INDIANA UNIVERSITY

# Solution Architecture
# For
# Shared Key Authentication and Encryption

Revision History

| Date | Revision | Author |
|---|---|---|
| 2012-07-18 | Created | ajk |
| | | |
| | | |

## Table of Contents

# I. Introduction

The Lustre filesystem currently supports Kerberos authentication to protect file data and metadata from network eavesdropping and tampering. While Kerberos is an excellent authentication protocol, it requires some infrastructure at the client end that is not trivial to deploy nor always permissible by site policy.

A coexisting shared-key, host-based authentication system would preserve the authenticity, integrity, and privacy of file data and metadata but without requiring any additional infrastructure on the client side. In such a system, a single key would be generated for each client host, and that key would be installed on both the client and the server.

Lustre's Kerberos support is provided through the Generic Security Services Application Program Interface (GSSAPI), a modular interface for providing authentication and encryption mechanisms. By implementing a shared-key system as a GSSAPI mechanism, Lustre's Kerberos functionality can be left untouched, and no new authentication and encryption code needs to be added.

We propose three new Lustre security flavors with accompanying GSSAPI mechanisms:

- *sknull:* no cryptography—for testing only, with a debug mode that logs incoming and outgoing packet signatures
- *ski:* data integrity through strong, host-based message authentication
- *skpi:* both data integrity and privacy through the same host-based authentication as *ski* plus strong encryption

# II. Solution Requirements

## Security

The shared key changes must provide mutual authentication and encryption between client and server using cryptographically strong, well-tested, and well-understood cipher algorithms, providing integrity and privacy protection equivalent to what is offered by the current Kerberos functionality. The key must be a simple shared secret.

## Modularity

Shared key authentication and encryption must not interfere with and must be usable independently from Lustre's existing Kerberos-based functionality. When shared key authentication is disabled, authentication and encryption must be indistinguishable from the current Kerberos functionality. The shared key changes must introduce no new authentication or encryption code into Lustre itself but rather must be implemented as GSSAPI mechanisms selected by Lustre security flavors. The shared key

cryptography shall have no direct knowledge of or communications with the Lustre code.

### Testability

The shared key functionality must be testable and must not reduce the testability of Lustre's existing code paths. A null security flavor and GSSAPI mechanism must be implemented to ease testing of these code paths, and unit and functional tests must be implemented for the null code paths as well as the shared key authentication and encryption code paths.

### Reuse of Cryptography Code

The pitfalls of reimplementing cryptographic algorithms must be avoided. No new cipher algorithms will be invented, instead opting for industry standards such as AES. Well-tested, well-understood existing code and services (such as the Linux keychain) will be reused wherever practical.

### Easy Key Management

Userspace tools to create, load safely into Lustre, list, and revoke keys must be available.

### Easy Configuration

Shared key authentication and encryption must be enabled or disabled using a single, simple configuration parameter. It must be disabled by default to avoid creating a backdoor to the Kerberos flavor.

## III. Use Cases

### Sensitive Data

Medical researchers at different sites want to share research data. Due to regulations on protected health information (PHI), the researchers want the data to be encrypted in transit and protected by strong authentication[1]. The *skpi* security flavor, proposed below, allows the researchers to use Lustre to collaborate with minimal additional configuration or maintenance overhead.

### Strong Cryptography Without Kerberos

The two researchers above could use Lustre's existing Kerberos support to fulfill their ethical and compliance obligations. Researcher A, however, works at a site with no existing Kerberos infrastructure. To use Kerberos,

[1] Note that strong authentication and encryption are not the only requirements imposed by regulations such as HIPAA, and this solution does not attempt to satisfy all those requirement. This solution does provide strong authentication and encryption, however, which may be a part of a site's HIPAA alignment strategy.

Researcher A would have to maintain an independent Kerberos realm; expending resources on this operational overhead is undesirable.

Researcher B's site does have a Kerberos infrastructure, but its procedures for issuing credentials to external people (such as Researcher A or countless future collaborators) are cumbersome and time-consuming. What's more, Researcher B's site does not permit Researcher B to run an independent authentication realm.

Kerberos is an excellent authentication system for some sites. But for Researchers A and B, it's not a good fit. These researchers can get strong cryptography with the *ski* or *skpi* security flavors proposed below with minimal overhead.

### GSSAPI Unit Testing

A Lustre developer making changes to the GSSAPI support code wants to test those changes. The *sknull* security flavor proposed below will make this testing simpler because it won't involve maintaining test keys or a test Kerberos infrastructure.

## IV. Solution Proposal

In the Lustre architecture, all the components, including the client, are trusted, meaning if the client is compromised, no choice of authentication or encryption mechanism will protect file data. That said, cryptographically strong algorithms will prevent network-based eavesdropping and monkey-in-the-middle attacks.

Implementing shared key as a GSSAPI mechanism and security flavors will achieve the modularity goal. Since the current functionality interfaces not directly with Kerberos but indirectly through the GSSAPI, implementing a separate GSSAPI mechanism will keep the shared key functionality totally separate. Two new security flavors are proposed: *ski*, which provides data integrity through strong message authentication; and *skpi,* which provides both data integrity and privacy through encryption.

Testability of the shared key functionality starts with the GSSAPI code path. A null GSSAPI mechanism will be developed to allow easier testing than the shared key and Kerberos mechanisms. We also propose a third security flavor to use this mechanism (*sknull*), complete with a debug mode that logs incoming and outgoing packet signatures. Unit tests can be written to examine the debug output. The shared key mechanism will also include this debug mode.

To avoid the common but egregious pitfall of implementing cryptography algorithms incorrectly, the shared key mechanism and security flavors will use

existing kernel cryptography modules and the Linux keychain for key management.

A key management interface will be achieved by developing userspace tools to create, list, and revoke keys and *lctl* methods to load the keys into a secure memory module so Lustre can then access them. Unit testing of these tools will be straightforward.

# V. Unit/Integration Test Plan

## Null GSSAPI Mechanism

Attempts to authenticate to a trivial GSSAPI server against the null mechanism using no credentials.

## Shared Key Null Security Flavor

Attempts to mount a Lustre filesystem using the shared key null security flavor (*sknull*).

## Userspace Key-Generation Tool

Attempts to generate a shared key using the provided userspace tool.

## Server-Side Key Loading Method

Attempts to load a shared key into the Lustre server using the provided *lctl* method.

## Client-Side Key Loading Method

Attempts to load a shared key into the Lustre client using the provided *lctl* method.

## Shared Key Privacy Security Flavor

Attempts to mount a Lustre filesystem using the shared key integrity security flavor (*ski*). Tests whether the filesystem can be mounted with an incorrect or missing key.

## Shared Key Privacy and Integrity Security Flavor

Attempts to mount a Lustre filesystem using the shared key privacy and integrity security flavor (*skpi*). Tests whether filesystem can be mounted with incorrect or missing key. Tests for encrypted communications.

# VI. Acceptance Criteria

SFS-DEV-002.2.AC1: All unit tests pass.

SFS-DEV-002.2.AC2: Shared keys can be successfully created, loaded into Lustre clients and servers, listed, and revoked.

SFS-DEV-002.2.AC3: A user can access the filesystem using either shared key and Kerberos authentication (not necessarily both at the same time).

SFS-DEV-002.2.AC4: A user cannot access the filesystem using an incorrect key or a zero-length key.