FIPS 140-2 Crypto
In the IoT
A. Intro to wolfSSL
B. Overview of wolfCrypt FIPS
C. FIPS 140-2 Challenges in the IoT
D. Doing new validations
E. Q&A
Overview

wolfSSL and wolfCrypt FIPS
Introduction to wolfSSL

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Needs “Clean Room” SSL</td>
<td>2004 - yaSSL (C++)</td>
<td>wolfSSL Name Change</td>
</tr>
<tr>
<td></td>
<td>2006 - CyaSSL (C)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Employees</th>
</tr>
</thead>
<tbody>
<tr>
<td>2011</td>
<td>3</td>
</tr>
<tr>
<td>2012</td>
<td>9</td>
</tr>
<tr>
<td>2013</td>
<td>11</td>
</tr>
<tr>
<td>2014</td>
<td>15</td>
</tr>
<tr>
<td>2015</td>
<td>19</td>
</tr>
<tr>
<td>2017</td>
<td>21</td>
</tr>
</tbody>
</table>

450 OEM Customers

15 Resale Partners

2 BILLION secure connections!
Introduction to wolfSSL - Products

- **wolfSSL**
  Lightweight SSL/TLS

- **wolfCrypt**
  Crypto Engine

- **wolfMQTT**
  Lightweight MQTT Client

- **wolfSSH**
  Lightweight SSH Server

- **wolfSSL JNI**
  wolfSSL Java Wrapper

- **wolfSSL C#**
  wolfSSL C# Wrapper

- **wolfSSL Python**
  wolfSSL Python Wrapper

- **SSL Proxy**
  On top of Squid Proxy

- **Secure memcached**

- **wolfSCEP**

- **Secure Update**
- Independent of SSL/TLS
- Design simplifies updates
- Most bugs and vulnerabilities happen in SSL/TLS, not crypto
# Current wolfCrypt FIPS OE List

## Certificate #2425

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Processor</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Linux 3.13 (Ubuntu)</td>
<td>Intel® Core™ i7-3720QM CPU @2.60GHz x 8</td>
<td>HP EliteBook</td>
</tr>
<tr>
<td>2. iOS 8.1</td>
<td>Apple™ A8</td>
<td>iPhone™ 6</td>
</tr>
<tr>
<td>3. Android 4.4</td>
<td>Qualcomm Krait 400</td>
<td>Samsung Galaxy S5</td>
</tr>
<tr>
<td>4. FreeRTOS 7.6</td>
<td>ST Micro STM32F</td>
<td>uTrust TS Reader</td>
</tr>
<tr>
<td>5. Windows 7 (64-bit)</td>
<td>Intel® Core™ i5</td>
<td>Sony Vaio Pro</td>
</tr>
<tr>
<td>6. Linux 3.0 (SLES 11 SP4, 64-bit)</td>
<td>Intel® Xeon® E3-1225</td>
<td>Imprivata OneSign</td>
</tr>
<tr>
<td>7. Linux 3.0 (SLES 11 SP4, 64-bit) on Microsoft Hyper-V 2012R2 Core</td>
<td>Intel® Xeon® E5-2640</td>
<td>Dell® PowerEdge™ r630</td>
</tr>
<tr>
<td>8. Linux 3.0 (SLES 11 SP4, 64-bit) on VMWare ESXi 5.5.0</td>
<td>Intel® Xeon® E5-2640</td>
<td>Dell® PowerEdge™ r630</td>
</tr>
<tr>
<td>9. Windows 7 (64-bit) on VMWare ESXi 5.5.0</td>
<td>Intel® Xeon® E5-2640</td>
<td>Dell® PowerEdge™ r630</td>
</tr>
<tr>
<td>Algorithm</td>
<td>Description</td>
<td>Cert #</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>AES</td>
<td>[FIPS 197, SP 800-38A] (Encryption, Decryption)</td>
<td>3157, 3330, 3417, 3490, 3508</td>
</tr>
<tr>
<td></td>
<td>Modes: CBC, CTR, Key sizes: 128, 192, 256 bits</td>
<td></td>
</tr>
<tr>
<td>DRBG</td>
<td>[SP 800-90A] (Hash_DRBG)</td>
<td>650, 775, 821, 863, 875</td>
</tr>
<tr>
<td></td>
<td>Security Strengths: 256 bits</td>
<td></td>
</tr>
<tr>
<td>HMAC</td>
<td>[FIPS 198-1] (Generation, Verification)</td>
<td>1990, 2121, 2175, 2228, 2241</td>
</tr>
<tr>
<td></td>
<td>SHA sizes: SHA-1, SHA-256, SHA-384, and SHA-512</td>
<td></td>
</tr>
<tr>
<td>RSA</td>
<td>[FIPS 186-4, and PKCS #1 v2.1 (PKCS1.5)] (Signature Generation, Signature Verification)</td>
<td>1602, 1710, 1749, 1791, 1803</td>
</tr>
<tr>
<td></td>
<td>Key sizes: 1024 (verification only), 2048</td>
<td></td>
</tr>
<tr>
<td>SHA</td>
<td>[FIPS 180-4] (Digital Signature Generation, Digital Signature Verification, non-Digital Signature Applications). SHA sizes: SHA-1, SHA-256, SHA-384, SHA-512</td>
<td>2614, 2763, 2823, 2882, 2893</td>
</tr>
<tr>
<td>Triple-DES (TDES)</td>
<td>[SP 800-20] (Encryption, Decryption)</td>
<td>1800, 1901, 1928, 1966, 1972</td>
</tr>
<tr>
<td></td>
<td>Modes: TCBC, Key sizes: 3-key</td>
<td></td>
</tr>
</tbody>
</table>
FIPS 140-2
Challenges in the IoT
FIPS 140-2 Challenges in the IoT

Predominant challenges include:

- Porting default shared library entry point
- Running CAVP test vectors
- Fitting FIPS module into available memory
- Porting library to target environment
Porting Default Entry Point

- When library is first loaded, two things need to happen:
  1. Power-On Integrity Check
  2. Run Known Answer Tests

- Shared library default entry point is used for this

```c
#define INITIALIZER(f) static void __attribute__((constructor)) f(void)
```

- Needs to be ported on new compiler/linkers (gcc, VS, etc)
FIPS 140-2 Module Runtime Requirements

**wolfCrypt FIPS Module**

Self Tests

a) Power-On Integrity Check (HMAC-SHA256)

b) Known Answer Tests

PASS?

NO → ABORT!

YES → Allow access to Module

AES, 3DES, RSA, etc.
Running CAVP Test Vectors

Challenges:

- No standard File I/O
- Need to process large vectors in smaller pieces

Solution:

- Test Harness (Host/Agent)
Fitting FIPS Module into Memory

- IoT Device Memory Constraints Pose a Challenge

- Mitigations / Resolutions
  - Configure algorithms differently (speed vs. size)
  - Configure memory usage differently (stack vs. heap)
  - Shrink module boundary
Porting Library to Target Environment

- IoT Devices Pose Portability Concerns
- Platform Details Can Vary:
  - Variety of RTOS’s
  - Different toolchains / compilers
  - Memory configurations (stack vs. heap preference)
  - Threading / Mutexes
  - Seeding PRNG / sources of randomness

- wolfCrypt platform-dependencies have been abstracted out
FIPS 140-2
Doing New Validations
New FIPS 140-2 Validations

- **Validation Options:**
  - Adding new Operating Environment (OE)
  - Rebranding Validation
  - Growing (or Shrinking) Module Boundary

- **Timeframe dependent on scope, platform, and lab/CMVP**
Step 1: CAVP Testing and Algorithm Certificates

1. Define desired cryptographic module boundary
   - From customer:
     - **Exact platform** (hardware, OS version)
     - **Example app** demonstrating I/O from device (for test harness)
2. **Port** and **test** module and harness on desired validation target
3. **Request** test vectors from FIPS Lab
4. **Run** test vectors through module, return to FIPS Lab
5. Obtain **Algorithm Certificates** from CAVP
Adding a New Operating Environment

Step 2: CMVP and FIPS 140-2 Certificate

1. Update **Security Policy**, send to FIPS Lab
2. **On-site** testing at FIPS Lab with module
3. FIPS Lab submits to **CMVP**… wait…
4. **FIPS 140-2 Certificate Issued**, or **Existing Updated**

[CMVP Main Page]

Validated FIPS 140-1 and FIPS 140-2 Cryptographic Modules


All
wolfCrypt FIPS Rebranded Validations

- Rebranded wolfCrypt FIPS validations possible
- One recent IoT-based example - reduced FIPS boundary

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenRTOS v9.0.0</td>
<td>ATSAM4L</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>[FIPS 197, SP 800-38A] (Encryption, Decryption) Modes: CBC, CTR, Key sizes: 256 bits</td>
</tr>
<tr>
<td>HMAC</td>
<td>[FIPS 198-1] (Generation, Verification) SHA sizes: SHA-256</td>
</tr>
<tr>
<td>SHA</td>
<td>[FIPS 180-4] (Message Digest) SHA sizes: SHA-256</td>
</tr>
</tbody>
</table>
Summary

A. FIPS 140-2 Challenges in the IoT
   a. Porting default shared library entry point
   b. Running CAVP test vectors
   c. Fitting FIPS module into available memory
   d. Porting library to target environment

B. Doing new validations
   a. Adding a new OE
   b. Rebranded Validation
   c. Growing (Shrinking) Module Boundary