***Detailed Design Specifications***

***For***

**(RS-EMS)**

Document ID:

Document Revision: 1.0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ***Function*** | ***Name*** | ***Elec. Signature & Date*** | ***Date*** |
| **Prepared by:** | SW. Engineer | Tanya Birger |  | 04.01.16 |
| **Reviewed by:** | Project lead | Shai Zaban |  |  |
| **Approved by:** |  |  |  |  |

Contents

[1.0 Background 2](#_Toc439680393)

[2.0 Tools and Technologies 2](#_Toc439680394)

[3.0 RS-EMS Overview 2](#_Toc439680395)

[3.1. IDT-EM is composed from sub-modules according to the following diagram: 2](#_Toc439680396)

[3.2. Operational IDT-EM is composed from four sub-modules: 2](#_Toc439680397)

[3.3. IDT-Srv interface for the application that uses the library 2](#_Toc439680398)

[4.0 Integration Raccon with Raccoon Enhanced Dongle (RED) 3](#_Toc439680399)

[4.1. RED-App provide the IDT-the following services: 3](#_Toc439680400)

[4.2. Major changes in Raccoon SafeNet: 3](#_Toc439680401)

[5.0 Key Sequences. 4](#_Toc439680402)

[A. Remove cartridge 4](#_Toc439680403)

[B. Cartridge Authentication 4](#_Toc439680404)

[C. Consumption Update 7](#_Toc439680405)

[6.0 Appendix 1 – IDT-HW-Inf API 9](#_Toc439680406)

[7.0 Appendx 2 – FCB driver API 9](#_Toc439680407)

# Background

The **Raccoon** project alerts and blocks the usage of non-genuine cartridges inserted to the printer. This system is composed of a tag with a cryptographic chip, attached to the cartridge and software that read the tag data and update it with the material left in the cartridge.

The RS-EMS library takes care of all tag related operations and provides the printer software with authentication and consumption update services.

The original RS-EMS used a cryptographic library for verifying the tag responses. The new version takes advantage of the cryptographic chip embedded in the Dongle. The RS-EMS should be adapted to the following changes:

* All verification functions were moved to the dongle
* Only the dongle keeps the Stratasys public key
* Only the dongle keeps the public key and the weight of all active cartridges
* The dongle generates random values and verifies them

# Tools and Technologies

As a module embedded in SSYS printer, IDT-EM is developed in C++.

REDApp\_Utils API is used to communicate with Dongle.

VaultIC100 API is used to communicate with the IDT.

### 

# RS-EMS Overview

## IDT-EM is composed from sub-modules according to the following diagram:

|  |  |
| --- | --- |
| IDT-Srv | |
| IDT-API | |
| IDT-Comm | IDT-API Comm | |
| FCB-Driver | Aardvark driver | |

## Operational IDT-EM is composed from four sub-modules:

* IDT-Srv: the interface for identification and monitoring.
* IDT-API: API for using the VaultIC100 tag*.*
* IDT-Comm: communication layer between IDT-API and FCB driver.
* FCB-Driver: communication layer for front connection board*.*

## IDT-Srv interface for the application that uses the library

* InitApi(): Initializes module components and resources.
* InitHw(): Initializes underlying hardware.
* GetInPlaceStatus(): gets the connection status of all IDTs.
* AuthenticateCartridge(): Validates digital signature of a specified cartridge over its IDC and a random-generated token. The IDC is returned as an output parameter.
* RemoveCartridge(): Mark cartridge as unauthenticated.
* UpdateConsumption():validate and update consumption.

# Integration RS-EMS with Raccoon Enhanced Dongle (RED)

The RED solution takes advantage of the cryptographic chip embedded in the dongle, as well as RAM and JVM that can run general purpose software.

The RED software is composed of:

* RED-Crypto API - Methods that activated the crypto algorithms runs on the embedded chip
* RED-App - Authentication & consumption update functions that will be called from the printer software and use the crypto methods

## RED-App provide the IDT-the following services:

* **public static boolean verifyTag(byte tagNo, boolean isActive, byte[] certificate, byte[] random)**

tagNo, isActive, certificate – input parameters

random – output parameter (however byte array of length 8 should be initialized in the client code)

* **public static boolean verifyChallenge(byte tagNo, boolean isActive, byte[] signature)**

tagNo, isActive, signature – input parameters

signature – 72 bytes

* **public static boolean removeTag(byte tagNo)**

tagNo – input parameter

* **public static boolean updateConsumption(byte tagNo, boolean isActive, int consumption, byte[] random)**

tagNo, isActive, consumption – input parameters

random – output parameter (however byte array of length 8 should be initialized in the client code)

* **public static boolean verifyWeight(byte tagNo, boolean isActive, byte[] signedWeight)**

tagNo, isActive, signedWeight – input parameters

## **Major changes in RS-EMS:**

In the current project 3 Raccon API functions should be changed by adding calls RED-App functions on dongle for verification intead of crypto pp functions:

* AuthenticateCartridge()
* RemoveCartridge()
* UpdateConsumption()

Key Sequences below describe these changes.

Questions:

# Key Sequences.

# Remove cartridge

Remove cartridge is executed by calling RemoveCartridge(numCart) method - MaterialMonitor requests the IDT reset catridge data. We should add to this function call to Dongle :

***removeTag(in tagNo): hasp\_status\_t***



# Cartridge Authentication

Cartridge authentication is executed by calling AuthenticateCartridge() method. It is executed on power-up, or when a cartridge is replaced. Cartridge authentication steps after getting relevant data with calls:

***ResetCartridgeInfo*** / ***SwitchToCartridge***/ ***ReadMaterialInfo*** are:

1. ***ReadSignature (in out signature,in out signature) : int***– MaterialMonitor sends a request to the IDT for its signature.
2. ***GenerateCertificate():int***
3. ***verifyTag (in certificate: IDCertificate, in isActive, out random, out result): hasp\_status\_t***– call to Dongle for IDC verifying according to Stratasys public key (PubKS);

Receive random array for the next step

1. ***ChallengeHostDevice (in cartridgeID: char, in token: Token, out signedToken: Token): int*** – MaterialMonitor sends the received random token to the IDT for signing by PrvKC; the signed token is set into out parameter.
2. ***verifyChallenge (in signature, in isActive, out result): hasp\_status\_t***– call to Dongle with signed random token to verify it.

In any step fails, i.e. a false or non-zero value is returned, process stops and fails. A corresponding error code is returned. The IDC is returned at the end of the process as out parameter.



# Consumption Update

Resin consumption update is executed by calling UpdateResinConsumption() before printing a slice or bitmap. Consumption update steps are:

1. ***SwitchToCartridge (in cartridgeID)*** ***: int***
2. ***updateConsumption(in handle***, ***in tagNo, in isActive, in consumption, out random, out result):*** ***hasp\_status\_t –*** call to Dongle with input consumption and receive success and random array for weight signing in the next step.
3. ***DecreaseVolume(in cartridgeID: char, in consumption: unsigned int, in random, out newVolume: CounterResponse): int*** – MaterialMonitor requests the IDT to subtract the consumption from current volume. The IDT returns the new volume, signed with PrvKC.

**If** ***DecreaseVolume return error= OVERCONSUMPTION*** MaterialMonitor calls to Adapter DecreaseVolume(0) in order to receive current volume and return ***error= OVERCONSUMPTION***

1. ***verifyWeight(in handle***, ***in tagNo, in isActive, in consumption, in signedWeight, in signedWeightLen***, ***out result):*** ***hasp\_status\_t –*** call to Dongle to verify signed weight***,***

which composed from weight signed with PrvKC and received random in step 2.

In any step fails, i.e. a false or non-zero value is returned, process stops and fails.



# Appendix 1 – IDT-HW-Inf API

* int IDThwInit(int \*cartridgesCount) – Init or rest HW.
* int IDThwGetInPlaceStatus(int \*status) – Gets a bitmask representing the switching status of all cartridges.
* int IDThwSelectChannel(int cartridgeNum) – Selects a specified cartridge to send and receive datas from.
* int IDTSendData(int bufferSize, char \*buffer) – Sends data to the selected cartridge.
* int IDTHWINF\_LIB IDTReceiveData(int bufferSize, char \*buffer) – Receive data from the selected cartridge.

# Appendx 2 – FCB driver API

* Int FCBInit (Num) – init or reset the FCB and its driver
* Int FCBGetStatus (Num)– performs a self-test and returns its status
* Int FCBGetCartridgesStatus (Num, &Status) – return status with 8 bits for the state of each cartridge
* Int FCBReadIDTFile (IDTn, Filename, &DataBuffer) &
* Int FCBWriteIDTFile (IDTn, Filename, DataBuffer) – we do not know yet which files we have and what are their structure. I think that the driver should have a single read and write pair to read and write a buffer. The assignment into specific structure shall be in the wrapping library.

**End of document**