Crypto SDK

Software Requirements Specifications (SRS)

**Change Control Information**

|  |  |  |  |
| --- | --- | --- | --- |
| Version No. | Date | Changed by | Scope of Change |
| 1.00 | June 7, 2010 | Yael Ejgenberg, Meital Levy | First draft |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1 INTRODUCTION 3](#_Toc264365742)

[*1.1* Context 3](#_Toc264365743)

[*1.2* Product Overview 3](#_Toc264365744)

[*1.3* Purpose 3](#_Toc264365745)

[*1.4* Scope 4](#_Toc264365746)

[*1.5* Reference 5](#_Toc264365747)

[1.6 Definition And Abbreviation 5](#_Toc264365748)

[2 OVERALL DESCRIPTION 5](#_Toc264365749)

[2.1System environment 5](#_Toc264365750)

[2.1Product Functions 5](#_Toc264365751)

[2.2User Characteristics 5](#_Toc264365752)

[2.3General Constraints 6](#_Toc264365753)

[2.4Assumptions and Dependencies 6](#_Toc264365754)

[3 6](#_Toc264365755)

[3.1 Functional Requirements Definition 6](#_Toc264365756)

[3.1.1 Overview of the system 6](#_Toc264365757)

[3.1.2 General SDK 8](#_Toc264365758)

[3.1.3 Get random string 9](#_Toc264365759)

[3.1.4 Encrypt/Decrypt 9](#_Toc264365760)

[3.1.5 1 out of n Oblivious Transfer 11](#_Toc264365761)

[3.1.6 Zero knowledge 12](#_Toc264365762)

[3.1.7 Secure Broadcasting 13](#_Toc264365763)

[3.2 External Interface Requirements 14](#_Toc264365764)

[3.2.1 User Interfaces 14](#_Toc264365765)

[3.2.2 Hardware Interfaces 14](#_Toc264365766)

[3.2.3 Software Interfaces 14](#_Toc264365767)

[3.2.4 Communications Protocols 14](#_Toc264365768)

[3.2.5 Memory Constraints 14](#_Toc264365769)

[3.2.6 Operation 14](#_Toc264365770)

[3.2.7 Product function 14](#_Toc264365771)

[3.2.8 Assumption and Dependency 14](#_Toc264365772)

[3.3 Software Product Features 14](#_Toc264365773)

[3.4 Software System Attributes 14](#_Toc264365774)

[3.4.1 Reliability 14](#_Toc264365775)

[3.4.2 Availability 14](#_Toc264365776)

[3.4.3 Security 14](#_Toc264365777)

[3.4.4 Maintainability 14](#_Toc264365778)

[3.4.5 Portability 14](#_Toc264365779)

[3.4.6 Performance 14](#_Toc264365780)

[3.5 Database Requirements 14](#_Toc264365781)

[3.6 Other Requirements 14](#_Toc264365782)

[4 ADDITIONAL MATERIALS 14](#_Toc264365783)

## INTRODUCTION

* 1. Context

The Crypto SDK exists in the context of the LAST project. The objective of the LAST project is to develop **comprehensive technological solutions** that can serve as **privacy-enhancing** computing platforms in practice, where huge amounts of data need to be processed. (Reference: [Lindell Abstract.pdf](file:///C:\Documents%20and%20Settings\Lenovo\Local%20Settings\Temporary%20Internet%20Files\Content.IE5\Y45VRG6X\Lindell%20Abstract.pdf))

The Crypto SDK exists as a service for high-level multi-party computation protocols as well as a standalone product. High-level protocols will be built upon the low-level primitives in the SDK.

* 1. Product Overview

The Crypto SDK is a wide set of cryptographic tools that is the basis to higher-level cryptographic protocols. However, the tools in the SDK can also be used as stand-alone applications.

The SDK has three basic layers:

1. Low-level crypto functions.
2. Non-interactive mid-level crypto functions.
3. Interactive mid-level crypto functions.

Functions in the first level are independent of other functions.

Functions in the second and third level might depend on functions in previous layers and/or same layer.

* 1. Purpose

#### General

* The main purpose is to provide a reliable, efficient, flexible cryptographic platform for the development of high-level cryptographic protocols.

As the LAST project evolves, new protocols and schemes will be developed by the researchers. The platform should provide a straightforward way to implement these protocols.

* Since the SDK is a broad easy-to-use kit, the secondary goal of comparing between different protocols can be effortlessly achieved.
* We seek to have a big impact on the crypto community. The goal is to have the community use the SDK.

#### Internal

* To make the SDK very flexible so that adding new functions and capabilities to it, (for example, a new type of OT is devised) will not alter the existing components and will naturally integrate with the SDK.
* To provide an API for developers who need cryptographic tools. Even though for the first and second layers there exist already a few tools in the market, none of these provides a full solution. Each may implement a sub-set of each layer but no comprehensive platform has been written yet. For the third layer it seems that more local solutions have been provided following specific needs of the developing teams.
  1. Scope

#### Setup

((Cryptographic schemes are usually based on complexity assumptions, which state that some problem, e.g. factorization, cannot be solved in polynomial time. (Wikipedia). ))

There is a whole set of assumptions or previous knowledge, that crypto protocols use or refer to. For example, PKI, CRS or the Plain Model.

We call this set of assumptions the Setup.

It is within the scope of our product to allow each protocol to be initialized with a different Setup depending on the user’s requirements for a certain usage of the protocol.

#### Security levels

Crypto primitives as well as HL protocols have different levels of security. The same protocol should be able to be implemented using different levels of security depending on the user’s definition. Moreover, we require that the protocol developer need not specify which sub-protocol to use, as long as the sub-protocol “chosen” belongs to the required security level.

#### Input/output

The SDK should provide full flexibility on the length of input arguments and output results for every function. For example, for some function (e.g. RSA) we work today with 1024 bit but in the future we may need to extend it. The implementation may change, but the interface needs to remain the same.

#### Ease of use (shall we mention this here?)

In order to allow a non-programmer to use the SDK and **compose** higher-level protocols a side tool with GUI should be provided. This tool will facilitate the composition of protocols by exposing different options according to specifications of the user.

Another tool we thought of is a GUI application that will show the use of different capabilities of the SDK.

* 1. Reference
  2. Definition And Abbreviation

|  |  |
| --- | --- |
| LAST |  |
| SDK | Software development kit |
| HLP | High level protocols |
| MPC | Multi party computation |
| PKI | Public key infrastructure |
| CRS | Common reference string |
| OT | Oblivious transfer |
|  |  |
|  |  |
|  |  |
|  |  |

## OVERALL DESCRIPTION

### 2.1System environment

The SDK should be platform independent. The Operating Systems can be any version of Windows, Linux, Unix or Mac which supports TCP/IP protocols.

### 2.1Product Functions

The SDK is not an application, but an API to write applications that seek privacy and security. Therefore, it does not have any specific functionality, as you would expect from other software applications.

### 2.2 User Characteristics

There are two main types of users, cryptography researchers (who may not necessarily be expert programmers) and developers.

The SDK is mainly aimed at developers. These will be the ones that will implement higher-level protocols as defined by the researchers. Internal and external developers will be able to extend existing functionality as needed.

At a second stage, we may provide a GUI application that will act as a “code generator” and will allow the researcher to compose high-level protocols as she wishes. This tool will not necessarily produce the most efficient code but it will be good enough to serve as a prototype.

### 2.3 General Constraints

In the future the SDK should become open-source. Therefore, very thorough documentation and high-quality coding are mandatory.

### 2.4 Assumptions and Dependencies

* Communication: Second and third layers of the SDK use Communication—between two parties or more
* Database: what do we need here?
* Secure coding: Easily avoided software defects are a primary cause of commonly exploited software vulnerabilities. We use secure coding principles to guaranty a robust code.
* Code reuse: Wherever there is already a high quality implementation of any of the functions that need to be included, we will wrap that implementation within our SDK.

3 SPECIFIC REQUIREMENTS

### Functional Requirements Definition

### Overview of the system

* **Name**: Overview of the system.



### General SDK

* **Name**: General SDK.

### 

* **Description**: This use-case includes the overall functionality of the system.
* **Actors**:
  + Programmer : Either an internal or an external programmer. Internal programmer is a programmer in the LAST project that develops the SDK. External programmer is a programmer that may extend the functionality of the SDK.
  + HLP: High level protocols that are composed from the SDK components.
* **Preconditions**: At least one concrete class for each published interface is provided (default implementation).
* **Postconditions**: *A list of conditions that must be true after the use case is complete.*
* **Use case interactions**: Heavily uses the send/receive communication use-case.

### Get random string

* **Name:** Get random string
* **Description**: Returns a random string.
* **Preconditions**:
* **Postconditions**: *A list of conditions that must be true after the use case is complete.*
* **Use case interactions**:
* **Flow of events**: A list of events that happen during the execution of the use case. This could also contain alternative paths.
* **Activity diagram**: An activity diagram or diagrams of the flow of events or some part of the flow of events.
* **Secondary scenarios**: If the flow of events contains only a primary scenario, then here secondary scenarios might also be documented.
* **User interface**: A simplified picture of the user interface for the use case. A prototype of the user interface helps the development team see if the design is on the right track.
* **Sequence diagrams**: Sequence diagrams of the different scenarios.
* **View of participating classes**: A diagram of all the classes whose instances work together to implement the use case.
* **Other requirements**: Other requirements might include quality attributes if you do not have a specific document for them.

### Encrypt/Decrypt

### 

* **Name**: Encrypt/Decrypt Use Case



* **Description**: Encrypting and decrypting using symmetric or asymmetric cryptosystems.
* **Preconditions**: To provide related key.
* **Postconditions**: *A list of conditions that must be true after the use case is complete.*
* **Use case interactions**: The Encrypt/Decrypt use case contains within itself Symmetric and Asymmetric operations.
* **Flow of events**: A list of events that happen during the execution of the use case. This could also contain alternative paths.
* **Activity diagram**: An activity diagram or diagrams of the flow of events or some part of the flow of events.
* **Secondary scenarios**: If the flow of events contains only a primary scenario, then here secondary scenarios might also be documented.
* **User interface**: A simplified picture of the user interface for the use case. A prototype of the user interface helps the development team see if the design is on the right track.
* **Sequence diagrams**: Sequence diagrams of the different scenarios.
* **View of participating classes**: A diagram of all the classes whose instances work together to implement the use case.
* **Other requirements**: Other requirements might include quality attributes if you do not have a specific document for them.

### Oblivious Transfer

### 

* **Name**: Oblivious Transfer Use Case



* **Description**: A protocol by which a sender sends some information to the receiver, but remains oblivious as to what is received. Specifically, a sender has n messages, and the receiver has an index i, and the receiver wishes to receive the i-th among the sender's messages, without the sender learning i, while the sender wants to ensure that the receiver receive only one of the n messages.
* **Preconditions**:
* **Postconditions**: The server does not learn the query of the receiver.
* **Use case interactions**: The receiver asks the server for a piece of information. The server returns an answer to that query. The protocol may take more than 2 rounds in so that the information is hidden.
* **Flow of events**: A list of events that happen during the execution of the use case. This could also contain alternative paths.
* **Activity diagram**: An activity diagram or diagrams of the flow of events or some part of the flow of events.
* **Secondary scenarios**: If the flow of events contains only a primary scenario, then here secondary scenarios might also be documented.
* **User interface**: A simplified picture of the user interface for the use case. A prototype of the user interface helps the development team see if the design is on the right track.
* **Sequence diagrams**: Sequence diagrams of the different scenarios.
* **View of participating classes**: A diagram of all the classes whose instances work together to implement the use case.
* **Other requirements**: Other requirements might include quality attributes if you do not have a specific document for them.

### Zero knowledge

### 

* **Name**: Zero knowledge Use Case



* **Description**: zero-knowledge protocol is an interactive method for one party to prove to another that a (usually mathematical) statement is true, without revealing anything other than the veracity of the statement (wikipedia).
* **Preconditions**:
* **Postconditions**: A zero-knowledge proof must satisfy three properties: (wikipedia)
* **Completeness**: if the statement is true, the honest verifier (that is, one following the protocol properly) will be convinced of this fact by an honest prover.
* **Soundness**: if the statement is false, no cheating prover can convince the honest verifier that it is true, except with some small probability.
* **Zero-knowledge**: if the statement is true, no cheating verifier learns anything other than this fact. This is formalized by showing that every cheating verifier has some *simulator* that, given only the statement to be proven (and no access to the prover), can produce a transcript that "looks like" an interaction between the honest prover and the cheating verifier.
* **Use case interactions**: There is an interactive protocol between the prover and the verifier.
* **Flow of events**: A list of events that happen during the execution of the use case. This could also contain alternative paths.
* **Activity diagram**: An activity diagram or diagrams of the flow of events or some part of the flow of events.
* **Secondary scenarios**: If the flow of events contains only a primary scenario, then here secondary scenarios might also be documented.
* **User interface**: A simplified picture of the user interface for the use case. A prototype of the user interface helps the development team see if the design is on the right track.
* **Sequence diagrams**: Sequence diagrams of the different scenarios.
* **View of participating classes**: A diagram of all the classes whose instances work together to implement the use case.
* **Other requirements**: Other requirements might include quality attributes if you do not have a specific document for them.

### Secure Broadcasting

### 

* **Name**: Secure broadcasting Use Case



* **Description**: Broadcasting a message securely. There are several variants of secure broadcast. The main version requires that all honest parties receive the same value and if the sender is honest, honest parties receive broadcasted value.
* **Preconditions**: There may be an upper bound on the number of malicious parties depending on the variant of secure broadcast used.
* **Postconditions**:
* **Use case interactions**: There is an interaction between all the parties. All parties participate in the protocol, meaning they also send messages to each other.
* **Flow of events**: A list of events that happen during the execution of the use case. This could also contain alternative paths.
* **Activity diagram**: An activity diagram or diagrams of the flow of events or some part of the flow of events.
* **Secondary scenarios**: If the flow of events contains only a primary scenario, then here secondary scenarios might also be documented.
* **User interface**: A simplified picture of the user interface for the use case. A prototype of the user interface helps the development team see if the design is on the right track.
* **Sequence diagrams**: Sequence diagrams of the different scenarios.
* **View of participating classes**: A diagram of all the classes whose instances work together to implement the use case.
* **Other requirements**: Other requirements might include quality attributes if you do not have a specific document for them.

### External Interface Requirements

### User Interfaces

### Hardware Interfaces

### Software Interfaces

### Communications Protocols

### Memory Constraints

### Operation

### Product function

### Assumption and Dependency

### Software Product Features

### Software System Attributes

### Reliability

### Availability

### Security

### Maintainability

### Portability

### Performance

### Database Requirements

### Other Requirements

## 4 ADDITIONAL MATERIALS

Symmetric.

symmetric