Homomorphic Encryption Standard API

Kim Laine, Yuriy Polyakov, Gerard W. Ryan

## Purposes of Standardization (Organizational items)

* We need to add a concise mission statement for the standardization.
* Discuss the official name of the standardization effort: HomomorphicEncryption.org
* Create a github repo for reference implementation examples.
* The website has to be improved and made more accessible. More applications. Add a news section.

## 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

### Reference implementations of the API

Availability of library-independent reference implementations

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

### Next Steps: Compilers

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

## 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*.

## API git repo

https://github.com/kimlaine/HEStd

## 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.

## 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

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.