Kunlun: A Modern Crypto Library

Overview

Download

The source code of Kunlun is available at https://github.com/yuchen1024/Kunlun. A local copy of the Git Repository can be obtained by cloning, using

```
$ git clone https://github.com/yuchen1024/Kunlun.git
```

or

```
$ git clone git@github.com:yuchen1024/Kunlun.git
```

Building Kunlun

Kunlun is build on OpenSSL, detailed instrucion of installing OpenSSL can be found at https://github.com/openssl/openssl. Check if OpenSSL have installed successfully by

```
$ openssl version
OpenSSL 1.1.1f  31 Mar 2020
```

In order to support shared memory parallel programming, you also need to install OpenMP, using

```
$ brew install libomp
```

on Mac OS, or

```
$ sudo apt install libomp-dev
```

on linux.
After the above required libraries are settled, compiling and running test program with

```
$ cd ./Kunlun
$ mkdir build && cd build
$ cmake ..
$ make
$ ./test_xxx
```

Code Description

Routine Algorithms

./utility/routines.hpp implements some routine algorithms

```
inline bool FileExist(const std::string& filename)
```

`FileExist` checks whether the input file exists, returns true if it exists, return false if not.

```
std::string FormatToHexString(std::string byte_str)
```

`FormatToHexString` returns a hexadecimal string with input "byte_str" which is an octet string.

```
bool IsPowerOfTwo(size_t x)
```

`IsPowerOfTwo` checks whether there exists an integer n satisfies x = 2^n for x > 0, returns true if integer n exists, return false if not.

```
std::vector<int64_t> GenRandomIntegerVectorLessThan(size_t LEN, int64_t MAX)
```

`GenRandomIntegerVectorLessThan` returns a random integer vector of size "LEN", and each value is less than "MAX". The vector generated in this way does not require cryptographic security.

```
template <typename ElementType> std::ofstream &operator<<(std::ofstream &fout, const ElementType& element)
template <typename ElementType> std::ifstream &operator>>(std::ifstream &fin, ElementType& element)

template <typename ElementType>
std::ofstream &operator<<(std::ofstream &fout, const std::vector<ElementType>& vec_element)
template <typename ElementType>
std::ifstream &operator>>(std::ifstream &fin, std::vector<ElementType>& vec_element)

template < > std::ofstream &operator<<<std::string>(std::ofstream &fout, const std::string& str)
template < > std::ifstream &operator>><std::string>(std::ifstream &fin, std::string& str)
```

The above template function overload operator `>>` and `<<`, the input/output stream objects ElementType can be any C++ POD type. Note: if the input/output stream object is string, it will call the specialized template function.

AES

./crypto/aes.hpp implements AES using Streaming SIMD Extensions2 (SSE2) instructions. The block showed in this file is a `__m128i` data type in SSE2.

```
struct Key{
    block roundkey[11];
    size_t ROUND_NUM;
};
```

It describes the struct of AES key, roundkey[i] is the key used in each round i, ROUND_NUM is the total times of encryption round or decryption round. Because round key is 128bits, ROUND_NUM equals 10.

```
inline Key GenEncKey(const block &salt)
inline Key GenDecKey(const block &salt)
```

`GenEncKey` and `GenDecKey` return the encryption key and decryption key of AES with a random salt input. To ensure correct decryption, the input salt of `GenDecKey` should be the same as `GenEncKey`.

```
inline void ECBEnc(const Key &key, block* data, size_t BLOCK_LEN)
inline void ECBDec(const Key &key, block* data, size_t BLOCK_LEN)

inline void CBCEnc(const Key &key, block* data, size_t BLOCK_LEN)
inline void CBCDec(const Key &key, block* data, size_t BLOCK_LEN)
```

The above function implements ECB mode and CBC mode of AES. For encryption function, input parameters are AES key, a pointer of plaintext and length of the plaintext. After encryption, the ciphertext will cover plaintext in memory.

Hashing

./crypto/hash.hpp

Pseudo Random Generator

./crypto/prg.hpp

Bloom Filter

./filter/bloom_filter.hpp

Oblivious Transform

Naor-Pinkas OT

./mpc/ot/naor_pinkas_ot.hpp

IKNP OT Extension

./mpc/ot/iknp_ote.hpp

Reverse Private Membership Test

./mpc/rpmt/cwprf_mqrpmt.hpp

Private Set Operation

./mpc/pso/pso_from_mqrpmt.hpp

Testing Examples

Flow of RPMT Demo

Flow of PSO Demo