Botan FIPS 140-2 Security Policy

Jack Lloyd
lloyd@randombit.net

Contents

1 Introduction

Note that this is a draft, and almost certainly does not comply with what FIPS 140-2 wants (also it's incomplete). In any case, there is no way for me to afford paying the validation lab, so this is all theoretical.

I would welcome comments from people who are familiar with the FIPS 140 process. I am currently basing this off a few dozen other security policies and the FIPS itself.

1.1 Purpose

This document is a security policy for the Botan C++ crypto library for use in a FIPS 140-2 Level 1 validation process. It describes how to configure and use the library to comply with the requirements of FIPS 140-2.

This document is non-proprietary, and may be freely reproduced and distributed in unmodified form.

1.2 Product Description

The Botan C++ crypto library (hereafter "Botan" or "the library") is an open source C++ class library providing a general-purpose interface to a wide variety of cryptographic algorithms and formats (such as X.509v3 and PKCS #10). It runs on most Win32 and POSIX-like systems, including Windows NT/2000/XP, MacOS X, Linux, Solaris, FreeBSD, and QNX. However, only versions running on *(goal:)* Windows XP, Linux, and Solaris have been validated by FIPS 140-2 at this time.

1.3 Algorithms

The library contains the following FIPS Approved algorithms: RSA, DSA, DES, TripleDES, Skipjack, AES, SHA-1, HMAC, the X9.19 DES MAC, and the FIPS 186-2 SHA-1 RNG. Other (non-Approved) algorithms, such as MD5 and Diffie-Hellman, are also included.

2 Initialization

Certain tests are only performed if the flag "fips140" is passed as part of the initialization process to the library (the argument to `LibraryInitializer` or **Init::initialize**). Known answer tests and key generation self-checks for RSA and DSA are always performed, regardless of this setting. This flag must be passed by any application which desires using the FIPS 140 mode of operation.

3 Roles and Services

Botan supports two roles, the User and the Crypto Officer. Authentication is not performed by the module; all authentication is implicitly done by the operating system.

3.1 User Role

The user has the ability to access the services of the module. This role is implicitly selected whenever the module's services are accessed.

3.2 Crypto Officer Role

The crypto officer has all of the powers of the user, and in addition has the power to install and uninstall the module and to configure the operating system. This role is implicitly selected whenever these actions are performed.

4 Key Management

4.1 Key Import/Export

Symmetric keys can be imported and exported in either unencrypted, encrypted, or split-knowledge forms, as the application desires. Private keys for asymmetric algorithms can be imported and exported as either encrypted or unencrypted PKCS #8 structures. The library natively supports PKCS #5 encryption with TripleDES for encrypting private keys.

4.2 Key Storage

In no case does the library itself import or export keys from/to an external storage device; all such operations are done explicitly by the application. It is the responsibly of the operator to ensure than any such operations comply with the requirements of FIPS 140-2 Level 1.

4.3 Key Generation

Keys for symmetric algorithms (such as DES, AES, and HMAC) are generated by an Approved RNG, by generating a random byte string of the appropriate size, and using it as a key.

DSA keys are generated as specified in FIPS 186-2 (or not?). RSA keys are generated as specified in ANSI X9.31 (*I think...*). Diffie-Hellman keys are generated in a manner compatible with ANSI X9.42. All newly created DSA and RSA keys are checked with a pairwise consistency test before being returned to the caller. A pairwise consistency check can be performed on any RSA, DSA, or Diffie-Hellman key by calling the **check_key** member function with an argument of `true`.

4.4 Key Establishment

Botan supports using RSA or Diffie-Hellman to establish keys. RSA can be used with PKCS #1 v1.5 or OAEP padding. None of these methods are FIPS Approved, but Annex D of FIPS 140-2 allows for their use until such time as a FIPS Approved asymmetric key establishment method is established.

4.5 Key Protection / Zeroization

Keys are protected against external access by the operating system's memory and process protection mechanisms. If the library is used by multiple processes at once, the OS virtual memory mechanisms ensure that each version will have it's own data space (and thus, keys are not shared among multiple processes).

All keys and other sensitive materials are zeroed in memory before being released to the system.

On Windows systems the **VirtualLock** system call is used to notify the operating system that the memory containing potentially sensitive keying material is not swapped to disk, preventing an attacker from applying disk forenistics techniques to recovery data.

On Unix systems, Botan allocates memory from file-backed memory mappings, which are thoroughly erased when the memory is freed.

5 References