

# Online Election System

**95.413 Project Report**  
Winter 2001, Carleton University

Dan DuFeu 219041 (ddufeu@chat.carleton.ca)  
Jon Harris 267309 (jbharris@magma.ca)  
April 10, 2001

## **Abstract**

An election is an excellent example of a scenario where security and confidentiality of data between parties is critical. The voters must be securely provided with a list of candidates to select from, their selection of a candidate must be securely recorded and their choice kept confidential. It is also necessary for the list of voters and candidates to be kept secure, as well as the final results of the election.

This project aims to provide a sample implementation of a secure voting system in which all of the prerequisites mentioned above are met.

Through the use of Public Key Cryptography, Symmetric Key Cryptography, Certificates and Signatures all of the data in the sample implementation provided with this document is kept both secure and confidential.

# Table of Contents

Abstract .....	2
Table of Contents .....	3
Introduction .....	4
Requirements.....	4
Assumptions.....	4
Components.....	5
Central Legitimization Agency (CLA) .....	5
Central Tabulating Facility (CTF).....	5
Voter Client.....	5
System Overview .....	6
Protocol Description.....	7
Requirement Analysis .....	12
Implementation.....	13
User Guide.....	13
Quick Start.....	13
System Data Files.....	14
System Execution.....	15
Build Instructions .....	16
Javadocs of Source Code.....	16
References .....	17
General .....	17
Implementation Specific .....	17

## Introduction

We have chosen to implement the online election system project as provided by the instructor. The following describes the requirements, the assumptions, and the components that form the online voting system.

## Requirements

The design of the online election system will ensure that:

- Only authorized voters can vote.
- No one can vote more than once.
- No one can determine for whom anyone else voted.
- No one can duplicate anyone else's votes.
- Every voter can make sure that his vote has been taken into account in the final tabulation.

## Assumptions

The following are assumptions for the Online Election System:

- Each voter will be securely delivered a VoterId and Password from an authorized source. For example, the VoterId and Password could be delivered by secure courier to the voter using two separate packages. The VoterId and Password is only known to the Voter and the Central Legitimization Agency (described below). The VoterId and Password combination is of significant complexity that it is computationally infeasible for an attacker to produce a valid pair.
- The results of the election may be retrieved at any time, even before a ballot has been cast. This is primarily for demonstration purposes.
- The implementation uses a simple certificate that simply signs a public key and name pair. These certificates do not have expiry dates or the ability to recall, however they are simplified for the purposes of this project. The certificates are placeholders for a more complete certificate scheme.
- The signatures used in this project comprise of an RSA encrypted SHA-1 hash of the previous message. The signatures are generally not encrypted in another message. In order to achieve a higher level of security with these signatures, a timestamp has been placed in the encrypted message and factored into the hash code, however a more elaborate scheme would likely be used in real life.
- We assume that the servers maintain their information in a persistent data store, so that in the event of a crash or power failure the information remains intact.

## Components

The components that implement the online election system are as follows:

- **Voter.** (Client/GUI for interaction with voters)
- **Central Legitimization Agency.** (Server for authenticating voters)
- **Central Tabulation Facility.** (Server for accepting and tallying votes)

### Central Legitimization Agency (CLA)

The **Central Legitimization Agency's (CLA)** main function is to authenticate and authorize the voters. Each voter will send a secure message to the CLA requesting a ValidationId. The CLA will generate a ValidationId, register it securely with the CTF, and return it securely to the voter. The ValidationId is of a significant complexity so as it is computationally infeasible for an attacker to produce a valid Id. The CLA retains a list of ValidationIds as well as a list identifying the recipient of each ValidationId, in order to prevent a voter from receiving two of them, and thus being able to vote twice.

### Central Tabulating Facility (CTF)

The **Central Tabulating Facility (CTF)** provides the following functionality:

- Allows a user to request a certified list of candidates.
- Accepts secure ValidationIds certified and signed from CLA.
- Accepts secure vote requests from authorized voters (with ValidationId).
- Securely returns the candidate of an authorized voter's ballot upon request for verification.
- Allows a user to request a certified election result.

In order to authorize the voter, the CTF checks the ValidationId against the list received from the CLA. If the ValidationId is there, the CTF crosses it off (to prevent someone from changing their ballot) and sets their ballot to the requested candidate. As a design of the system (for demonstration purposes), the CTF can produce a certified result at any time.

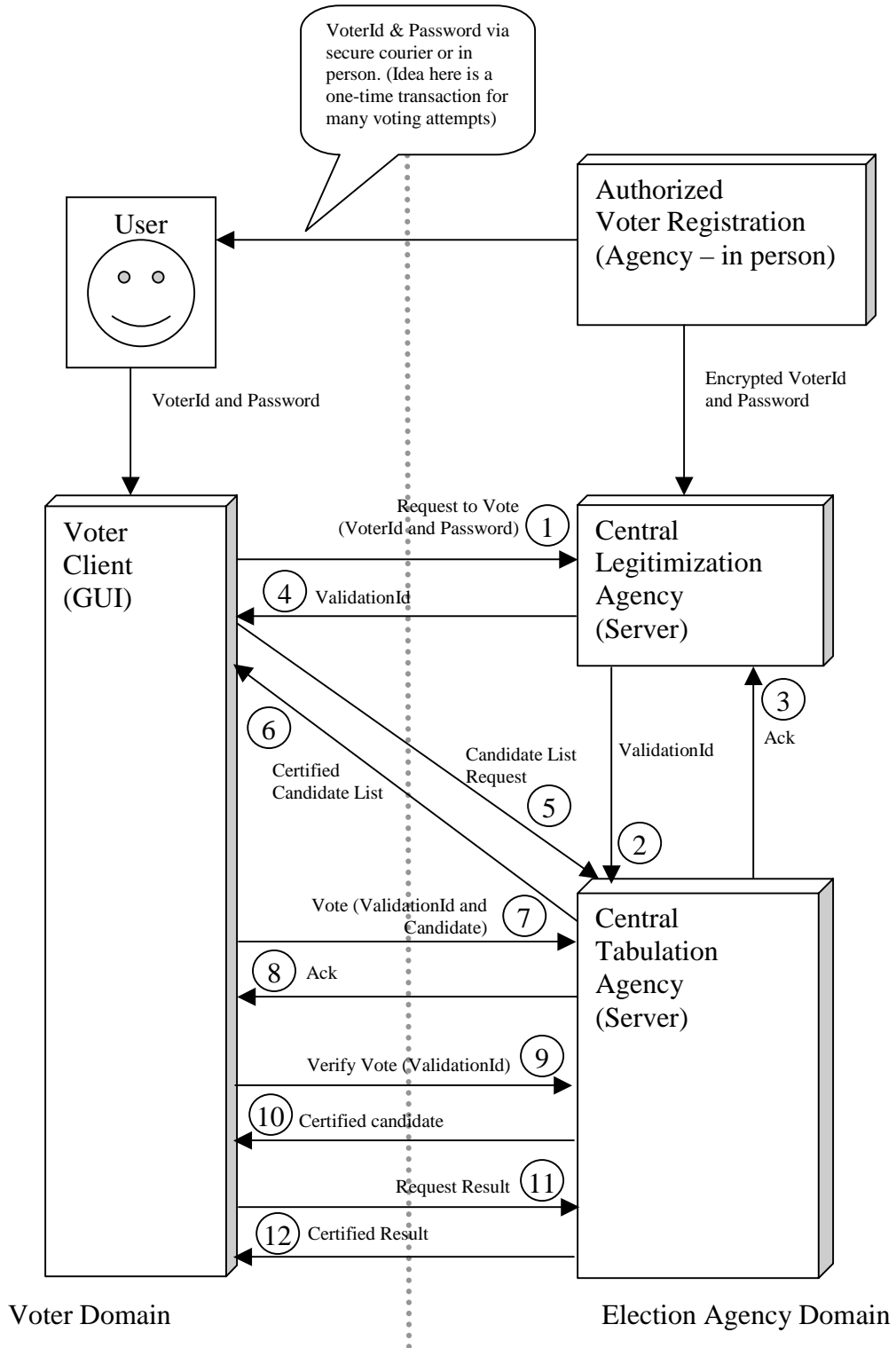
### Voter Client

The **Voter Client** is responsible for allowing the user to input a previously provided VoterId and Password to be sent securely to the Central Legitimization Agency to securely obtain a unique ValidationId. After receiving the voter's ValidationId, the Voter Client will securely request and retrieve a list of candidates from the Central Tabulation Facility, to provide the voter with a choice in casting their vote, which is also securely sent to the CTF. The Voter Client can also verify the candidate for whom the voter's ballot has been registered through a secure request with the Central Tabulation Facility using the voter's ValidationId.

The Voter Client can also display the certified final result of the election from the Central Tabulation Facility.

# System Overview

The following is a diagram providing an overview of the components and dataflow which is implemented in the Online Voting System.



## Protocol Description

The following protocol descriptions correspond to the messages outlined in the system overview (the items listed below reference the numbers in the diagram). In particular, the security aspects of each message (i.e. how each message achieves the goals of the system) are described here. In the following protocol descriptions, the term “Voter” refers to the Voter Client.

### 1) Voter requests ValidationId from CLA.

#### *Messaging Description*

- i. The voter initiates a connection with the machine it assumes to be the CLA.
- ii. The CLA immediately sends its public RSA key and certificate.
- iii. The voter reads in the CLA’s public key and verifies the certificate using its CA public key. Once verified, the voter knows that even though the other machine may not be an actual CLA, *we trust that anything we encrypt with the CLA’s public key can only be read by an actual CLA.*
- iv. The Voter generates a symmetric Blowfish key, and encrypts that with the CLA’s public key and sends it to the CLA. The Voter then sends its login (VoterId, Password, and public key) to the CLA using the symmetric cipher.
- v. The CLA decrypts the symmetric key using its RSA private key, and decrypts the login with the symmetric key.
- vi. The CLA generates a ValidationId, and proceeds to protocol item 2, below.

If the case is that the voter already has a ValidationId, the CLA proceeds to protocol item 4, below returning a “REPEAT” code.

### 2) CLA requests to add an authorized ValidationId to CTF.

#### *Messaging Description*

- i. The CLA initiates a connection with the machine it assumes to be the CTF.
- ii. The CTF immediately sends its public RSA key and certificate.
- iii. The CLA reads in the CTF’s public key and verifies the certificate using its CA public key. Once verified, the CLA knows that even though the other machine may not be an actual CTF, *we trust that anything we encrypt with the CTF’s public key can only be read by an actual CTF.*
- iv. The CLA generates a symmetric Blowfish key, and encrypts that with the CTF’s public key and sends it to the CTF. The CLA then sends its request (public key, certificate, ValidationId) to the CTF using the symmetric cipher. The CLA then sends its RSA signature of the request to the CTF.

- v. The CTF decrypts the symmetric key using its RSA private key, and decrypts the request with the symmetric key. Before adding the ValidationId to its authorized pool the CTF ensures that the CLA actually sent the message by:
  - a. Verify the CLA's RSA public key with its certificate using the CA public key. This ensures that only the actual CLA knows the private RSA key to RSA sign the message.
  - b. The CTF checks the RSA signature sent by the CLA using the certified CLA public key.
- vi. If the signature matches, then the ValidationId is added to the system.

### **3) CTF returns acknowledgement of new ValidationId to CLA.**

#### *Messaging Description*

- i. Using the symmetric cipher and key previously set up in step 2, the CTF encrypts an acknowledgement of "OK" back to the CLA. The CLA also signs the message. In case of error, the response of ("ERROR", errorMessage) would be sent back to the CLA.
- ii. The CLA verifies the response before returning the ValidationId to the voter. We have previously received the CTF public key and certificate in step 2.

### **4) CLA returns ValidationId to Voter.**

#### *Messaging Description*

- i. Using the symmetric cipher and key previously set up in step 1, the CLA encrypts an acknowledgement of "NEW" or "REPEAT" along with the vote ValidationId back to the voter. The CLA also signs the message. In case of error, the encrypted response of ("ERROR", errorMessage) would be sent back to the Voter.
- ii. The Voter verifies the response before requesting the list of candidates from the CTF. We have previously received the CLA public key and certificate in step 1.

## 5) Voter requests candidate list from CTF.

### *Messaging Description*

- i. The voter initiates a connection with the machine it assumes to be the CTF.
- ii. The CTF immediately sends its public RSA key and certificate.
- iii. The voter reads in the CTF's public key and verifies the certificate using its CA public key. Once verified, the voter knows that even though the other machine may not be an actual CTF, *we trust that anything we encrypt with the CTF's public key can only be read by an actual CTF.*
- iv. The Voter generates a symmetric Blowfish key, and encrypts that with the CTF's public key and sends it to the CTF. The Voter then sends its simple request of "LIST" to the CTF using the symmetric cipher.
- v. The CTF decrypts the symmetric key using its RSA private key, and decrypts the request with the symmetric key.
  - a. For the candidate list, no authentication is required of the user.

## 6) CTF returns candidate list to voter.

### *Messaging Description*

- i. Using the symmetric key established in protocol item 5, the CTF encrypts the list of candidates and sends it to the voter. The CTF then signs the list of candidates and sends that signature to the voter.
- ii. The Voter decrypts the candidate list and verifies the signature of the CTF. We have previously received the CTF public key and certificate in step 5.

## 7) Voter requests to vote to CTF.

### *Messaging Description*

- i. The voter initiates a connection with the machine it assumes to be the CTF.
- ii. The CTF immediately sends its public RSA key and certificate.
- iii. The voter reads in the CTF's public key and verifies the certificate using its CA public key. Once verified, the voter knows that even though the other machine may not be an actual CTF, *we trust that anything we encrypt with the CTF's public key can only be read by an actual CTF.*
- iv. The Voter generates a symmetric Blowfish key, and encrypts that with the CTF's public key and sends it to the CTF. The Voter then sends its vote request consisting of ("VOTE", ValidationId, candidate) to the CTF using the symmetric cipher.

- v. The CTF decrypts the symmetric key using its RSA private key, and decrypts the request with the symmetric key.
- vi. Before the vote is cast, the CTF verifies that the ValidationId is in the system, but has not been used, and that the candidate exists. Having satisfied this criterion, the CTF records the vote and returns the candidate confirmation to the voter.

**8) CTF returns vote acknowledgement to voter.**

*Messaging Description*

- i. Using the symmetric key established in protocol item 7, the CTF encrypts the acknowledgement of (“OK”) and sends it to the voter. In case of error, the encrypted response would be (“ERROR”, errorMessage). The CTF then signs the message and sends that signature to the voter.
- ii. The Voter decrypts the result and verifies the signature of the CTF. We have previously received the CTF public key and certificate in step 7.

**9) Voter requests to verify his/her vote.**

*Messaging Description*

- i. The voter initiates a connection with the machine it assumes to be the CTF.
- ii. The CTF immediately sends its public RSA key and certificate.
- iii. The voter reads in the CTF’s public key and verifies the certificate using its CA public key. Once verified, the voter knows that even though the other machine may not be an actual CTF, *we trust that anything we encrypt with the CTF’s public key can only be read by an actual CTF.*
- iv. The Voter generates a symmetric Blowfish key, and encrypts that with the CTF’s public key and sends it to the CTF. The Voter then sends its confirmation request consisting of (“CHECK”, ValidationId) to the CTF using the symmetric cipher.
- v. The CTF decrypts the symmetric key using its RSA private key, and decrypts the request with the symmetric key.
- vi. The CTF verifies that the ValidationId is in the system, and has been used. The CTF returns the candidate confirmation to the voter.

## 10) CTF returns vote verification status and selected candidate to voter.

### *Messaging Description*

- i. Using the symmetric key established in protocol item 9, the CTF encrypts the acknowledgement of (“OK”, ballot candidate) and sends it to the voter. In case of error, the encrypted response would be (“ERROR”, errorMessage). The CTF then signs the message and sends that signature to the voter.
- ii. The Voter decrypts the result and verifies the signature of the CTF. We have previously received the CTF public key and certificate in step 9.

## 11) Voter requests election results from CTF.

### *Messaging Description*

- i. The voter initiates a connection with the machine it assumes to be the CTF.
- ii. The CTF immediately sends its public RSA key and certificate.
- iii. The voter reads in the CTF’s public key and verifies the certificate using its CA public key. Once verified, the voter knows that even though the other machine may not be an actual CTF, *we trust that anything we encrypt with the CTF’s public key can only be read by an actual CTF.*
- iv. The Voter generates a symmetric Blowfish key, and encrypts that with the CTF’s public key and sends it to the CTF. The Voter then sends its simple request of “RESULT” to the CTF using the symmetric cipher.
- v. The CTF decrypts the symmetric key using its RSA private key, and decrypts the request with the symmetric key.

For the election result, no authentication is required of the user.

## 12) CTF returns election results.

### *Messaging Description*

- i. Using the symmetric key established in protocol item 11, the CTF encrypts the election results and sends it to the voter. The CTF then signs the results and sends that signature to the voter.
- ii. The Voter decrypts the results and verifies the signature of the CTF. We have previously received the CTF public key and certificate in step 11.

## Requirement Analysis

The following section analyses how the security features of the Online Voting System satisfies the requirements:

- *Only authorized voters can vote.*

Each is given a VoterId and Password pair that is computationally infeasible for an attacker to guess at a valid pair. After certifying the CLA, the voter encrypts this information and sends it to the CLA. The CLA verifies the login, and if it has not done so already for the voter, it securely requests the CTF to add a ValidationId (also computationally infeasible for an attacker to produce a valid Id). Only those ValidationIds recorded in the CTF can cast a vote.

- *No one can vote more than once.*

Each voter only has one VoterId and Password pair. Only one ValidationId is generated per VoterId and Password pair, and only one vote is allowed per ValidationId.

- *No one can determine for whom anyone else voted.*

This property is maintained mostly due to the fact that there are two separate agencies, the CLA and the CTF. The CLA will authenticate the voter, and authorize the voter to vote. The CTF records the votes using the authorization codes. Therefore, the candidate selected and the VoterId is never on the same machine. All transactions regarding the actual casting of the ballot and the ValidationId are secured by a cipher.

- *No one can duplicate anyone else's votes.*

Since the voter keeps his VoterId, Password and any ValidationId secret to the general public, and only sends these to the CLA or CTF after it has verified their public key certificates, nobody can know the ValidationId of someone else.

- *Every voter can make sure that his vote has been taken into account in the final tabulation.*

The CTF provides a "CHECK" functionality where using the ValidationId, it will return the candidate selected secured with the requestor's symmetric cipher key.

## Implementation

The above-mentioned protocol was implemented using Java and the Cryptix toolkit for the Java Cryptography Extensions (JCE). The system uses RSA for public key cryptography and Blowfish™ for a symmetric cipher, and SHA-1 as a Hash/Signature generation algorithm.

Please see the references section to obtain further information about the Cryptix toolkit.

The following sections describe finer details about building and running the sample implementation as well as descriptions of the data files that are required for execution.

## User Guide

The following describes how to invoke and use the system. In order to simplify the testing process, we have packaged the class files with the Cryptix library into one jar file. This eliminates the need to set additional Classpaths.

The following instructions illustrate how to test the system using the executable jar file, however the procedure to run via the direct class files is similar and is explained in `/src/README.txt`.

### Quick Start

The following describes how to quickly invoke and test the program:

1. In the project directory, invoke the `runServices.bat` file. This will invoke the CLA server, the CTF server and the Voter client on the localhost. If you are not executing this on the SCS network, you may have to set the `JAVA_HOME` environment variable in `setpaths.bat`.
2. Use a VoterId and Password combination contained in the `CLA.voters` to login to the system. The system has a straightforward interface, messages to the user are placed within a text area to the left of the GUI, and messages regarding the underlying security and protocol are delivered to the console window. Generating the first symmetric keys takes a few moments, please be patient.

## System Data Files

On the diskette provided, you will find the following files:

- CA.private CA's (Certificate Authority) Private RSA Key File.
- CA.public CA's (Certificate Authority) Public RSA Key File.
  
- CLA.cert CLA's (Central Legitimization Agency) Certificate.
- CLA.private CLA's (Central Legitimization Agency) Private RSA Key File.
- CLA.public CLA's (Central Legitimization Agency) Public RSA Key File.
- CLA.voters CLA's (Central Legitimization Agency) VoterId to ValidationId records.
  
- CTF.cert CTF's (Central Tabulating Facility) Certificate.
- CTF.private CTF's (Central Tabulating Facility) Private RSA Key File.
- CTF.public CTF's (Central Tabulating Facility) Public RSA Key File.
- CTF.candidate CTF's (Central Tabulating Facility) List of candidates for election.
  
- Voter.private Voter's Private RSA Key File.
- Voter.public Voter's Public RSA Key File.
  
- VirtualElectionBooth.jar Executable Jar File containing all Source Code and Libraries needed for execution.
- \src\ Directory containing Source Code for the Virtual Election Booth Executables.

## System Execution

Before running the system you may need to modify setpaths.bat to ensure that the JAVA\_HOME environment variable correctly points to the base of your java installation directory.

There are a series of batch files in this directory that automate the execution of several components of the system:

- setpaths.bat                Sets the environment variable of the location of the java installation directory for compilation and execution.
- runCLA.bat                Batch file to execute the CLA server, you must pass in the host of the CTF server on the command line.
- runCTF.bat                Run the CTF server, no parameters are required.
- runCertGen.bat            Generate a certificate, see the javadocs in the /src/javadoc directory for details on the arguments to RSACertificateGenerator.java.
- runCertVerify.bat        Verifies a certificate, see the javadocs in the /src/javadoc directory for details on the arguments to RSACertificateVerifier.java
- runKeyGen.bat            Generates a RSA key, see the javadocs in the /src/javadoc directory for details on the arguments to RSAKeyGenerator.java.
- runVoter.bat             Batch file to run the Voter Client, you must pass in the CLA host and CTF host on the command line.
- runServices.bat          Batch file to run all three services in new windows (CTF, CLA, Voter) on the local machine.
- renGen.bat                Batch file to generate the required RSA keys, RSA certificates, the voter list and the candidate list needed for execution.

## Build Instructions

Before building the system you must modify setpaths.bat to ensure that the JAVA\_HOME environment variable correctly points to the base of your java installation directory.

The system is built by compiling all source files in the source (\virtualElectionBooth\src) directory, building a jar file out of these classes and the classes that make up the Cryptix Security Provider libraries, and placing that jar in this directory.

This process is automated and can be started by running the following in this directory:

- makeJar.bat                      Batch file for automatically generating the executable jar file from the source files and Cryptix Security Provider libraries. Requires the location of the java installation directory as specified by setpaths.bat

The resulting jar file has the advantage of being compressed and requiring no environment variables to be set in order to run.

The class ServiceProvider was added to the source to allow for routing of commands when executing the jar file since jar files only allow for one class to be allocated as the class to execute.

As a result, when executing the jar file, the first parameter must be the name of the service to be run.

For example: java -jar VirtualElectionBooth.jar CTFService

## **Javadocs of Source Code**

Sun provides a javadoc tool for automatic generation of API documentation of given Java source code provided the comments in the source code follow the appropriate format.

We have generated API docs of our source code and it is available for perusal in the /src/javadoc/ directory.

Please see the file README.txt in the /src/ directory for more information.

## References

### General

[1] Schneier, Bruce, *Applied Cryptography*, Second Edition, Jon Wiley & Sons, 1996

[2] Stallings, William, *Cryptography and Network Security*, Second Edition, Prentice Hall, 1999

[3] *RSA Laboratories – Public Key Cryptography Standards (PKCS)*  
<http://www.rsasecurity.com/rsalabs/pkcs/index.html>, 2001

[4] *Javadoc 1.2*  
<http://java.sun.com/products/jdk/1.2/docs/tooldocs/javadoc/>, 2001

### Implementation Specific

[4] *Cryptix JCE*  
<http://www.cryptix.org/products/jce/index.html>, 2001

[5] *Java™ Cryptography Extension (JCE) 1.2.1*  
<http://www.java.sun.com/products/jce/index.html>, 2001

[6] *Java™ Cryptography Architecture API Specification & Reference*  
<http://java.sun.com/products/jdk/1.2/docs/guide/security/CryptoSpec.html>, 1999