

Forum:[Hack Forums](#)

Thread: [Encryption, decryption, hash, encode/decode, RSA keys, Signatures, etc..](#)

SovietBastard - August 17, 2024, 9:36 am

so i shared this in the coding lounge section yesterday, but honestly i thought the right section was this one(cryptography), i will drop the original thread content here so anyone can read it, don't kill me :|

Im just sharing my own cryptography utility module/library written in c++ for c++, doxygen comments or comments in general might contain insignificant grammar errors cuz IM NOT english speaker, i really like opinions so RAID can begin.

Implementation of the module is easy to follow and to understand. the source code is too big to fit here so i wont drop it, i will share the gh repo address for full source code, tests, screen-s, PoC, and full documentation as well as an introduction to cryptography to prepare you for the reading of the documentation in case you're not familiar with basic cryptography, if anyone wants it, just let me know and i'll share the address here in the comments.

The library/module is about Symmetric/Asymmetric cryptography, encryption/decryption, hashing, encoding/decoding, RSA key management||generation, block authentication, message signature, signature verification. I implemented most of the available cipher mode of operations(CBC, GCM, EAX, CFB, OFB, CTR) or at least the most secure and up2date, GCM mode of operation for instance is more secure than CFB or OFB but doesn't support algorithms like GOST or SERPENT, GOST was created by Russian Gov and the most secure hash function version atm is "28147-89", and a GOST block cipher... GOST "28147-89" as i know. Each mode comes with a set of algorithms available(the ones i implemented), for example CBC mode comes with +-17 algorithms, GCM with 5+- algorithms, EAX with 15+-, etc...

Available block cipher algorithms (AES, BlowFish, twofish, Cast128, Cast256, Idea, RC2, RC5, RC6, Mars, Serpent, GOST, ARIA, HIGHT, LEA, SEED, SPECK128, SIMON128...), note that GOST requires a 32 bytes kbs and 16 bytes ivbs. Functions do no propagate errors but instead the error(if any) is scoped and saved into the return block structure(op_frame<>, etc... if used) and can have access to it through member "error.error_msg" or "error.has_error" to check if returned with error. I implemented a couple of methods to prevent side channel attacks or other memory related attack vectors, compiler optimization, but i don't promise anything. 10000 IRs by default, this can be set to any value. generates/manages RSA public/private keys, creates messages authentication signatures, verifies signatures, etc...

hashing algorithms (SHA1...SHA512, Tiger, Blake2, MD5, Whirlpool, Ripemd160...). The single translation-unit containing the whole module, is about 2k+ LOC so i split/divided the module in segments(symmetric based division) to divide it into sub-modules.

this is basically for myself when i deal with encryption and related stuff, makes my life much easier, makes encryption easier...

IF THE POLL REACHES THE THRESHOLD(10 votes) i will keep sharing otherwise i will stop, let me know!

the following is the implementation of the module, not the module!

```
Code #include <string>
#include <iostream>
#include "/path/to/ByteCrypt.hpp"
```

```
using namespace Types;
using namespace ByteCryptModule;
```

```

/**
 * im using *auto* instead of explicit types bcz of max characters allowed here
 */
int main(int argc, char *argv[]) {
    ByteCrypt instance;

    const std::string plaintext("some text"), secret_key("secret123");

    // ----- HASH (sha1, sha224, sha256, sha384, sha512, tiger, blake2, md5, ripemd160)
    const auto h1 = instance.hash(plaintext); // hashed block, default hash algo = SHA256
    const auto h2 = instance.hash(plaintext, e_hash_algo_option::SHA1);
    // ... SHA224/256/384/512 ...

    auto h512 = instance.hash(plaintext, e_hash_algo_option::SHA512);
    auto htiger = instance.hash(plaintext, e_hash_algo_option::TIGER);
    auto hblake2 = instance.hash(plaintext, e_hash_algo_option::BLAKE2);
    auto hripemd = instance.hash(plaintext, e_hash_algo_option::RIPEMD160);
    std::cout << "Hashed Result: " << h1.result << '\n'; // `result` contains what it says, `error` returns
error...

    // ----- ENCRYPTION
    // Modes (GCM, CBC, EAX, CFB, OFB, CTR mode of operation), algorithms (AES, GOST,
SERPENT, Blowfish, Cast128, Cast256, RC2, RC5, RC6, SEED, SIMON128, ... Mars).
    auto e1 = instance.gcm_encrypt(plaintext, secret_key); // gcm mode encryption, aes block
cipher(default cipher)
    auto e2 = instance.gcm_encrypt(plaintext, secret_key, e_gcm_algorithm::RC6);
    // ... other GCM mode algorithms ...
    std::cout << "Encrypted GCM-AES: " << e2.result << '\n';

    auto e3 = instance.cbc_encrypt(plaintext, secret_key, e_cbc_algorithm::GOST);
    auto e4 = instance.cbc_encrypt(plaintext, secret_key, e_cbc_algorithm::SERPENT);
    auto e5 = instance.cbc_encrypt(plaintext, secret_key, e_cbc_algorithm::BLOWFISH);
    // ... other CBC mode algorithms ...

    auto e6 = instance.eax_encrypt(plaintext, secret_key);
    auto e7 = instance.eax_encrypt(plaintext, secret_key, e_eax_algorithm::SEED);
    auto e8 = instance.eax_encrypt(plaintext, secret_key, e_eax_algorithm::CAST256);
    // ... other modes of operation ...

    // ----- DECRYPTION
    auto e0a = instance.gcm_decrypt(gcm_aes_encrypted.result, secret_key); // gcm mode
encryption, aes block cipher(default cipher)
    auto e0b = instance.gcm_decrypt(gcm_aes_encrypted.result, secret_key,
e_gcm_algorithm::RC6);
    // ... other GCM mode algorithms ...

    auto e0c = instance.cbc_decrypt(cbc_gost_encrypted.result, secret_key,
e_cbc_algorithm::GOST);
    auto e0d = instance.cbc_decrypt(cbc_serpent_encrypted.result, secret_key,
e_cbc_algorithm::SERPENT);
    auto e0e = instance.cbc_decrypt(cbc_blowfish_encrypted.result, secret_key,
e_cbc_algorithm::BLOWFISH);
    // ... other CBC mode algorithms ...

```

```

    auto d1 = instance.eax_decrypt(eax_aes_encrypted.result, secret_key, e_eax_algorithm::AES);
    auto d2 = instance.eax_decrypt(eax_seed_encrypted.result, secret_key,
e_eax_algorithm::SEED);
    auto d3 = instance.eax_decrypt(eax_cast256_encrypted.result, secret_key,
e_eax_algorithm::CAST256);
    // ... other EAX mode algorithms ...
    // ... other modes of operation CFB, OFB, CTR...

    // ----- Asymmetric Cryptography, RSA DER/PEM keys, Digital Signature, Message
Authentication...
    auto der_def = instance.generate_rsa_key_der_pair(); // DER, default key size = 2048
    auto der512 = instance.generate_rsa_key_der_pair(512);
    // ... 1024 - 2048 - 3072 - 4096 ...

    auto pem = instance.generate_rsa_key_pem_pair(); // PEM, default size = 2048
    // ... other block size

    bool store_key = instance.save_rsa_key("public.pem",
*gen_rsa_PEM_key_pair.result.public_key).result;
    bool store_key = instance.save_rsa_key("private.pem",
*gen_rsa_PEM_key_pair.result.private_key).result;

    auto load_public_key = instance.load_rsa_key("public.pem");
    auto load_private_key = instance.load_rsa_key("private.pem");
    std::string message("message block");
    auto msg_signature = instance.sign_message(message, load_private_key.key);
    bool sig_verify = instance.verify_signature(message, msg_signature.result, load_public_key.key);

    // ----- ENCODING
    auto base64_encoded = instance.base64_encode(plaintext);
    auto base64_decode = instance.base64_decode(base64_encoded);
    auto hex_encoded = instance.hex_encode(plaintext);
    auto random_block = instance.generate_random_bytes();
    return 0;
}

```

dobroy_000 - August 17, 2024, 9:49 am

i saw this on coding lounge as well, yeah i also thought that this should be in this category, but good that you moved it here, more relevant to the context haha, definitely share more :)

SovietBastard - August 17, 2024, 9:58 am

(Aug 17, 2024 09:49 AM)dobroy_000 Wrote: [\[CITE\]](#) i saw this on coding lounge as well, yeah i also thought that this should be in this category, but good that you moved it here, more relevant to the context haha, definitely share more :)

yeah i noticed just couple of minutes ago and moved it here where it belongs

i attached some screenshots showcasing the result of each functionality.

CBC test:

GCM test:

EAX test:

RSA DER

RSA PEM

Encoding, Hashing...