***Detailed Design Specifications***

***For***

**IDT-EM**

Document ID:

Document Revision: 1.0

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

Table of Contents

[1.0 Identification 3](#_Toc352596061)

[2.0 Introduction 3](#_Toc352596062)

[2.1. Purpose 3](#_Toc352596063)

[2.2. Scope 3](#_Toc352596064)

[2.3. Definitions, Acronyms, and Abbreviations 3](#_Toc352596065)

[2.4. References 3](#_Toc352596066)

[3.0 Design Considerations 3](#_Toc352596067)

[3.1. Major Design Constraints/Limitations 4](#_Toc352596068)

[3.2. Major Design Decisions 4](#_Toc352596069)

[3.3. Tools and Technologies Used 4](#_Toc352596070)

[3.4. Safety Mitigation 4](#_Toc352596071)

[4.0 Module Design 4](#_Toc352596072)

[4.1. System Architecture 4](#_Toc352596073)

[4.2. High Level Design 5](#_Toc352596074)

[4.3. Detailed Component Design 7](#_Toc352596075)

[5.0 Requirements Cross Reference 9](#_Toc352596076)

[6.0 Packaging 9](#_Toc352596077)

[7.0 Revision History 9](#_Toc352596078)

[Appendix 1 11](#_Toc352596079)

# Identification

This document describes the detailed design of the IDT-EM module to be integrated with SSYS printer software and its testing aids.

# Introduction

## Purpose

The document describes in details the detailed design of the IDT-EM module, focusing on the IDT-Srv sub-module. The detailed design includes the module components and the inter-relations between the module and Stratasys printer software.

## Scope

The document concentrates on the IDT-EM module only; it deals neither with other component of the Raccoon system, nor with simulators or debugging modules.

## Definitions, Acronyms, and Abbreviations

|  |  |  |
| --- | --- | --- |
| EM-SW | - | Embedded (printer) SW |
| ECC | - | Elliptic Curve Cryptography |
| FCB | - | Front Connection Board |
| FCB Driver | - | FCB-based hardware driver |
| IDC | - | ID Certificate |
| IDT | - | ID Tag |
| IDT-API | - | API library provided by the IDT vendor |
| IDT-Comm | - | Comm. Layer between IDT API and the IDT-HW |
| IDT-HW | - | General term for the IDT interfacing HW (FCB is the specific implementation for Keshet) |
| IDT-HW-Inf | - | Raccoon hardware interface library |
| IDT-Lib | - | The main RS-EMS deliverable (IDT-Srv + IDT-Comm) |
| IDT-Srv | - | IDT services requires by the application |
| PrvKC | - | Cartridge Private Key |
| PrvKS | - | Stratasys Private Key |
| PubKC | - | Cartridge Public Key |
| PubKS | - | Stratasys Public Key |
| RS-Sim | - | Raccoon System (FCBs & IDTs) Simulator |
| RS-Tester | - | Independent testing & acceptance application |
| SSYS | - | Stratasys |

## References

RPS-SW SRS

# Design Considerations

## Major Design Constraints/Limitations

IDT-EM shall be fully integrated in Raccoon system. Hence:

1. It shall be executed over Windows 7 and Linux OS.
2. It shall interop with IDT.
3. It shall agree with the IDT cryptography.
4. It shall be embedded in SSYS printer.
5. It shall be testable by an IDT simulator and IDT-tester.

## Tools and Technologies Used

As a module embedded in SSYS printer, IDT-EM is developed in C++. It uses an external cryptography library, the Crypto++ 5.6.2.

VaultIC100 API is used to communicate with the IDT.

# Module Design

## System Architecture

RS-EMS is composed from modules according to the following diagram.

The green items are provided by Giga Ltd. and delivered to SSYS.

The orange items are provided by Giga Ltd. for testing and debugging purposes only, and delivered to SSYS by request:

|  |  |
| --- | --- |
| EM-SW (Printer) | IDT-Tester |
| **IDT-EM** | |
| IDT | |

IDT-EM is compiled and built as a native dll in two different modes:

* Operational mode – to be used by printer SW.
* Testing mode – to be used by IDT-Tester. IDT-EM is built in testing mode for testing and debugging purposes only, and it is not meant to be shipped in this mode.

On testing mode, IDT-Tester, IDT-EM and IDT-Sim shall be executable on different machines; to access IDT-EM from remote, library will be wrapped by Windows service or Linux daemon.

## High Level Design

### 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:

1. IDT-Srv: the interface for identification and monitoring.
2. IDT-API: API for using the VaultIC100 tag; *provided by Inside Secure.*
3. IDT-Comm: communication layer between IDT-API and FCB driver.
4. FCB-Driver: communication layer for front connection board; *provided by SSYS.*

Alternatively, IDT-Comm and FCB-Driver can be replaced with original IDT-API communication layer to work directly with a single IDT, or with IDT-SimmCom to work with IDT simulator. Those alternatives are out of this document’s scope.

### IDT-Srv

API of IDT-EM has to expose two functions for the printer SW: cartridge authentication and consumption update, which are both a part of the ResinMonitor component. In addition, ResinMonitor can set identification data into IDT for testing and integration purposes only.

Cartridge authentication is executed on power-up, or when a cartridge is replaced; consumption update is executed on every bitmap or slice printing.

The ResinMonitor component accesses the IDT by an IDTagAdapter.

The concrete type of IDTagAdapter is determined by IDTagAdapterFactory component, according to module configuration. IDT-EM provides one concrete implementation of IDTagAdapters, the VaultIC100TagAdapter.



### IDT-Srv main components

* ResinMonitor: The main component of the IDT-EM. This class is responsible for executing the authentication and consumption update processes, as well as setting cartridge identification data in testing.
* IDTagAdapter: An abstract class which adapts the ResinMonitor interface to the IDT interface.
* IDTagAdapterFactory: A factory class which creates the concrete type of IDTagAdapter, according to IDT-EM configuration, determined in run-time.
* VaultIC100Adater: A concrete IDTagAdapter for VaultIC100 tag.

### IDT-Comm

This module is a part of the IDT-EM communication layer, as illustrated in the following diagram:

vaultic\_comms

vaultic\_protocol

vaultic\_block\_protocol

vaultic\_fcb\_peripheral

IDT-API

IDT-Comm

vaultic\_spi\_peripheral

vaultic\_twi\_peripheral

IDT-Comm contains a single component, vaultic\_fcb\_peripheral, which contains global functions which communicates with FCB. Pointers to those peripheral functions are set into vaultic\_block\_protocol component of the IDT-API communication layer.

### Provided interface

The IDT-EM API exposes the IResinMonitor interface, which provides entry points of executing authentication and volume consumption, as well as setting identification data in testing mode. It includes the following methods:

* InitApi(): Initializes module components and resources.
* InitHw(): Initializes underlying hardware.
* GetInPlaceStatus(): gets the connection status of all IDTs.
* BurnIDC(): Generates certificate out of resin information and generates tag public key pair. This method is accessible in testing mode only.
* 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.
* UpdateResinConsumption(): Checks if cartridge is allowed to consume a specified volume of resin.

All methods return integer codes: 0 if operation completed successfully, other values for errors. See apeendix 1 for a full list of error codes.

### Threading Model

IDT-EM is a single threaded module. To ensure thread safety, AuthenticateCartridge() and UpdateResinConsumption() method are locked by critical section objects.

### Error Handling

All API methods are exception-safe. All methods use return codes to report errors. See apeendix 1 for a full list of error codes.

### Security

#### IDT-EM Security

ResinMonitor component uses Crypto++ library for signing data and verifying signatures. The digital signature algorithm chosen is ECDSA over finite fields (EC2N), using K-283 curve and SHA256 hash function. The component holds no private keys. No other component of IDT-EM deals with security.

See appendix 2 for more details.

#### IDT Security

PrvKC is burnt on VaultIC100 tag in a burning station. IDT includes illegal code execution prevention and protection against side channels attacks and probing. It can detect tampering attempts and destroy sensitive data on such events.

#### Traffic Security

On power-up, or when cartridge is replaced, it performs mutual authentication with IDT-EM. IDT-EM retrieves writing privileges only on testing mode.

## Detailed Component Design – IDT-Srv

### IResinMonitor

IResinMonitor defines the module interface (see 4.2.5).

### ResinMonitor

ResinMonitor is the main component of the IDT-EM, which implements IResinMonitor. This class is responsible for executing the authentication and consumption update processes, as well as setting identification data to tag in testing mode. It is a singleton object, initialized on power-up stage, and destroyed on shutdown. It holds a reference to IDTagAdapter for executing operations on IDT, a common verifier for SSYS signature, and an array of certificates of all authenticated cartridges.



### IDCertificate

This class composes cartridge identification data, PubKC and message signature and holds it in the following properties:

* IDD – a data structure holding cartridge identification data: resin name, resin batch number, production date, material expiration date, full cartridge weight and chip serial number.
* PublicKey – holds PubKC.
* SignedIDD – digital signature of IDD message digest.

In addition, it contains an Encode() and Decode() methods, to convert its content into a textual message and vice versa.



### IDTagAdapter

This class is an abstract class, which provides abstract methods for initialization and executing the steps of authentication and consumption update processes on any IDT. The provided methods are:

* Init
* SetIDCertificate
* GetIDCertificate
* ChallengeToken
* GetCurrentVolume
* DecreaseConsumption

In addition, IDTagAdapter holds the current cartridge ID for thread synchronization.

### IDTagAdapterFactory

This class is responsible for instantiating IDTagAdapter concrete type. It contains a single static method, CreateTagAdapter(), called by ResinMonitor at constructor. The type of IDTag adapter to create is defined as enum and passed to this method as an argument.

ResinMonitor is responsible for destroying the concrete IDTagAdapter.

ResinMonitor holds a reference to IDTagAdapter, and it is unaware of its concrete type.



### VaultIC100Adapter

This class instantiates IDTagAdapter in order to use VaultIC100 as IDT.

#### Initialization

General Initialization is implemented by overriding the Init() method, according to Vault\_IXX sample code:

* Call OpenLibrary
* Call GetLibSymbol() to get pointers to the following functions:
  + VltInitLibrary
  + VltCloseLibrary
  + VltGetApi
  + VltFindDevices
  + VltGetStrongAuthentication
* Build communication parameters and call VltInitLibrary.
* Call VltGetApi and set vltApi handle.
* Call VltGetStrongAuthentication.

If any of the steps fails, call VltCloseLibrary to exit gracefully with the corresponding error code.

#### Burning Certificate

VaultIC100Adapter uses the VaultIC100 API methods to instantiate the SetIDCertificate. SetIDCertificate executes the following steps:

* VltInitializeAlgorithm: set elliptic curve and hash function.
* VltWriteFile: writes the certificate file, containing identification data, its signature by PrvKS and PubKC.

## Detailed Component Design – IDT-Comm

### vaultic\_rshw\_peripheral

This component implements global functions of Init, Close, IOCTL (IO control), SendData and ReceiveData to fully communicate with FCB via rshw component. Pointers to those functions are set into vaultic\_block\_protocol component of the IDT-API.

### rshw

This component is a mediator between vaultic\_rshw\_peripheral and IDT-HW-Inf module, which loads IDT-HW-Inf module and its entry points (see Appendix 3 for IDT-HW-API).

## Key Sequences

#### 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 are:

1. ***GetIDCertificate(in cartridgeID: char, out certificate: IDCertificate) : int***– ResinMonitor sends a request to the IDT for its IDC.
2. ***VerifyCertificate(in certificate: IDCertificate): bool*** – IDC is verified according to Stratasys public key (PubKS); PubKC is extracted from IDC and stored for step 5.
3. ***GenerateToken(): Token*** – a random token is generated using Crypto++ random numbers pool.
4. ***ChallengeToken(in cartridgeID: char, in token: Token, out signedToken: Token): int*** – ResinMonitor sends the random token to the IDT for signing by PrvKC; the signed token is set into out parameter.
5. ***VerifyToken(in pubKC: PulibKey, in originalToken: Token, in signedToken: token): bool*** – ResinMonitor signs the original token with PubKC and compares the result with the signed token retrieved from IDT and returns true value if they are equal.

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. ***GetCurrentVolume(in cartridgeID: char, out currentVolume: CounterResonse): int*** – ResinMonitor reads the current resin volume of a cartridge.
2. ***DecreaseVolume(in cartridgeID: char, in consumption: unsigned int, out newVolume: CounterResponse): int*** – ResinMonitor requests the IDT to subtract the consumption from current volume. The IDT returns the new volume, signed with PrvKC.
3. ***VerifyVolume(signedNewVolume: CounterResponse): bool*** – the new volume is verified by PubKC and compared with the expected volume.
4. ***GetCurrentVolume(in cartridgeID: char, out currentVolume: CounterResponse): int*** – ResinMonitor reads the current resin volume of a cartridge. The decreased volume shall be returned in output parameter this time.
5. Sign volume retrieved by step #4 by PubKC.
6. ***VerifyVolume(signedNewVolume: CounterResponse): bool*** – the new volume is verified by PubKC and compared with the expected volume.

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



## Major Design Decisions

### Resources Management

In order to obtain stability and to avoid memory leaks, resources, such as sockets or file handlers, are initialized in a dedicated Init() method; if resources are already initialized, calling Init() frees the old handlers first. Data members which will (almost) definitely initialized successfully are initialized in constructors.

### Authentication and Consumption Update

Processes of both authentication and consumption update are specified in IDT-EM SRS. Both processes will be under the responsibility of a single component, the ResinMonitor, for two reasons:

* Conceptual: both processes are a part of the same role of monitoring resin consumption.
* Simplicity: easier to develop and maintain a single component and manage its resources.

### Abstraction of IDT

The IDT of Raccoon is VaultIC100, provided by Inside Secure. However, the testability of IDT-EM requires that it would be able to connect to IDT-Sim; besides, the IDT vendor could be replaced. To achieve the required modularity, ResinMonitor module uses an abstract adapter to the IDT, the IDTagAdapter; the concrete type of IDTagAdapter is determined by a factory class according to module configuration.

### Logging

Operational IDT-EM requires no logging, so module interface methods returns error codes, varied as possible. On testing, a simple trace file will be handled.

# Requirements Cross Reference

|  |  |
| --- | --- |
| **SRS** | **DDS** |
|  |  |

# Packaging

NA

# Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Revision** | **Date** | **Status** | **Changed by:** | **Reason/ Description of change.  (Insert Technical Review record)** |
| 1.0 | 11.05.2014 | Done | Itay Sofer | Document Creation |

**Appendix 1 – API Error Codes**

This list is TBD.

**Appendix 2 – setting ECC curve**

If P-256 is not supported, we should use K-283 (or B-283) which are equivalents from crypto strength perspective (although not recommended on NIST Suite B crypto algorithms)

The difference is the curves used for the ECC crypto.

Each curve its unique parameters (computing a curve is a task by itself)

The P curves are curves over the field of GF(P) , while K are Koblitz curves over GF(2\*\*M) and B are non-Koblitz curves also over GF(2\*\*M)

The Koblitzs curves are usually faster then P curves.

*(by David Moshkovitz)*

**Appendix 3 – 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.

**Appendix 4 – 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**