2006

Jini distributed key exchange and file transfer service with digital signatures

Kevin Ligozio

Masters Thesis Proposal

Rochester Institute of Technology

Masters of Computer Science

Masters Candidate: Kevin Ligozio

Committee Chairman: Stanislaw Radziszowski
Committee Reader: Alan Kaminsky
Committee Observer: Edith Hemaspaandra

Table of Contents

Revision History

5/03/2001 – Initial Revision - *Draft*
5/08/2001 – Revised per comments from S. Radziszowski – *Final Draft*
5/12/2001 – Revised per comments from A. Kaminsky
5/19/2001 – Revised per additional comments from A. Kaminsky

- Changed Jini Service model
- Removed command-line interface examples in functional specification.

5/22/2001 – Changed from "Project" proposal to "Thesis" proposal.

Summary

The Java Cryptography Extension (JCE) includes so-called "strong cryptography". It includes implementations of some well-known algorithms, including the Digital Signature Algorithm (DSA), and the Diffie-Hellman Key Agreement Algorithm. It also provides the Java Cryptography Architecture (JCA), which allows other cryptographic algorithms to be plugged in. A *provider* is a collection of cryptographic algorithm classes that implement a certain interface. This project will utilize the JCA to implement a provider that implements the Diffie-Hellman Public Key Agreement Algorithm, the Rijndael Symmetric Key Encryption Algorithm (as specified by the Advanced Encryption Standard (AES)), and the Digital Signature Algorithm (DSA) specified in the Digital Signature Standard (DSS). A software package will be developed that will allow users to exchange information encrypted with AES and signed with DSA. The users will exchange Rijndael keys using the Diffie-Hellman Public Key Agreement Algorithm with DSA signatures. The software package will be part of the JINI Architecture, providing two JINI services: the *JINI Key Agreement Service*, and the *JINI Secure File Transfer Service*.

Overview

This project primarily involves the study of cryptography. It uses public key (Diffie-Hellman), private key (Rijndael/AES), and digital signature algorithms (DSA) to study the methods in which information can be securely transferred. In addition to understanding the theory of how information can be (relatively) securely transferred, the project covers the number theory involved in the mechanics of the algorithms. Finally, the project involves the development of a software package that allows users to share information securely as described in the project summary. This development requires an understanding of the Java programming language, JINI, general software development practices, and the Java Cryptography Extension (JCE). In general, this involves the understanding of good programming practice, the ability to effectively use an existing architecture and API, and an understanding of distributed software technologies and techniques.

Architectural Overview

There are two primary components to the software developed for this project. The first is a JCE *Cryptographic Service Provider* (provider) that implements the Diffie-Hellman Key Agreement Algorithm, the Digital Signature Algorithm, and the Advanced Encryption Standard Algorithm (Rijndael). The provider will have three "engines", each engine implementing one of the three primary algorithms. The second is the development of two JINI services, a JINI Key Agreement Service and a JINI Secure File Transfer Service.

Implementation of Cryptographic Algorithms using JCE

The JCE 1.2.1 includes the following engines related to this project:

- Cipher – used to encrypt or decrypt some specified data
- KeyAgreement – used to execute a key agreement protocol between 2 or more parties.

This project implements a Cipher engine for AES, and a KeyAgreement algorithm for the Diffie-Hellman Key Agreement. The DSA algorithm will be implemented using interfaces in the standard Java2 SDK (J2SDK). The JCE defines **"Service Provider Interfaces" (SPI)** which define the application interfaces implemented by each engine.

Interface Summary by Engine:

- Diffie-Hellman Engine
  - Implements
    - KeyAgreement – provides the functionality of the key agreement protocol
    - AlgorithmParameterGenerator – generates the Diffie-Hellman parameters (prime (p) and primitive element (g))
    - KeyPairGenerator – generates the public and private variables for key exchange (e.g., x and $X = g^x \bmod p$)
    - KeyFactory – used to save keys for use after the application has been closed and re-opened
  - Algorithms required:
    - Prime Generation
    - Primitive Root Generation
    - Random Number Generation
    - Modular exponentiation (provided by Java's BigInteger)
- DSA
  - Implements
    - Signature – provides the functionality of the digital signature algorithm
    - KeyStore – used to create and manage a private database of keys
    - KeyPairGenerator – generates the public and private keys for signing
  - Algorithms required:
    - Prime Generation
    - Primitive Root Generation

        - Extended Euclidean Algorithm
- AES
    - Implements:
        - KeyGenerator – generates the symmetric key (note that this is optional since the symmetric key will be generated by the Diffie-Hellman Key Agreement – this is only useful for testing without the Key Agreement algorithm being used).
        - SecretKeyFactory – used to save keys for use after the application has been closed and re-opened
        - Other interfaces required are still being researched and will be included in the next draft.
    - Transformations required (as per AES specification – I haven't researched these enough yet to know specific algorithms required):
        - SubBytes()
        - ShiftRows()
        - MixColumns()
        - AddRoundKey()
- Number Theory (Shared Utilities between Engines)
    - Euclid's Algorithm (greatest common divisor)
    - Extended Euclidean Algorithm (multiplicative inverses)
    - Primitive Root Generation
    - Prime Number Generation
        - Store all of the primes up to 2000 to test before using probabilistic primality testing (the number 2000 is recommended by Schneier).
        - Use Miller-Rabin primality test
    - Random Number Generation (will use Sun implementation if available)
    - Hash Function (will use Sun implementation if available)

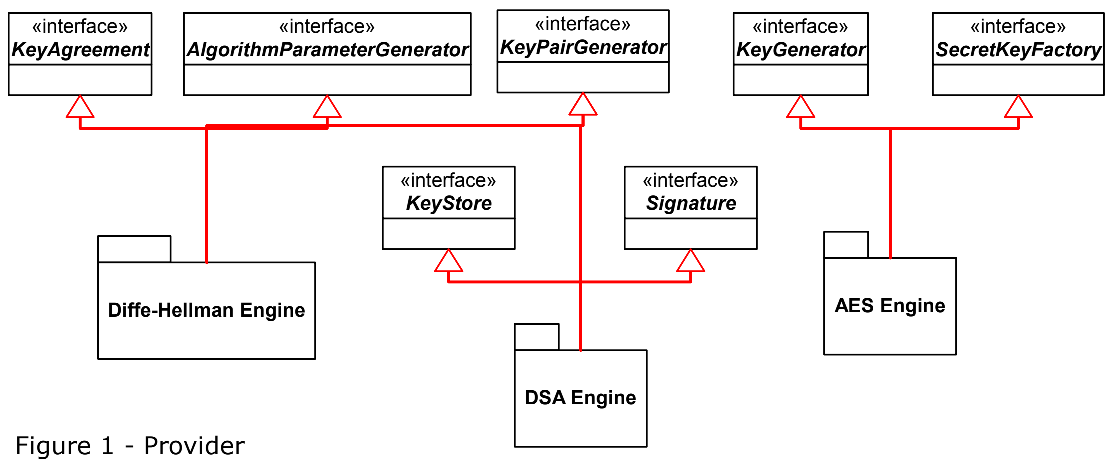

Figure 1 - Provider

JINI Services for Distributed Key Agreement and Encrypted File Transfer

The second primary component is the software package that will utilize these encryption algorithms. The package will, when possible, allow the user to select which provider to use (i.e., Sun's or the provider written for this project), so that comparisons can be made.

The application will provide either a command-line or graphical user interface that allows the user to generate and view both private and public keys used in all algorithms (for verification). The application will support three JINI Services created specifically for this project: The JINI Key Agreement Service (JKAS), the JINI Signature Service (JSS) and the JINI Secure File Transfer Service (JSFTS).

The Key Agreement Phase

Each user that is willing to be the destination of key secure file transfers will register instances of two services: The JINI Key Agreement Service (JKAS) and the JINI Secure File Transfer Service (JSFTS). The registration will include service attributes that tell which user the services are associated with. When user Bob registers his JKAS, JSS and JSFTS, he includes a service attribute with the user name "Bob". When user Alice decides to send a secure message to Bob, she will search the JINI Lookup Service (JLUS) for an instance of JKAS and JSS associated with "Bob". Alice will be returned a service proxy object for Bob's JKAS, and for his JSS. The JKAS is already connected via RMI to Bob's backend JKAS service. After receiving the service proxy, Alice initializes Bob's JKAS proxy with a reference to her JSS backend service, and calls a method in Bob's JKAS service proxy to start the key agreement session. Bob's backend JKAS will lookup an instance of the JSS associated with "Alice" in the JLUS so that he can verify her signature. During the exchange, Bob signs using his backend JSS service, and verifies Alice's signature using her JSS proxy. Similarly, Alice signs using her backend JSS service and verifies Bob's signature using his JSS proxy. When the session has completed, Bob's backend JKAS service stores the secret key along with some identifier, such as a serial number, that distinguishes the key from all other keys that Bob's JKAS has generated. The secret key is also calculated by Bob's JKAS service proxy on Alice's machine and is returned in a wrapper object that encapsulates both the key and the identifier of the secret key assigned by the backend service. Finally, the key-wrapping object obtains a lease of the secret key with the backend service. The secret key is a leased resource, and if Alice cancels the lease or fails to renew it, or if Bob decides to cancel the lease, then the backend service will discard the key.

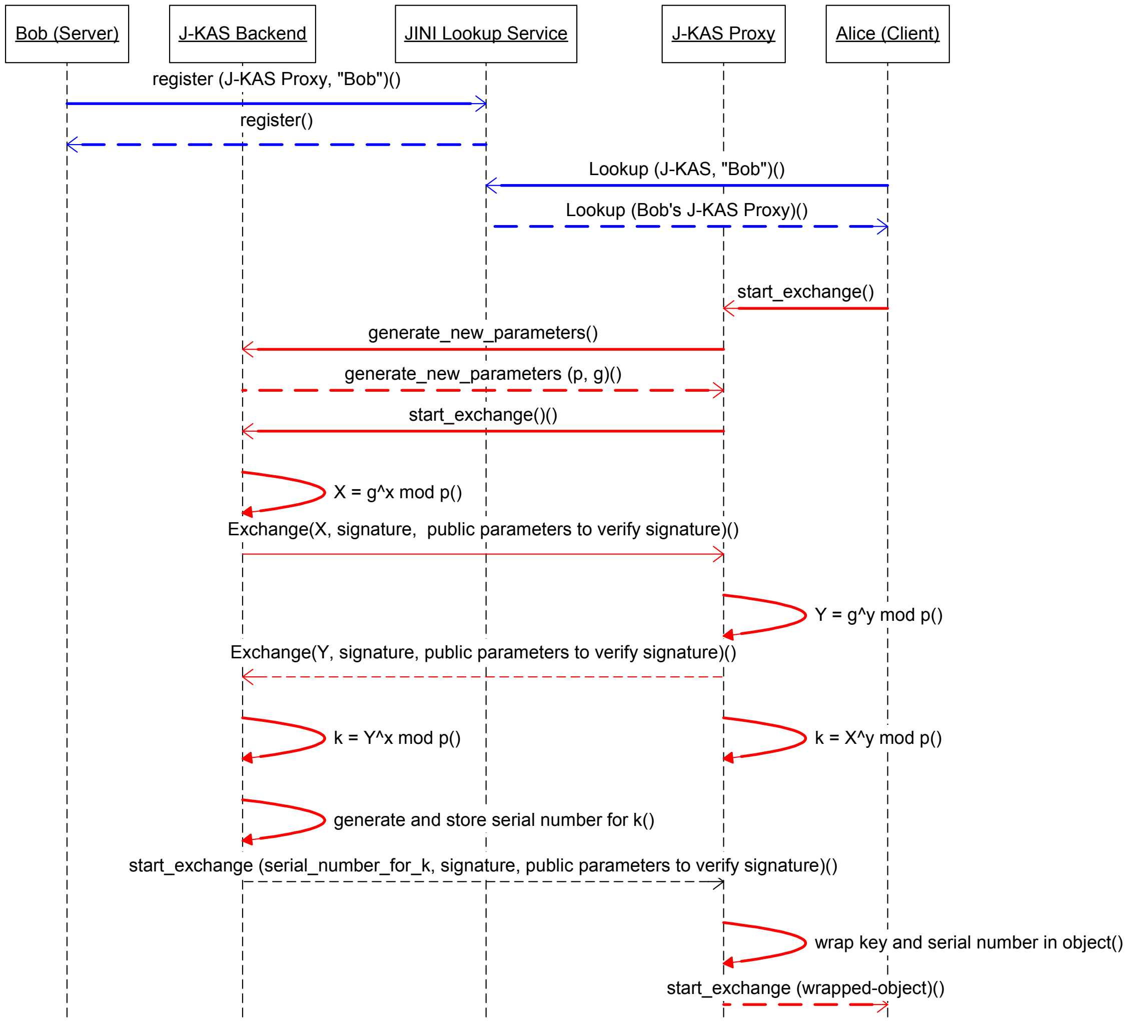
Bob (Server)
J-KAS Backend
JINI Lookup Service
J-KAS Proxy
Alice (Client)
register (J-KAS Proxy, "Bob")()
register()
Lookup (J-KAS, "Bob")()
Lookup (Bob's J-KAS Proxy)()
start_exchange()
generate_new_parameters()
generate_new_parameters (p, g)()
start_exchange()()
X = g^x mod p()
Exchange(X, signature, public parameters to verify signature)()
Y = g^y mod p()
Exchange(Y, signature, public parameters to verify signature)()
k = Y^x mod p()
k = X^y mod p()
generate and store serial number for k()
start_exchange (serial_number_for_k, signature, public parameters to verify signature)()
wrap key and serial number in object()
start_exchange (wrapped-object)()

Figure - The Key Agreement Phase - the Diffie-Hellman Key Agreement interactions are in red. The JINI registration and lookup are in blue. The signature is not included for simplicity.

Transferring a File Securely

After a secret key has been securely agreed upon, Alice searches the JLUS for and instance of the JSFTS associated with "Bob". She is returned the service proxy object for Bob's JSFTS, and initializes it with a reference to Bob's JSS proxy. Alice calls a method in the service proxy to begin a file transfer, passing in both the name of the file and the key-wrapping object returned to her from the key agreement phase. The service proxy communicates, via RMI, to the JSFTS backend service running on Bob's machine, giving it the identifier of the secret key to use (which was stored in the wrapping-object). The service proxy then digitally signs a hash of the file's contents, encrypts the file and signed hash using the secret key, and sends it all to Bob's JSFTS backend service. The backend service uses the identifier to retrieve the associated secret key from the local key store, decrypts the message, verifies the signature, and stores the file's contents in a pre-determined location on Bob's machine.

Principal Deliverables

- A project report that contains, at a minimum:
    - Abstract
    - Project Details – i.e., Interaction of the AES, DSA and Diffie-Hellman Algorithms (how they were used together)
    - Mathematical Background (Overview of algorithms used)
    - Software Design Overview
    - Source Code
    - Software Usage Instructions
    - Input/Output Examples
    - Possible improvements to algorithms (e.g., efficiency, etc.) and to the software (e.g., multi-user, etc.)
- Source Code Archive
- Software Demonstration

Functional Specification

The deliverable software shall, at a minimum, be tested on the Solaris operating system. Time permitting, it may be built and tested on the Win2K operating system. There will be one executable, and two instances of it will be started, either on the same machine or different machines on a LAN. The user interface will provide commands similar to the following commands. An example command-line command string is given to help illustrate the use of the software, but these are subject to change and may be replaced with a graphical representation.

Setting up the environment for JINI:

As described in the architectural overview, JINI will be used as the distributed medium for this project. This section describes the commands associated with initializing the client with the correct services.

Note: JINI requires that (1) a web server is running, (2) an instance of the RMI Activation Daemon *(RMID)* is running (for Sun's JINI lookup service – *reggie*), and (3) at least one JINI lookup service is running. This setup should be done *before* running the software delivered for this project.

The Application Software:

There are two standalone programs for this project. The first is a server, which is run by every user wishing to receive securely transferred files. The server runs all backend services as described in the Architectural Overview section. The server also registers the appropriate instances of the JKAS, JSS and JSFTS service proxies with the JLUS, and maintains the service registration leases. The second standalone program is a client program. This program allows a user to select the desired instances of other's JKAS, JSS and JSFTS services in the JLUS, and then perform a secure file transfer using those service instances.

References

1. Cryptography, Theory and Practice; Douglas R. Stinson.
2. Applied Cryptography; Bruce Schneier.
3. http://csrc.nist.gov/encryption/aes/rijndael/
4. Java Cryptography; Jonathan Knudsen.
5. Handbook of Applied Cryptography; Alfred J. Menezes, et alia.

Note that it is expected that the references will increase. For example, code examples may be used.

Draft Table of Contents

Detailed Schedule

Current Status:

I have compiled information on all algorithms. I have spent the most time on the Diffie-Hellman Key Agreement Algorithm, and have created a prototype using Sun's DH Provider. The prototype establishes an RMI connection with another instance of the prototype on the same machine and exchanges a key. Note that the actual algorithm has not been implemented – it is Sun's implementation. The prototype was an attempt to better understand the JCE, and the Diffie-Hellman algorithm, before actually implementing it.

I read the AES specification, but need to review it in greater detail and start to determine all of the different algorithms that need to be implemented it. The specification seems to be clear in this regard. No prototyping has been done for this algorithm.

I have read and need to review the DSA algorithm, as it seems to be the most complicated one. The current plan is that the key agreement will not to be signed, but the text will be. Although this allows the "intruder in the middle" attack, this is only an exercise in learning the fundamentals of cryptography, and not an attempt to make a "rock solid" application.

I have taken a considerable amount of time studying the JCE, and understanding how I can write and use a provider that plugs into the architecture.

Milestones:

- Completed Proposal – Signed and approved – 5/30
- Solid understanding of all algorithms with prototypes – ____
- Software Implementation Complete - ____
- Software Test Complete - ____
- Demonstration to Advisor - ____
- Project Report Draft - ____
- Project Report Final - ____
- Practice Defense Date - ____
- Target Defense Date - ____