Homomorphic Encryption Standard API

Kim Laine, Yuriy Polyakov

# Introduction

<http://HomomorphicEncryption.org>

<https://github.com/kimlaine/HEStdAPI>

## Purposes of API

1. Make HE easier to use
2. Library agnostic code
3. Best practices for HE library design
4. Examples showing how to use API for applications
5. Drive adoption
6. Main code should scheme/library agnostic; only configuration profile contains scheme/library specific information

## Core API Specification

C++ header file containing function declarations (template) and class declarations. Three classes at least: HEContext, HEPlaintext, HECiphertext. HEContext is a wrapper for configuration profile and keys. HEPlaintext and HECiphertext are wrappers for library specific implementations of the concrete classes.

HEContext contains:

* LoadProfile mechanism to load library specific configurations/settings
* Serialization and deserialization of keys, ciphertexts, plaintexts
* Encrypt/decrypt/homomorphic operations

Our goal is not to create new classes for each type of object (e.g. keys), but instead keep them encapsulated by HEContext. The only classes exposed are HEContext, HECiphertext, and HEPlaintext, which represent the actual data.

## API Specification Extensions

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

## Configuration Profiles

The idea of a configuration profile is to be a library specific container of parameters (referring to standard parameters) and configurations so that the rest of the API is library/scheme agnostic.

Recommendation: ideally human-readable and easy to see which parameters are being used, for example JSON. This can include scheme/library ID, and a parameter identifier string describing which parameters to use. The parameter identifier should follow a standard naming convention and either refer to the standard parameters directly, or to custom parameters implemented by the library only with some security level guarantee. The parameter identifier is a required field. The configuration can contain any additional number of library dependent settings.

/\*

{

library\_id: "SEAL v2.3",

library\_descriptor: ...,

param\_id: "custom/hestd128\_cyclotomic\_8192\_SEAL\_1",

library\_dep:

{

plain\_modulus: ...

use\_memory\_pool: true

coeff\_modulus: [ 0x123451, 0x132412341 ]

}

}

\*/

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*.

## Supporting Multiple Keys

We assume that each secret key carries a unique identifier (e.g. a hash). All objects derived from this secret key (public key, evaluation keys, ciphertexts) will carry this identifier.

## 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.

## Examples

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.