数据安全文档

一、 加密算法介绍：

1.对称加密算法 通过一个公共的密钥，双方进行加密，解密操作。

对称加密算法是应用较早的加密算法，技术成熟。在对称加密算法中，数据发信方将明文（原始数据）和加密密钥（mi yao）一起经过特殊加密算法处理后，

使其变成复杂的加密密文发送出去。收信方收到密文后，若想解读原文，则需要使用加密用过的密钥及相同算法的逆算法对密文进行解密，

才能使其恢复成可读明文。在对称加密算法中，使用的密钥只有一个，发收信双方都使用这个密钥对数据进行加密和解密，这就要求解密方事先必须知道加密密钥。

2. hash 算法 对于一些数据 计算得到一个固定位数的值。（对于数据量大的计算时间大，内存消耗也比较大）

Hash，一般翻译做“散列”，也有直接音译为“哈希”的，就是把任意长度的输入（又叫做预映射pre-image）通过散列算法变换成固定长度的输出，该输出就是散列值。这种转换是一种压缩映射，也就是，散列值的空间通常远小于输入的空间，不同的输入可能会散列成相同的输出，所以不可能从散列值来确定唯一的输入值。简单的说就是一种将任意长度的消息压缩到某一固定长度的消息摘要的函数

3. 非对称加密算法：

非对称加密算法需要两个密钥：公开密钥（publickey）和私有密钥（privatekey）。公开密钥与私有密钥是一对，如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密；如果用私有密钥对数据进行加密，那么只有用对应的公开密钥才能解密。

因为加密和解密使用的是两个不同的密钥，所以这种算法叫作非对称加密算法。 非对称加密算法实现机密信息交换的基本过程是：甲方生成一对密钥并将其中的一把作为公用密钥向其它方公开；得到该公用密钥的乙方使用该密钥对机密信息进行加密后再发送给甲方；甲方再用自己保存的另一把专用密钥对加密后的信息进行解密。

另一方面，甲方可以使用乙方的公钥对机密信息进行签名后再发送给乙方；甲方再用自己的私匙对乙方发送回来的数据进行验签。

甲方只能用其专用密钥解密由其公用密钥加密后的任何信息。 非对称加密算法的保密性比较好，它消除了最终用户交换密钥的需要。

非对称密码体制的特点：算法强度复杂、安全性依赖于算法与密钥但是由于其算法复杂，而使得加密解密速度没有对称加密解密的速度快。对称密码体制中只有一种密钥，并且是非公开的，

如果要解密就得让对方知道密钥。所以保证其安全性就是保证密钥的安全，而非对称密钥体制有两种密钥，其中一个是公开的，这样就可以不需要像对称密码那样传输对方的密钥了。这样安全性就大了很多

4. 数字签名

数字签名（又称公钥数字签名、电子签章）是一种类似写在纸上的普通的物理签名，但是使用了公钥加密领域的技术实现，用于鉴别数字信息的方法。一套数字签名通常定义两种互补的运算，一个用于签名，另一个用于验证。

数字签名，就是只有信息的发送者才能产生的别人无法伪造的一段数字串，这段数字串同时也是对信息的发送者发送信息真实性的一个有效证明。

数字签名是非对称密钥加密技术与数字摘要技术的应用。

5. 密码协商 (主要是openssl)

密钥协商：两个或多个实体协商，共同建立会话密钥，任何一个参与者均对结果产生影响，不需要任何可信的第三方（TTP）。

密钥协商协议：会话密钥由每个协议参与者分别产生的参数通过一定的计算得出。常见的密钥协商协议，如IKE。

密钥协商协议的生成方式：可分为证书型和无证书型。证书型是指在会话密钥的产生过程中，由一个可信的证书中心（CA）给参与密钥协商的各方各分发一个证书，此证书中含有此方的公钥，ID及其他信息。证书型密钥协商协议的优点是提供认证，目前PKI（公钥密码体制）广泛部署，比较成熟，

应用面广，且由PKG管理公私钥对有利于统一管理，缺点是计算代价大，需要一个可信的CA，同时证书还需要维护。无证书型是指各方在进行会话密钥的协商过程中不需要证书的参与，这是目前密钥协商协议的主流种类，

优点是不需要CA的参与，减少了计算量，尤其是在低耗环境下应用的更多，同时安全性也不比证书型弱。几乎没有明显的缺点，只是设计一个安全的更加低耗的无证书密钥协商方案不是很容易。

二、加密算法选择：

根据华为安全文档相关的介绍和它提供的相关具体算法的安全性。

公司认定的不安全密码算法包括：

1）分组密码算法：Blowfish, DES, DESX, RC2,RC4,Skipjack, 2TDEA, TEA

2）流密码算法：SEAL, CYLINK\_MEK, RC4 (<128 bits)

3）哈希算法：SHA0, MD2, MD4, MD5, RIPEMD, RIPEMD-128；SHA-1禁止用于数字签名的场景

4）数字签名算法：RSA (<1024 bits)、DSA (<1024 bits)、 ECDSA(≤160bits) 5）密钥交换算法：DH (< 1024 bits)、ECDH(≤223bits)

公司推荐使用的密码算法包括：

1）分组密码算法：AES128

2）流密码算法： AES128（OFB或CTR模式）

3）哈希算法：SHA256

4）数字签名算法：RSA(2048)、DSA(2048) 、ECDSA(256)

5）密钥交换算法：DH(2048)

1. 相关数据加密的选择(加密算法库 ：cryptopp ++ , openssl)

主要针对 应用层的数据加密 和 加密算法的选择： sha256 + AES + base64, sha256 + RSA(2048) + base 64.

1. Web 登陆到终端设备：
   1. 主要是http + json 数据通信方式 或者 http + xml or html.加密可以采用整体数据 和 分段数据加密的方式。见 下表格。

端口管理：

1. 对于所有的端口，都要进行鉴权。

|  |  |
| --- | --- |
| 整体数据加密方式 | 对于http打包的数据整体可以采用 sha256 计算一个key值，在用AES 算法加密json等数据，最后采用 base64加密一次。 |
| 分段数据加密 | 可以针对json字段进行加密,在打包成json字段发送。例如：用户名，密码字段 “params”:{“currentusername”:”admin”, “username”:”admin”}  可以先用 sha256 + AES + base64 加密 admin，在打包成json字符串。“params”:{“currentusername”:”VGhpcyBpcyBhIGV4YW1wbGUA”, “username”:”VGhpcyBpcyBhIGV4YW1wbGUA”} |

sha256 + AES + base64 加密具体流程：

Client ------------ 申请一个key ---------------------> Server

Client -----------AES 加密数据，进行base64编码----------------------> Server

Server 端进行base64 解码，在进行AES 解码。

最后附上cryptopp ++ 接口封装代码：

Crypto.h:

#include <iostream>

#include <string>

namespace Crypto {

typedef enum{

AES\_ECB = 0,

AES\_CBC,

AES\_CFB,

AES\_OFB,

AES\_CTR

}ASE\_MODE;

//base Function

int cryptopp\_generate\_key(std::string str,std::string & key,int len = 16);

int cryptopp\_save\_key(std::string path,std::string key);

int cryptopp\_load\_key(std::string path,std::string & key);

// Hash Function

int cryptopp\_md5(std::string data,int size,std::string & md5);

int cryptopp\_sha256(std::string data,std::string key,std::string & md5);

// ASE Function

int cryptopp\_ase\_encode(std::string data,ASE\_MODE mode,std::string key,std::string iv,std::string & out);

int cryptopp\_ase\_decode(std::string data,ASE\_MODE mode,std::string key,std::string iv,std::string & out);

// \\ RSA fuction

/\*\*

\* param seed : in, Random seed.

\* keylen : in,key`s len.

\* prikey : out,RSA private key,The size is keylen.

\* pubkey : out,RSA public key,The size is keylen.

\* Function : generate a pair of key for RSA.

\*\*/

int cryptopp\_generate\_RSAkey(std::string & prikey,std::string& pubkey,std::string seed,int keylen);

int cryptopp\_generate\_RSAkey1(std::string& prikey,std::string& pukey,int keylen);

int cryptopp\_rsa\_signer(std::string prikey,std::string seed,std::string & signer\_data,std::string data);

bool cryptopp\_rsa\_verify(std::string pubkey,std::string & signature,std::string data);

int cryptopp\_rsa\_encode(std::string pubkey,const std::string message,std::string & chiper);

int cryptopp\_rsa\_decode(std::string prikey,const std::string & ciphertext,std::string &result);

// base64 function

int cryptopp\_base64\_encoder(std::string in,std::string & out);

int cryptopp\_base64\_decoder(std::string in,std::string & out);

}

Cryptopp.cpp:

#include <iostream>

#include <string>

#include "osrng.h"

#include "cryptlib.h"

#include "hex.h"

#include "filters.h"

#include "aes.h"

#include "modes.h"

#include "hmac.h"

#include "sha.h"

#include "secblock.h"

#define CRYPTOPP\_ENABLE\_NAMESPACE\_WEAK 1

#include "md5.h"

#include "rsa.h"

#include "randpool.h"

#include "files.h"

#include "pssr.h"

#include "base64.h"

#include "crypto.h"

using namespace std;

using namespace CryptoPP;

// fuction : generate key by input str, The len is generated`s key len, must be more than 16. ( len >= 16)

// str : input

// len : input key len

// key : output

int Crypto::cryptopp\_generate\_key(string str,string & key,int len)

{

//string encode;

if( len < 16 || str.size() < len)

{

std::cerr << " str size" << str.size() << "less " << len <<std::endl;

return -1;

}

StringSource((const unsigned char \*)str.c\_str(),len,true,new HexEncoder(new StringSink(key)));

return key.size();

}

// Hash

int Crypto::cryptopp\_md5(string data,int size,string & md5)

{

CryptoPP::Weak::MD5 md;

string encode;

StringSource(data.c\_str(),true,new HashFilter(md,new StringSink(encode)));

StringSource(encode.c\_str(),true,new HexEncoder(new StringSink(md5)));

//cout << "MD5 : " << md5.c\_str() << endl;

return md5.size();

}

int Crypto::cryptopp\_sha256(string data,string key,string & sha256)

{

string key1;

string encode;

if(key.size() < 16)

{

std::cerr << " key lenght is error \n";

return -1;

}

key1 = key.substr(0,16);

//cout << "key str : " << key1.c\_str()<<" key size : " << key1.size()<<endl;

SecByteBlock key2((const unsigned char\*)key1.c\_str(),key1.size());

StringSource(key2,key2.size(),true,new HexEncoder(new StringSink(encode)));

//cout << "sha key : " << encode.c\_str() <<" Len :" << encode.size()<< endl;

HMAC< SHA256 > hmac(key2, key2.size());

StringSource(data, true, new HashFilter(hmac,new StringSink(encode)) /\*HashFilter \*/);

StringSource(encode.c\_str(),true,new HexEncoder(new StringSink(sha256)));

return 0;

}

/\*\*

\*

\* AES : ECB CBC CFB OFB CRT

\*

\*\*/

int Crypto::cryptopp\_ase\_encode(string data,ASE\_MODE mode,string key,string iv,string & out)

{

// key >= 16 位

// iv 16 位

// ECB 模式下不需要初始化iv

string encode;

//byte \_\_iv[AES::BLOCKSIZE];

//byte \_\_key[AES::DEFAULT\_keyLength];

string key1;

string iv1;

if(key.size() < AES::DEFAULT\_KEYLENGTH)

{

std::cerr << "key lenght less than " << AES::DEFAULT\_KEYLENGTH << std::endl;

return -1;

}

if(iv.size() < AES::BLOCKSIZE && mode != AES\_ECB)

{

std::cerr << "Iv key len less then " << AES::BLOCKSIZE << std::endl;

return -1;

}

key1 = key.substr(0,AES::DEFAULT\_KEYLENGTH);

iv1 = iv.substr(0,AES::BLOCKSIZE);

//std::cout <<" key : " << key1.c\_str() << " len : " << key1.size() << std::endl;

//std::cout <<" iv key : " << iv1.c\_str() << " iv len : " << iv1.size() << std::endl;

SecByteBlock \_\_key((const unsigned char\*)key1.c\_str(),key1.size()),\_\_iv((const unsigned char\*)iv1.c\_str(),iv1.size());

switch (mode)

{

case AES\_ECB:

{

CryptoPP::ECB\_Mode<AES>::Encryption e;

e.SetKey(\_\_key,\_\_key.size());

StringSource(data, true,

new StreamTransformationFilter(e,

new StringSink(encode)

) // StreamTransformationFilter

); // StringSource

#if 1

StringSource(encode, true,

new HexEncoder(

new StringSink(out)

) // HexEncoder

);

#else

out = encode;

#endif

break;

}

case AES\_CBC:

{

CryptoPP::CBC\_Mode<AES>::Encryption e;

e.SetKeyWithIV(\_\_key,\_\_key.size(),\_\_iv,\_\_iv.size());

StringSource(data, true,

new StreamTransformationFilter(e,

new StringSink(encode)

) // StreamTransformationFilter

); // StringSource

StringSource(encode, true,

new HexEncoder(

new StringSink(out)

) // HexEncoder

);

break;

}

case AES\_CFB:

{

CryptoPP::CFB\_Mode<AES>::Encryption e;

e.SetKeyWithIV(\_\_key,\_\_key.size(),\_\_iv,\_\_iv.size());

StringSource(data, true,

new StreamTransformationFilter(e,

new StringSink(encode)

) // StreamTransformationFilter

); // StringSource

StringSource(encode, true,

new HexEncoder(

new StringSink(out)

) // HexEncoder

);

break;

}

case AES\_OFB:

{

CryptoPP::OFB\_Mode<AES>::Encryption e;

e.SetKeyWithIV(\_\_key,\_\_key.size(),\_\_iv,\_\_iv.size());

StringSource(data, true,

new StreamTransformationFilter(e,

new StringSink(encode)

) // StreamTransformationFilter

); // StringSource

StringSource(encode, true,

new HexEncoder(

new StringSink(out)

) // HexEncoder

);

break;

}

case AES\_CTR:

{

CryptoPP::CTR\_Mode<AES>::Encryption e;

e.SetKeyWithIV(\_\_key,\_\_key.size(),\_\_iv,\_\_iv.size());

StringSource(data, true,

new StreamTransformationFilter(e,

new StringSink(encode)

) // StreamTransformationFilter

); // StringSource

StringSource(encode, true,

new HexEncoder(

new StringSink(out)

) // HexEncoder

);

break;

}

default:

std::cerr << "This mode not impl \n";

break;

}

return out.size();

}

int Crypto::cryptopp\_ase\_decode(string data1,ASE\_MODE mode,string key,string iv,string & out)

{

// key >= 16 位

// iv 16 位

// ECB 模式下不需要初始化iv

//string encode;

string data;

//byte \_\_iv[AES::BLOCKSIZE];

//byte \_\_key[AES::DEFAULT\_KEYLENGTH];

string key1;

string iv1;

if(key.size() < AES::DEFAULT\_KEYLENGTH)

{

std::cerr << "key lenght less than " << AES::DEFAULT\_KEYLENGTH << std::endl;

return -1;

}

if( mode != AES\_ECB && iv.size() < AES::BLOCKSIZE)

{

std::cerr << "Iv key len less then " << AES::BLOCKSIZE << std::endl;

return -1;

}

key1 = key.substr(0,AES::DEFAULT\_KEYLENGTH);

iv1 = iv.substr(0,AES::BLOCKSIZE);

//std::cout <<" key : " << key1.c\_str() << " len : " << key1.size() << std::endl;

//std::cout <<" iv key : " << iv1.c\_str() << " iv len : " << iv1.size() << std::endl;

SecByteBlock \_\_key((const unsigned char\*)key1.c\_str(),key1.size()),\_\_iv((const unsigned char\*)iv1.c\_str(),iv1.size());

//Decoder into byte

StringSource(data1, true,new HexDecoder(new StringSink(data)) /\*HexDecoder\*/);

switch (mode)

{

case AES\_ECB:

{

CryptoPP::ECB\_Mode<AES>::Decryption e;

e.SetKey(\_\_key,\_\_key.size());

StringSource(data, true,

new StreamTransformationFilter(e,

new StringSink(out)

) // StreamTransformationFilter

); // StringSource

break;

}

case AES\_CBC:

{

CryptoPP::CBC\_Mode<AES>::Decryption e;

e.SetKeyWithIV(\_\_key,\_\_key.size(),\_\_iv,\_\_iv.size());

StringSource(data, true,

new StreamTransformationFilter(e,

new StringSink(out)

) // StreamTransformationFilter

); // StringSource

break;

}

case AES\_CFB:

{

CryptoPP::CFB\_Mode<AES>::Decryption e;

e.SetKeyWithIV(\_\_key,\_\_key.size(),\_\_iv,\_\_iv.size());

StringSource(data, true,

new StreamTransformationFilter(e,

new StringSink(out)

) // StreamTransformationFilter

); // StringSource

break;

}

case AES\_OFB:

{

CryptoPP::OFB\_Mode<AES>::Decryption e;

e.SetKeyWithIV(\_\_key,\_\_key.size(),\_\_iv,\_\_iv.size());

StringSource(data, true,

new StreamTransformationFilter(e,

new StringSink(out)

) // StreamTransformationFilter

); // StringSource

break;

}

case AES\_CTR:

{

CryptoPP::CTR\_Mode<AES>::Decryption e;

e.SetKeyWithIV(\_\_key,\_\_key.size(),\_\_iv,\_\_iv.size());

StringSource(data, true,

new StreamTransformationFilter(e,

new StringSink(out)

) // StreamTransformationFilter

); // StringSource

break;

}

default:

std::cerr << "This mode not impl \n";

break;

}

return out.size();

}

static RandomPool & GlobalRNG()

{

static RandomPool randomPool;

return randomPool;

}

// RSA 2048

int Crypto::cryptopp\_generate\_RSAkey(string & prikey,string& pubkey,string seed,int keylen)

{

RandomPool randPool;

randPool.IncorporateEntropy((byte \*)seed.c\_str(),seed.size());

RSAES\_OAEP\_SHA\_Decryptor priv(randPool, keylen);

HexEncoder privFile(new StringSink(prikey));

priv.AccessMaterial().Save(privFile);

privFile.MessageEnd();

RSAES\_OAEP\_SHA\_Encryptor pub(priv);

HexEncoder pubFile(new StringSink(pubkey));

pub.AccessMaterial().Save(pubFile);

pubFile.MessageEnd();

return 0;

}

int Crypto::cryptopp\_generate\_RSAkey1(string& prikey,string& pukey,int keylen)

{

string prikey\_\_,pukey\_\_;

AutoSeededRandomPool rng;

InvertibleRSAFunction parameters;

parameters.GenerateRandomWithKeySize(rng, keylen);

//生成公钥和私钥

RSA::PrivateKey privateKey( parameters );//私钥用于加密(n,d)

RSA::PublicKey publicKey( parameters ); //公钥用于解密(n,e)

privateKey.Save(StringSink(prikey\_\_).Ref());

publicKey.Save(StringSink(pukey\_\_).Ref());

//std::cout << "privateKey : " << prikey\_\_ << std::endl;

//std::cout << "publickey : " << pukey\_\_ << std::endl;

// HexEncode

StringSource((const unsigned char \*)prikey\_\_.c\_str(),prikey\_\_.size(),true,new HexEncoder(new StringSink(prikey)));

StringSource((const unsigned char \*)pukey\_\_.c\_str(),pukey\_\_.size(),true,new HexEncoder(new StringSink(pukey)));

return 0;

}

int Crypto::cryptopp\_rsa\_encode(string pubkey,const string message,string & chiper)

{

// HexDecode

StringSource s((const byte \*)pubkey.c\_str(),pubkey.size(),true,new HexDecoder);

RSAES\_OAEP\_SHA\_Encryptor pub(s);

//RandomPool randPool;

//randPool.IncorporateEntropy((byte \*)seed.c\_str(),seed.size());

StringSource(message, true, new PK\_EncryptorFilter(GlobalRNG(), pub, new HexEncoder(new StringSink(chiper))));

return chiper.size();

}

int Crypto::cryptopp\_rsa\_decode(string prikey,const string & ciphertext,string &result)

{

// HexDecode

StringSource s((const byte \*)prikey.c\_str(),prikey.size(),true,new HexDecoder);

RSAES\_OAEP\_SHA\_Decryptor priv(s);

StringSource(ciphertext, true, new HexDecoder(new PK\_DecryptorFilter(GlobalRNG(), priv, new StringSink(result))));

return result.size();

}

/\*\*

\* RSA 签名

\*\*/

int Crypto::cryptopp\_rsa\_signer(string prikey,string seed,string & signer\_data,string data)

{

HexDecoder decoder;

RSA::PrivateKey prikey\_\_;

decoder.Put((const byte \*)prikey.c\_str(),prikey.size());

decoder.MessageEnd();

prikey\_\_.Load(decoder);

RSASS<PSS, SHA1>::Signer signer(prikey\_\_);

// signer.AccessKey().Load(decoder);

// Create signature space

size\_t length = signer.MaxSignatureLength();

SecByteBlock signature(length);

// Sign message

length = signer.SignMessage(GlobalRNG(),(const byte\*) data.c\_str(),data.length(),signature);

// Resize now we know the true size of the signature

signature.resize(length);

//signer\_data = \*signature;

StringSource(signature,signature.size(),true,new HexEncoder(new StringSink(signer\_data)));

return signer\_data.size();

}

/\*\*

\* RSA 验证签名

\*\*/

bool Crypto::cryptopp\_rsa\_verify(string pubkey,string & signa,string data)

{

RSA::PublicKey pubkey\_\_;

HexDecoder decoder;

decoder.Put((const byte \*)pubkey.c\_str(),pubkey.size());

decoder.MessageEnd();

pubkey\_\_.Load(decoder);

RSASS<PSS, SHA1>::Verifier verifier(pubkey\_\_);

std::string signare;

StringSource(signa,true,new HexDecoder(new StringSink(signare)));

SecByteBlock signature((const byte \*)signare.c\_str(),signare.size());

bool result = verifier.VerifyMessage((const byte\*)data.c\_str(),data.length(), signature, signature.size());

return result;

}

// save a key into file

int Crypto::cryptopp\_save\_key(string path,string key)

{

StringSource(key.c\_str(),true,new Base64Encoder(new FileSink(path.c\_str())));

return 0;

}

// Load a key from file

int Crypto::cryptopp\_load\_key(string path,string & key)

{

FileSource(path.c\_str(),true,new Base64Decoder(new StringSink(key)));

return 0;

}

int Crypto::cryptopp\_base64\_encoder(string in,string & out)

{

StringSource(in.c\_str(),true,new Base64Encoder(new StringSink(out)));

return 0;

}

int Crypto::cryptopp\_base64\_decoder(string in,string & out)

{

StringSource(in.c\_str(),true,new Base64Decoder(new StringSink(out)));

return 0;

}