Semantics-Driven Testing of the PKCS11 API

Matthew Bauer, Mike Dodds
Joint work with Charisee Chiw, Joey Dodds and Stephen Magill
Galois Consultancy

Founded in 1999 • Based in Portland, OR • 100 Employees
Galois / AWS Collaboration

- Collaboration: Key Security related projects in partnership with AWS
- Approach: continuous formal methods
  - Tight integration with engineering processes
  - Integration into CI / CD pipeline
  - High levels of automation
  - Low cost of maintenance
  - Actionable bug reports
- Ongoing formal methods success story: we’re helping bringing high assurance software to AWS end users (*ie.* everyone)
- NB: lots of other AWS formal methods work - see other talks at HCSS!
Galois / AWS Projects

- Cryptographic Algorithm verification
  - SIKE / BIKE - Post-quantum algorithms
  - HMAC/DRBG - TLS core algorithms

- Cryptographic Protocol verification
  - s2n - Amazon TLS handshake protocol

- Cryptographic APIs testing
  - PKCS11 - Public-Key Cryptography Standards
This Project: Assuring API Implementations

- Each API defines expected sets of behaviours - an API is pretty much a specification.
- Library implementations should match the spec i.e.
  - Not crash
  - Return expected values
- Often not the case!
Target: The PKCS11 API

- A platform independent API standard for interacting with cryptographic tokens such as hardware security modules (HSMs) and smart cards

- Functionality:
  - Store cryptographic tokens on devices
  - Generate cryptographic keys and random numbers
  - Encrypt, decrypt, hash, sign and verify data
  - Wrap and unwrap keys
The PKCS11 API - Keys

- Keys hold cryptographic data and properties, which include:
  - Key type (e.g. AES)
  - Key class (e.g. private key, public key, secret key)
  - Storage characteristics (e.g. does it persist on the device)
  - Supported operation types (e.g. encryption, decryption, etc...)
  - User defined labels
The PKCS11 API - Mechanisms

- Each cryptographic operation is parameterized by a mechanism that describes the underlying algorithms used in the cryptographic operation.

  - Example: AES-CBC for encryption and decryption describes:
    - The algorithm (AES) and mode (CBC)
    - Parameters to the algorithm (such as an initialization vector)
The PKCS11 API is Complicated

- ~350 pages of specification (~150 base spec, ~200 key/mechanism spec)
- ~50 function specifications
- ~45 cryptographic algorithms
- ~90 error codes

→ size makes it challenging to formally verify code.
Instead: Model-based Test Synthesis

- Write a strict formal model of the API - values and transitions
- Synthesize a test set by exploring the model
- Use formal techniques to ensure a high level of coverage
- Test the implementation library, add tests to CI

→ Achieve a high level of API confidence.
PKCS11 Testing in Detail
PKCS11 API Function Descriptions

Function name and return type

Argument types and order

![Function Description](image_url)
Informal Description: C_EncryptInit initializes an encryption operation. *hSession* is the session’s handle; *pMechanism* points to the encryption mechanism; *hKey* is the handle of the encryption key.
Stateful Behavior: After calling C_EncryptInit, the application can either call C_Encrypt to encrypt data in a single part; or call C_EncryptUpdate zero or more times, followed by C_EncryptFinal, to encrypt data in multiple parts ....
### PKCS11 API Function Descriptions

#### C_EncryptInit

```c
CK_DEFINE_FUNCTION(CK_RV, C_EncryptInit){
    CK_SESSION_HANDLE hSession,
    CK_MECHANISM_PTR pMechanism,
    CK_OBJECT_HANDLE hKey
}
```

C_EncryptInit initializes an encryption operation. hSession is the session's handle; pMechanism points to the encryption mechanism; hKey is the handle of the encryption key.

The CKA_ENCRYPT attribute of the encryption key, which indicates whether the key supports encryption, MUST be CK_TRUE.

After calling C_EncryptInit, the application can either call C_Encrypt to encrypt data in a single part; or call C_EncryptUpdate zero or more times, followed by C_EncryptFinal, to encrypt data in multiple parts. The encryption operation is active until the application uses a call to C_Encrypt or C_EncryptFinal to actually obtain the final piece of ciphertext. To process additional data (in single or multiple parts), the application MUST call C_EncryptInit again.

**Return values:** CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED, CKR_FUNCTION_FAILED, CKR_GENERAL_ERROR, CKR_HOST_MEMORY, CKR_KEY_FUNCTION_NOT_PERMITTED, CKR_KEY_HANDLE_INVALID, CKR_KEY_SIZE_RANGE, CKR_KEY_TYPE_INCONSISTENT, CKR_MECHANISM_INVALID, CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE, CKR_PIN_EXPIRED, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_USER_NOT_LOGGED_IN.

**Examples:** see C_EncryptInit

---

**Error Handling:** CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR, CKR_DEVICE_REMOVED, CKR_FUNCTION_CANCELED … (20 Returns in total)
PKCS11 API Return Code Descriptions

- Returns codes are organized by section, where a section’s order in the document defines a precedence on the return codes it contains.
- Each section also defines an order on the return codes within in.
  - In some sections, this is a total order according to order of appearance.
  - In some sections, this is a partial order where all returns are unordered unless explicitly stated.

5.1 Universal Cryptoki function return values
Any Cryptoki function can return any of the following values:
- CKR_GENERAL_ERROR: Some unforeseen, unrecoverable error has occurred. In the worst case, it is possible that the function call failed, the situation is not necessarily totally hopeless, as it is likely to be the case when CKR_OK.
- CKR_MISSING_MEMORY: The computer that the Cryptoki Library is running on has insufficient memory to perform the request. The requested function could not be performed, but detailed information about why not is not available.

5.1.1 Universal Cryptoki function return values
- CKR_SESSION_HANDLE_INVALID: The specified session handle was invalid at the time that the function was invoked.
- CKR_DEVICE_RENDER: The token was removed from its slot during the execution of the function.
- CKR_SESSION_CLOSED: The session was closed during the execution of the function.
- CKR_FUNCTION_FAILED: The request function could not be performed, but detailed information about why not is not available.

5.1.2 Cryptoki function return values for functions that use a session handle
Any Cryptoki function that takes a session handle as one of its arguments (i.e., any Cryptoki function except for C_Initialize) will return these values:
- CKR_SESSION_HANDLE_INVALID: The specified session handle was invalid at the time that the function was invoked.
- CKR_DEVICE_RENDER: The token was removed from its slot during the execution of the function.
- CKR_SESSION_CLOSED: The session was closed during the execution of the function.

5.1.3 Cryptoki function return values for functions that use a token
Any Cryptoki function that uses a particular token (i.e., any Cryptoki function except for C_Initialize, C_Finalize, C_GetToken)
- CKR_DEVICE_MEMORY: The token does not have sufficient memory to perform the requested function.
- CKR_DEVICE_ERROR: Some problem has occurred with the token and/or slot. This error code can be returned by a function that does not have sufficient memory to perform the requested function.
- CKR_DEVICE_TOKEN_PRESENT: The token was not present in its slot at the time that the function was invoked.
- CKR_DEVICE_TOKEN_RENDER: The token was removed from its slot during the execution of the function.

5.1.4 Special return values for application-supplied callbacks
There are some special return values which are not returned by any function in the actual Cryptoki API, but which may be:
- CKR_CANCEL: When a function executing in a user-supplied callback function decides to give the application a chance to do something.

5.1.5 Special return values for mutex-handling functions
There are some return values which are not returned by any actual Cryptoki functions. These values are:
- CKR_MUTEX_BAD: This error code is returned by mutex-handling functions that are passed a bad mutex object.
- CKR_MUTEX_NOT_LOCKED: This error code is returned by mutex-unlocking functions. It indicates that the mutex could not be unlocked.

5.1.6 All other Cryptoki function return values
Described in this chapter are the return values that are not included in the descriptions of other Cryptoki functions. They are:
- CKR_ACTION_PROHIBITED: This error code can only be returned by C_OpenObject, C_SetAttribute, C_DestroyObject, and C_GetAttribute.
- CKR_ARGUMENTS_BAD: This error code indicates that the arguments supplied to the Cryptoki function are invalid.
- CKR_ATTRIBUTE_READ_ONLY: An attempt was made to get a value for an attribute which may not be set by the application.
PKCS11 API Pitfalls

- Behavior is underspecified
  - Possible return codes may not be listed
  - Extra return codes may be listed
  - Not all return codes are described

- Return code precedences are not concisely and uniformly described

- The description of stateful behavior is imprecise and scattered across the document

→ Need a formal description of the API!
Formally Modeling the API

- Cryptographic algorithms
- Error conditions associated with each function
- Stateful behavior that defines how functions interact
Formal Cryptographic Specifications

- Specified using Cryptol, a domain specific language for cryptography
- Cryptol specifications are executable programs that closely resemble their mathematical definitions
- We have specified the following algorithms
  - AES
  - Triple DES
  - ECDSA
  - RSA
  - SHA
Formal Return Code Specifications

- What error conditions are possible and how are they triggered?
  - We describe errors as constraints over function arguments and the (model of) the system state
  - Return code precedence complicates constraints, all conditions of higher priority errors must not be true
  - SMT solvers are used to synthesize the necessary system state and function inputs that generate a particular error
Formal State Models

Encryption

UnInitialized

Encryption

Updated

(Init, Ok)

(Init, OperationActive)

(Update, Ok)

(Update, OperationActive)

Transitions = (Function Name, Return Code)

Encryption

Initialized

Encryption

Updating

(*, BufferTooSmall)
## State Model Encoding in Haskell

<table>
<thead>
<tr>
<th>State Transition</th>
<th>Current State</th>
<th>Function</th>
<th>Error Code</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incremental State Transition</td>
<td>UnInit</td>
<td>C_EncryptInit</td>
<td>CKR_OK</td>
<td>Initialized</td>
</tr>
<tr>
<td>Incremental State Transition</td>
<td>UnInit</td>
<td></td>
<td></td>
<td>UnInit</td>
</tr>
<tr>
<td>Incremental State Transition</td>
<td>Initialized</td>
<td>C_EncryptInit</td>
<td>CKR_OPERATION_ACTIVE</td>
<td>Initialized</td>
</tr>
<tr>
<td>Incremental State Transition</td>
<td>Initialized</td>
<td>C_EncryptUpdate</td>
<td>CKR_OK</td>
<td>Updating</td>
</tr>
<tr>
<td>Incremental State Transition</td>
<td>Initialized</td>
<td></td>
<td></td>
<td>UnInit</td>
</tr>
<tr>
<td>Incremental State Transition</td>
<td>Updating</td>
<td>C_EncryptUpdate</td>
<td>CKR_OK</td>
<td>Updating</td>
</tr>
<tr>
<td>Incremental State Transition</td>
<td>Updating</td>
<td>C_EncryptInit</td>
<td>CKR_OPERATION_ACTIVE</td>
<td>Updating</td>
</tr>
<tr>
<td>Incremental State Transition</td>
<td>Updating</td>
<td>C_EncryptFinal</td>
<td>CKR_OK</td>
<td>UnInit</td>
</tr>
<tr>
<td>Incremental State Transition</td>
<td>Updating</td>
<td></td>
<td></td>
<td>UnInit</td>
</tr>
</tbody>
</table>
System Architecture

Return Code Constraints

Cryptol Algorithm Specs

Test Vectors

C Tests

\[ F(A_1, \ldots, A_N) = RV \]

Function: \( F \)
Return: \( RV \)

Args: \( A_1, \ldots, A_N \)
Test Generation

- Generated tests that exercised every software triggered return code for every stateful operation
- Explored every path through the finite state machines as well as all meaningful compositions of different paths
- Over 1,500 test cases in total

<table>
<thead>
<tr>
<th>Feature</th>
<th>Number of Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encryption</td>
<td>442</td>
</tr>
<tr>
<td>Decryption</td>
<td>438</td>
</tr>
<tr>
<td>Digest</td>
<td>182</td>
</tr>
<tr>
<td>Sign</td>
<td>128</td>
</tr>
<tr>
<td>Verify</td>
<td>150</td>
</tr>
<tr>
<td>Sign Recover</td>
<td>30</td>
</tr>
<tr>
<td>Verify Recover</td>
<td>30</td>
</tr>
<tr>
<td>Session Management</td>
<td>42</td>
</tr>
</tbody>
</table>
Test Results

- Tested against **OpenCryptoki** and **pre-release Amazon CloudHSM**

- Bugs fixed before production:
  - Library segmentation faults
  - Invalid state machines
  - Improper return codes
  - Missing null pointer handling
  - Lossy object copies

**CloudHSM**

- **7** Library Segfaults
- **2** State machine errors
- **20+** Improper return codes
Example State Machine Error

**Spec**

- Encryption UnInitialized
  - (Final, Ok)
  - (Init, Ok)
  - (Update, Ok)
  - (Update, Ok) (Init, OperationActive)

- Encryption Initialized
  - (*, BufferTooSmall)
  - (Init, OperationActive)

**OpenCryptoki**

- Encryption UnInitialized
  - (Final, Ok)
  - (Init, Ok)
  - (*, BufferTooSmall)
  - (*, Ok)

- Encryption Initialized
  - (Update, Ok)
  - (Init, OperationActive)
Example State Machine Error

**Spec**

- **Encryption UnInitialized**
  - (Final, Ok) -> **Encryption Updated**
  - (Init, Ok) -> **Encryption Initialized**
  - (Update, Ok)

- **Encryption Updated**
  - (Init, OperationActive)

- **Encryption Initialized**
  - (*, BufferTooSmall)
  - (Init, OperationActive)

**CloudHSM**

- **Encryption UnInitialized**
  - (Final, Ok) -> **Encryption Updated**
  - (Init, Ok) -> **Encryption Initialized**
  - (*, Ok)

- **Encryption Updated**
  - (Update, Ok)

- **Encryption Initialized**
  - (*, BufferTooSmall)
  - (Init, OperationActive)
Keys to Deployment Success

- Categorization of failures by type and severity
- Test case reproducibility
- Speed
- Metrics
- Configurability
Amazon Deployment

- The test suite is deployed in Amazon’s CI/CD pipeline.
  - Test suite and specification compliance is continuously maintained
  - Searchable metrics are published with each run
  - Detailed logs capturing all object instantiations, function calls and return values are captured
  - Over 10,000 tests execute in less than 30 minutes
Thank you!