Protected Mobility LLC.

# PMCryptolib

Version 1.0

# FIPS 140-2 Non-Proprietary Security Policy

**Level 1 Validation**

**Document Version 1.04**

# Table of Contents

# Table of Figures

# Table of Tables

# 1 Introduction

## 1.1 Purpose

This is a non-proprietary Cryptographic Module Security Policy for the PMCryptolib version 1.0 from Protected Mobility LLC. This security policy provides detailed information relating to the PMCryptolib version 1.0 module as it relates to the FIPS 140-2 Level 1 security requirements along with instructions on setting up the module in a secure FIPS 140-2 mode.

## 1.2 Product Overview

The PMCryptolib is a FIPS 140-2 Level 1, software module which provides a number of cryptographic services. The PMCCryptolib is purely a software product.

The FIPS 140-2 validation of the PMCryptolib module includes the follow components:

- libpmcrypto.so -binary file for use on Android
  or
- cryptolib.framework –binary file for use on iOS

For the purposes of the FIPS 140-2 validation, the Protected Mobility Cryptographic Library object modules are linked into the PMCryptolib .so, or .framework file by Protected Mobility. This linked library is then linked in a run-time environment to some other library, application, or system, running on an Android or iOS operating system. In FIPS 140-2 terminology, the Protected Mobility Cryptographic Library is a multi-chip standalone cryptographic module.

The PMCryptolib module contains cryptographic functions that are accessible to the software developer through the Application Programming Interface (API). The PMCryptolib cryptographic functions include encryption and decryption using the Advanced Encryption Standard (AES), message hashing , keyed message hashing, digital signatures, digital signature verification, key pair generation and random bit generation. The library also contains a , generation of a common point on the elliptic curve function[1] for use by the underlying application and a PKCS #5 function for generating keying material which can be used by the underlying application. The PMCryptolib module is a C programming language API and supports platforms/operating systems such as Google Android and Apple's iOS. The devices that support these operating systems are single operator mobile computing devices.

The PMCryptolib module was tested and validated for FIPS 140-2 Level 1 on the platforms defined in Table 1.

[1] For the purpose of the PMCryptolib module the "Generate Common Point on the Elliptic Curve" function does not have a cryptographic functionality inside the boundary of the module but is a function that is available to an application developer.

**Table 1 – Tested Operational Environments**

| Operating System | Processor | Library File |
|---|---|---|
| Android 2.2 | ARM Cortex-A8 | libpmcrypto.so |
| Android 2.3 | ARM Cortex-A9 | |
| Android 3.0 | ARM Cortex-A9 | |
| iOS 4.2 | ARM 6 | cryptolib.framework |
| iOS 4.2 | ARM 7 | |
| iOS 4.3 | ARM 7 | |

## 1.3 Cryptographic Module Specification

The PMCryptolib is a software-based cryptographic module evaluated for use on Android OS versions 2.2, 2.3 and 3.0 and Apple iOS versions 4.2 and 4.3. The logical cryptographic boundary of the module is the PMCryptolib software binary file (linkable library) which is encapsulated in the libpmcrypto.so file for the Android operating systems and the cyptolib.framework file for the iOS operating systems. The PMCryptolib module is contained in the physical boundary of the General Purpose Computing Mobile Device (GPCMD) on which the module resides as shown in Figure 1. The PMCryptolib is solely contained in the block labeled PMCryptolib Module.

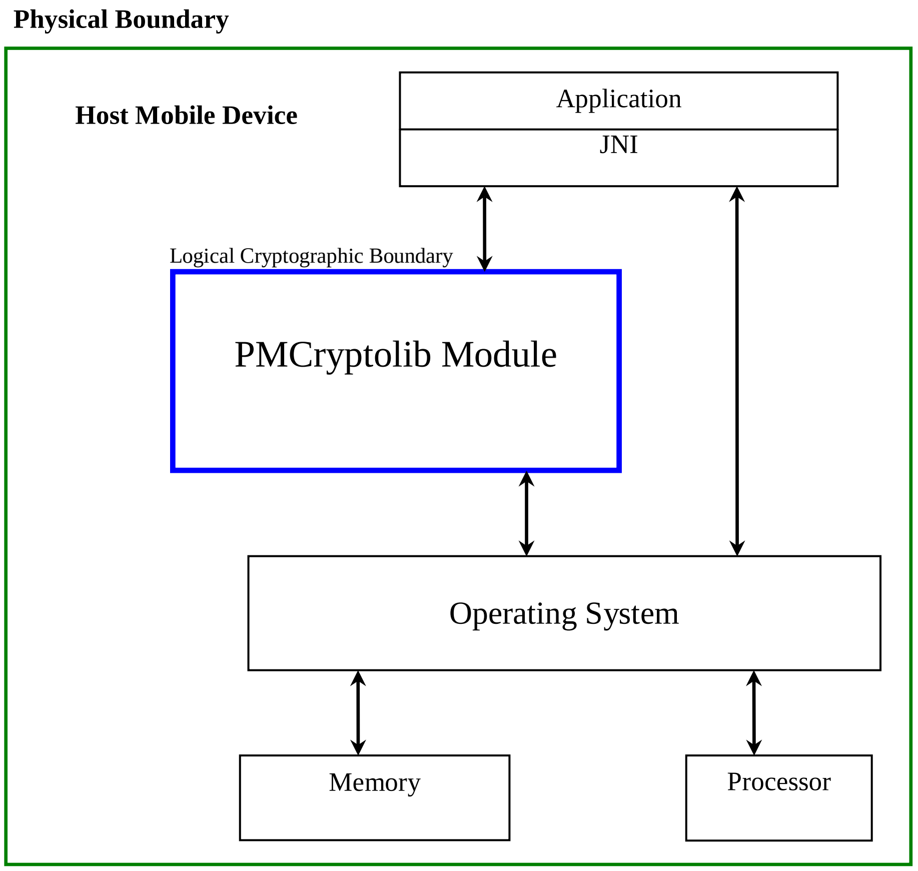

**Figure 1 - PMCryptolib Block Diagram**

The PMCrypto module has five distinct functionalities (Figure 2), generating keying material (PKCS #5), hashing and keyed hashing (SHA2 and HMAC SHA2), elliptic curve functions, Encryption and Decryption, and generating random data. The keying material produced by the PKCS #5 can be used by the application linking to the library and is outside the scope of the FIPS 140-2 Level 1 validation.

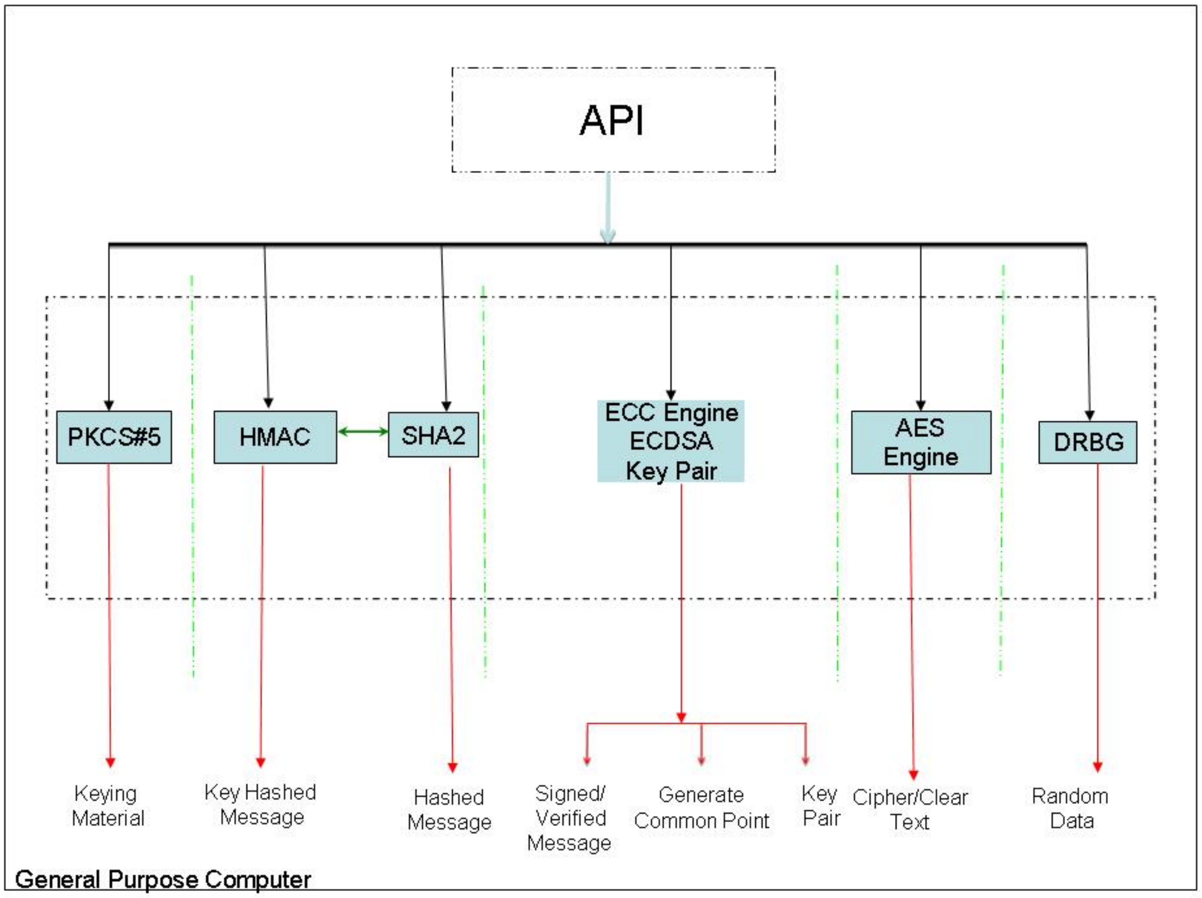

**Figure 2 – Functional Divisions of the PMCryptolib**

Per FIPS 140-2 terminology, the PMCryptolib is a multi-chip standalone module that meets overall level 1 FIPS 140-2 requirements. The PMCryptolib is validated as shown in Table 2.

**Table 2 - Security Level per FIPS 140-2 Section**

| Section | Section Title | Level |
|---|---|---|
| 1 | Cryptographic Module Specification | 1 |
| 2 | Cryptographic Module Ports and Interfaces | 1 |
| 3 | Roles, Services, and Authentication | 1 |
| 4 | Finite State Model | 1 |
| 5 | Physical Security | NA |
| 6 | Operational Environment | 1 |
| 7 | Cryptographic Key Management | 1 |
| 8 | EMI/EMC | 1 |
| 9 | Self-tests | 1 |
| 10 | Design Assurance | 3 |
| 11 | Mitigation of Other Attacks | NA |

### 1.3.1 Mode of Operation

The PMCryptolib is a cryptographic library that functions only in a FIPS Approved mode of operation. The module is in the FIPS Approved mode of operation once the GPCMD on which the library resides is powered on and

PMCryptolib passes the power-up self-tests successfully. The module does not have a non-approved mode of operation. The module will also run all the power- up self-tests each time the library gets loaded into working memory. This is accomplished if the application which links to the library is not running on the device (it cannot be running in the background) and then gets loaded into memory by starting the application. If the module passes the self-test and is operational it indicates that the module is in a FIPS Approved mode of operation. If the self-tests are not passed the module will not operate.

## 1.4 Module Ports and Interfaces

The logical interfaces of the PMCryptolib are the Application Programming Interfaces (API) of the module library. Physically, ports and interfaces are provided by the GPCMD on which the PMCryptolib module is installed. The interfaces can be categorized into following logical interfaces defined by FIPS 140-2:

- Data Input Interface
- Data Out Interface
- Data Control Interface
- Status Output Interface

All of these logical interfaces are described in the following table:

**Table 3 – FIPS 140-2 Logical Interfaces**

| FIPS 140-2 Interface | Data Type | PMCryptolib Logical Interfaces |
|---|---|---|
| Data Input | Plaintext<br>Ciphertext | API function calls that have input data that is processed through the function arguments. |
| Data Output | Plaintext<br>Ciphertext | API function calls that return data that was processed through the function arguments. |
| Control Input | Plaintext | Call to the API to initiate the module functions |
| Status Output | Plaintext | Result codes returned by the API function |

## 1.5 Roles, Services and Authentication

### 1.5.1 Crypto Officer and User Roles

Two roles are supported by the module: a Crypto-Officer (CO) role and a User role. The role of crypto-officer or user is implicitly assumed by the service. The Crypto-Officer (CO) is responsible for installation/removal of the application containing the PMCryptolib on to the GPCMD on which the module is validated. The user implements an API function call to the PMCryptolib. Descriptions of the services available to each role are provided in the tables in the services section below.

### 1.5.2 Services

There are two services assigned to the crypto-officer. The first is the installation/removal of the application containing the PMCryptolib module onto/from the GPCMD on which the module is validated. The second service is performing of the power-up self-test on demand. The services available to the crypto-officer are provided in Table 4.

**Table 4 – Mapping of Crypto-Officer Services to Inputs, Outputs, CSPs, and Type of Access**

| Service | Description | Input | Output | CSP and Type of Access |
|---|---|---|---|---|
| Installation/Removal | Installation or removal of the application containing the PMCryptolib onto/from the GPCMD | N/A | N/A | None |
| Perform Self-tests | Perform self-tests on demand | Power off then power on the device on which the PMCryptolib or Force stop the application linking to the PMCryptolib and then reload it into working memory. | Success or Failure indicator of self-tests | None |

There are several services assigned to the user role. Descriptions of the services available to the User role are provided in the Table 5.

**Table 5 – Mapping of User Services to Inputs, Outputs, CSPs, and Type of Access**

| Service | Description | Input | Output | CSP and Type of Access |
|---|---|---|---|---|
| Show Status | The show status service is the Return from each of the API functions. | API function call | Return code indicating success or failure | None |

| Service | Description | Input | Output | CSP and Type of Access |
|---|---|---|---|---|
| Encrypt Data | Data is encrypted using the AES algorithm | API function call | Cipher text<br><br>Return code indicating success | Cipher key-read |
| Decrypt Data | Data is decrypted using the AES algorithm | API function call | Plain text<br><br>Return code indicating success | Cipher key- read |
| Create key pair | Generates a public/private key pair for use with the digital signature algorithm | API function call | Public key, Private key<br><br>Return code indicating success | public key-write<br>private key-write |
| Generate Common Point on Elliptic Curve | Uses the elliptic curve to generate a Common Point on the Elliptic Curve. The Common Point is generated from values passed into the module from outside application. | API function call | common point on elliptic curve<br><br>Return code indicating success | public key-read<br>private key- read |
| Validate Public Key | This function will validate that the given public key is valid on the NIST-recommended curve (P-256, P-384, P-521). | API function call | Success or Failure | public key-read |
| Verify Domain | This function will validate the NIST specified prime curve parameters for elliptic curve with a specified number of bits. | API function call | Success or Failure | None |
| ECDSA sign | Creates a digital signature | API function call | Digital signature<br><br>Return code indicating success | Private key-read |
| ECDSA verify | Verifies a digital signature | API function call | Return code indicating success | Public key-read |
| Calculate Message Digest | Create a SHS message digest of data | API function call | Hashed message | none |

| Service | Description | Input | Output | CSP and Type of Access |
|---|---|---|---|---|
| Create Keyed HASH (HMAC) | Computes a keyed hash | API function call | Keyed hash<br><br>Return code indicating success | Key -read |
| Generated Random Bits | Generates random data | API function call- entropy, personalization string, nonce | Random data<br><br>Return code indicating success | |
| Zeroize | All keys, CSP and memory allocated to the API used by the process get zeroized when API call is complete | Completed function call | None | All used in the function that was called (write) |
| PKCS #5 | Combines a password with salt to produce keying material | Password | Keying material | None |
| Perform power-up self-tests | Power-up self-tests are performed automatically | Power is applied to the device that contains the module | Success or Failure indicator of self-tests | none |

### 1.5.3 Authentication

The PMCryptolib does not employ any authentication mechanisms. FIPS 140-2 does not require authentication for a level 1 module. If an authentication mechanism is desired for an operator role it can be obtained using the log in feature of the GPCMD.

## 1.6 Physical Security

The physical security requirements do not apply to the PMCryptolib Module. The PMCryptolib module is a software module and the physical security is provided by the host platform (FIPS 140-2).

The module is a software module and was tested on standard GPCMD platforms that meet the applicable Federal Communication Commission (FCC) Electromagnetic Interference (EMI) and Electromagnetic Compatibility (EMC) requirements for business use as defined in Subpart B of FCC Part 15.

## 1.7 Operational Environment

The PMCryptolib cryptographic module is a multi-chip standalone cryptographic module that runs on a GPCMD. The PMCryptolib cryptographic module is validated on the following operational environments: Android OS versions 2.2, 2.3 and 3.0 and Apple iOS versions 4.2 and 4.3.

The module supports only single users. The operational environments ensure that process space and memory locations allocated to the PMCryptolib cannot be accessed by outside processes. The PMCryptolib is a non-modifiable binary file.

## 1.8 Cryptographic Key Management

The PMCryptolib implements the following FIPS-approved algorithms:

- AES Algorithm (certificate #1716 )
- DRBG 800-90 Algorithm (certificate #108)
- SHS Algorithm (certificate #1499 )
- HMAC Algorithm (certificate #991 )
- ECDSA Algorithm (certificate #222 )

The PMCryptolib does not implement any non-FIPS approved algorithms.

### 1.8.1 Key Generation

The PMCryptolib implements two instances of the FIPS Approved SP800-90 (validation #108) Deterministic Random Bit Generator (DRGB) for the generation of random bits. The first instance of the DRBG is an ephemeral DRBG which seeds the primary DRBG. The primary DRBG generates random bits that feed into the FIPS Approved ECDSA algorithm for the Approved generation of ECDSA keys.

### 1.8.2 Key Establishment & Distribution

The PMCryptolib module does not perform any type of key establishment or distribution. Any key establishment or distribution is performed by the underlying application and is outside the scope of the PMCryptolib module validation.

### 1.8.3 Key entry/output

Keys are input to the PMCryptolib from the application that invokes the module, which is running on the same GPCMD. Keys are output from the PMCryptolib to the application that invokes the module, which is running on the same GPCMD. Keys enter in to and output from the module in plaintext form. (FIPS 140-2 IG 7.7).

### 1.8.4 Key Storage

The PMCryptolib module does not provide permanent storage of keys or key material. The keys or key material is entered from the GPCMD to the PMCrypto module via an API function call. The keys are maintained in volatile memory only until the function call is complete. No keys or CSPs are maintained by the module once the API function call is complete.

The protection of keys is performed by the underlying OS memory management mechanisms

### 1.8.5 Key Zeroization

Prior to returning from an API function call the PMCrypto module runs zeroization on memory space allocated by the function call to zeroize any keys or CSPs used by the module.

The module supports the following critical security parameters:

**Table 6 – List of Cryptographic Keys, Cryptographic Key Components, and CSPs**

| Key/CSP | Key/CSP Type | Generation / Input | Output | Storage | Zeroization | Use |
|---|---|---|---|---|---|---|
| Seed | The SP800-90 CTR_DRBG uses entropy, a nonce and a personalization string to create DRBG seed | SP800-90 using entropy, nonce and personalization string | Never exits the module | Stored in volatile memory for the API call. | Performed automatically when the API call is complete | Seeding the DRGB 800-90 |
| AES Cipher key | CBC-128 & 256 GCM 128 & 256 | Passed in from the applicataion | Never exits the module | Stored in volatile memory for the API call. | Performed automatically when the API call is complete. | CBC & GCM Used to encrypt and decrypt messages |
| Elliptic curve public key | NIST–recommended Curves P-256, P-384, P-521 | Generated in the module or passed in from the application | Key is output to the application executing operations on the same GPCMD in plaintext form.. | Stored in volatile memory for the API call. | Performed automatically when the API call is complete. | Used by the linking application |
| Elliptic curve private key | NIST–recommended Curves P-256, P-384, P-521 | Generated in the module or passed in from the application | Key is output to the application executing operations on the same GPCMD in plaintext form. | Stored in volatile memory for the API call. | Performed automatically when the API call is complete. | Used by the linking application |
| HMAC SHA | 256, 384, 512 | Key is input from the application executing operations on the same GPCMD in plaintext form. | Never exits the module | Stored in volatile memory for the API call | Performed automatically when the API call is complete. | Used to create a keyed hash message |
| HMAC SHA 256 | 256 | Generated from known information about the crypto module. | Never exits the module | Stored in volatile memory for the software integrity test at power-up or on demand | Performed automatically when the integrity test is complete. | To Create HASH for the software integrity test. |

| Key/CSP | Key/CSP Type | Generation / Input | Output | Storage | Zeroization | Use |
|---|---|---|---|---|---|---|
| HMAC SHA 256 HASH | HMAC SHA 256 HASH | Hard coded in the module binary | Never exits the module | In module binary | Uninstalling the module | Software integrity test |

## 1.9 Self-Tests

The PMCryptolib performs a series of self-tests in order to ensure that the module is functioning properly. It performs power up self-test automatically when the module is powered up (device on which the module is running is powered up and/or when the library gets loaded on to the device) and conditional self-tests when the specific security function is invoked.

- Power-Up Self-Tests:
    - Software Integrity Test – HMAC SHA-256
    - Known Answer Tests (KATs)
        - AES CBC KAT (128, 256)
        - AES GCM KAT (128, 256)
        - SHA-256 KAT
        - SHA-384 KAT
        - SHA-512 KAT
        - HMAC SHA-256 KAT
        - HMAC SHA-384 KAT
        - HMAC SHA-512 KAT
        - ECDSA KAT (P-256) SigGen/SigVer/KeyGen
        - ECDSA KAT (P-384) SigGen/SigVer/KeyGen
        - ECDSA KAT (P-521) SigGen/SigVer/KeyGen
        - DRBG KAT

The PMCryptolib performs the following conditional self-tests:

- Continuous Random Number Generator (ephemeral DRBG)
- Continuous Random Number Generator (primary DRBG)
- Pairwise Consistency- Performed on the Elliptic Curve Key Generation

If self-tests are passed it is indicated by the module entering a normal operational state. If any of the power up self-tests fail, the module returns an error, goes into a catastrophic failure state and does not operate. The errors will be in the form of codes returned by the self-test API.

The corrective action for any self test failure is to power cycle the device on which the application linking to the PMCryptolib is running or to force stop the application on the device and then start it back up which reloads it into the working memory and performs the power-up self-tests. Consult the device manual for instructions to verify the application is not running in the background on the device. The operator may need to go into the settings to force stop the application. If any self-test continues to fail, remove the application linking to the PMCryptolib module from the device and try reinstalling it. If any of the self-tests continues to fail contact the PMCryptolib developers.

The power on self-tests can be performed by the operator on demand as described in the paragraph above for corrective action for failure of a self-test.

## 1.10 Design Assurance

Configuration management for all of the Protected Mobility source code and document files is provided by a source code management system that requires a username and password to access any files in the system. The system keeps track of who made changes, what changes were made and the date the changes were made. The system will increment the unique id assigned to the file anytime changes are made.

## 1.11 Mitigation of Other Attacks

The PMCryptolib does not perform any mitigation of other attacks.

# 2 Secure Operation

The PMCryptolib meets Level 1 requirements for FIPS 140-2. The sections below describe how to configure and maintain the module in a FIPS-Approved mode of operation. There are no specific installation instructions for a developer to use the PMCryptolib module in an application outside of the normal linking to a PMCryptolib binary file in the application.

## 2.1 Crypto-Officer Guidance

### 2.1.1 Initial Setup

The crypto-officer is responsible for the installation of an application using the PMCryptolib on an operational environment on which the module has been validated. The module is a binary object module (linked library) linked to an application. The binary code for the object module has a computed HMAC SHA256 hash value embedded in the code for the software integrity test.

### 2.1.2 Management

The crypto-officer can perform the power-on self-tests on demand by "power cycling the module." This can be done by unloading and then reloading the application into the working memory of the device. This is not necessarily accomplished by closing the application. Some devices keep applications running in the background even when the application has been closed. The application will need to be force stopped using the task manager. The crypto-officer can also perform the power-up self-tests on demand by power cycling the device on which the application that links to the library is running.

## 2.2 User Guidance

### 2.2.1 Setup/Operation

The user is the process in the application that performs the function calls to the module's API's. There are no special set-up or operation instructions. The module only operates in a FIPS Aproved mode.

# 3 Acronyms

**Table 7 - Acronyms**

| Acronym | Definition |
|---|---|
| AES | Advanced Encryption Standard |
| API | Application Programming Interface |
| CSP | Critical Security Parameter |
| DRBG | Deterministic Random Bit Generator |
| FIPS | Federal Information Processing Standard |
| GPCMD | General Purpose Computing Mobile Device |
| HMAC | Keyed-Hashing for Message Authentication |
| KAT | Known Answer Test |
| OS | Operating System |
| SHA | Secure Hash Algorithm |
| | |