185023528 关注

原创 8 粉丝 3 喜欢 2 评论 0

等级： 博客 2 访问： 1万+

积分： 282 排名： 30万+

转 ios使用rsa加解密

2016年05月10日 11:01:41 185023528 阅读数：1708

在iOS中使用RSA加密解密，需要用到.der和.p12后缀格式的文件，其中.der格式的文件存放的是公钥（Public
式的文件存放的是私钥（Private key）用于解密. 首先需要先生成这些文件，然后再将文件导入工程使用，不多

一、使用openssl生成所需秘钥文件

生成环境是在mac系统下，使用openssl进行生成，首先打开终端，按下面这些步骤依次来做：

1. 生成模长为1024bit的私钥文件private_key.pem

```
openssl genrsa -out private_key.pem 1024
```

2. 生成证书请求文件rsaCertReq.csr

```
openssl req -new -key private_key.pem -out rsaCerReq.csr
```

注意：这一步会提示输入国家、省份、mail等信息，可以根据实际情况填写，或者全部不用填写，直接全部敲

3. 生成证书rsaCert.crt，并设置有效时间为1年

```
openssl x509 -req -days 3650 -in rsaCerReq.csr -signkey private_key.pem -out rsaCert.crt
```

4. 生成供iOS使用的公钥文件public_key.der

```
openssl x509 -outform der -in rsaCert.crt -out public_key.der
```

5. 生成供iOS使用的私钥文件private_key.p12

```
openssl pkcs12 -export -out private_key.p12 -inkey private_key.pem -in rsaCert.crt
```

注意：这一步会提示给私钥文件设置密码，直接输入想要设置密码即可，然后敲回车，然后再验证刚才设置
然后敲回车，完毕！
在解密时，private_key.p12文件需要和这里设置的密码配合使用，因此需要牢记此密码.

6. 生成供Java使用的公钥rsa_public_key.pem

```
openssl rsa -in private_key.pem -out rsa_public_key.pem -pubout
```

7. 生成供Java使用的私钥pkcs8_private_key.pem

```
openssl pkcs8 -topk8 -in private_key.pem -out pkcs8_private_key.pem -nocrypt
```

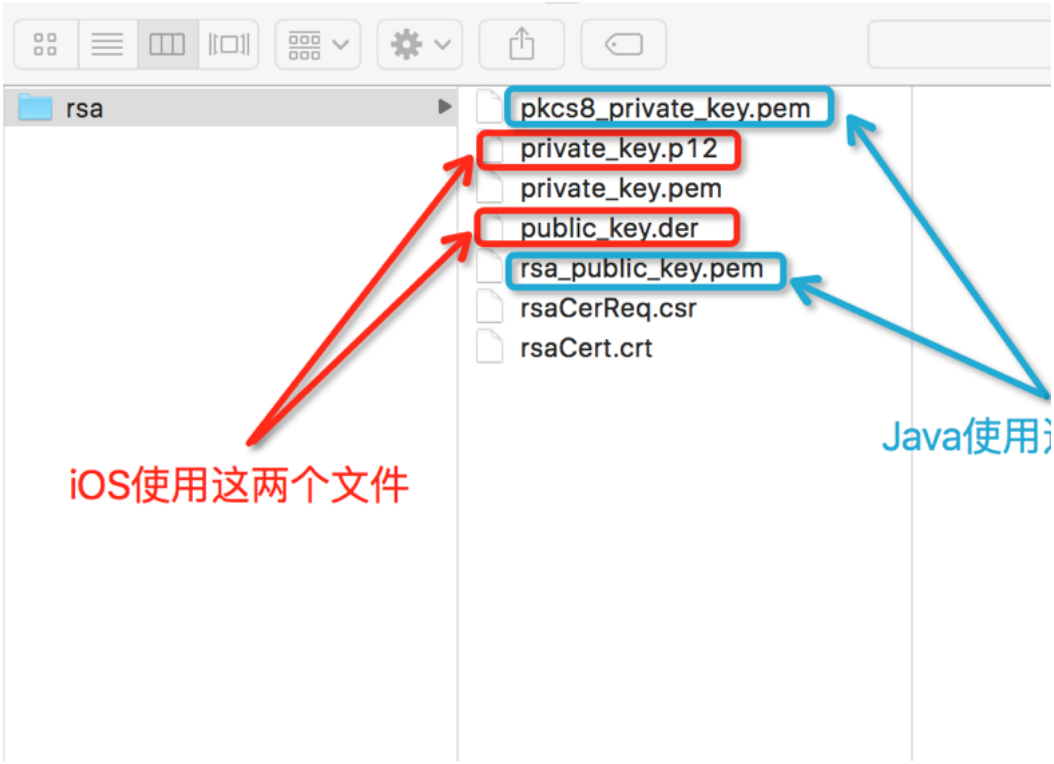
全部执行成功后，会生成如下文件，其中public_key.der和private_key.p12就是iOS需要用到的文件，如下图：

版本的库需要稍作调整
阅读量：3871

关于串口发送16进制编码及解码问题
阅读量：2191

ios使用rsa加解密
阅读量：1706

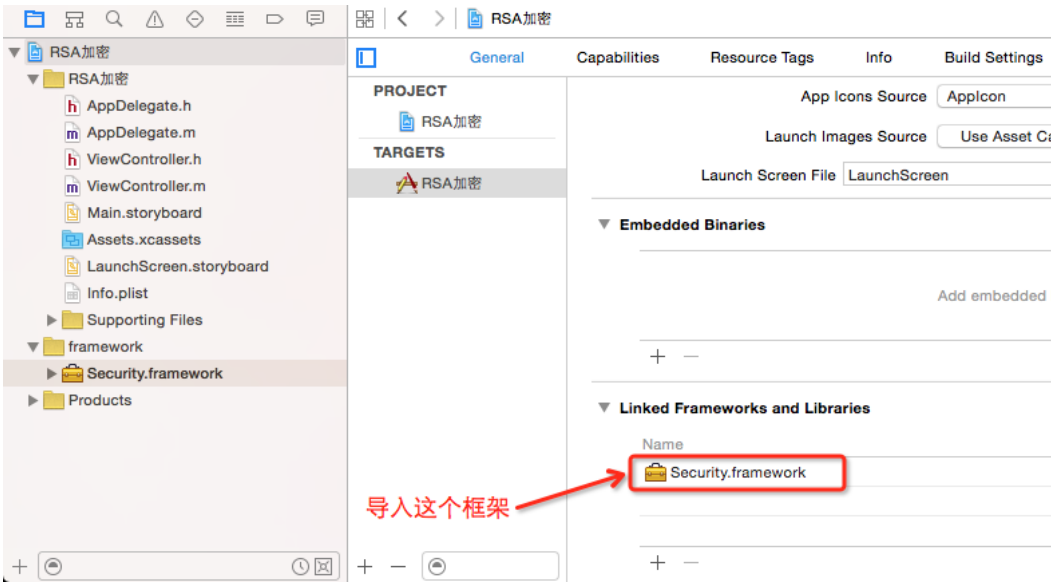
VM12安装centos7安装VMare-tool后，hgfs下没有共享文件夹
阅读量：1575



生成的文件

二、将文件导入工程使用

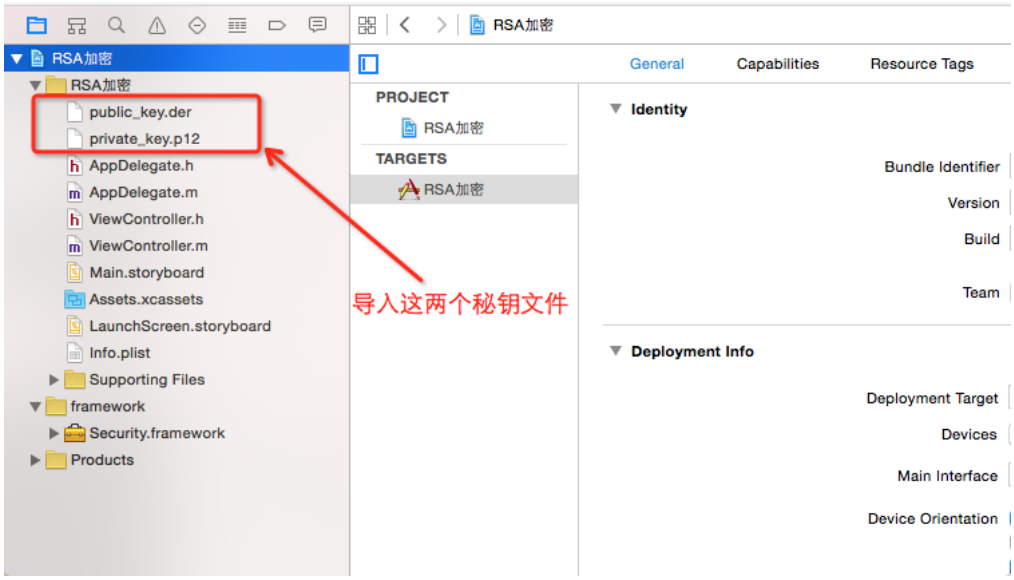
1.新建工程, 并导入Security.framework框架, 如下图:



新建工程并添加框架

2.导入秘钥文件

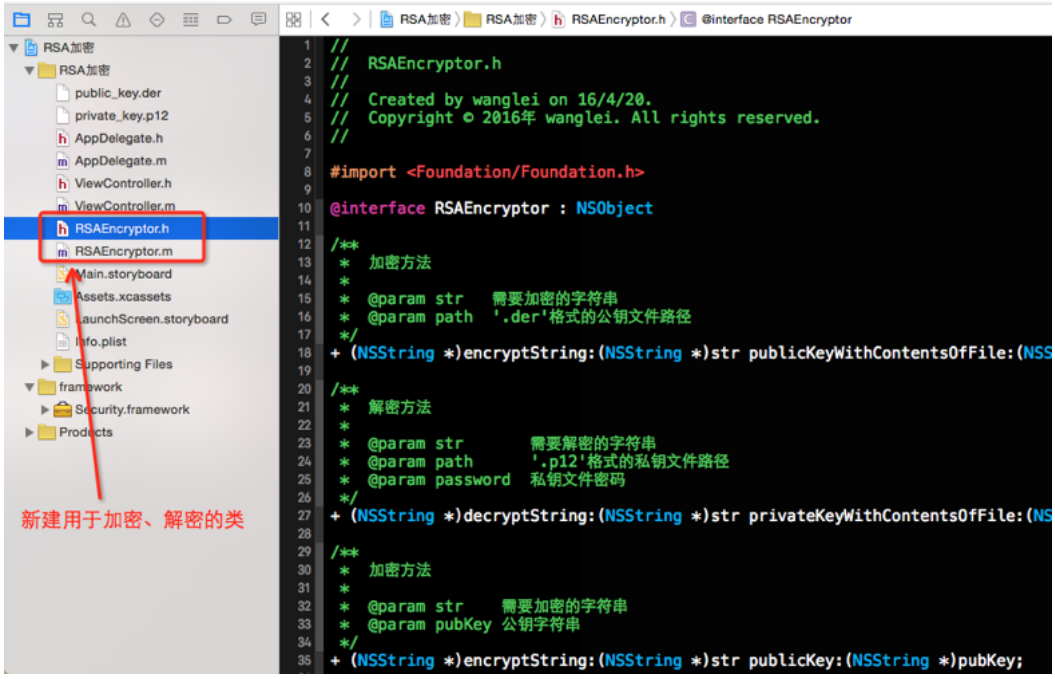
导入.der和.p12格式的秘钥文件, 如下图:



导入秘钥文件

3.新建用于加密、解密的类RSAEncryptor, 并实现相关方法

新建RSAEncryptor类, 如下图:



新建用于加密解密的类

下面开始上代码, 可以直接复制过去用:

RSAEncryptor.h代码如下:

```
#import <Foundation/Foundation.h>

@interface RSAEncryptor : NSObject

/**
 * 加密方法
 *
 * @param str 需要加密的字符串
 * @param path '.der'格式的公钥文件路径
 */
+ (NSString *)encryptString:(NSString *)str publicKeyWithContentsOfFile:(NSString *)path;

/**
```

```
 *  解密方法15 *
 *  @param str       需要解密的字符串
 *  @param path      '.p12'格式的私钥文件路径
 *  @param password  私钥文件密码
 */
+ (NSString *)decryptString:(NSString *)str privateKeyWithContentsOfFile:(NSString *)pa

/**
 *  加密方法
 *
 *  @param str    需要加密的字符串
 *  @param pubKey 公钥字符串
 */
+ (NSString *)encryptString:(NSString *)str publicKey:(NSString *)pubKey;

/**
 *  解密方法
 *
 *  @param str     需要解密的字符串
 *  @param privKey 私钥字符串
 */
+ (NSString *)decryptString:(NSString *)str privateKey:(NSString *)privKey;

@end
```

RSAEncryptor.m代码如下:

```
#import "RSAEncryptor.h"
#import <Security/Security.h>

@implementation RSAEncryptor

static NSString *base64_encode_data(NSData *data){
    data = [data base64EncodedDataWithOptions:0];
    NSString *ret = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
    return ret;
}

static NSData *base64_decode(NSString *str){
    NSData *data = [[NSData alloc] initWithBase64EncodedString:str options:NSDataBase64
    return data;
}

#pragma mark - 使用'.der'公钥文件加密

//加密
+ (NSString *)encryptString:(NSString *)str publicKeyWithContentsOfFile:(NSString *)pat
    if (!str || !path)  return nil;
    return [self encryptString:str publicKeyRef:[self getPublicKeyRefWithContentsOfFile
}

//获取公钥
+ (SecKeyRef)getPublicKeyRefWithContentsOfFile:(NSString *)filePath{
    NSData *certData = [NSData dataWithContentsOfFile:filePath];
    if (!certData) {
        return nil;
    }
    SecCertificateRef cert = SecCertificateCreateWithData(NULL, (CFDataRef)certData);
    SecKeyRef key = NULL;
    SecTrustRef trust = NULL;
    SecPolicyRef policy = NULL;
    if (cert != NULL) {
        policy = SecPolicyCreateBasicX509();
        if (policy) {
            if (SecTrustCreateWithCertificates((CFTypeRef)cert, policy, &trust) == noEr
```

```
                SecTrustResultType result;
                if (SecTrustEvaluate(tru
                    key = SecTrustCopyPublicKey(trust);
                }
            }
        }
    }
    if (policy) CFRelease(policy);
    if (trust) CFRelease(trust);
    if (cert) CFRelease(cert);
    return key;
}

+ (NSString *)encryptString:(NSString *)str publicKeyRef:(SecKeyRef)publicKeyRef{
    if(![str dataUsingEncoding:NSUTF8StringEncoding]){
        return nil;
    }
    if(!publicKeyRef){
        return nil;
    }
    NSData *data = [self encryptData:[str dataUsingEncoding:NSUTF8StringEncoding] withK
    NSString *ret = base64_encode_data(data);
    return ret;
}

#pragma mark - 使用'.12'私钥文件解密

//解密
+ (NSString *)decryptString:(NSString *)str privateKeyWithContentsOfFile:(NSString *)pa
    if (!str || !path) return nil;
    if (!password) password = @"";
    return [self decryptString:str privateKeyRef:[self getPrivateKeyRefWithContentsOfFi
}

//获取私钥
+ (SecKeyRef)getPrivateKeyRefWithContentsOfFile:(NSString *)filePath password:(NSStringg

    NSData *p12Data = [NSData dataWithContentsOfFile:filePath];
    if (!p12Data) {
        return nil;
    }
    SecKeyRef privateKeyRef = NULL;
    NSMutableDictionary * options = [[NSMutableDictionary alloc] init];
    [options setObject: password forKey:(__bridge id)kSecImportExportPassphrase];
    CFArrayRef items = CFArrayCreate(NULL, 0, 0, NULL);
    OSStatus securityError = SecPKCS12Import((__bridge CFDataRef) p12Data, (__bridge CF
    if (securityError == noErr && CFArrayGetCount(items) > 0) {
        CFDictionaryRef identityDict = CFArrayGetValueAtIndex(items, 0);
        SecIdentityRef identityApp = (SecIdentityRef)CFDictionaryGetValue(identityDict,
        securityError = SecIdentityCopyPrivateKey(identityApp, &privateKeyRef);
        if (securityError != noErr) {
            privateKeyRef = NULL;
        }
    }
    CFRelease(items);

    return privateKeyRef;
}

+ (NSString *)decryptString:(NSString *)str privateKeyRef:(SecKeyRef)privKeyRef{
    NSData *data = [[NSData alloc] initWithBase64EncodedString:str options:NSDataBase64
    if (!privKeyRef) {
        return nil;
    }
    data = [self decryptData:data withKeyRef:privKeyRef];
    NSString *ret = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
```

```
    return ret;106 }

#pragma mark – 使用公钥字符串加密

/* START: Encryption with RSA public key */

//使用公钥字符串加密
+ (NSString *)encryptString:(NSString *)str publicKey:(NSString *)pubKey{
    NSData *data = [self encryptData:[str dataUsingEncoding:NSUTF8StringEncoding] publi
    NSString *ret = base64_encode_data(data);
    return ret;
}

+ (NSData *)encryptData:(NSData *)data publicKey:(NSString *)pubKey{
    if(!data || !pubKey){
        return nil;
    }
    SecKeyRef keyRef = [self addPublicKey:pubKey];
    if(!keyRef){
        return nil;
    }
    return [self encryptData:data withKeyRef:keyRef];
}

+ (SecKeyRef)addPublicKey:(NSString *)key{
    NSRange spos = [key rangeOfString:@"-----BEGIN PUBLIC KEY-----"];
    NSRange epos = [key rangeOfString:@"-----END PUBLIC KEY-----"];
    if(spos.location != NSNotFound && epos.location != NSNotFound){
        NSUInteger s = spos.location + spos.length;
        NSUInteger e = epos.location;
        NSRange range = NSMakeRange(s, e-s);
        key = [key substringWithRange:range];
    }
    key = [key stringByReplacingOccurrencesOfString:@"\r" withString:@""];
    key = [key stringByReplacingOccurrencesOfString:@"\n" withString:@""];
    key = [key stringByReplacingOccurrencesOfString:@"\t" withString:@""];
    key = [key stringByReplacingOccurrencesOfString:@" "  withString:@""];

    // This will be base64 encoded, decode it.
    NSData *data = base64_decode(key);
    data = [self stripPublicKeyHeader:data];
    if(!data){
        return nil;
    }

    //a tag to read/write keychain storage
    NSString *tag = @"RSAUtil_PubKey";
    NSData *d_tag = [NSData dataWithBytes:[tag UTF8String] length:[tag length]];

    // Delete any old lingering key with the same tag
    NSMutableDictionary *publicKey = [[NSMutableDictionary alloc] init];
    [publicKey setObject:(__bridge id) kSecClassKey forKey:(__bridge id)kSecClass];
    [publicKey setObject:(__bridge id) kSecAttrKeyTypeRSA forKey:(__bridge id)kSecAttrK
    [publicKey setObject:d_tag forKey:(__bridge id)kSecAttrApplicationTag];
    SecItemDelete((__bridge CFDictionaryRef)publicKey);

    // Add persistent version of the key to system keychain
    [publicKey setObject:data forKey:(__bridge id)kSecValueData];
    [publicKey setObject:(__bridge id) kSecAttrKeyClassPublic forKey:(__bridge id)
     kSecAttrKeyClass];
    [publicKey setObject:[NSNumber numberWithBool:YES] forKey:(__bridge id)
     kSecReturnPersistentRef];

    CFTypeRef persistKey = nil;
    OSStatus status = SecItemAdd((__bridge CFDictionaryRef)publicKey, &persistKey);
    if (persistKey != nil){
```

```
        CFRelease(persistKey);
    }
    if ((status != noErr) && (status != errSecDuplicateItem)) {
        return nil;
    }

    [publicKey removeObjectForKey:(__bridge id)kSecValueData];
    [publicKey removeObjectForKey:(__bridge id)kSecReturnPersistentRef];
    [publicKey setObject:[NSNumber numberWithBool:YES] forKey:(__bridge id)kSecReturnRe
    [publicKey setObject:(__bridge id) kSecAttrKeyTypeRSA forKey:(__bridge id)kSecAttrK

    // Now fetch the SecKeyRef version of the key
    SecKeyRef keyRef = nil;
    status = SecItemCopyMatching((__bridge CFDictionaryRef)publicKey, (CFTypeRef *)&key
    if(status != noErr){
        return nil;
    }
    return keyRef;
}

+ (NSData *)stripPublicKeyHeader:(NSData *)d_key{
    // Skip ASN.1 public key header
    if (d_key == nil) return(nil);

    unsigned long len = [d_key length];
    if (!len) return(nil);

    unsigned char *c_key = (unsigned char *)[d_key bytes];
    unsigned int  idx     = 0;

    if (c_key[idx++] != 0x30) return(nil);

    if (c_key[idx] > 0x80) idx += c_key[idx] - 0x80 + 1;
    else idx++;

    // PKCS #1 rsaEncryption szOID_RSA_RSA
    static unsigned char seqiod[] =
    { 0x30,   0x0d, 0x06, 0x09, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x01,
        0x01, 0x05, 0x00 };
    if (memcmp(&c_key[idx], seqiod, 15)) return(nil);

    idx += 15;

    if (c_key[idx++] != 0x03) return(nil);

    if (c_key[idx] > 0x80) idx += c_key[idx] - 0x80 + 1;
    else idx++;

    if (c_key[idx++] != '\0') return(nil);

    // Now make a new NSData from this buffer
    return ([NSData dataWithBytes:&c_key[idx] length:len - idx]);
}

+ (NSData *)encryptData:(NSData *)data withKeyRef:(SecKeyRef) keyRef{
    const uint8_t *srcbuf = (const uint8_t *)[data bytes];
    size_t srclen = (size_t)data.length;

    size_t block_size = SecKeyGetBlockSize(keyRef) * sizeof(uint8_t);
    void *outbuf = malloc(block_size);
    size_t src_block_size = block_size - 11;

    NSMutableData *ret = [[NSMutableData alloc] init];
    for(int idx=0; idx<srclen; idx+=src_block_size){
        //NSLog(@"%d/%d block_size: %d", idx, (int)srclen, (int)block_size);
        size_t data_len = srclen - idx;
        if(data_len > src_block_size){
```

```
            data_len = src_block_size;
                                        240         }

        size_t outlen = block_size;
        OSStatus status = noErr;
        status = SecKeyEncrypt(keyRef,
                               kSecPaddingPKCS1,
                               srcbuf + idx,
                               data_len,
                               outbuf,
                               &outlen
                               );
        if (status != 0) {
            NSLog(@"SecKeyEncrypt fail. Error Code: %d", status);
            ret = nil;
            break;
        }else{
            [ret appendBytes:outbuf length:outlen];
        }
    }

    free(outbuf);
    CFRelease(keyRef);
    return ret;
}

/* END: Encryption with RSA public key */

#pragma mark - 使用私钥字符串解密

/* START: Decryption with RSA private key */

//使用私钥字符串解密
+ (NSString *)decryptString:(NSString *)str privateKey:(NSString *)privKey{
    if (!str) return nil;
    NSData *data = [[NSData alloc] initWithBase64EncodedString:str options:NSDataBase64
    data = [self decryptData:data privateKey:privKey];
    NSString *ret = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
    return ret;
}

+ (NSData *)decryptData:(NSData *)data privateKey:(NSString *)privKey{
    if(!data || !privKey){
        return nil;
    }
    SecKeyRef keyRef = [self addPrivateKey:privKey];
    if(!keyRef){
        return nil;
    }
    return [self decryptData:data withKeyRef:keyRef];
}

+ (SecKeyRef)addPrivateKey:(NSString *)key{
    NSRange spos = [key rangeOfString:@"-----BEGIN RSA PRIVATE KEY-----"];
    NSRange epos = [key rangeOfString:@"-----END RSA PRIVATE KEY-----"];
    if(spos.location != NSNotFound && epos.location != NSNotFound){
        NSUInteger s = spos.location + spos.length;
        NSUInteger e = epos.location;
        NSRange range = NSMakeRange(s, e-s);
        key = [key substringWithRange:range];
    }
    key = [key stringByReplacingOccurrencesOfString:@"\r" withString:@""];
    key = [key stringByReplacingOccurrencesOfString:@"\n" withString:@""];
    key = [key stringByReplacingOccurrencesOfString:@"\t" withString:@""];
    key = [key stringByReplacingOccurrencesOfString:@" "  withString:@""];

```

```
    // This will be base64 encoded, decode it.
    NSData *data = base64_decode(key
    data = [self stripPrivateKeyHeader:data];
    if(!data){
        return nil;
    }

    //a tag to read/write keychain storage
    NSString *tag = @"RSAUtil_PrivKey";
    NSData *d_tag = [NSData dataWithBytes:[tag UTF8String] length:[tag length]];

    // Delete any old lingering key with the same tag
    NSMutableDictionary *privateKey = [[NSMutableDictionary alloc] init];
    [privateKey setObject:(__bridge id) kSecClassKey forKey:(__bridge id)kSecClass];
    [privateKey setObject:(__bridge id) kSecAttrKeyTypeRSA forKey:(__bridge id)kSecAttr
    [privateKey setObject:d_tag forKey:(__bridge id)kSecAttrApplicationTag];
    SecItemDelete((__bridge CFDictionaryRef)privateKey);

    // Add persistent version of the key to system keychain
    [privateKey setObject:data forKey:(__bridge id)kSecValueData];
    [privateKey setObject:(__bridge id) kSecAttrKeyClassPrivate forKey:(__bridge id)
     kSecAttrKeyClass];
    [privateKey setObject:[NSNumber numberWithBool:YES] forKey:(__bridge id)
     kSecReturnPersistentRef];

    CFTypeRef persistKey = nil;
    OSStatus status = SecItemAdd((__bridge CFDictionaryRef)privateKey, &persistKey);
    if (persistKey != nil){
        CFRelease(persistKey);
    }
    if ((status != noErr) && (status != errSecDuplicateItem)) {
        return nil;
    }

    [privateKey removeObjectForKey:(__bridge id)kSecValueData];
    [privateKey removeObjectForKey:(__bridge id)kSecReturnPersistentRef];
    [privateKey setObject:[NSNumber numberWithBool:YES] forKey:(__bridge id)kSecReturnF
    [privateKey setObject:(__bridge id) kSecAttrKeyTypeRSA forKey:(__bridge id)kSecAttr

    // Now fetch the SecKeyRef version of the key
    SecKeyRef keyRef = nil;
    status = SecItemCopyMatching((__bridge CFDictionaryRef)privateKey, (CFTypeRef *)&ke
    if(status != noErr){
        return nil;
    }
    return keyRef;
}

+ (NSData *)stripPrivateKeyHeader:(NSData *)d_key{
    // Skip ASN.1 private key header
    if (d_key == nil) return(nil);

    unsigned long len = [d_key length];
    if (!len) return(nil);

    unsigned char *c_key = (unsigned char *)[d_key bytes];
    unsigned int  idx     = 22; //magic byte at offset 22

    if (0x04 != c_key[idx++]) return nil;

    //calculate length of the key
    unsigned int c_len = c_key[idx++];
    int det = c_len & 0x80;
    if (!det) {
        c_len = c_len & 0x7f;
    } else {
        int byteCount = c_len & 0x7f;
```

```
        if (byteCount + idx > len) {
            //rsa length field longer than buf
            return nil;
        }
        unsigned int accum = 0;
        unsigned char *ptr = &c_key[idx];
        idx += byteCount;
        while (byteCount) {
            accum = (accum << 8) + *ptr;
            ptr++;
            byteCount--;
        }
        c_len = accum;
    }

    // Now make a new NSData from this buffer
    return [d_key subdataWithRange:NSMakeRange(idx, c_len)];
}

+ (NSData *)decryptData:(NSData *)data withKeyRef:(SecKeyRef) keyRef{
    const uint8_t *srcbuf = (const uint8_t *)[data bytes];
    size_t srclen = (size_t)data.length;

    size_t block_size = SecKeyGetBlockSize(keyRef) * sizeof(uint8_t);
    UInt8 *outbuf = malloc(block_size);
    size_t src_block_size = block_size;

    NSMutableData *ret = [[NSMutableData alloc] init];
    for(int idx=0; idx<srclen; idx+=src_block_size){
        //NSLog(@"%d/%d block_size: %d", idx, (int)srclen, (int)block_size);
        size_t data_len = srclen - idx;
        if(data_len > src_block_size){
            data_len = src_block_size;
        }

        size_t outlen = block_size;
        OSStatus status = noErr;
        status = SecKeyDecrypt(keyRef,
                               kSecPaddingNone,
                               srcbuf + idx,
                               data_len,
                               outbuf,
                               &outlen
                               );
        if (status != 0) {
            NSLog(@"SecKeyEncrypt fail. Error Code: %d", status);
            ret = nil;
            break;
        }else{
            //the actual decrypted data is in the middle, locate it!
            int idxFirstZero = -1;
            int idxNextZero = (int)outlen;
            for ( int i = 0; i < outlen; i++ ) {
                if ( outbuf[i] == 0 ) {
                    if ( idxFirstZero < 0 ) {
                        idxFirstZero = i;
                    } else {
                        idxNextZero = i;
                        break;
                    }
                }
            }

            [ret appendBytes:&outbuf[idxFirstZero+1] length:idxNextZero-idxFirstZero-1];
        }
    }

```

```
    free(outbuf);
                440     CFRelease(keyRef);
    return ret;
}

/* END: Decryption with RSA private key */

@end
```

4. 测试加密、解密

首先先测试使用.der和.p12秘钥文件进行加密、解密, 在ViewController.m中进行测试, 代码如下:

```
#import "ViewController.h"
#import "RSAEncryptor.h"

@interface ViewController ()

@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    //原始数据
    NSString *originalString = @"这是一段将要使用'.der'文件加密的字符串!";

    //使用.der和.p12中的公钥私钥加密解密
    NSString *public_key_path = [[NSBundle mainBundle] pathForResource:@"public_key.der'
    NSString *private_key_path = [[NSBundle mainBundle] pathForResource:@"private_key.p

    NSString *encryptStr = [RSAEncryptor encryptString:originalString publicKeyWithConte
    NSLog(@"加密前:%@", originalString);
    NSLog(@"加密后:%@", encryptStr);
    NSLog(@"解密后:%@", [RSAEncryptor decryptString:encryptStr privateKeyWithContentsOfF

}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```

运行后, 输出信息如下:

```
2016-04-21 12:18:42.076 RSA加密[1414:95647] 加密前:这是一段将要使用'.der'文件加密的字符串!
2016-04-21 12:18:42.076 RSA加密[1414:95647] 加密后:URJhXPK27XBfK0uXgQvw6WH0zHQJdVjwODvD7croToBDkL/ElAcmQRSglei10PAwSL6z
NFr5S2kCtZW5tkzvFlMk/ZbiVMWkXm+yVrgd9jH354EeYwvkMqv7k04pfDyb/RFCKmUIy0CPIyONKuXLCmPqkzbXkAaU=
2016-04-21 12:18:42.082 RSA加密[1414:95647] 解密后:这是一段将要使用'.der'文件加密的字符串!
```

输出结果

可以看到已经可以成功加密、解密了.

下面接着测试使用秘钥字符串进行加密、解密, 那么秘钥字符串从哪里来? 可以来这里:http://web.chacuo.net 在线生成RSA秘钥的网站, 生成公钥和秘钥后, 复制出来用于测试. 然后在ViewController.m中使用RSAEntrypt方法进行加密, ViewController.m中代码如下:

```
#import "ViewController.h"
#import "RSAEncryptor.h"

@interface ViewController ()

```

```
@end

@implementation ViewController

- (void)viewDidLoad {
    [super viewDidLoad];

    //原始数据
    NSString *originalString = @"这是一段将要使用'秘钥字符串'进行加密的字符串!";

    //使用字符串格式的公钥私钥加密解密
    NSString *encryptStr = [RSAEncryptor encryptString:originalString publicKey:@"MIGfM

    NSLog(@"加密前:%@", originalString);
    NSLog(@"加密后:%@", encryptStr);
    NSLog(@"解密后:%@", [RSAEncryptor decryptString:encryptStr privateKey:@"MIICeAIBADANI

}

- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

@end
```

运行后, 输出信息如下:

```
2016-04-21 12:30:42.903 RSA加密[1437:102498] 加密前:这是一段将要使用'秘钥字符串'进行加密的字符串!
2016-04-21 12:30:42.903 RSA加密[1437:102498] 加密后:hQ1TVly2OR0OHNdRU9C3RWW4r4teesRcOUIMnc6QORxVH2FKgYOhazAvRMyYx/
mc1dphJRbsUUTGimYytgt3w404QlD9Udd1HtWAWds1y+Z1VYYG6qcqdzHvTBK+FttAigmkFSiW8Gwjzrrd0ZdSlAtw+qtmnll7P1TdHBNN7BM=
2016-04-21 12:30:42.908 RSA加密[1437:102498] 解密后:这是一段将要使用'秘钥字符串'进行加密的字符串!
```

输出结果

可以看到,也成功加密、解密了.

至此, RSA加密演示完毕!

文 / jianshu_wl（简书作者）
原文链接：http://www.jianshu.com/p/74a796ec5038

iOS客户端与JAVA服务器之间的RSA加密解密

文章转载自：http://www.cnblogs.com/makemelike/articles/3802518.html 在网上找了许多篇关于RSA加密解密的文章与博客与不...

想对作者说点什么? 我来说一句

iOS之RSA加密解密(亲测可用)

iOS之纯代码实现非对称加密和解密过程,亲测可用,不能实现退分!... iOS之RSA

iOS加密解密之rsa完整代码

内附rsa双向加密完整代码,适合iOS开发初中级开发人员。

iOS之RSA加密解密与后台之间的双向加密详解

注：本文全部转载自：https://www.jianshu.com/p/43f7fc8d8e14iOS之RSA加密解密与后台之间的双向加密详解序言因为项目中

IOS中RSA的加密解密

1. 生成私钥: openssl req -x509 -days 3650 -new -newkey rsa:2048 -keyout private_key.pem -out private_key...

iOS与JAVA之间的RSA加解密

RSA算法是一种非对称加密算法,常被用于加密数据传输.如果配合上数字摘要算法, 也可以用于文件签名. 本文将讨论如何在iOS中

iOS 与Java 配合在用户登录的时候对用户名和密码进行RSA加密

首先RSA加密不是很难，，，你首先要找到靠谱的第三方RSA算法，，那么我来讲一下我们项目对于这方面的设计吧....

java与IOS之间的RSA加解密

转自：http://yuur369.iteye.com/blog/1769395 很简单的一个需求，ipad端给密码RSA加密，传到java后台，解密。RSA加密算法

下载 iOS加密解密之rsa完整代码

内附rsa双向加密完整代码，适合iOS开发初中级开发人员。

通过ios实现RSA加密解密中的 RSAEncryptor.h/m相关代码

RSAEncryptor.h代码 // // RSAEncryptor.h // 121mai // // Created by 薛XX on 16/2/16. // // #import ...

ios 安卓 javaweb RSA加密解密

ios版， 公钥私钥一键加密解密 @interface RSA : NSObject // return base64 encoded string + (NSString *)encryptSt...

iOS 客户端进行 RSA 加密并在 PHP 服务端进行解密

前言本文是对 Js Lim 的一篇博客的翻译，原文链接如下： http://jslim.net/blog/2013/06/24/rsa-decryption-on-ios/ 若本文有翻译不

openssl ios 公钥分段加解密

由于苹果官方api只支持公钥加密，私钥加密，不支持公钥解密来自服务器端私钥加密的数据，故改用openssl实现这个 - (NSDat

使用RSA算法实现对数据的加解密

在我们现实当中经常会存在需要对某些数据进行加密保护 然后进行解密的操作,比方,我们需要对某些XML配置信息里面的某些数

RSA公私钥进行数据加解密

本文出处：http://blog.csdn.net/chaijunkun/article/details/7275632，转载请注明。由于本人不定期会整理相关博文，会对相应内

MAC OSX下的RSA加解密实现

MAC OSX下的RSA加解密实现

JAVA RSA加解密和数字签名、DES加解密 在项目中的实际使用

RSA： 1、生成随机秘钥对 2、用公钥加密私钥解密 客户端：RSA用公钥加密之后，需要对加密后的数据在进行Base64加密，

rsa 加密解密工具类

rsa 加解密工具类

非对称加解密——RSA加密、解密以及数字签名

对称与非对称加解密，最主要区别在于：对称加密，加解密的密钥是一致的；非对称加密，加解密的密钥是不一致的； 对称加密

RSA对文件加解密

纯手打代码，借鉴了下面网址中的已有的函数，已完整实现 package RSA; import java.io.FileInputStream; import java.io.FileOu

Java 进行 RSA 加解密的例子

加密是保证数据安全的手段之一。加密是将纯文本数据转换为难以理解的密文；解密是将密文转换回纯文本。数据的加解密属于

下载 JAVA中RSA加密解密工具类加强版

博客地址 http://blog.csdn.net/sbsujjbcy/article/details/46873403

python实现一款rsa加解密工具

由于最近本人老忘记许多密码，直接把密码明文传到网盘或者其他地方太不安全。便想着利用rsa的公钥将容易忘记的密码加密后

java实现RSA的简单加密解密

RSAUtil package com.zhuyun.rsa; import java.io.IOException; import java.security.KeyFactory; i...

下载 RSA生成密钥对、公钥加密和私钥解密

支持最大2048位RSA计算，主要是生成公私钥对、公钥加密、私钥解密功能。每次重新生成公私钥对，随机产生一定长度的随机数作为输入数据，公

RSA加密算法加密与解密过程解析

RSA加密算法剖析

OpenSSL中RSA的简易加解密流程

RSA是公钥密码体制的典范，在本实验中，我们的将使用OpenSSL自带的RSA相关函数生成RSA加密体系。下面是可能要使用到

python3 使用Pycrypto进行RSA加解密

为方便自己使用，直接记录下代码，修改公约和私钥就行 import base64 from Crypto import Random from Crypto.Cipher import

python 利用pycrypto进行rsa生成公钥、私钥，加密、解密、签名、解签

1、安装 pycrypto pip install pycrypto 2、利用pycrypto进行rsa生成公钥、私钥，加密、解密、签名、解签 # -*- coding: utf-8 -*-..

常用加密解密算法【RSA、AES、DES、MD5】介绍和使用

为了防止我们的数据泄露，我们往往会对数据进行加密，特别是敏感数据，我们要求的安全性更高。下面将介绍几种常用的加密

漫谈iOS RSA非对称加密与解密

前言最近公司的客户端安全性出现了严重的问题，如今这个出解决方案并自我测试验证可行性的重任落在了我的身上，学习了很

下载 Java实现RSA加解密工具类Demo

这个RSA加解密的Demo是我从别的地方花了好多金币下载的，希望对大家有帮助，资源共享嘛。

RSA加解密工具类讲解（Java实现）

这里我写了一个工具类： import java.security.KeyFactory; import java.security.KeyPair; import java.security.KeyP...

跨平台rsa签名与验签

本文主要讲java和python平台的rsa签名与验签，java使用的是16进制密钥，python使用的是pkcs8编码格式的密钥，其原理其他

java使用RSA加密方式实现数据加密解密

全栈工程师开发手册 （作者：栾鹏） java教程全解 java使用RSA加密方式实现数据加密解密，需要首先产生私钥和公钥测试代码

RSA加解密算法

RSA加解密算法 算法简介 假设资料要由A机器传至B机器,那,由B机器用乱数决定一个key,我们称之为privatekey,这个key自始至终

RSA加密解密以及内容超长时采用分段加密

RSA加密解密以及内容超长时采用分段加密 1、在使用 RSA加密解密内容时会出现这样的异常： Data must not be longer than

RSA加解密算法java实现（已添加分段加密算法处理）

一 RSA简介 这种算法1978年就出现了，它是第一个既能用于数据加密也能用于数字签名的算法。它易于理解和操作，也很流行，

DES、RSA（分段加解密） Android中常用的两种加密方式

DES加解密 public class DESEncrypt { /** * DES解密 * @param decryptString 密文 * @param iv...

下载 RSA算法公钥私钥加解密C语言源码调试通过

RSA非对称加解密算法，目前主流的加密算法，采用大数库生成大素数，然后根据算法原理，进行大数运算；算法在生成大素数时候相对耗时，但是在

基于openssl的RSA加解密实现

一、引言 openssl是一套第三方的关于数据完整性的安全协议，有一些常用的密码算法，数字证书，数字签名等等方面的一些应用

openssl命令行进行RSA加密解密

http://www.cnblogs.com/aLittleBitCool/archive/2011/09/22/2185418.html openssl是一个功能强大的工具包，它集成了众多...

java-RSA加密解密，支持分段加解密

java-RSA加密解密，支持分段加解密 RSA公钥加密算法是1977年由罗纳德·李维斯特（Ron Rivest）、阿迪·萨莫尔（Adi Shamir

python实现rsa加密

一 代码 import rsa key = rsa.newkeys(3000)#生成随机秘钥 privateKey = key[1]#私钥 publicKey = key[0]#公钥 ...

iOS和java之间的RSA加密解密、加签认证对接

iOS – java RSA 非对称加密对接

Client-ServerRSA加解密通信方案-Server端(C++)（一）

0. 背景最近，需要新做一个游戏demo，类似《部落冲突·皇室战争》的推塔玩法。客户端使用Unity，编程语言为C#，服务端使用

rsa加密算法的加解密所踩过的坑

RSA加解密遇到的问题。 1首先是私钥和公钥的读取 项目组使用的是.key格式的公私钥。已有读取代码，但是读取不到。百度说

Java和android及iOS对接RSA加密经验

1.网上找的java生成RSA密钥对的例子,产生的字附串实际上是hax后和密钥串 你可以将他们当成静态字附串存在java代码里 2.an

RSA加密解密（附源码工程）

一、RSA加密介绍RSA公钥加密算法是1977年由罗纳德·李维斯特（Ron Rivest）、阿迪·萨莫尔（Adi Shamir）和伦纳德·阿德曼

RSA原理和手工，工具解密

2018/3/30criedcat密码学writeup笔者借阅一些网络文献来总结笔者一周来对密码学以及rsa加密方法的认知。本writeup提纲：1rs

没有更多推荐了，返回首页