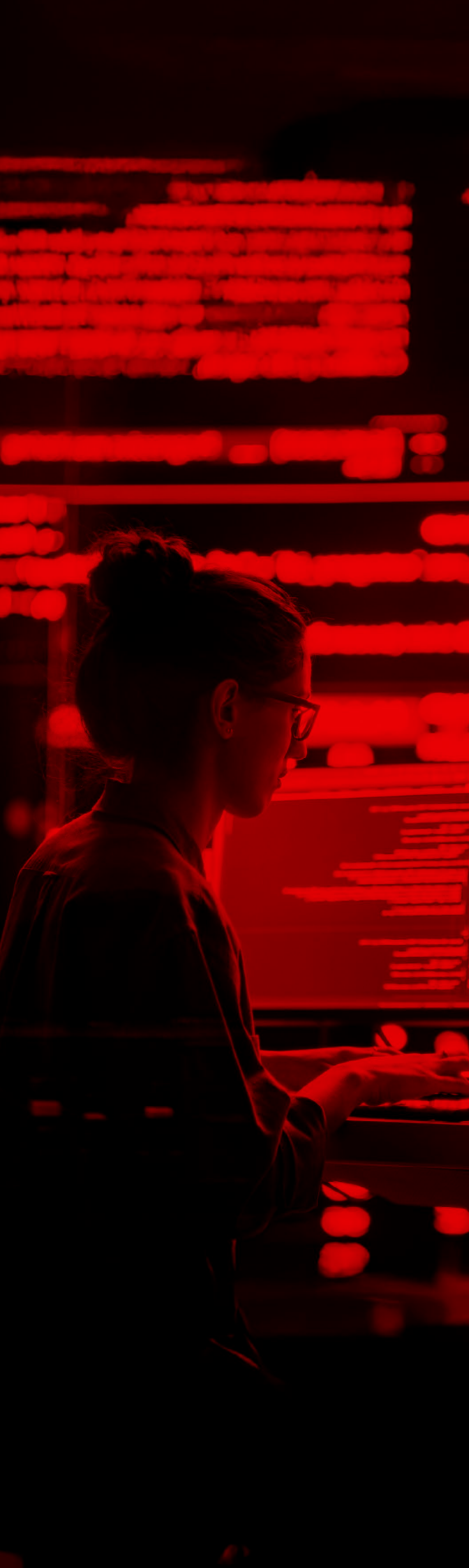


Acronis
Threat Research Unit

01

Best Practices Guide

FIPS Compliance Check

Contents

Acronis
Threat Research Unit

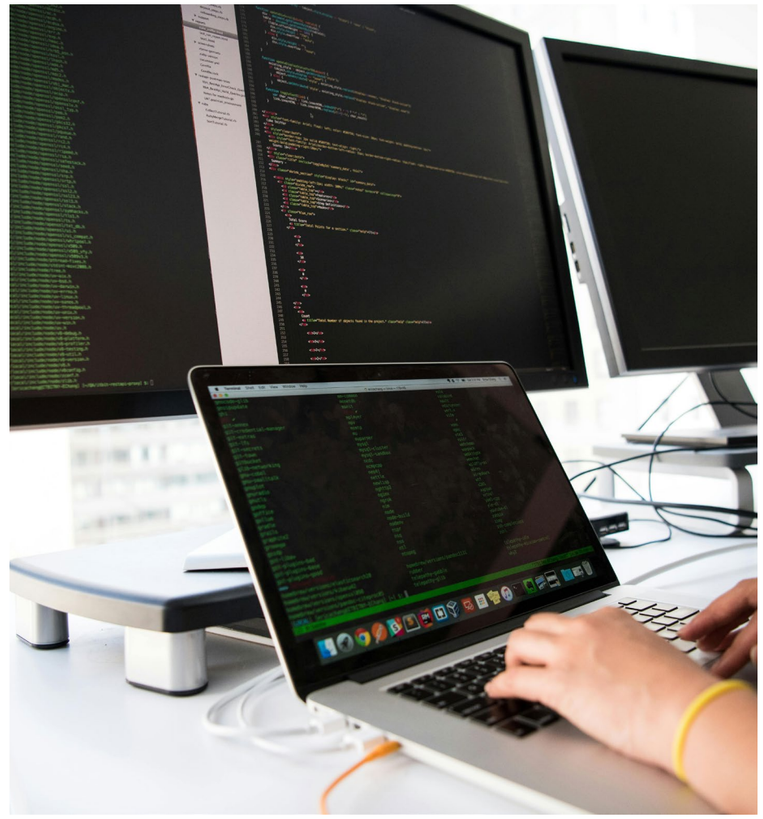
Introduction

This guide is designed to assist developers in ensuring their software meets the stringent requirements of the Federal Information Processing Standard (FIPS) 140-2.

FIPS 140-2 is a NIST standard that specifies the security requirements for cryptographic modules, ensuring that sensitive data is protected effectively. This guide covers essential aspects of FIPS compliance, including the use of FIPS-certified cryptographic providers and the avoidance of non-FIPS algorithms. It provides a comprehensive overview of setting up a FIPS-compliant environment, with detailed steps for configuring cryptographic providers in Java, Go and Python.

Additionally, the document lists approved cryptographic algorithms and libraries, such as AES, RSA and ECDSA, along with the Bouncy Castle FIPS Provider and OpenSSL with FIPS support. By following the guidance provided, developers can ensure their software meets the necessary security standards and helps protect sensitive data.

This document also delves into the validation process for FIPS 140-2, outlining the steps required to obtain certification for cryptographic modules. It includes information on the role of Cryptographic Module Validation Program (CMVP) and how developers can submit their modules for testing and evaluation. This guide emphasizes the importance of continuous compliance, detailing how to maintain FIPS 140-2 certification through regular updates and security assessments.



01 FIPS compliance summary for developers

1. Ensure the host is FIPS compliant: The OS should be installed with a FIPS-enabled kernel module, OpenSSL should use a FIPS-compliant provider, and your software should rely on system cryptographic mechanisms.
2. If not using system cryptography, any custom cryptographic components should be FIPS compliant.
3. MD5 is allowed but **only for nonsecurity purposes**, such as file-integrity checks.

02 Key points about FIPS compliance

- **FIPS 140-2** is a NIST standard for cryptographic modules.
- Your **Java environment** should use only **FIPS-certified** crypto providers (e.g., Bouncy Castle FIPS, SunPKCS11 with an approved module, etc.).
- **Non-FIPS algorithms** (like MD5, RC4, etc.) should be avoided in security contexts.
- Typically, **the OS** should also be running in **FIPS mode** (e.g., RHEL, Ubuntu with `fips=1`).

03 Set up FIPS-compliant environment

On RHEL / CentOS, for example:

```
sudo grubby --update-kernel=ALL --args="fips=1"
sudo dracut -f
sudo reboot
```

Verification

After the system starts, check that FIPS mode is enabled:

```
fips-mode-setup --check
FIPS mode is enabled.
```

or

```
cat /proc/sys/crypto/fips_enabled
```


04 Approaches to FIPS in Java

4.1 Set up FIPS-compliant environment

See section 02

Use a FIPS-compliant JDK, such as OpenJDK or Oracle JDK with FIPS-certified cryptographic providers.

4.2 Install a FIPS-validated security provider

Java requires a FIPS-certified cryptographic module. The commonly used ones are:

- **Bouncy Castle FIPS Provider:** Add the Bouncy Castle FIPS provider to your JVM.
- **FIPS-Validated JCE** (Java Cryptography Extension).

Configure the provider in the java.security file:

```
security.provider.1=com.sun.net.ssl.internal.ssl.Provider
security.provider.2=org.bouncycastle.jcajce.provider.
BouncyCastleFipsProvider
```

4.3 Configure your service for FIPS

Here is an example of how to configure Logstash. In the same way, configure your service.

Modify Logstash to use the FIPS-enabled JVM and cryptographic modules:

1. Set the Java home:

- Point Logstash to the FIPS-compliant JVM in logstash.yml or the environment variable:

```
export JAVA_HOME=/path/to/fips-enabled-java
```

2. Enable SSL/TLS with FIPS-certified libraries:

- Ensure that any SSL / TLS communications use the FIPS-certified cryptographic libraries.
- Update the logstash.yml or pipeline configuration files with the proper certificates and keys:

```
xpack.security.enabled: true
xpack.ssl.keystore.path: /path/to/keystore.jks
xpack.ssl.keystore.password: changeit
xpack.ssl.truststore.path: /path/to/truststore.jks
xpack.ssl.truststore.password: changeit
```

3. Check plugins

Ensure all Logstash plugins that use cryptography (e.g., beats, elasticsearch output) are compatible with FIPS mode.

4.4 Verify FIPS mode

test_algorithms.java

test_tls.java

1. Verify FIPS mode:

The first test (test_tls_regular) only displays availability of FIPS cryptography provider in the current runtime and availability of algorithms and cipher suites.

The second test (test_tls_bcfips) tries to establish a TLS connection to dh1024.badssl.com — this host can only use DH1024 key exchange mechanism.

```
# cat Makefile
MY_CLASSPATH := -cp /usr/local/share/logstash-fips/bc-fips-2.0.0.jar:/usr/local/share/logstash-fips/
bctls-fips-2.0.19.jar:/usr/local/share/logstas-fips/bcutil-fips-2.0.3

test_tls_regular:
        java \
          test_tls.java

test_tls_bcfips:
        java \
          -Djava.security.properties=/usr/local/share/logstash-fips/java.security \
          -Dorg.bouncycastle.fips.approved_only=true \
          ${MY_CLASSPATH} \
          test_tls.java

test_algorithms_regular:
        java \
          test_algorithms.java

test_algorithms_bcfips:
        java \
          -Djava.security.properties=/usr/local/share/logstash-fips/java.security \
          -Dorg.bouncycastle.fips.approved_only=true \
          ${MY_CLASSPATH} \
          test_algorithms.java

# make test_algorithms_regular
java \
  test_algorithms.java
> Checking if JVM is in FIPS mode...
> FIPS mode is NOT enabled.

>Testing FIPS-Unapproved Algorithms:
> MD5 is available (Provider: SUN version 11)
> DES is available (Provider: SunJCE version 11)
> RC4 is available (Provider: SunJCE version 11)

> Testing FIPS-Approved Algorithms:
> SHA-256 is available (Provider: SUN version 11)
> AES is available (Provider: SunJCE version 11)

# make test_algorithms_bcfips
java \
  -Djava.security.properties=/usr/local/share/logstash-fips/java.security \
  -Dorg.bouncycastle.fips.approved_only=true \
  -cp /usr/local/share/logstash-fips/bc-fips-2.0.0.jar:/usr/local/share/logstash-fips/bctls-fips-2.0.19.
jar:/usr/local/share/logstas-fips/bcutil-fips-2.0.3 \
  test_algorithms.java
> Checking if JVM is in FIPS mode...
> FIPS mode is enabled (Provider: BCFIPS)

> Testing FIPS-Unapproved Algorithms:
> MD5 is available (Provider: SUN version 11)
> DES is available (Provider: SunJCE version 11)
> RC4 is available (Provider: SunJCE version 11)

> Testing FIPS-Approved Algorithms:
> SHA-256 is available (Provider: BCFIPS version 2.0)
> AES is available (Provider: BCFIPS version 2.0)


# make test_tls_regular
java \
  test_tls.java
> TLS Handshake Successful!
> TLS Version: TLSv1.2
> Cipher Suite: TLS_DHE_RSA_WITH_AES_128_GCM_SHA256

> Server Certificate Details:
> Subject: CN=*.badssl.com
> Issuer: CN=R11, O=Let's Encrypt, C=US
> Expiry Date: Mon Apr 07 03:02:50 UTC 2025
> Serial Number: 406704991044954629750146633906297610196195
> Subject: CN=R11, O=Let's Encrypt, C=US
> Issuer: CN=ISRG Root X1, O=Internet Security Research Group, C=US
> Expiry Date: Fri Mar 12 23:59:59 UTC 2027
> Serial Number: 184083759606652600789093070426744763640

# make test_tls_bcfips
java \
  -Djava.security.properties=/usr/local/share/logstash-fips/java.security \
  -Dorg.bouncycastle.fips.approved_only=true \
  -cp /usr/local/share/logstash-fips/bc-fips-2.0.0.jar:/usr/local/share/logstash-fips/bctls-fips-2.0.19.
jar:/usr/local/share/logstas-fips/bcutil-fips-2.0.3 \
  test_tls.java
> TLS connection failed: Could not generate DH keypair
javax.net.ssl.SSLException: Could not generate DH keypair
        at java.base/sun.security.ssl.Alert.createSSLException(Alert.java:133)
        at java.base/sun.security.ssl.TransportContext.fatal(TransportContext.java:353)
        at java.base/sun.security.ssl.TransportContext.fatal(TransportContext.java:296)
        at java.base/sun.security.ssl.TransportContext.fatal(TransportContext.java:291)
        at java.base/sun.security.ssl.SSLSocketImpl.handleException(SSLSocketImpl.java:1689)
        at java.base/sun.security.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:471)
        at java.base/sun.security.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:427)
        at TLSSecurityChecker.checkTLSCompliance(test_tls.java:22)
        at TLSSecurityChecker.main(test_tls.java:11)
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
```

```
        at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.
java:62)
        at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.
invoke(DelegatingMethodAccessorImpl.java:43)
        at java.base/java.lang.reflect.Method.invoke(Method.java:566)
        at jdk.compiler/com.sun.tools.javac.launcher.Main.execute(Main.java:404)
        at jdk.compiler/com.sun.tools.javac.launcher.Main.run(Main.java:179)
        at jdk.compiler/com.sun.tools.javac.launcher.Main.main(Main.java:119)
Caused by: java.lang.RuntimeException: Could not generate DH keypair
        at java.base/sun.security.ssl.DHKeyExchange$DHEPossession.<init>(DHKeyExchange.java:164)
        at java.base/sun.security.ssl.DHClientKeyExchange$DHClientKeyExchangeProducer.
produce(DHClientKeyExchange.java:185)
        at java.base/sun.security.ssl.ClientKeyExchange$ClientKeyExchangeProducer.
produce(ClientKeyExchange.java:65)
        at java.base/sun.security.ssl.SSLHandshake.produce(SSLHandshake.java:436)
        at java.base/sun.security.ssl.ServerHelloDone$ServerHelloDoneConsumer.consume(ServerHelloDone.
java:182)
        at java.base/sun.security.ssl.SSLHandshake.consume(SSLHandshake.java:392)
        at java.base/sun.security.ssl.HandshakeContext.dispatch(HandshakeContext.java:443)
        at java.base/sun.security.ssl.HandshakeContext.dispatch(HandshakeContext.java:421)
        at java.base/sun.security.ssl.TransportContext.dispatch(TransportContext.java:183)
        at java.base/sun.security.ssl.SSLTransport.decode(SSLTransport.java:172)
        at java.base/sun.security.ssl.SSLSocketImpl.decode(SSLSocketImpl.java:1506)
        at java.base/sun.security.ssl.SSLSocketImpl.readHandshakeRecord(SSLSocketImpl.java:1416)
        at java.base/sun.security.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:456)
        ... 10 more
Caused by: java.security.InvalidAlgorithmParameterException: Attempt to create key of less than 2048
bits: DH
        at org.bouncycastle.jcajce.provider.ProvDH$KeyPairGeneratorSpi.initialize(Unknown Source)
        at java.base/sun.security.ssl.DHKeyExchange$DHEPossession.<init>(DHKeyExchange.java:156)
        ... 22 more
Caused by: org.bouncycastle.crypto.fips.FipsUnapprovedOperationError: Attempt to create key of less
than 2048 bits: DH
        at org.bouncycastle.crypto.fips.FipsDH$KeyPairGenerator.<init>(Unknown Source)
```


4.5 Summary cheat sheet

1. **OS FIPS mode:**
   - e.g. `fips=1`, reboot, verify `/proc/sys/crypto/fips_enabled == 1`.
2. **Install / use FIPS-certified Java:**
   - Vendor or custom JDK with FIPS modules (IBM, Red Hat, Azul, etc.).
3. **Configure `java.security`:**
   - Add a FIPS provider at highest priority.
   - Set `org.bouncycastle.fips.approved_only=true` if using BC FIPS.
4. **Restrict algorithms:**
   - No MD5, no SHA-1 for secure hashing, no RC4, etc.
   - Use FIPS-approved cipher suites in TLS.
5. **Test and validate:**
   - Check logs for crypto errors.
   - Confirm your crypto calls are using the FIPS provider.
6. **Document the entire setup for compliance audits:**
   - JDK version, provider settings, code changes, OS-level FIPS mode, etc.

05 Approaches to FIPS in Go

5.1 Set up FIPS-compliant environment

See section 02

5.2 Use Go with BoringCrypto (BoringSSL)

1. **Find a FIPS-enabled Go release**
   - Google releases special Go versions (`go1.xx.xb4` or similar) containing the BoringCrypto module.
   - Some Linux vendors (RHEL, SUSE) may provide Go binaries with FIPS patches.
2. **Install FIPS Go**
   - Download / install the FIPS build.
   - Verify with `go version` or `go version -m <binary>` (look for “boringcrypto” mention).
3. **Build your service with FIPS Go**
   - Ensure `$GOROOT` points to the FIPS-enabled Go.
   - `go build` your app with that toolchain.
4. **Remove non-FIPS crypto**
   - Check for `crypto/md5`, `crypto/sha1`, etc. Replace them if needed.
   - Use only approved algorithms (e.g., `SHA-256`, `AES`).
5. **Enable FIPS on the OS**
   - On RHEL / CentOS:

```
sudo grubby --update-kernel=ALL --args="fips=1"
sudo dracut -f sudo reboot
```

Confirm with:

```
cat /proc/sys/crypto/fips_enabled → 1.
or
fips-mode-setup --check
```

6. **Test**
   - Run your service and ensure there are no crypto-related errors.
   - Confirm you're using the correct binary (boringcrypto build).

5.3 Use cgo + OpenSSL FIPS

1. **FIPS-validated OpenSSL**
   - Make sure your system has OpenSSL built with FIPS support.
   - Typically found on RHEL / SUSE with packages like `openssl-fips.`
2. **Write cgo wrappers**
   - By default, Go does not use OpenSSL.
   - Implement C wrappers calling OpenSSL FIPS functions.
   - In Go, import `"C"` and invoke your wrappers instead of `crypto/*.`
3. **Avoid standard Go crypto**
   - If you stick to `crypto/*`, it bypasses OpenSSL, breaking FIPS compliance.
   - All crypto operations should go through your OpenSSL FIPS wrappers.
4. **Build with cgo**
   - `CGO_ENABLED=1 go build.`
   - Ensure linking to the correct FIPS library paths (check `LD_LIBRARY_PATH`).
5. **Test**
   - Verify your calls use FIPS-enabled OpenSSL.
   - No usage of disallowed algorithms.

5.4 FIPS-approved algorithms and TLS

1. **Encryption:** AES (CBC, GCM), Triple-DES (deprecated in many setups, but still technically FIPS approved, up to certain key lengths).
2. **Hashing:** SHA-256, SHA-384, SHA-512.
3. **Signatures:** RSA, ECDSA (with FIPS-valid curve parameters).
4. **TLS:**
   - Avoid unsafe ciphers like RC4, MD5-based ciphers, etc.

5.5 Common steps to verify FIPS compliance

1. **Check OS FIPS mode**

```
cat /proc/sys/crypto/fips_enabled
```

should be 1.

2. **Check Go version**

```
go version
```

   - Look for a mention of `boringcrypto` if using that build.
   - f using a distro build, see your distro docs.

3. **Search your codes**

```
grep -r "crypto/md5" . (also check for sha1, rc4, etc.)
```

   - Remove/replace if found.

4. **Inspect TLS config — for example:**
   - If you stick to `crypto/*`, it bypasses OpenSSL, breaking FIPS compliance.
   - All crypto operations should go through your OpenSSL FIPS wrappers.

```
&tls.Config{
  MinVersion: tls.VersionTLS12,
  CipherSuites: []uint16{
    tls.TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256,
    tls.TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384,
    // ...
  },
}
```

5. **Run tests**
   - Validate that all cryptographic operations succeed.
   - If anything breaks due to missing crypto functions, you're likely using non-FIPS-approved algorithms.

5.6 Summary

1. **Choose a strategy:**
   - Go+BoringCrypto (most straightforward)
   - cgo+OpenSSL-FIPS (more complex)
   - Vendor-supplied FIPS Go
2. **Build your service** with that FIPS-enabled toolchain.
3. **Eliminate non-FIPS algorithms** in code.
4. **Enable OS FIPS mode.**
5. **Test** thoroughly in a real FIPS environment.

06 Approaches to FIPS in Python

6.1 Set up FIPS-compliant environment

1. **Install Python that links to a FIPS-validated OpenSSL**
   - Many Linux distros (RHEL, Ubuntu, SUSE) provide a Python package compiled against their system OpenSSL.
   - If the system OpenSSL is FIPS-enabled, Python can (in most cases) leverage that for its crypto operations.
2. **Enable FIPS mode at the OS level**

See section 4.5

3. **Validate Python's OpenSSL**
   - In a Python shell:
   - `import ssl print(ssl.OPENSSL_VERSION)`
   - It should show a version that corresponds to your FIPS-capable OpenSSL (e.g., "OpenSSL 1.1.1… FIPS").
4. **Check for non-FIPS algorithms**
   - Python's `hashlib` and `ssl` modules typically respect the system OpenSSL's FIPS policy.
   - But if you explicitly use `hashlib.md5()` or `hashlib.sha1()` for security-critical operations, this is typically not FIPS compliant.
   - If your environment truly enforces FIPS mode, calls to MD5 for cryptographic purposes may fail or produce errors.

6.2 Use a custom-built Python with FIPS

1. **Compile Python from source**
   - Configure it to use a **FIPS-validated OpenSSL.** For example:

```
./configure --with-openssl=/path/to/fips/openssl ...
make
make install
```

   - Ensure your custom OpenSSL is already in FIPS mode or compiled with FIPS support.
2. **Confirm build**
   - After install, check:

```
python3 -c "import ssl;
print(ssl.OPENSSL_VERSION)"
```

   - The output should reflect a FIPS-enabled library.
5. **Test**
   - If you attempt to use a non-FIPS algorithm, Python may raise an error.
   - Some distros also patch Python to restrict usage of insecure ciphers if `fips=1` is set.

6.3 Third-party crypto libraries (PyCryptodome, cryptography, etc.)

1. **Library should use a FIPS-validated crypto backend**
   - For example, the `cryptography` Python package can build against OpenSSL. If your system OpenSSL is FIPS enabled, `cryptography` can leverage that.
   - Validate that the library indeed uses the system's OpenSSL rather than a statically linked or vendored OpenSSL.
2. **Disable or remove non-FIPS ciphers**
   - Check if any third-party libraries do their own internal crypto (some might ship with non-FIPS code).
   - Consult the library docs on FIPS compliance.

6.4 Common steps for FIPS in Python

1. **Enable OS FIPS mode**
   - On Linux distros, set fips=1 in the boot loader, rebuild initramfs, reboot.
   - Confirm with:

```
cat /proc/sys/crypto/fips_enabled
```

2. **Install Python and OpenSSL from Trusted Repo**
   - Prefer distro packages that explicitly mention FIPS support.
   - Or build Python yourself, linking to a known FIPS-validated OpenSSL.
3. **Check Python version and OpenSSL link**

```
python3 --version python3 -c "import ssl; print(ssl.OPENSSL_VERSION)"
```

   - Ensure the reported OpenSSL is your FIPS build.

4. **Audit your code for Non-FIPS usage**
   - Search for import `hashlib, hashlib.md5`, or any direct usage of MD5, SHA-1 or weak ciphers.
   - If used for security (e.g., password hashing), replace with FIPS-approved algorithms (e.g., `SHA-256, SHA-512`).

5. **Restrict SSL / TLS cipher suites**
   - If using `ssl` or requests, you can specify ciphers. For example:

```
import ssl context = ssl.SSLContext(ssl.PROTOCOL_TLSv1_2)
context.set_ciphers('ECDHE-RSA-AES256-GCM-
SHA384:ECDHE-RSA-AES128-GCM-SHA256')
```

6. **Test and verify**
   - Run your app in a fully FIPS-enabled environment.
   - If Python or a library tries to use disallowed algorithms, you may see runtime errors or handshake failures.

6.5 Summary cheat sheet

1. **Install or build Python with a FIPS-validated OpenSSL.**
2. **Confirm with:**

```
import ssl print(ssl.OPENSSL_VERSION)
```

   - Should mention a FIPS-capable OpenSSL.

3. **Enable FIPS mode at the OS level.**
4. **Remove non-FIPS crypto:**
   - Avoid `md5`, `sha1` (if used for security).
   - Restrict ciphers to AES and SHA-2 families.
5. **Validate with real usage tests:**
   - TLS connections (e.g., `requests, ssl.SSLContext`).
   - Hashing and encryption operations (e.g., `hashlib`, `cryptography` library).

07 Approaches to FIPS in Windows

7.1 Enabling FIPS mode in Windows

7.1a Using Group Policy / Local Security Policy

1. **Open Local Security Policy:**
   - Press **Win + R**, type `secpol.msc,` and press **Enter.**
2. **Navigate to:**
   Security Settings → **Local Policies** → **Security Options.**
3. **Find the policy:**
   System cryptography: Use FIPS-compliant algorithms for encryption, hashing, and signing.
4. **Double-click and set it to Enabled.**
5. **Close the policy editor.**
   In many cases, a reboot is advised to ensure all services pick up the change.

7.1b Via the Registry

1. **Open Registry Editor:**
   - Press **Win + R**, type `regrdit,` and press **Enter.**
2. **Locate the key:**
   `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\FipsAlgorithmPolicy`
3. **Set Enabled (DWORD) to 1 to enforce FIPS mode (set to `0` to disable it).**
4. **Close Registry Editor.**
   - A reboot may be required for full effect.

> **Note:** Group Policy changes typically write to the same registry path, so either method achieves the same result.

7.2 Verifying FIPS mode on Windows

7.2a Local Security Policy

Check if the **System cryptography: Use FIPS-compliant** algorithms policy is **Enabled.**

7.2b Registry Check

Verify `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\FipsAlgorithmPolicy\Enabled` is set to 1

7.2c PowerShell Script Example

```
$regKey = “HKLM:\SYSTEM\CurrentControlSet\Control\Lsa\
FipsAlgorithmPolicy”
try {
    $val = Get-ItemProperty -Path $regKey -Name Enabled -ErrorAction
Stop
    if ($val.Enabled -eq 1) {
        Write-Host “FIPS Mode is ENABLED.”
    } else {
        Write-Host “FIPS Mode is DISABLED.”
    }
} catch {
    Write-Host “FIPS Mode registry key not found; likely DISABLED.”
}
```

7.2d Registry Check

In .NET applications, you can also check:

```
bool fipsCompliant = System.Security.Cryptography.CryptoConfig.
AllowOnlyFipsAlgorithms;
Console.WriteLine(“System Crypto FIPS compliance: “ + fipsCompliant);
```

7.3 Practical considerations

1. **Applications must use FIPS-compliant APIs**
   - Even if Windows is in FIPS mode, your application or libraries must avoid non-FIPS algorithms (e.g., MD5 for security, RC4 ciphers, etc.).
   - Some .NET or C++ libraries automatically throw exceptions when attempting to use disallowed algorithms in FIPS mode.
2. **MD5 for Nonsecurit use**
   - Under FIPS guidelines, **MD5 can be used for noncrytographic purposes** (e.g., file integrity checks).
   - In strict .NET FIPS mode, calling MD5 for any reason may still throw an exception, unless special handling is in place.
   - For a truly FIPS-certified environment, ensure cryptographic operations rely on approved algorithms (SHA-2, AES, etc.).
3. **Impact on third-party tools**
   - Some older applications or SSL libraries may break in FIPS mode. Test thoroughly.
4. **Installer automation**
   - For large deployments (e.g., via RMM or other management tools), you can incorporate a “FIPS check” step before installing software.
   - If FIPS is **enabled**, install or configure the **FIPS-compliant** package / agent. If **disabled**, proceed with the regular version or show a warning.

08 Quick tips

- **SHA-1** is generally disallowed in strict FIPS mode for cryptographic use, but it might still appear for noncrypto tasks (e.g., Git commits). Audit carefully.
- **MD5** is never FIPS-approved for secure hashing.
- **Logging:** Watch for errors like “FIPS_mode_set: FIPS is not supported” if you’re using OpenSSL incorrectly.
- **Docker / container:** Ensure the host kernel is in FIPS mode and you’re using the correct FIPS Go in your container image.
- **Documentation:** If you need formal certification, record your steps, versions, and build-process thoroughly.

09 List of RHEL 9 applications using cryptography that is not compliant with FIPS 140-3

Application	Reason for FIPS 140-3 noncompliance
Bacula	Implements the CRAM-MD5 authentication protocol.
Cyrus SASL	Uses the SCRAM-SHA-1 authentication method.
Dovecot	Uses SCRAM-SHA-1.
Emacs	Uses SCRAM-SHA-1.
FreeRADIUS	Uses MD5 and SHA-1 for authentication protocols.
Ghostscript	Custom cryptography implementation (MD5, RC4, SHA-2, AES) to encrypt and decrypt documents.
GRUB	Supports legacy firmware protocols requiring SHA-1 and includes the libgcrypt library.
iPXE	Implements TLS stack.
Kerberos	Preserves support for SHA-1 (interoperability with Windows).
Lasso	The lasso_wsse_username_token_derive_key() key derivation function (KDF) uses SHA-1.
MariaDB, MariaDB Connector	The mysql_native_password authentication plugin uses SHA-1.
MySQL	mysql_native_password uses SHA-1.
OpenIPMI	The RAKP-HMAC-MD5 authentication method is not approved for FIPS usage and does not work in FIPS mode.
Ovmf (UEFI firmware), Edk2, shim	Full cryptographic stack (an embedded copy of the OpenSSL library).
Perl	Uses HMAC, HMAC-SHA1, HMAC-MD5, SHA-1, SHA-224, etc.
Pidgin	Implements DES and RC4 ciphers.
PKCS #12 file processing (OpenSSL, GnuTLS, NSS, Firefox, Java)	All uses of PKCS #12 are not FIPS compliant due to unsafe KDF for calculating the whole-file HMAC.
Poppler	Can save PDFs with nonallowed algorithms (MD5, RC4, SHA-1) if present in the original PDF.
PostgreSQL	Implements Blowfish, DES and MD5. A KDF uses SHA-1.
QAT Engine	Mixed hardware and software implementation of cryptographic primitives (RSA, EC, DH, AES, ...)

Application	Reason for FIPS 140-3 noncompliance
Ruby	Provides insecure MD5 and SHA-1 library functions.
Samba	Preserves support for RC4 and DES (interoperability with Windows).
Syslinux	BIOS passwords use SHA-1.
SWTPM	Explicitly disables FIPS mode in its OpenSSL usage.
Unbound	DNS specification requires SHA-1-based algorithms in DNSKEY records for validation.
Valgrind	AES, SHA hashes.
zip	Custom cryptography implementation (insecure PKWARE encryption algorithm) for password-protected archives.

About the author

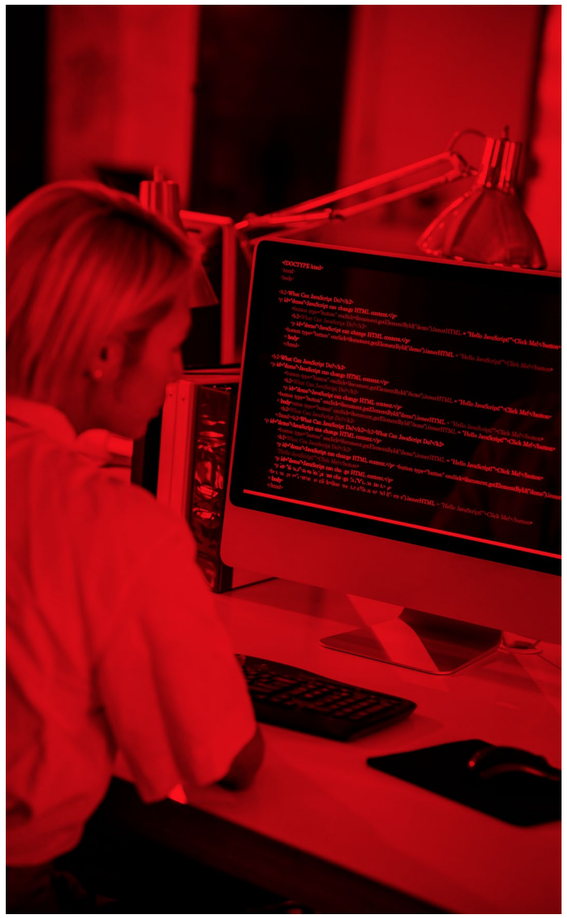
Christian Nikita
With over two decades of experience in cybersecurity, Christian Nicita (ISO 27001 LA, CCSK) is a recognized expert in cybersecurity governance, risk management and compliance. His career spans leadership roles in top-tier organizations where he has successfully led the implementation of international security standards, including ISO/IEC 27001, ISO/IEC 27017 and ISO/IEC 27018.

As a Cybersecurity Governance, Risk and Compliance Specialist at Acronis, he plays a key role in defining security policies, advancing risk mitigation strategies, conducting threat modeling, and ensuring compliance with key regulatory frameworks, including GDPR, ENS and 2G3M.

With a deep technical foundation in application security, penetration testing and secure software development, he bridges the gap between security strategy and technical execution, ensuring resilience against emerging cyberthreats. Christian has provided cybersecurity advisory services to global enterprises in telecommunications, utilities, banking and cloud services, enhancing their security posture and compliance readiness.

About TRU

Acronis Threat Research Unit (TRU) is a dedicated unit composed of experienced cybersecurity experts. Our team includes cross-functional experts in cybersecurity, AI and threat intelligence.

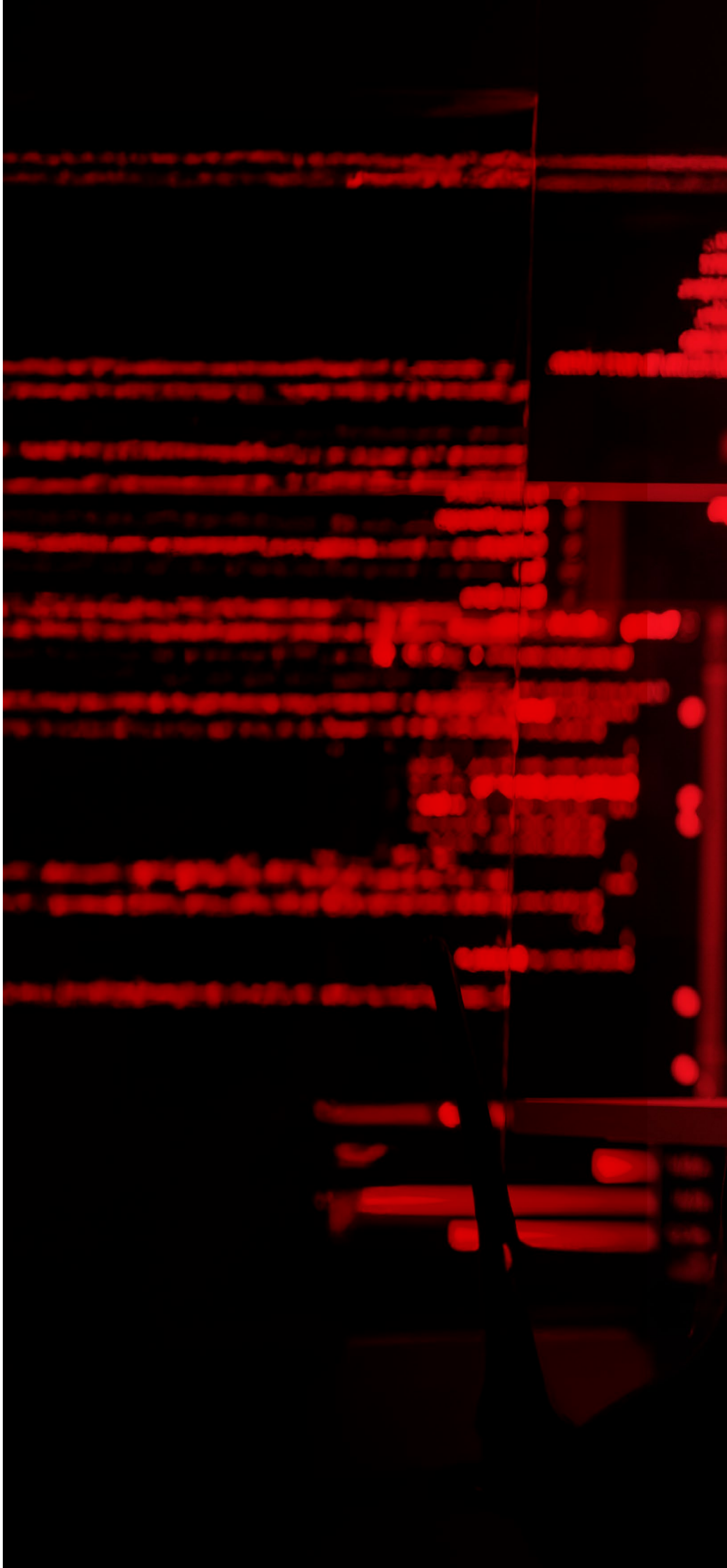


TRU conducts deep research into emerging cyberthreats, focusing on malware, ransomware, phishing and APTs.

We help proactively manage cyber risks and respond to incidents effectively. Our team leverages threat intelligence to prevent future attacks and compiles guidelines and recommendations to assist IT teams in building robust security frameworks.

01

Best Practices Guide



Acronis
Threat Research Unit

2025