "add TLS" on the TODO list, think no longer — adopt the TLS Pool. It will save you a lot of anxiety about security configuration, where to load certificates from and how to handle the accompanying private keys in a secure manner. All this is delegated to the TLS Pool. In fact, it will implement many more things than you would ever be likely to add — and you get it all for free.

## Adopting SteamWorks (LDAP configuration dissemination)

The idea of SteamWorks is to offer you with a configuration backbone.

**Status:** At this point, SteamWorks is in its infancy. A few test setups have shown to work though.

- SteamWorks is founded on LDAP, for which we hope to simplify management through a front-end that takes care of the intricacies of running a complex daemon. SteamWorks uses LDAP SyncRepl as a subscription mechanism for near-instant dissemination of configuration changes to those clients that have subscribed. The design of SteamWorks incorporates components that help to isolate networks from connectivity problems to remote parts of the Internet.

- SteamWorks delivers configuration information to backends, using configuration scripts to determine what information should be taken from where in LDAP, and how to pass it on to those backends. The backends will then store the information in a form suitable for the targeted software system.
- As a first example, the TLS Pool has a backend for SteamWorks, permitting the complexities of TLS configuration to be made centrally, for instance by an administrator. This releases individual users from the concerns that come with the specialised domain of security. As soon as a security problem pops up, a change can be made in the SteamWorks configuration and subscribed clients would virtually immediately pickup the change and process it locally, is the underlying idea.

## Adopting Kerberos (or await IdentiyHub)

We advise you to start using Kerberos as your central authentication mechanism.

**Status:** Kerberos is very robust, very secure and very well integrated into software. Setting it up is not always easy, but once setup life simplifies to a single-signon system. Note that the one weak point at this time is support for HTTPS — and that we are working on resolving that generically, by adding Kerberos to TLS in the form of TLS-KDH.

- Kerberos offers "single sign-on", which means that you login once (usually at the start of the day) and continue to use the initial credential received for hours to go. You need to do no further logins, as any new service ticket needed during those hours are derived from your initial start-of-day credential. On top of that, Kerberos is very fast, because it is built on symmetric-key algorithms such as AES-256.
- Kerberos is integrated into many protocols, including SMTP, IMAP, LDAP and SSH. You will be pleasantly surprised by how much faster your SSH logins will be with Kerberos, compared to using the SSH Agent or mundane/manual password entry.
- Be aware that we will offer Kerberos as part of the upcoming project phase, IdentityHub. This means that you should think ahead if you want to setup a quick test platform to cover the time until we get there, or that you prefer to wait a while. Be advised that setting up the KDC is the most difficult part of running Kerberos, which is why we will "web-wrap" it for less technical domain owners, but a straightforward setup on a single node is quite doable for an experienced administrator.
- Be advised that the IdentityHub works towards automated realm crossover between Kerberos realms, in a way that allows clients of on realm to authenticate to another, even if they had not been introduced before. The technique for doing so will likely derive its security from DANE. Note that authentication does not imply authorisation: you may be able to prove to anyone who you are, but the question remains if you are welcome to use any given service.

## Adopt RADIUS or Diameter (for authorisation)

The RADIUS system and its follow-up Diameter are widely used to centralise authentication, authorisation and accounting within a domain, or secure realm. Although authentication (who is it?) and authorisation (what may he do?) tend to blur in these systems, they will answer questions like *Can Jack access the files of Joe?* with *Yeah* or *Nay*.

**Status:** RADIUS is very stable, and in principle, so is Diameter. The latter is less often used, but has a few clever extensions. When we come to the ServiceHub, we will standardise on Diameter. Mature systems exist to connect RADIUS and Diameter, so that should not constrain your local choice. New systems should probably consider Diameter, and break with the somewhat overdue tradition of RADIUS.

- As part of IdentityHub, we will support a flexible authorisation framework. It is assumed that a user's identity is authenticated, and that its access to a resource is questioned. A few examples are:
  - Can this user act on behalf of that user, group, role, alias or pseudonym?

  - Is this user acceptable to the access control list for the given service?

- We have not standardised the method of asking these questions yet. This is in part due to our desire to retain privacy as much as we can; when we setup the SecureHub, we would prefer to not spread all information to each service, but hopefully answer little more than these direct questions in a way that they can easily cache. And of course that also introduces concerns of efficiency of any such caches and the expediency of updates to them.

## Adopt modern Internet standards (in general)

In our work, we are going to make a few radical assumptions about support for modern Internet protocol. Some things simply cannot be done well on IPv4 (thanks to NAT traversal nightmares) and some things are simply not secure without mechanisms like DNSSEC.

> **Status:** Most of these technologies have been around for quite a while, and their software has been well tested. You may find that you are still an early adopter, although you would also be surprised how well the uptake of various modern developments has been in practice.

- Adopt IPv6. Really, when it comes to end-to-end communication you do not want to be dealing with NAT traversal issues. Ask any self-acclaimed SIP expert about their fondness of NAT and they will show you their best daunting-nightmare grinn. We are going to *require* IPv6 on various parts of our design, simply because it evades problems that are now in the way of progress, and because the inherent complexities of NAT traversal lead to complex code, complex networking problems and poor end-user experiences.
- Adopt DNSSEC. Start signing your zones, and validate your DNS queries. We will *require* DNSSEC for a number of follow-up projects, for example to validate DANE entries, and perhaps SSHKEY records in DNS. With DNSSEC in place, the DNS is a reliable database of domain-specific information; without DNSSEC, it cannot be used as a foundation for *anything* that is at risk of being abused.
- Adopt SCTP. This protocol sits next to UDP and TCP, and can actually behave like either. In addition, it can run multiple streams at the same time. Particularly interesting is its support of a reliable, but not necessarily in-order delivery of frames, which happens to be very useful for systems that run on UDP to avoid head-of-line blockage but that need to resend at the application level to achieve a reliable service. We will *require* SCTP for several of our services, among others for the ServiceHub project where it will be used as a sort of *umbilical cord* between hosting providers. Chances are that SIP infrastructure will also require SCTP, in support of longer messages that can then include security information.
- Be advised that the IdentityHub and ServiceHub will export DANE records for services. Research by SURFnet had led to the conclusion that the spread administration of DNS, certificates and servers makes DANE a potentially fragile system, but automation can be helpful to overcome that.

## Embrace for more...

The InternetWide Architecture encompasses much more. As this project advances, we will update this page with more steps that you can take.