```
#include <iostream>
#include <string>
#include <vector>
#include <chrono>
#include <openssl/rand.h>
#include <openssl/evp.h>
#include <openssl/des.h>
#include <openssl/aes.h>
#include <algorithm>
#include <iomanip>


using namespace std;
using namespace chrono;

enum AlgorithmType{
        DES,
        TDES,
        AES128,
        AES192,
        AES256,

};

struct TestConfig {

        AlgorithmType algorithm;

        size_t dataSize;
        double encryptionTime;
        double decryptionTime;
        double throughput;

};

vector <unsigned char> generateRandomData(size_t size){

        vector<unsigned char> data(size);
        RAND_bytes(data.data(), size);
        return data;


}

vector<unsigned char> generateKey(AlgorithmType algo){

        vector<unsigned char> key;

        switch(algo){
                case DES:
```

```
                        key.resize(8);
                        RAND_bytes(key.data(), 24);
                        break;

                case TDES:
                        key.resize(24);
                        RAND_bytes(key.data(), 24);
                        break;

                case AES128:
                        key.resize(16);
                        RAND_bytes(key.data(), 16);
                        break;

                case AES192:
                        key.resize(24);
                        RAND_bytes(key.data(), 24);
                        break;

                case AES256:
                        key.resize(32);
                        RAND_bytes(key.data(), 32);
                        break;
        }

        return key;
}

TestConfig testDES(size_t dataSize){

        TestConfig config;
        config.algorithm = DES;
        config.dataSize = dataSize;

        vector <unsigned char> plaintext =
generateRandomData(dataSize);
        vector <unsigned char> key = generateKey(DES);


        vector <unsigned char> ciphertext(dataSize + 8);
        vector <unsigned char> decryption(dataSize);

        unsigned char iv[8];
        RAND_bytes(iv, 8);

        DES_key_schedule desKey;
        DES_set_key_unchecked((const_DES_cblock*)key.data(), &desKey);

        auto start = high_resolution_clock::now();
```

```
        size_t processed = 0;
        while(processed + 8 <= dataSize){
                DES_ncbc_encrypt(plaintext.data() + processed,
ciphertext.data() + processed, 8, &desKey, (DES_cblock*)iv,
DES_ENCRYPT);
                processed += 8;
        }

        auto end = high_resolution_clock::now();
        config.encryptionTime = duration<double>(end - start).count();

        RAND_bytes(iv, 8);

        start = high_resolution_clock::now();


        RAND_bytes(iv, 8);

        start = high_resolution_clock::now();

        processed = 0;

        while(processed + 8 <= dataSize){
                DES_ncbc_encrypt(ciphertext.data() + processed,
decryption.data() + processed, 8, &desKey, (DES_cblock*)iv,
DES_DECRYPT);
                processed += 8;

        }

        end = high_resolution_clock::now();
        config.decryptionTime = duration<double>(end - start).count();

        config.throughput = (dataSize / (1024.0 * 1024.0)) /
config.encryptionTime;

        return config;


}

TestConfig test3DES(size_t dataSize){

        TestConfig config;
        config.algorithm = TDES;
        config.dataSize = dataSize;

        vector<unsigned char> plaintext =
generateRandomData(dataSize);
        vector<unsigned char> key = generateKey(TDES);
```

```
        vector<unsigned char> ciphertext(dataSize + 8);
        vector<unsigned char> decrypted(dataSize);

        unsigned char iv[8];
        RAND_bytes(iv, 8);

        DES_key_schedule ks1, ks2, ks3;
        DES_set_key_unchecked((const_DES_cblock*)key.data(), &ks1);
        DES_set_key_unchecked((const_DES_cblock*)key.data() + 8,
&ks2);
        DES_set_key_unchecked((const_DES_cblock*)key.data() + 16,
&ks3);

        auto start = high_resolution_clock::now();

        size_t processed = 0;
        while (processed + 8 <= dataSize){
                DES_ecb3_encrypt((const_DES_cblock*)(plaintext.data()
+ processed), (DES_cblock*)(ciphertext.data() + processed), &ks1,
&ks2, &ks3, DES_ENCRYPT);

                if(processed == 0){

                        for(int i = 0; i < 8; i++){

                                ciphertext[processed + i] ^= iv[i];


                        }

                }else {

                        for (int i = 0; i < 8; i++){

                                ciphertext[processed + i] ^=
ciphertext[processed - 8 + i];

                        }

                }

                processed += 8;

        }

        auto end = high_resolution_clock::now();
        config.encryptionTime = duration<double>(end - start).count();

        start = high_resolution_clock::now();
```

```
        processed = 0;
        while(processed + 8 <= dataSize){

                if(processed == 0){

                        for(int i = 0; i < 8; i++){

                                ciphertext[processed + i] ^= iv[i];

                        }

                }else{

                        for(int i = 0; i < 8; i++){

                                ciphertext[processed + i] ^=
ciphertext[processed - 8 + i];

                        }


                }


                DES_ecb3_encrypt((const_DES_cblock*)(ciphertext.data()
+ processed),(DES_cblock*)(decrypted.data() + processed), &ks1, &ks2,
&ks3, DES_DECRYPT);


                processed += 8;

        }

        end = high_resolution_clock::now();
        config.decryptionTime = duration<double>(end - start).count();

        config.throughput = (dataSize / (1024.0 * 1024.0)) /
config.encryptionTime;

        return config;


}



TestConfig testAES(size_t dataSize, AlgorithmType algo){

        TestConfig config;
```

```
        config.algorithm = algo;
        config.dataSize = dataSize;

        int keySize = 0;
        switch(algo){
                case AES128:
                        keySize = 16;
                        break;

                case AES192:
                        keySize = 24;
                        break;

                case AES256:
                        keySize = 32;
                        break;

                default:
                        keySize = 16;

        }

        vector<unsigned char> plaintext =
generateRandomData(dataSize);
        vector<unsigned char> key(keySize);

        RAND_bytes(key.data(), keySize);

        unsigned char iv[16];
        RAND_bytes(iv, 16);

        AES_KEY aesKey;
        AES_set_encrypt_key(key.data(), keySize * 8, &aesKey);

        size_t paddedSize = ((dataSize + 15) / 16) * 16;
        vector<unsigned char> ciphertext(paddedSize);
        vector<unsigned char> iv_enc;
        iv_enc.assign(iv, iv + 16);

        auto start = high_resolution_clock::now();

        AES_cbc_encrypt(plaintext.data(), ciphertext.data(), dataSize,
&aesKey, iv_enc.data(), AES_ENCRYPT);

        auto end = high_resolution_clock::now();
        config.encryptionTime = duration<double>(end - start).count();

        AES_KEY aesDecKey;
        AES_set_decrypt_key(key.data(), keySize * 8, &aesDecKey);
```

```
        vector<unsigned char> decrypted(paddedSize);
        vector<unsigned char> iv_dec;
        RAND_bytes(iv, 16);
        iv_dec.assign(iv, iv + 16);

        start = high_resolution_clock::now();

        AES_cbc_encrypt(ciphertext.data(), decrypted.data(),
paddedSize, &aesDecKey, iv_enc.data(), AES_DECRYPT);
        end = high_resolution_clock::now();

        config.decryptionTime = duration<double>(end - start).count();

        config.throughput = (dataSize / (1024.0 * 1024.0)) /
config.encryptionTime;

        return config;


}


void printResults(const vector<TestConfig>& results) {
        cout << "\nAlgorithm,Data Size (MB), Encryption Time (s),
Decryption Time (s), Throughput (MB/s) \n";

        for ( const auto& result : results){
                string algoName;
                switch(result.algorithm) {
                        case DES:
                                algoName = "DES";
                                break;

                        case TDES:
                                algoName = "3DES";
                                break;

                        case AES128:
                                algoName = "AES-128";
                                break;

                        case AES192:
                                algoName = "AES-192";
                                break;

                        case AES256:
                                algoName = "AES-256";
                                break;

                }
```

```
                double dataSizeMB = result.dataSize / (1024.0 *
1024.0);

                cout << algoName << "," << dataSizeMB << "," << fixed
<< setprecision(6) << result.encryptionTime << "," <<
result.decryptionTime << "," << result.throughput << "\n";

        }

}


int main(){

        RAND_poll();

        vector<size_t> dataSize = {
                1 * 1024 * 1024,
                10 * 1024 * 1024,
                100 * 1024 * 1024,
                500 * 1024 * 1024,
                1 * 1024 * 1024 * 1024

        };

        vector<TestConfig> results;

        cout << " Test data sizes: 1MB, 10MB, 100MB, 500MB, 1GB \n\n";

        for (size_t size : dataSize) {
                cout << "Testing with " << size/(1024*1024) << " MB
data...\n";

                // Test DES
                cout << "  Testing DES...";
                results.push_back(testDES(size));
                cout << " done\n";

                // Test 3DES
                cout << "  Testing 3DES...";
                results.push_back(test3DES(size));
                cout << " done\n";

                // Test AES-128
                cout << "  Testing AES-128...";
                results.push_back(testAES(size, AES128));
                cout << " done\n";

                // Test AES-192
```

```
                cout << "  Testing AES-192...";
                results.push_back(testAES(size, AES192));
                cout << " done\n";

                // Test AES-256
                cout << "  Testing AES-256...";
                results.push_back(testAES(size, AES256));
                cout << " done\n";
        }
        printResults(results);

        return 0;


}
```