Initial Thoughts Regarding Homomorphic Encryption Standard API

Kim Laine, Yuriy Polyakov, Gerard W. Ryan

Contents

[Introduction 1](#_Toc525301321)

[Wrapper Implementations 1](#_Toc525301322)

[Configuration Profiles 2](#_Toc525301323)

[Core API Specification 3](#_Toc525301324)

[Purposes of API 3](#_Toc525301325)

[Benefits of configuration profiles 3](#_Toc525301326)

[Benefits of standard API calls 4](#_Toc525301327)

[Examples 4](#_Toc525301328)

[Next Steps 4](#_Toc525301329)

[Compilers 4](#_Toc525301330)

[Serialization strategy 5](#_Toc525301331)

[API specification extensions 5](#_Toc525301332)

[Supporting Multiple Keys 5](#_Toc525301333)

[API git repo 5](#_Toc525301334)

## Introduction

### Wrapper Implementations

The actual implementations conforming to the API standard specification have to be added to each library by the library developers. These implementations can come in the form of wrappers, targeting any language such as C++ or Python.

## Configuration Profiles

The idea of a configuration profile is to provide a human-readable container of both library-independent and library-dependent settings for homomorphic encryption. This allows the rest of the “core” API to be library/scheme agnostic.

A configuration profile is a JSON representation consisting of several groups of settings:

1. Security settings. For most schemes, these are RLWE parameters. These parameters should reference the security standard using simple identifier strings. An example:
   1. Security level: HEStd\_128, HEStd\_192, HEStd\_256, Custom
   2. Secret key distribution: ternary/error/uniform
   3. Ring dimension
   4. Gaussian error distribution parameter: e.g., 3.19
2. Scheme identifier: BGV, BFV, CKKS (HEAAN)
3. Standard scheme-specific parameters: Plaintext modulus (BGV/BFV)
4. Library-dependent settings
   1. Library identifier
   2. Library version number
   3. Configuration settings
      1. Scheme-specific settings
      2. Encoding-specific settings
      3. Optimization options (relinearization profile, memory management, etc.)
      4. Etc.

JSON EXAMPLE:

{

"Security": {

"TableID": "HEStd\_128\_classic",

"SKDistrib": "ternary",

"Dimension": "4096",

"ErrorDistribParm": "3.19"

},

"Scheme": {

"ID": "BFV",

"Parms": {

"PlaintextModulus": "65537",

"Mode": "Asymmetric"

}

},

"Libraries": {

"PALISADE": {

"Version": "1.2.0",

"Configuration": {}

},

"SEAL": {

"Version": "3.0",

"Configuration": {}

}

}

}

Notes:

Another attribute to specify is whether symmetric or asymmetric cryptography is used. Most schemes support both.

Evaluation/Galois keys are merged with public keys. Multiplication without relinearization would be described in *extensions*.

## Core API Specification

The API specifies required classes and methods for performing homomorphic encryption operations. The classes are Ciphertext, Plaintext, and Context. The API specifies conversions and operations between Plaintext and Ciphertext.

The Context encapsulates parameters, keys, and library-specific details. There is only secret key per context to avoid dictionary lookups and storing any key IDs. All public/evaluation keys are assumed to be derived from this secret key. The benefit of this approach is that it hides the details of key representation.

The API specifies serialization and deserialization methods for configuration profiles, keys, ciphertexts, and plaintexts. The serialization format for these objects is library-specific and not described in the API specifications.

Pseudocode for the API is in the file pseudocode.txt.

## Purposes of API

### Benefits of configuration profiles

1. For end users:
   1. Build trust and transparency (confidence in using an HE-enabled solution)
   2. Users can tune some parameters of configuration profiles
2. For application developers:
   1. They can focus on the logic of their application and can use HE as a feature rather than become experts in HE/crypto
   2. Easy to switch between libraries based on requirements; not locked into a specific library or version
3. For library developers:
   1. Ease of maintenance
   2. Reduce amount of help desk support
4. For security auditors and CSO office:
   1. Provides a mechanism to evaluate and enforce security policies (via internal/external certification process)

### Benefits of standard API calls

1. For application developers:
   1. They can focus on the logic of their application and can use HE as a feature rather than become experts in HE/crypto
   2. Easy to switch between libraries based on requirements; not locked into a specific library or version
2. For library developers:
   1. Best practices guidelines and minimum requirements
   2. Ease of maintenance
   3. Reduce amount of help desk support

## Examples

Availability of library-independent examples of the API

1. Build a community of HE users
2. Drive adoption of HE

Several examples using the API with different libraries will be made publicly available, for instance, Python scripts that look the same but call different libraries to perform their task. The examples will be uploaded to the HEStdAPI repository.

## Next Steps

### Compilers

Provide a foundation for automated HE-enabled code generation in the future

### Serialization strategy

We will need to support library interoperability. Motivating example: cloud computing. The data may get encrypted using different libraries, platforms, etc.

We need to define a standard format (like X.509) that serializable objects can be exported to and imported from in various libraries.

Ideally all library/scheme-specific parameters would be encapsulated in the configuration profiles, and the serialization format would simply deal with storing the ciphertext and key polynomials.

### API specification extensions

If any changes to function signatures are needed, an extension document can be created

### Supporting Multiple Keys

A new cryptocontext is needed for each secret key. Multi-party capabilities (such as threshold FHE or multi-key FHE) are not currently supported.

## API git repo

https://github.com/kimlaine/HEStd