Sentinel® LDK

Sentinel HL AppOnChip Feature v.1.1
Interim Solution Developer's Guide

gemalto
security to be free

Product Version: 1.1

Document Part Number: 000-000000-001, Rev. A

Document Build: 1511-3

Release Date: November 2015

Contents

Preface

Support Contacts

If you encounter a problem while installing, registering or operating this product, please make sure that you have read the documentation. If you cannot resolve the issue, contact your supplier or Gemalto Customer Support. Gemalto Customer Support operates 24 hours a day, 7 days a week. Your level of access to this service is governed by the support plan arrangements made between Gemalto and your organization. Please consult this support plan for further information about your entitlements, including the hours when telephone support is available to you.

Contact Method	Contact Information	
Address	Gemalto, Inc. 4690 Millennium Drive Belcamp, Maryland 21017, USA	
Phone	US	1-800-545-6608
	International	1-410-931-7520
Technical Support Customer Portal	https://serviceportal.safenet-inc.com Existing customers with a Technical Support Customer Portal account can log in to manage incidents, get the latest software upgrades, and access the Gemalto Knowledge Base.	

1 Introduction

Overview

This document provides guidance to the ISV developer for implementing an interim solution that utilizes the Sentinel HL *AppOnChip* feature to protect important application code. The document discusses concepts, architecture, tools and processes that need to be employed in order to take advantage of this capability.

The *AppOnChip* solution provide a high level of security for a protected application by enabling the ISV to move the execution of important functions in the application from the end user's machine to a VM located within the Sentinel HL (Driverless configuration) key. These functions are immune to attempts by software pirates to decompile or debug the protected code by using existing cracking resources.

Important!

The Licensing API libraries provided in this package uses vendor libraries for the DEMOMA Batch Code only! To obtain the special customized vendor libraries that you require for your own Batch Code, contact SafeNet Technical Support.

Terminology

Term	Description
VM Engine	Virtual machine that runs in a Sentinel HL (Driverless configuration) key. The VM contains a standalone CPU that is independent of the CPU in the machine to which the key is connected.
VM Licensing API	A modified version of the Sentinel Licensing API. This version of the API supports additional interfaces that a protected application would call to access methods that have been programmed into a Sentinel HL key.
VM License Generation API	A modified version of the Sentinel License Generation API. This version of the API supports the generation of V2C files that can program VM resources into Sentinel HL keys.
VMBuilder utility	Utility that accepts the class file that contains the function to program in the Sentinel HL key. The utility generates the VM resource and the License template.

Term	Description
VM Resource VM Image	The VM resource contains the VM image (that is, encrypted Java code that contains a function from the protected application). The VM resource is created by the VMBuilder utility and is programmed into the Sentinel HL key, where it is decrypted and then executed by the VM Engine.
VMBuilder Tool	A utility with a graphical user interface that calls the VMBuilder utility and can perform a number of additional functions, up to and including the programming of a VM resource in a Sentinel HL key.
License Template	Template containing the VM resource, for the VM License Generation API. The License Generation API accepts a C2V file generated for the Sentinel HL key, and uses the License template to generate a V2C file.

Package Contents

The AppOnChip Developer Package contains the following directories:

Directory	Description
API	Contains the modified VM Licensing API libraries and sample code. This includes static library and dlls for 32-bit and 64-bit. Note: The libraries are provided only for the DEMOMA Batch Code. To obtain the libraries for your own Batch Code, contact SafeNet Technical Support.
Vendor Suite	Contains the VMBuilder utility and VMBuilder Tool, and the modified VM License Generation API libraries.

Compatibility and Prerequisites

- The AppOnChip feature is based on the Sentinel LDK v.7.4 Embedded License Manager with support for Sentinel HL (Driverless Configuration) keys.
- The AppOnChip feature only supports standalone Sentinel HL keys. Network keys are not supported.
- The VM Licensing API can coexist with LDK 7.4 LMS without conflict.
- A protected application that contains the VM License Generation API libraries does NOT require the presence of the Run-time Environment.

2 Enhancements in AppOnChip SDK v.1.1

Native SHA-256 interfaces

Sentinel HL keys support the SHA-256 cryptographic hash function starting from firmware version 4.51. This functionality is exposed to code running in the VM using native interfaces.

The following native methods have to been added to the Crypto class:

```
public native static void sha256Init();

public native static void sha256Update(String str);

public native static void sha256Update(byte[] data);

public native static void sha256Update(byte[] data,int off,int size);

public native static void sha256Finish(byte[] digest);
```

Native ECDSA crypto interfaces

Sentinel HL keys support ECDSA K-283 sign and verify operations starting from firmware version 4.51. This functionality is exposed to code running in the VM using native interfaces.

The following native methods have to been added to the Crypto class:

```
public native static void k283LoadEccKey(byte[] priv_key);

public native static void k283SignMessage(byte[] message_hash, byte[] signature);

public native static int  k283VerifyMessage(byte[] message_hash, byte[] signature,
                                            byte[] public_key);
```

A sample application showing the use case of signing a message, and verifying a signature, can be found in the java_sample source directory under VendorTools/VendorSuite/java_sample/src/apps/ECCK283.java.

The sign and verify functions in the example application use the SHA-256 cryptographic hash function to create the 32 bytes digest of the message.

The corresponding host-side (PC) sample application using the C API can be found under API/Runtime/vm/c/hasp_vm_demo_k283.c. The sample application shows how to invoke the signing and verification functions from the host using the C API.

Exception Handling using the C API

In the previous AppOnChip SDK version, the handling of exceptions thrown in the application running in the Sentinel HL VM, was only supported using the Java code.

The C API was enhanced to support exception handling using C code. In case an exception is thrown, the C API function hasp_vm_invoke() returns with status code 707 (HASP_VM_METHOD_EXCEPTION_OCCUR).

In case of an exception, the vm_returncode value that is returned by hasp_vm_invoke() contains the exception code, and the exception info is placed in the iobuffer that is allocated by the caller.

A sample application can be found under API/Runtime/vm/c/hasp_vm_demo_exception.c.

Enhancements in VMBuilder

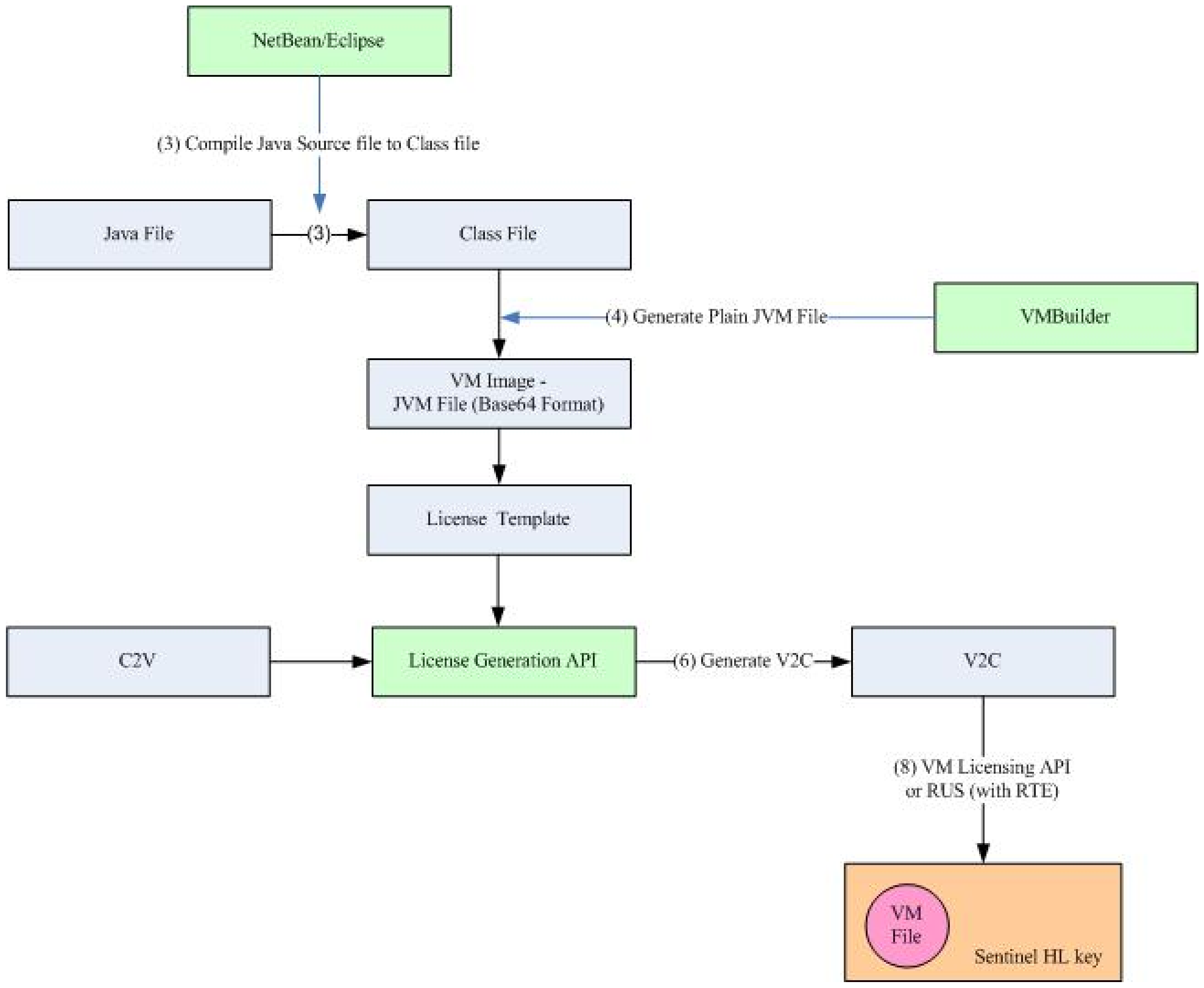
The VMBuilder Utility can now be used to program non-demo Sentinel HL keys. In the previous AppOnChip SDK, the VMBuilder Utility did only support programming of Sentinel HL demo keys. To program non-demo keys, the VMBuilder GUI Tool has to be used.

To program the VM resource containing your Java application into the non-demo Sentinel HL key, the VMBuilder Utility needs to access the Sentinel HL Master Key for the batch code.

The VMBuilder Utility also supports creating a LicGen template as output. This output can be used with the License Generation API to create V2Cs for individual Sentinel HL keys.

3 AppOnChip Feature - Implementation Process

This chapter describes the process for implementing the AppOnChip feature in the application that you want to protect.


NetBean/Eclipse
(3) Compile Java Source file to Class file
Java File
(3)
Class File
(4) Generate Plain JVM File
VMBuilder
VM Image - JVM File (Base64 Format)
License Template
C2V
License Generation API
(6) Generate V2C
V2C
(8) VM Licensing API or RUS (with RTE)
VM File
Sentinel HL key

The Sentinel HL AppOnChip feature is implemented as follows:

1. Identify one or more blocks of code in your application that should be moved to the VM on the Sentinel HL key. Translate these functions to Java.

   For information on the limitations on the Java code that you can use, see “Java VM Limitations” on page 25.

2. In the Java file, determine which function or method will be invoked by the protected application, and declare them as public and static.

   The following sample is a simple base16 algorithm java file:

```
public class Base16Encoder {

        private final static byte[] HEX = new byte[]{
              '0', '1', '2', '3', '4', '5', '6', '7',
              '8', '9', 'A', 'B', 'C', 'D', 'E', 'F' };

          /**
           * Convert bytes to a base16 string.
           */
          public static void encode(byte[] byteArray, byte[]hexBuffer) {
              for(int i = 0; i<byteArray.length ; i++ )
         {
                  hexBuffer[2*i] = HEX[(byteArray[i]& 0xff)>>4];
                  hexBuffer[2*i+1] = HEX[(byteArray[i]&0xF)];
         }
          }

          /**
           * Convert a base16 string into a byte array.
           */
           public static byte[] decode(String s) {
              int len = s.length();
              byte[] r = new byte[len / 2];
              for (int i = 0; i < r.length; i++) {
                  int digit1 = s.charAt(i * 2), digit2 = s.charAt(i * 2 + 1);
                  if (digit1 >= '0' && digit1 <= '9')
                      digit1 -= '0';
                  else if (digit1 >= 'A' && digit1 <= 'F')
                      digit1 -= 'A' - 10;
                  if (digit2 >= '0' && digit2 <= '9')
                      digit2 -= '0';
                  else if (digit2 >= 'A' && digit2 <= 'F')
                      digit2 -= 'A' - 10;

                  r[i] = (byte) ((digit1 << 4) + digit2);
              }
              return r;
          }
}
```

This sample contains two public static methods:

- `public static void encode(byte[] byteArray, byte[]hexBuffer)`
- `public static byte[] decode(String s)`

If the function is not static, the class object initialization method should be called first. For example:

```
<id>0</id>
<name>apps/Base16Encoder.init : ()V</name>
```

3. Use NetBean or Eclipse to compile the Java functions to Class file format.
4. Use the VMBuilder utility (or VMBuilder Tool) to convert this Base16Encoder class file.

   The files generated by the VMBuilder utility are:

   - VMBuilder configuration file (*.mvp): This file contains console parameters. This file can serve later as the input configuration file to the VMBuilder Tool for additional builds.
   - XML-based build result file (*.xml): This file contains the exposed methods list and the vm image.
   - License template file (*.xml): This file is used with the VM License Generation API to generate a V2C file to program the Sentinel HL key.

   For the sample java file listed earlier, the exposed methods list in the build result file would appear as follows:

```
<exposed_method>
<method>
<id>0</id>
<name>apps/Base16Encoder.init : ()V</name>
</method>
<method>
<id>1</id>
<name>apps/Base16Encoder.encode : ([B[B)V</name>
</method>
<method>
<id>2</id>
<name>apps/Base16Encoder.decode : (Ljava/lang/String;)[B</name>
</method>
</exposed_method>
```

As this list demonstrates, each exposed public method is assigned a unique method ID within this vm file. This method ID can be used by the VM Licensing API.

You can use the License Template file with the VM Licensing API to generate a V2C file for programming the VM resource into the Sentinel HL key. (A C2V file for the Sentinel HL key is also required.) When programming the VM resource file into the Sentinel HL key, you assign it a unique VM resource file ID. This ID is defined in license template.

You can also use the VMBuilder Tool to program the VM resource directly into the Sentinel HL key. For information on the VMBuilder Tool, see “VMBuilder Tool” on page 19.

The License Template file would appear similar to the following:

```
<?xml version="1.0" encoding="utf-8"?>
<sentinel_ldk:license schema_version="1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:sentinel_ldk="http://www.safenet-inc.com/sentinelldk">
<acknowledgement_request>Yes</acknowledgement_request>
<enforcement_type>HL</enforcement_type>
<dynamic_resource>
<vm_resource>
<!-- You can specify the VM resource file ID -->
<id>1</id>
<content> …</content>
</vm_resource>
</dynamic_resource>
</sentinel_ldk:license>
```

5. Use one of the available resources (RUS or the Licensing API) to generate a C2V file for the Sentinel HL key.
6. Using the VM License Generation API, generate a V2C file to program the VM resource file into the Sentinel HL key.
7. In the application to protect, use the VM Licensing API to invoke the method that is programmed in the Sentinel HL key.

The functions required to invoke methods that are programmed in the Sentinel HL key are described in “VM Licensing API – Additional Functions” on page 14.

The following C sample code demonstrates how these functions are used:

```
hasp_status_t      status;
hasp_handle_t      handle;
hasp_u32_t         vm_returncode = 0;
hasp_size_t             iolength = 0;
unsigned char      iobuffer[256];
char               byteArray[32] = { 0};
char               hexBuffer[64] ={ 0 };
VM_IO_BUFFER_T  iobuffer_cxt = { 0 };

iobuffer_cxt.cmdBuff = iobuffer;
 iobuffer_cxt.rspBuff = iobuffer;

/* vm resource file id = 1*/
status = hasp_vm_init(handle, 1);
 if (status) {
      hasp_logout(handle);
      printf("hasp_vm_init failed: %d!\n", status);
      exit(-1);
   }
   else
   {
```

```
            printf("hasp_vm_init succeeded!\n");
        }

        /*base 16 class input and output parameters preparation. */
        put_byte_array(&iobuffer_cxt, byteArray, 32);
        put_byte_array(&iobuffer_cxt, hexBuffer, 64);
        iolength = iobuffer_cxt.cmdSize;

        printf("public static void encode(byte[] byteArray, byte[]hexBuffer)
    \n");
        /* vm method id = 1*/
        status = hasp_vm_invoke(handle,
            1,
            iobuffer,
            &iolength,
            &vm_returncode);
        if (status) {
            hasp_logout(handle);
            printf("hasp_vm_invoke failed: %d!\n", status);
            exit(-1);
        }
        else
        {
            printf("hasp_vm_invoke succeeded!\n");
        }

        get_byte_array(&iobuffer_cxt, hexBuffer, 64);
        get_byte_array(&iobuffer_cxt, byteArray, 32);

       * hasp_vm_close must be called to free vm engine for other processes
    or threads usage*/
        status = hasp_vm_close(handle);
        if (status) {
            hasp_logout(handle);
            printf("hasp_vm_close failed: %d!\n", status);
            exit(-1);
        }
        else
        {
            printf("hasp_vm_close succeeded!\n");
        }
```

8. On the machine where the Sentinel HL key is connected, use the VM Licensing API to apply the V2C file to the HL key.

4
VM Licensing API – Additional Functions

VM Licensing API is identical to Sentinel Licensing API, but with the addition of three functions for working with the VM resource. This chapter describes the functions and error codes that were added for the VM Licensing API.

The additional three functions are provided as source code (hasp_vm_api.c and hasp_vm_api.h), which are implemented as wrappers around an internal function provided by the VM Licensing API. In future LDK versions, it is planned to integrate the three VM functions as part of the Sentinel Licensing API.

VM Functions

hasp_vm_init

Initializes the VM engine with the VM resource ID.

First call hasp_login to get the HL key session handle, and then call hasp_vm_init. After you call hasp_vm_init, the HL key VM engine is allocated to the session. The VM engine is freed when you call hasp_vm_close. Until the VM engine is freed, other applications or threads cannot use the VM engine.

```
hasp_status_t HASP_CALLCONV hasp_vm_init(
            hasp_handle_t *    handle,
            hasp_fileid_t      vm_id);
```

Field Name	Description
handle	Handle for the session.
vm_id	VM resource file ID

hasp_vm_invoke

Invokes a method located in a VM resource.

```
hasp_status_t  HASP_CALLCONV hasp_vm_invoke(
               hasp_handle_t      handle,
               hasp_methodid_t    method_id,
               void *             iobuffer,
               hasp_size_t *      length,
               hasp_u32_t *       returncode)
```

Field Name	Description
handle	Handle for the session.
method_id	Input for method ID of VM resource file.
iobuffer	Pointer to the io buffer of vm method. The maximum io buffer length is 256 bytes. The parameters in the io buffer must be stored as demonstrated by the following example: int methodA (int x, byte y, char [] z) will have the following layout in buffer:

Address	0	4	5	n
Content	X(4-byte)	Y(1-byte)	1-byte (char array size)	n-byte (char array data)

Field Name	Description
length	Pointer to the io buffer length.
returncode	Pointer to the vm method return code. This field is optional, depending on whether the method has return results. For example, if the method definition is like ”int method()”, the returncode field will contain the returned result.

hasp_vm_close

Frees the VM engine for other threads or applications.

```
hasp_status_t HASP_CALLCONV hasp_vm_close(
          hasp_handle_t *  handle)
```

Field Name	Description
handle	Handle for the session.

VM Return Codes

Number	Return Code	Description
701	HASP_VM_USED_ERR	VM engine is monopolized by other clients
702	HASP_VM_INT_ERR	VM engine internal error
703	HASP_VM_HEAP_OVERFLOW	Heap in VM engine execution overflow
704	HASP_VM_INVALID_CODE	Invalid vm code in the VM resource
705	HASP_VM_TYPE_ERR	Type or size is not correct in vm io buffer
706	HASP_VM_METHOD_NOT_FOUND	ID is not found in the VM resource
707	HASP_VM_METHOD_EXCEPTION_OCCUR	Exception was thrown

5 VMBuilder Utility

The VMBuilder utility accepts the class file that contains the methods to program in the Sentinel HL key. The utility generates the VM resource and License template. This chapter provides details regarding the VMBuilder utility.

VMBuilder Function

The VMBuilder utility performs the following tasks:

1. Reads and validates Java classes to make sure the target Sentinel HL key can support it.
2. Resolves class dependencies.
3. Resolves symbolic references.
4. Converts Java byte code to vm byte codes.
5. Removes unused data and code.
6. Identifies and reports access violation of objects and methods.
7. Assembles and create final VM image.
8. Creates a list of input and output parameter passing operations.

The VMBuilder utility generates the following output files:

1. VMBuilder configuration file (*.mvp): This file contains console parameters. The file can be used as input by VMBuilder Tool.
2. XML-based build result file (*.xml): This file contains the exposed method list and the vm image.
3. License template file (*.xml): This contains the VM resource and is used by the VM License Generation API. You can modify this file to specify the VM resource ID.

With this utility, no symbolic name is required in the final image. Even class names are converted to class IDs.

This version of VMBuilder supports up to 200 classes in a single image. Therefore, class IDs can be represented by a single byte.

Passing Configuration Information to VMBuilder

Use one of the following formats to invoke and pass information to the VMBuilder utility:

- **Configuration information provided in the command line:**

  ```
  VendorTools/VMBuilder [options] <classname> [<classname> ...]
  ```

  where:

classname	Package and class names (apps/license)
options	Available options are:

-c *classpath*	Package relative class path
-s *sourcepath*	Define root for source files
-o *outputpath*	Define path for java output files
-b binpath	Define path for class output files
-n *imagename*	Identifier for the image
-t serialnumber	Create image for the specified target (hex)
-l *filename*	Create log file
-v<n>	Set verbose level to n (n=0-3)
-u<n>	Burn vm image into the token (0: demoma, 1: non-demoma)
-i developerid	Developer Id (hex)
-p productid	Product Id (hex)
-f featureid	Feature Id (hex)
-j vmid	Java VM Id (hex)
-w size	VM Cache Size (hex)
-h	Display this help screen

- **Configuration information provided in a separate file:**

  ```
  VendorTools/VMBuilder @<filename>
  ```

  where:

filename	Name of file containing options, package and class names (apps/license). Each option and class name should be specified on a separate line in the file.

  The options that can be specified in the file are:

classname	Package and class names (apps/license)
-s *sourcepath*	Defines where source files are (debug only)
-o *outputpath*	Directory name where output files are created
-c *classpath*	Package relative class path Addressed class file: *classpath\\classname*.class
-n *imagename*	Name for the image (generated randomly if not defined)

VMBuilder Output Result File

The Output Result file is an XML file that is generated by VMBuilder. The file lists exposed methods and contains the vm image.

A sample result file is shown below.

```
<?xml version='1.0' encoding='utf-8'?>
<vmimage_file>
    <image_name>bbb</image_name>
    <exposed_method>
        <method>
            <id>0</id>
            <name>apps/Sum.init : ()V</name>
        </method>
        <method>
            <id>1</id>
            <name>apps/Sum.GetSum : (I)I</name>
        </method>
    </exposed_method>
    <vm_resource>
        <size>119</size>
        <content>
Wv4CAAABdwBLAFUABgAAAAgAUAAAAAAAAAAAAAAAAABbAGgAAAA/BAAAAAYFAAEBKDgAAQFACAA
GCQAGBQABARs8KQAGBQAaQAgAAwBiYmIAAD8AAAAAAAICABAABgAAJQskAAAAAgAsAAYAACUECy
kAAAACAgE=
        </content>
    </vm_resource>
    <image_checksum>8YbB8atHavdjTYw69r4Z2Q==</image_checksum>
</vmimage_file>
```

Note the following:

- Method ID is used in the protected application to call vm-related APIs.
- Base64 format VMImageCode is used by VM License Generation API to create a V2C file to program a VM resource file into the Sentinel HL key.

6
VMBuilder Tool

The VMBuilder Tool (VMBuilderView_windows.exe) provides all the functionality of the VMBuilder utility, but provides a graphical user interface to simplify the process of preparing inputs and invoking the utility. For use in an automated environment (for example, your build scripts or project file), the VMBuilder Tool cannot be used. In this case, use the VMBuilder Utility.

When the VMBuilder GUI tool is started, the VMBuilder Tool window is displayed.

The following workflows are available for the VMBuilder Tool:

- **Flow A**
  a. Generate a VM Resource using the VMBuilder Tool window. Click **Build**. Click **Apply**. The generated Template is automatically loaded.
  b. In the Apply VM Resource screen, optionally modify the VM Resource ID. Save the Template and/or apply the Template directly to a local Sentinel HL key or to a C2V file, to generate a V2C file.

- **Flow B**
  a. In the VMBuilder Tool window, just click **Apply**.
  b. In the Apply VM Resource screen, import an existing Template. Apply the Template directly to a local Sentinel HL key, or apply the Template to a C2V file, to generate a V2C file.

Using the VMBuilder Tool Window

If you want to proceed directly to the Apply VM Resource dialog box, launch the VMBuilder Tool and click **Apply** at the bottom of the VMBuilder Tool window.

To launch the VMBuilder Tool and generate the VMBuilder output files, do the following:

1. Run **Vendor Tools\VMBuilderView_windows.exe**. The VMBuilder Tool window is displayed.
2. (Optional) If you have a configuration file from a previous run of the VMBuilder Tool or VMBuilder utility, click **Load Config File** and browse to the VMP file. After loading the configuration files, you can modify the fields as necessary or skip directly to step 7.
3. In the **Image Name** field, enter an image name
4. In the **Output Path** field, enter or browse to the path in which VMBuilder output files should be placed.
5. In the **Class Path** field, enter or browse to the path that contains all the class files that you want to build.

6. In the **Class List**, select one or more classes for the build process.
7. Click **Build**. The build process executes.
8. In the **Output Information** pane, click any of the tabs at the bottom of the window to view the results of the build process. The information available includes:
   - Build result.
   - Method list, contains the method ID and the signature of the method.
   - VM image file in XML format.
9. (Optional) Click **Save Config File** to save the configuration information that you entered for a subsequent build process.
10. To apply the VM Resource to a local Sentinel HL key or to a C2V file, or to save the License Template file, click **Apply**. Continue with the Apply VM Resource dialog box described below.

Figure 1 shows an example of a completed VMBuilder Tool window.

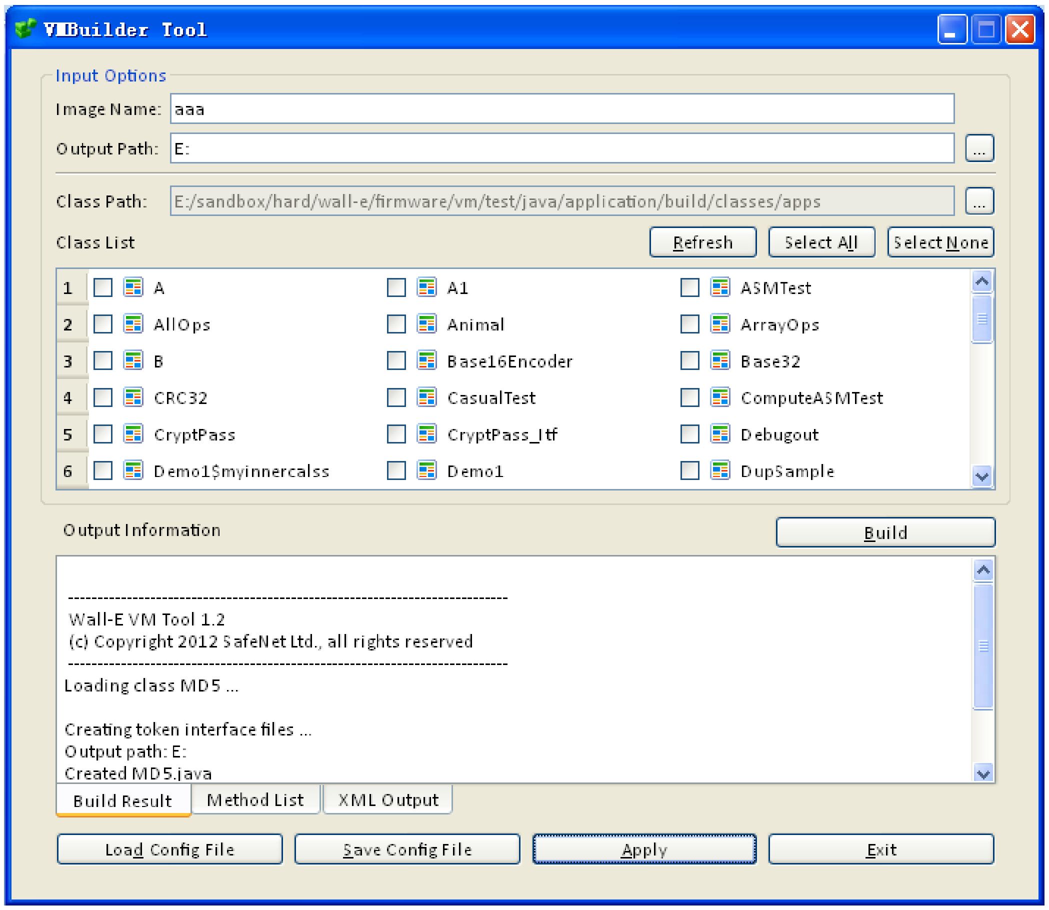


Image Name: aaa
Output Path: E:
Class Path: E:/sandbox/hard/wall-e/firmware/vm/test/java/application/build/classes/apps

1	A	A1	ASMTest
2	AllOps	Animal	ArrayOps
3	B	Base16Encoder	Base32
4	CRC32	CasualTest	ComputeASMTest
5	CryptPass	CryptPass_ltf	Debugout
6	Demo1$myinnercalss	Demo1	DupSample

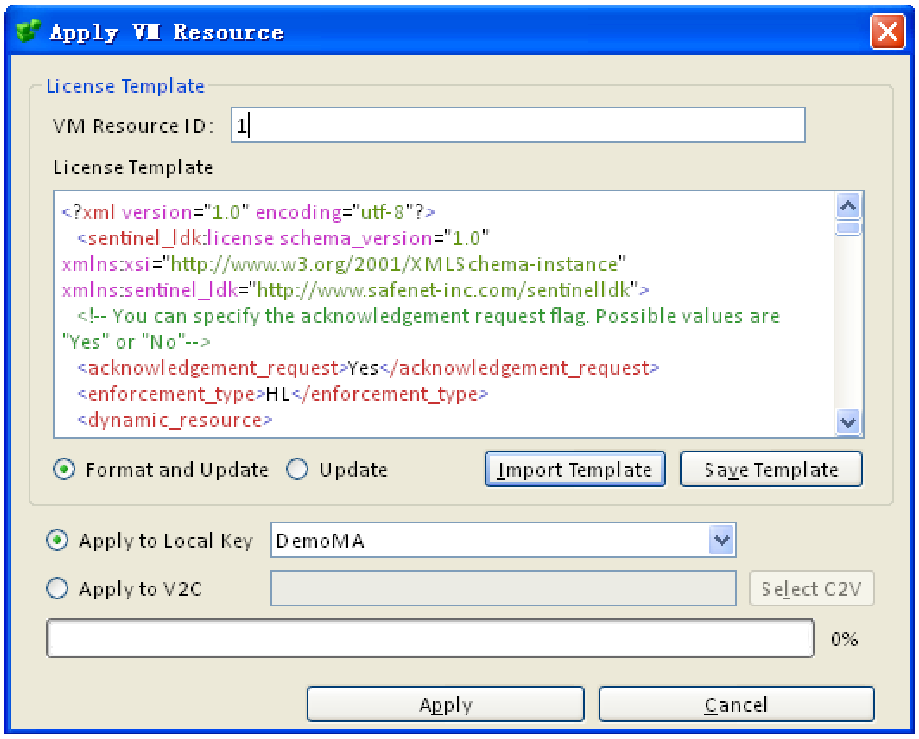
```
---------------------------------------------------------------------------
Wall-E VM Tool 1.2
(c) Copyright 2012 SafeNet Ltd., all rights reserved
---------------------------------------------------------------------------
Loading class MD5 ...

Creating token interface files ...
Output path: E:
Created MD5.java
```

Figure 1. VMBuilder Tool Window - Example

Using the Apply VM Resource Dialog Box

You can use Apply VM Resource dialog box to:

- Apply the VM Resource generated above to generate a License template. You can modify the Resource ID for the License template.
- Import a different License template from the local disk.
- Use the generated or imported License Template to program a local Sentinel HL key directly or to generate a V2C file from a C2V file.



VM Resource ID: 1

License Template

```
<?xml version="1.0" encoding="utf-8"?>
  <sentinel_ldk:license schema_version="1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:sentinel_ldk="http://www.safenet-inc.com/sentinelldk">
  <!-- You can specify the acknowledgement request flag. Possible values are
"Yes" or "No"-->
  <acknowledgement_request>Yes</acknowledgement_request>
  <enforcement_type>HL</enforcement_type>
  <dynamic_resource>
```

[✓ Format and Update] Update

[✓ Apply to Local Key] DemoMA

Apply to V2C

0%

Figure 2. Apply VM Resource dialog box

To use the Apply VM Resource dialog box, do the following:

1. In the VMBuilder Tool window, click the **Apply** button.
2. If you generated a VM Resource using the VMBuilder Tool, the dialog box displays the License Template.

   You can optionally modify the VM Resource ID (assign a value from 1 to 65471, inclusive). Each new VM Resource in a protection key must be assigned a unique VM Resource ID. Default value is 1. The display of the License Template is automatically refreshed to show the value that you entered.

   OR

   Click **Import Template** to import a License Template into the dialog box.
3. Select one of the following program modes:

Update	Adds the VM license directly to the Sentinel HL key.
Format and Update	Formats the Sentinel HL key and then adds the VM license.

4. To program a local Sentinel HL key, select **Apply to Local Key**, and then select the relevant Batch Code from the list.
5. To apply the Template to a V2C file, select **Apply to V2C**, and then click Select C2V to locate the relevant C2V file.
6. Click **Apply**. The License Template is applied to the selected Sentinel HL key or V2C file. The progress bar indicates the progress of the process.
7. To save the displayed License Template file, click **Save Template**. You can use this License Template with the VM License Generation API to generate a V2C file to update a local or remote Sentinel HL key.

7 VM License Generation API Library

Template definition

The modified VM License Generation API library provided in the package has been enhanced to handle VM resources under the “HL” Enforcement type. Data of the VM resources is taken as input through license definitions for Sentinel HL (Driverless configuration) keys. You can manage VM resource data under the <vm_resource> tag in the following manner (within the <dynamic_resource> tag).

```
……
<memory>
   <!-- default ro memory -->
   <ro_memory_segment>
      <offset>0</offset>
      <content>pd94bwwgdmvyc2lvbj</content>
   </ro_memory_segment>
</memory>
……
<!- dynamic resource -->
<dynamic_resource>
   <!– isv data resource can be added here -->
  ……
   <!– vm resource-->
   <vm_resource>
      <id>1</id>
      <content>xyz</content>
   </vm_resource>
  ……
   <vm_resource action = "cancel">
      <id>2</id>
   </vm_resource>
   <!– some other resources can be here in future(i.e. Aes res etc.) -->
</dynamic_resource>
```

License definition for VM resource file should have the following characteristics:

1. VM data should be specified under the <vm_resource> element.
2. A VM resource can be updated as a single data blob only. Multiple segments are not supported here.
3. A VM resource has only executable properties. Therefore, no “attribute” is required explicitly; the attribute is by default “executable”.
4. The only explicit attribute that is supported for <vm_resource> element is **action = “cancel”**. This attribute is used to delete the VM resource from the Sentinel HL key.
5. The file ID can be assigned a value from 1 to 65471 (inclusive).

Modifying Dynamic Memory in the Sentinel HL Key

VM resource files are stored in Sentinel HL keys in the dynamic memory.

The table that follows describes how the License Template is used with the VM License Generation API to update VM resource files in the dynamic memory of a Sentinel HL key

Action	License Template	Description
Create a file in dynamic memory and update its content	`<dynamic_resource> <vm_resource> <id>1</id> <content>xyz</content> </vm_resource> </dynamic_resource>`	1. Creates a VM resource file with the given ID. The size of the file is determined by the content provided. (No <size> tag is required.) 2. If content is not provided, an error for Invalid xml definition is returned.
Update the file in dynamic memory	`<dynamic_resource> <vm_resource> <id>1</id> <content> MTIK</content> </vm_resource> </dynamic_resource>`	If a VM resource file was already created for the given ID, updates the content of that file. If the new content size is different from the original size, the size of the file is modified as required.
Remove the file from dynamic memory	`<dynamic_resource> <vm_resource action = "cancel"> <id>1</id> </vm_resource> </dynamic_resource>`	Action “cancel” is required to remove a VM resource file. If a file with the given ID does not exist, the definition is ignored. Only the ID tag is expected here.
Apply “Clear and Update” into the license definition		VM resource files are not removed from the dynamic memory.
Apply “format and Update” into the license definition		All VM resource files are removed from dynamic memory.

8 Java VM Limitations

This chapter describes limitation on the Java code that is programmed in the Sentinel HL key.

Java VM Compromises

The Sentinel HL key is a resource-constrained device. To support Java while providing an acceptable level performance, certain compromises had to be made.

The Sentinel HL key supports a subset of the full Java VM:

- Single threaded support only. This is a limitation at the device level. However, at the host side, multiple threads may access Java classes inside the Sentinel HL key. The host side library and driver is expected to serialize device access.
- Primitive types:
  - Boolean, byte, char, short, long, and int are supported.
  - Float and double are NOT supported in the current version of the firmware (v.4.5.1). Fixed point float may be added in a future version.
- Exceptions are treated like errors in the Sentinel HL key. However, they are propagated to the host and raised by the Sentinel HL key library.
- Automatic garbage collection is not provided. Instead, a native method called Free(Object) is provided and should be invoked by the Java applications whenever an object goes out of scope.
- Memory of non-static fields is freed after each method invocation. Data that is required to live across multiple method invocations must be declared as static.
- There is a limit of 256 bytes for the data exchange on every method call issued by the host side.
- Limited support exists for the base classes.

Base Class Limitations

- Object class: Basic methods are supported. Since multi-threaded is not supported, the synchronization methods are not provided.
- String class: Limited string support is provided. Input strings are converted to UTF-8 and converted back on the return.
- Exception classes: Exceptions can be generated in the Sentinel HL key and can be thrown, but catch is not supported in the Sentinel HL key.