rxgk - GSSAPI based security for AFS

Simon Wilkinson
Your File System Inc

rxgk

- Design
- Implementation
- Deployment

History

- 2004 - Initial rxgk design at Stockholm Hackathon
- 2007 - Further design work in Stockholm
- 2007 - Prototype implementation for arla
- 2007 - AFS Best Practices Workshop
  Love Hörnquist Åstrand: “rxgk - why and how far”
- 2009 - Your File System Inc fund rxgk development
- 2009 - Edinburgh Hackathon discusses rxgk design
- 2009 - European AFS Workshop
  Simon Wilkinson: “rxgk : GSSAPI based security for AFS”
- 2010 - Complete set of rxgk Internet Drafts published

Design Goals

(and limitations)

mechanism independence

- All the world is not Kerberos
- Permit the use of any authentication mechanism with a GSSAPI interface
- But don’t require kernel implementations of every GSSAPI mechanism we support

algorithm agility

- No more fcrypt

- Really no more DES

- Agility required so we don’t have to do this dance again for every crypto change

defend against cache poisoning attacks

- Using only the user's key to secure connections opens us to cache poisoning attacks

- Malicious user forges traffic from the fileserver, and populates cache with bogus data

- Other users on the server read bogus data (and, potentially, write it back to the fileserver)

- Particularly dangerous in the case of executables

departmental fileservers

- rxkad has a single, cell wide, key
- All servers have a copy of this key
- Knowledge of the key conveys super user powers - any user may be impersonated to any server
- Currently impossible to run servers with a smaller set of delegated powers

secure the callback channel

- Callbacks are currently unauthenticated
- In AFS3, this only leads to potential denial of service attacks
- With extended callbacks, an attacker could manipulate the contents of the client’s cache
- Extended callbacks requires a secure callback channel to be deployable across the internet

anonymous cache managers

- Secure callbacks, and cache poisoning prevention require cache manager keys

- Have to allow for cache managers which don’t have key material

- Use work on anonymous GSSAPI to permit anonymous, but keyed, cache managers

server enforced security policy

- aka ***“The First Packet Problem”***
- Client sends its first packet to a server before the security challenge
- Server cannot prevent that first packet containing unencrypted data
- Server, therefore, cannot require that all communication with it be encrypted

preserve location independence

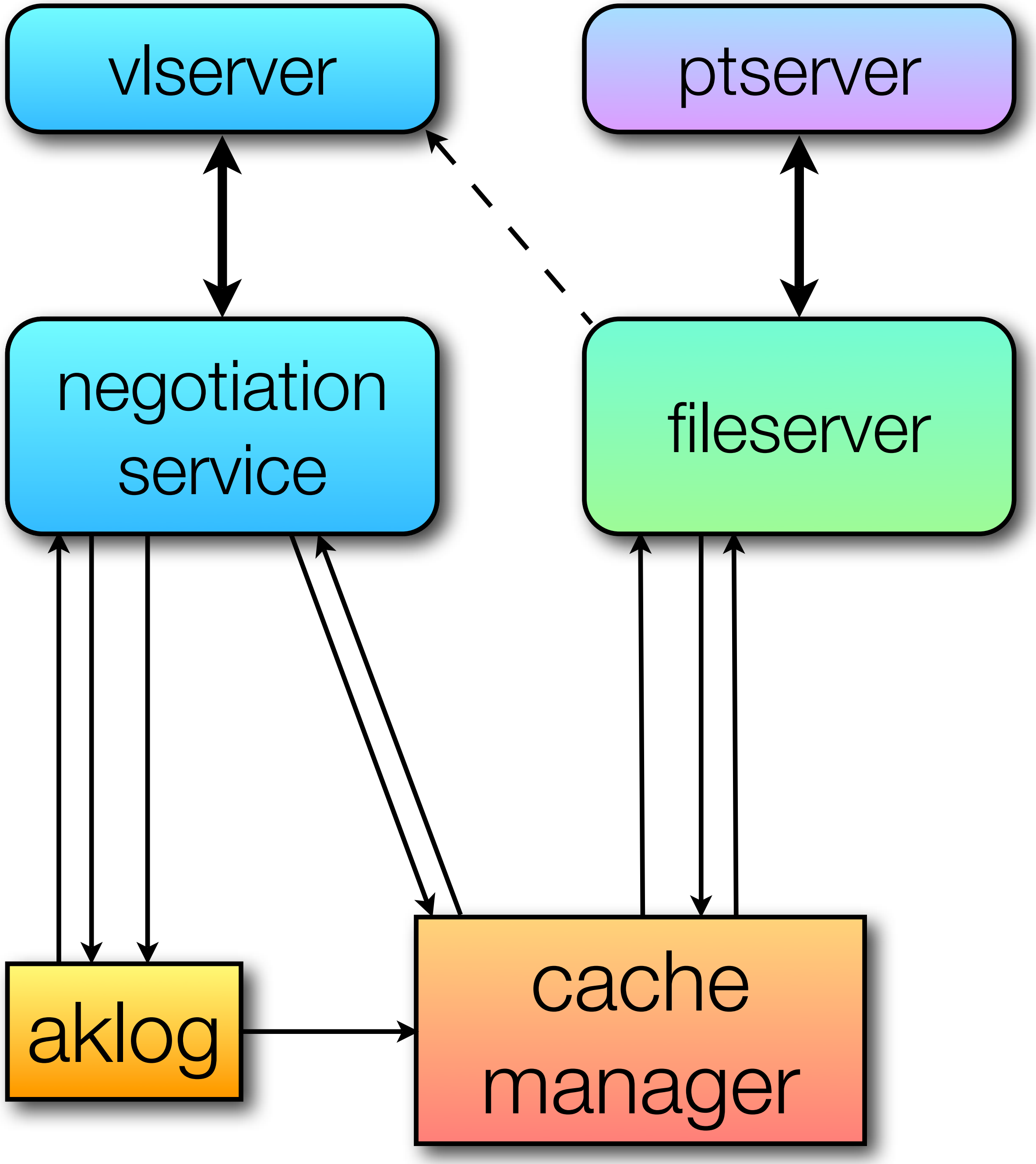
- Authenticate to “AFS”, not to individual servers
- User shouldn’t be involved when cache manager uses a different server
- Particular issue with smartcard based GSSAPI mechanisms

rx limitations

- Authentication is limited to single challenge / response
- Challenge is server initiated
- Restrictions on maximum packet size

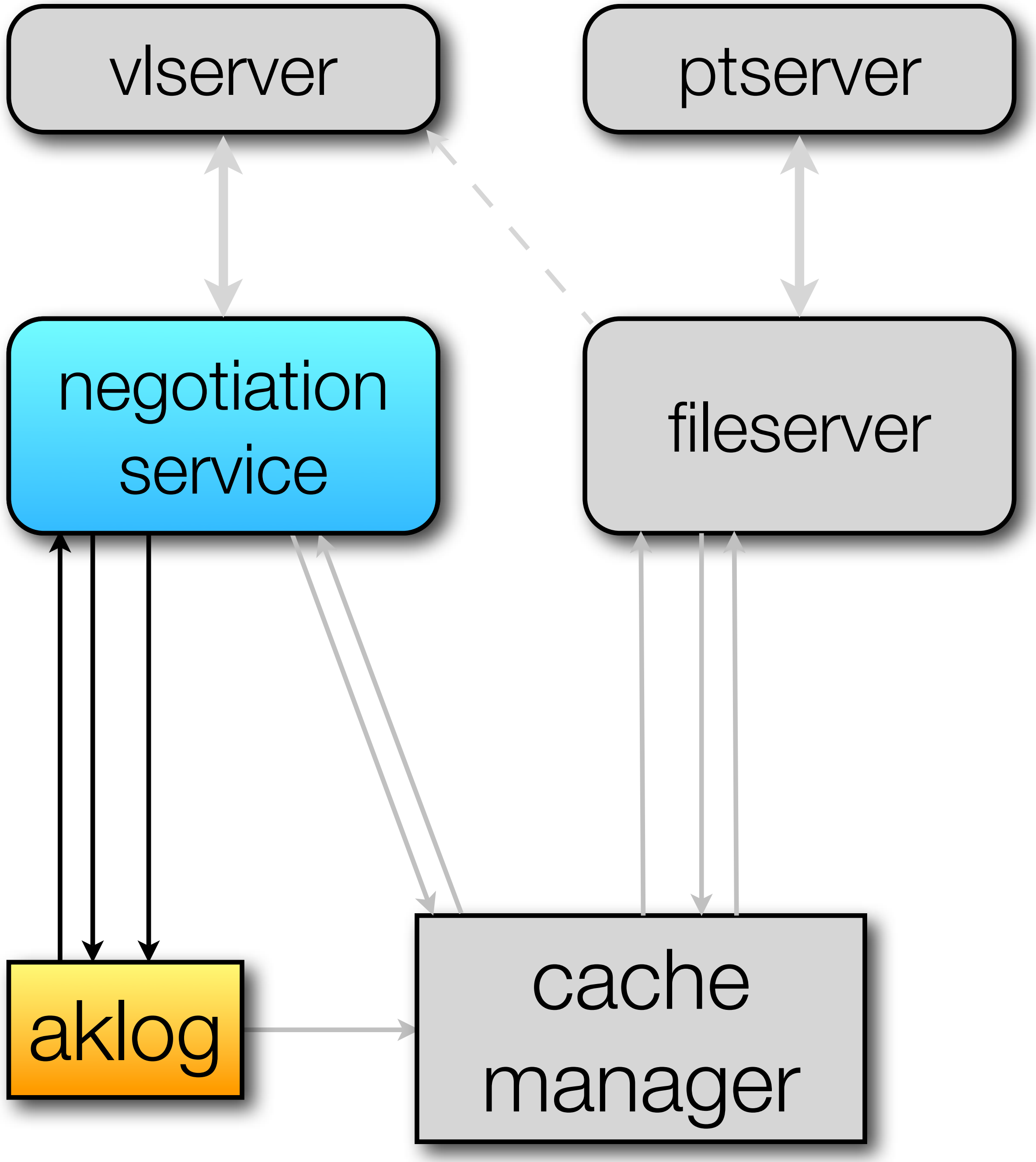
rxgk design

overview



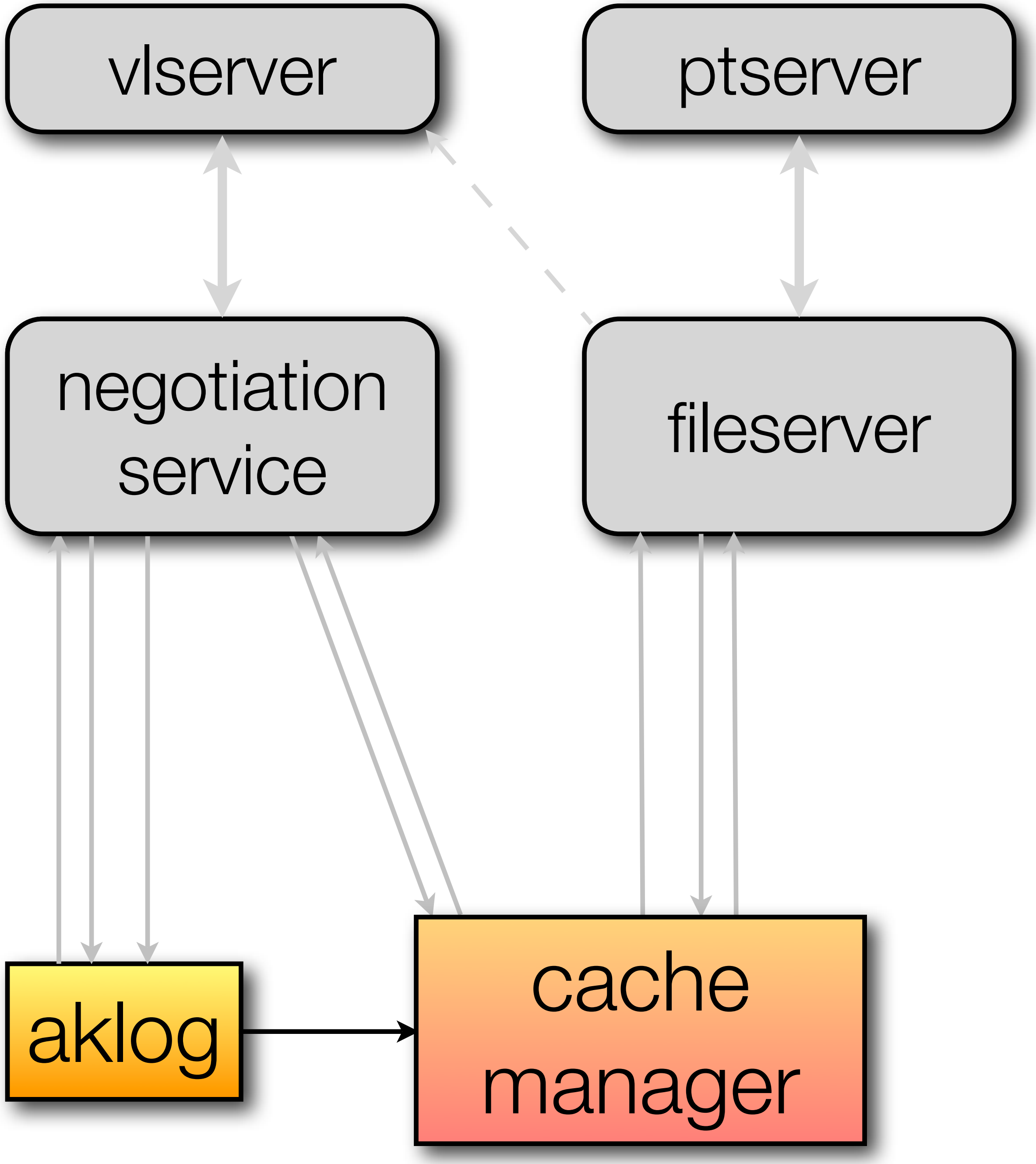
vlserver
ptserver
negotiation service
fileserver
aklog
cache manager

token acquisition



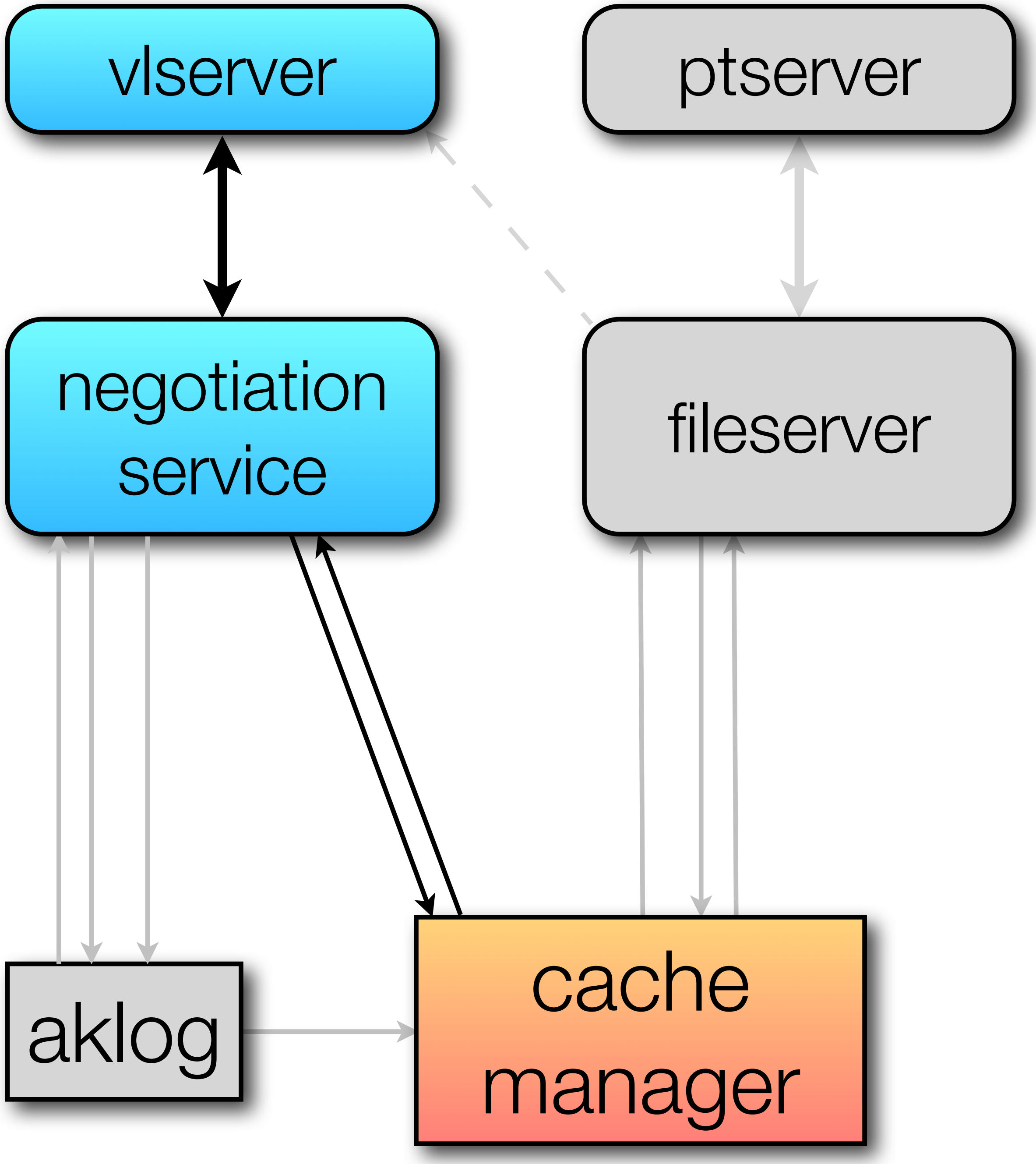
- aklog on client
- performs GSSAPI handshake with negotiation service
- gets rxgk token containing session key

token storage



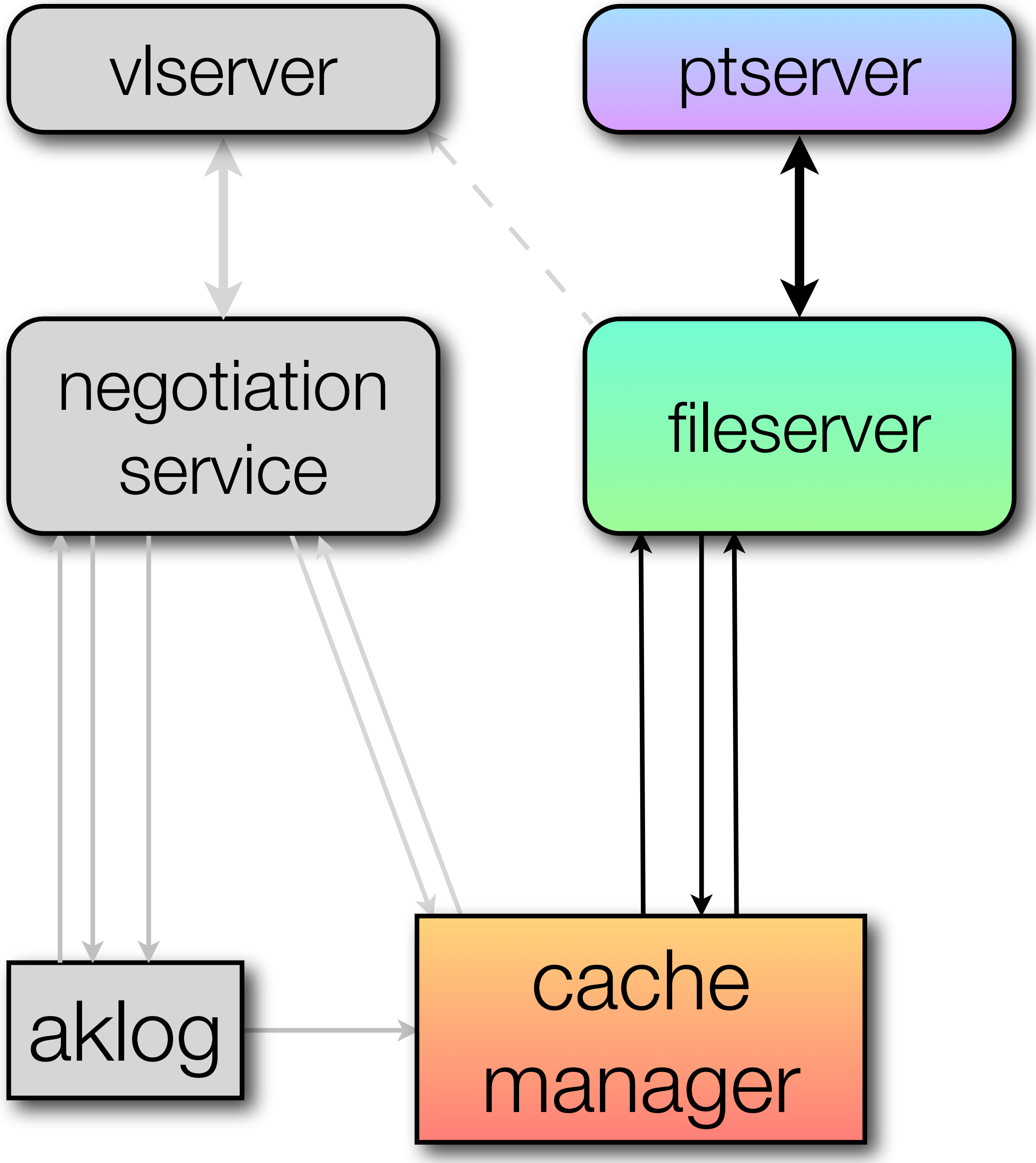
- aklog uploads token (and session key) to cache manager

fileserver first contact



- Cache manager calls negotiation service with:
  - user’s token
  - cache manager’s token
  - fileserver uuid
- Gets an rxgk token specific to that fileserver, and an indication of minimum security level

fileserver connection



- Cache manager sends first packet to fileserver
- Fileserver sends rxgk challenge
- Client sends rxgk response (using fileserver token)
- fileserver converts client identity (from token) to pts identity

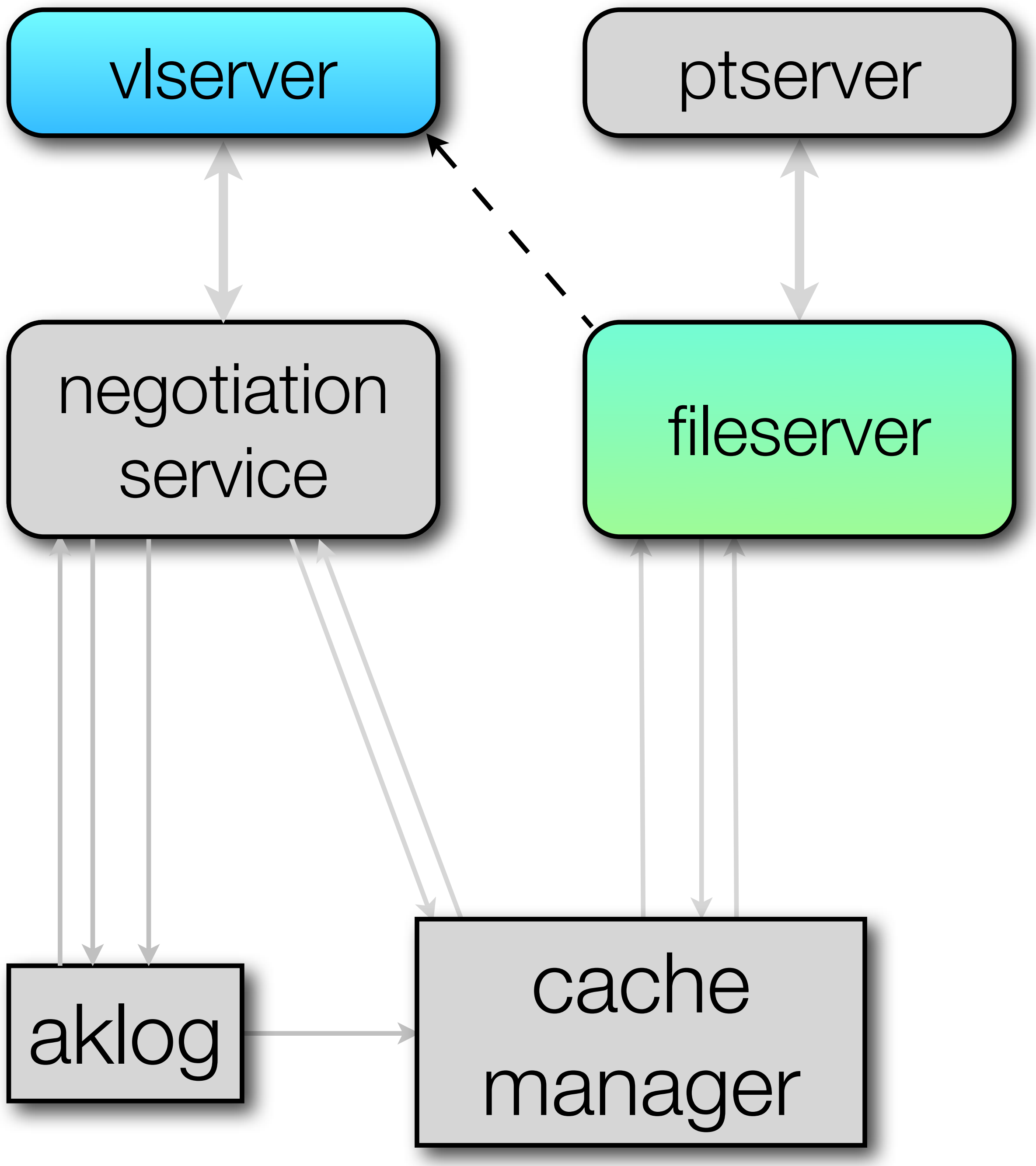
connection encryption

- Three permitted encryption levels:
  - **clear**: Only connection establishment is authenticated. An active attacker can mount MITM attacks. Good for speed, poor for security.
  - **integrity**: Data in connection is integrity protected. An active attacker cannot add or remove information, but a passive attacker can read all information.
  - **encryption:** Data in connection is privacy and integrity protected. An attacker can neither read, nor amend, the data straeam.

connection encryption

- Any algorithm defined by RFC3961 can be used for connection encryption
- Initially working with AES, but algorithm (and hash) agility is built in.

fileserver registration



- At start up, fileserver registers with vlserver
- vlserver marks fileserver as rxgk capable
- fileserver may also register rxgk server key

implementation

abstraction

- Lots of places assume rxkad keys
- Lots of places contain duplicated code to initialise, and accept, rxkad tokens and keys
- Unify all of these into single functions in libauth
- In 1.5.x series now

code cleanup

```
#define u (*(get_user_struct()))
```

is just plain evil...

tokens

- kernel token storage, and interface pioctls assume rxkad
- New expanded pioctl interface from Arla, and prototyped in rxk5
- Mechanism agnostic token storage, and pioctls, implemented as part of rxgk work
- In the *new-tokens* branch of YFS's github, queued for inclusion after 1.6

xdr

- OpenAFS's XDR was an interesting mixture of vendor and local code (sometimes in the same process!)
- Unify on using our own XDR routines everywhere
- Add support for xdr_free()
- Fix xdr_mem and add xdr_len mechanisms
- All applied to the 1.5.x series

crypto

“leave cryptography to the cryptographers”

crypto

- Use an external crypto library wherever possible
  - Heimdal’s hcrypto
  - OpenSSL
  - Mozilla NSS

- Hardware acceleration support will “Just Work”

- Use Heimdal’s RFC3961 implementation when crypto library doesn’t offer its own

crypto

- Different rules apply for kernel code
  - Some kernels have no crypto
  - Others won’t let us use the crypto they have

- Local import of hcrypto
  - Build system tailored for kernel use

- Same Heimdal-derived 3961 library as used in userspace

deployment

1: deploy rxgk capable clients

- rxgk clients won’t do anything differently in cells without rxgk support

- Safe to deploy them first

2: upgrade database servers

- rxgk requires new ptserver and vlserver
  - ptserver for support of GSSAPI name types
  - vlserver for rxgk capability flags, and negotiation service

- New servers must be deployed to all Ubik replication sites before new features can be used

- It may be possible to build test instances without requiring new dbservers, but production use will require them.

3: register GSSAPI names in ptserver

- Register GSSAPI names for all rxgk users in the ptserver
- Will automatically happen for Kerberos
- Other GSSAPI mechanisms will require per-site scripting (we don’t know what form your X509 names take!)

4: Create rxgk service key

- rxgk uses a cell-wide service key for token encryption.
- This must be replicated between all database servers, and all non-departmental fileservers

5: Create rxgk GSSAPI key

- rxgk uses the GSSAPI identity `afs-rxgk@_afs.<cellname>`

- Key material for this identity must be available to all database servers

6: Bring up rxgk capable fileservers

- Install the rxgk service key
- Restart the fileserver
- **NB:** Downgrading the fileserver requires administrator intervention

7: delete the old afs/cell key

- Once all clients, and all fileservers are rxgk capable ...
- Remove the old rxkad afs/cell key
- Downgrade attacks are, sadly, unavoidable whilst this key is still present

Questions

(internet willing)