

185023528

关注

原创8

粉丝3

喜欢2

评论0

等级: 博客 已

访问: 1万+

积分: 282

排名: 30万+

电路板抄板

最新文章

MFC中Rich Edit 2.0 控件中字体不一致的问题

字体函数 -- GetDeviceCaps

cstring与char *的转换

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

C语言中的二级指针和二维数组问题

个人分类

嵌入式UI5篇

C#4篇

ios app4篇

ios1篇

linux1篇

展开

归档

2017年6月2篇

2017年5月1篇

2017年4月1篇

2016年11月1篇

2016年8月1篇

展开

热门文章

C语言中的二级指针和二维数组问题

阅读量: 4036

(转载) NPOI使用手册, 实践发现使用2.2

转

ios使用rsa加解密

2016年05月10日 11:01:41185023528 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 key)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生成所需密钥文件

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
```

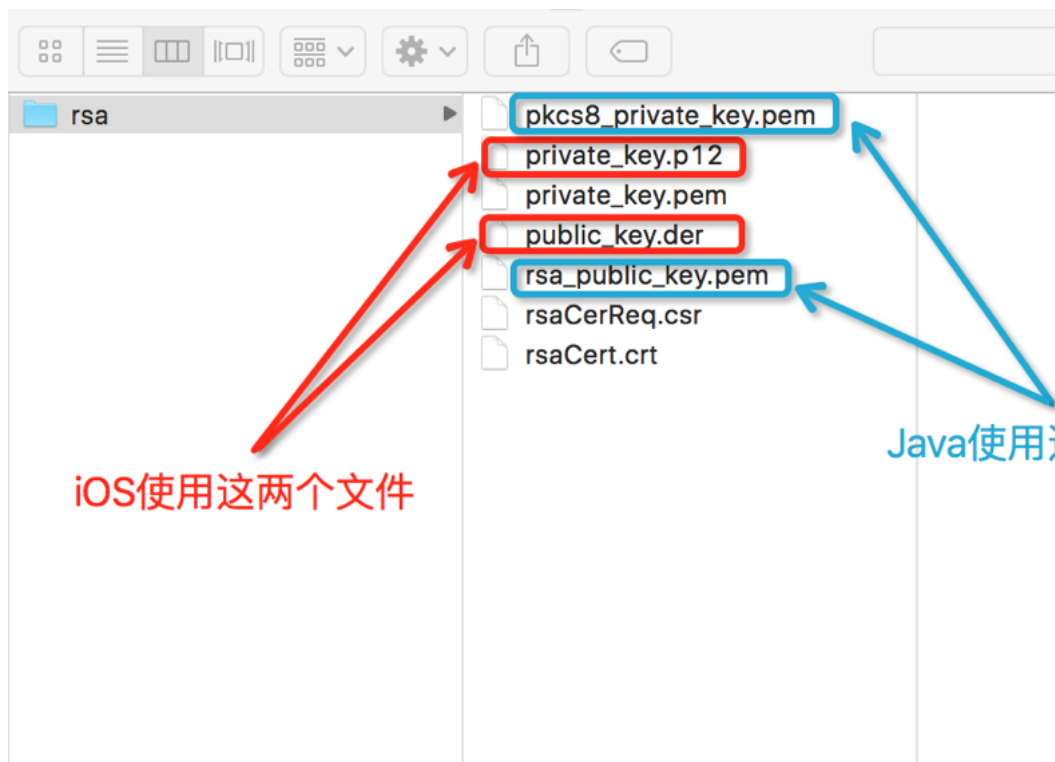
全部执行成功后, 会生成如下文件, 其中public_key.der和private_key.p12就是iOS需要用到的文件, 如下图:

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

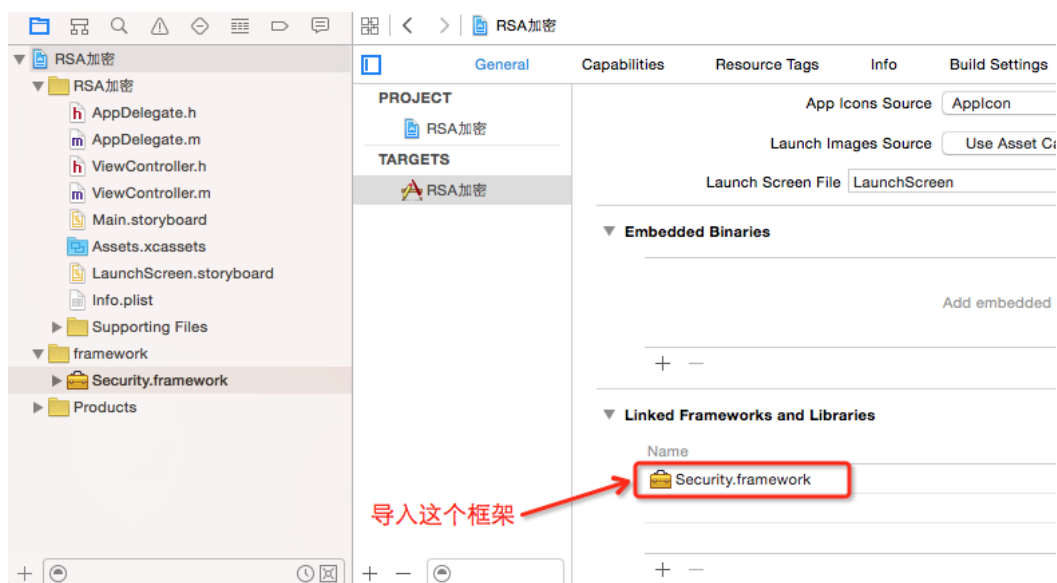
VM12安装centos7安装VMare-tool后, hgfs
下没有共享文件夹
阅读量: 1575



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.导入密钥文件

导入.der和.p12格式的密钥文件, 如下图:



单片机 解密



联系我们



扫码联系客服



扫码下载APP

关于 招聘 广告服务 网站地图
京ICP证09002463号 百度提供站内搜索
©2018 CSDN版权所有

kefu@csdn.net 400-660-0108

QQ客服 客服论坛

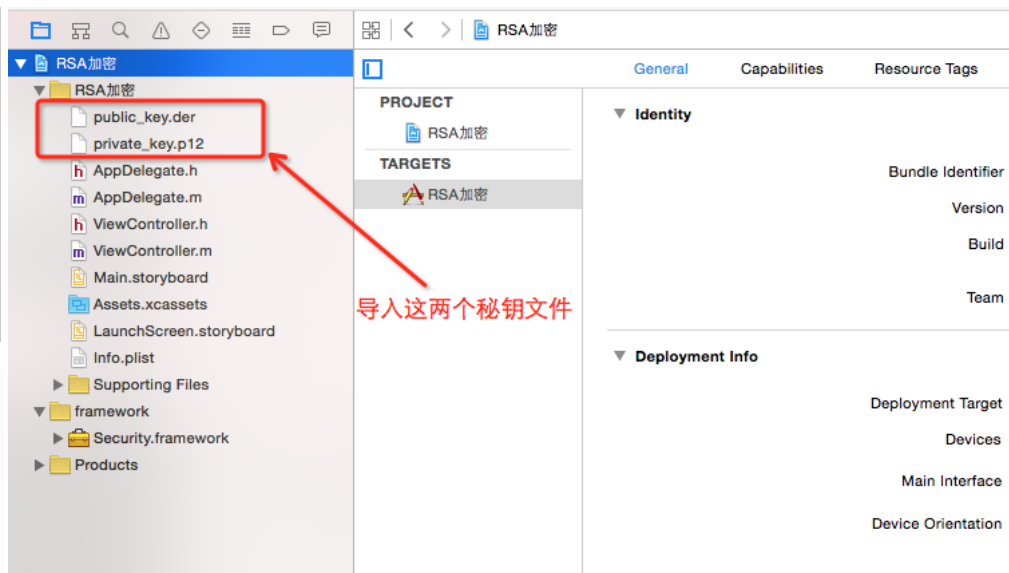
经营性网站备案信息 网络110报警服务
中国互联网举报中心 北京互联网违法和不良信息举报中心



官方公众号



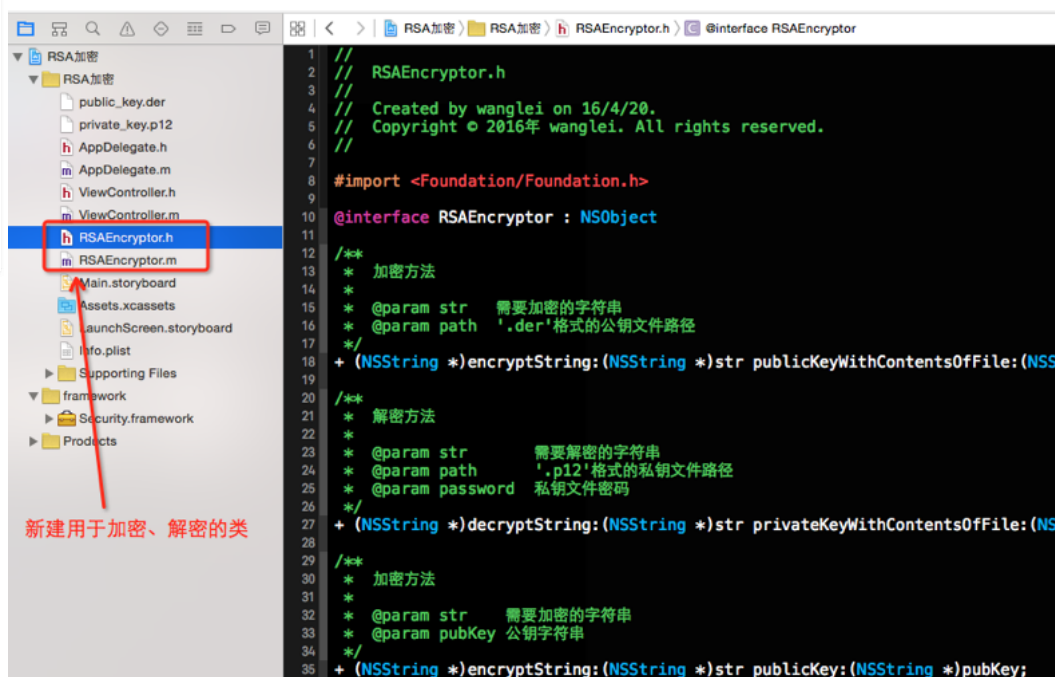
区块链大本营



导入密钥文件

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:

```
1 #import <Foundation/Foundation.h>
2
3 @interface RSAEncryptor : NSObject
4
5 /**
6  * 加密方法
```

开发者调查

AI开发者大会日程曝光

告别知识焦虑, 即刻启程

敏感词过滤算法

扫描原理

登录

注册

X

```
9 * @param path 需要加密的字符串
10 */
11 + (NSString *)encryptString:(NSString *)str publicKeyWithContentsOfFile:(NSString *)path
12
13 /**
```

```

14  * 解密方法15 | *
15  * @param str      需要解密的字符串
16  * @param path      '.p12' 格式的私钥文件路径
17  * @param password  私钥文件密码
18  */
19  + (NSString *)decryptString:(NSString *)str privateKeyWithContentsOfFile:(NSString *)pa
20  /**
21  * 加密方法
22  *
23  * @param str      需要加密的字符串
24  * @param pubKey  公钥字符串
25  */
26  + (NSString *)encryptString:(NSString *)str publicKey:(NSString *)pubKey;
27  /**
28  * 解密方法
29  *
30  * @param str      需要解密的字符串
31  * @param privKey  私钥字符串
32  */
33  + (NSString *)decryptString:(NSString *)str privateKey:(NSString *)privKey;
34  @end

```

RSAEncryptor.m代码如下:

```

1  #import "RSAEncryptor.h"
2  #import <Security/Security.h>
3
4  @implementation RSAEncryptor
5
6  static NSString *base64_encode_data(NSData *data){
7      data = [data base64EncodedDataWithOptions:0];
8      NSString *ret = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
9      return ret;
10 }
11
12 static NSData *base64_decode(NSString *str){
13     NSData *data = [[NSData alloc] initWithBase64EncodedString:str options:NSDataBase64
14     return data;
15 }
16
17 #pragma mark - 使用'.der'公钥文件加密
18
19 // 加密
20 + (NSString *)encryptString:(NSString *)str publicKeyWithContentsOfFile:(NSString *)pa
21     if (!str || !path) return nil;
22     return [self encryptString:str publicKeyRef:[self getPublicKeyRefWithContentsOfFile
23 }
24
25 // 获取公钥
26 + (SecKeyRef)getPublicKeyRefWithContentsOfFile:(NSString *)filePath{
27     NSData *certData = [NSData dataWithContentsOfFile:filePath];
28     if (!certData) {
29         return nil;
30     }
31     SecCertificateRef cert = SecCertificateCreateWithData(NULL, (CFDataRef)certData);
32     SecKeyRef key = NULL;
33     SecTrustRef trust = NULL;
34     SecPolicyRef policy = NULL;
35     if (cert != NULL) {
36         policy = SecPolicyCreateBasicX509();
37         if (policy) {
38             if (SecTrustCreateWithCertificates((CFTypeRef)cert, policy, &trust) == noEr

```

```

39         SecTrustResultType result;
40         if (SecTrustEvaluate(tru
41             key = SecTrustCopyPublicKey(trust);
42         }
43     }
44 }
45 }
46 if (policy) CFRelease(policy);
47 if (trust) CFRelease(trust);
48 if (cert) CFRelease(cert);
49 return key;
50 }
51
52 + (NSString *)encryptString:(NSString *)str publicKeyRef:(SecKeyRef)publicKeyRef{
53     if(![str dataUsingEncoding:NSUTF8StringEncoding]){
54         return nil;
55     }
56     if(!publicKeyRef){
57         return nil;
58     }
59     NSData *data = [self encryptData:[str dataUsingEncoding:NSUTF8StringEncoding] withK
60     NSString *ret = base64_encode_data(data);
61     return ret;
62 }
63
64 #pragma mark - 使用'.12'私钥文件解密
65
66 // 解密
67 + (NSString *)decryptString:(NSString *)str privateKeyWithContentsOfFile:(NSString *)pa
68     if (!str || !path) return nil;
69     if (!password) password = @"";
70     return [self decryptString:str privateKeyRef:[self getPrivateKeyRefWithContentsOfFi
71 }
72
73 // 获取私钥
74 + (SecKeyRef)getPrivateKeyRefWithContentsOfFile:(NSString *)filePath password:(NSString
75
76     NSData *p12Data = [NSData dataWithContentsOfFile:filePath];
77     if (!p12Data) {
78         return nil;
79     }
80     SecKeyRef privateKeyRef = NULL;
81     NSMutableDictionary * options = [[NSMutableDictionary alloc] init];
82     [options setObject: password forKey:(__bridge id)kSecImportExportPassphrase];
83     CFArrayRef items = CFArrayCreate(NULL, 0, 0, NULL);
84     OSStatus securityError = SecPKCS12Import((__bridge CFDataRef) p12Data, (__bridge CF
85     if (securityError == noErr && CFArrayGetCount(items) > 0) {
86         CFDictionaryRef identityDict = CFArrayGetValueAtIndex(items, 0);
87         SecIdentityRef identityApp = (SecIdentityRef)CFDictionaryGetValue(identityDict,
88         securityError = SecIdentityCopyPrivateKey(identityApp, &privateKeyRef);
89         if (securityError != noErr) {
90             privateKeyRef = NULL;
91         }
92     }
93     CFRelease(items);
94
95     return privateKeyRef;
96 }
97
98 + (NSString *)decryptString:(NSString *)str privateKeyRef:(SecKeyRef)privKeyRef{
99     NSData *data = [[NSData alloc] initWithBase64EncodedString:str options:NSDataBase64
100     if (!privKeyRef) {
101         return nil;
102     }
103     data = [self decryptData:data withKeyRef:privKeyRef];
104     NSString *ret = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];

```

```
105     return ret;106 }
107
108 #pragma mark - 使用公钥字符串加密
109
110 /* START: Encryption with RSA public key */
111
112 // 使用公钥字符串加密
113 + (NSString *)encryptString:(NSString *)str publicKey:(NSString *)pubKey{
114     NSData *data = [self encryptData:str dataUsingEncoding:NSUTF8StringEncoding] publi
115     NSString *ret = base64_encode_data(data);
116     return ret;
117 }
118
119 + (NSData *)encryptData:(NSData *)data publicKey:(NSString *)pubKey{
120     if(!data || !pubKey){
121         return nil;
122     }
123     SecKeyRef keyRef = [self addPublicKey:pubKey];
124     if(!keyRef){
125         return nil;
126     }
127     return [self encryptData:data withKeyRef:keyRef];
128 }
129
130 + (SecKeyRef)addPublicKey:(NSString *)key{
131     NSRange spos = [key rangeOfString:@"-----BEGIN PUBLIC KEY-----"];
132     NSRange epos = [key rangeOfString:@"-----END PUBLIC KEY-----"];
133     if(spos.location != NSNotFound && epos.location != NSNotFound){
134         NSUInteger s = spos.location + spos.length;
135         NSUInteger e = epos.location;
136         NSRange range = NSMakeRange(s, e-s);
137         key = [key substringWithRange:range];
138     }
139     key = [key stringByReplacingOccurrencesOfString:@"\\r" withString:@""];
140     key = [key stringByReplacingOccurrencesOfString:@"\\n" withString:@""];
141     key = [key stringByReplacingOccurrencesOfString:@"\\t" withString:@""];
142     key = [key stringByReplacingOccurrencesOfString:@" " withString:@""];
143
144     // This will be base64 encoded, decode it.
145     NSData *data = base64_decode(key);
146     data = [self stripPublicKeyHeader:data];
147     if(!data){
148         return nil;
149     }
150
151     // a tag to read/write keychain storage
152     NSString *tag = @"RSAUtil_PubKey";
153     NSData *d_tag = [NSData dataWithBytes:[tag UTF8String] length:[tag length]];
154
155     // Delete any old lingering key with the same tag
156     NSMutableDictionary *publicKey = [[NSMutableDictionary alloc] init];
157     [publicKey setObject:(__bridge id) kSecClassKey forKey:(__bridge id)kSecClass];
158     [publicKey setObject:(__bridge id) kSecAttrKeyTypeRSA forKey:(__bridge id)kSecAttrK
159     [publicKey setObject:d_tag forKey:(__bridge id)kSecAttrApplicationTag];
160     SecItemDelete((__bridge CFDictionaryRef)publicKey);
161
162     // Add persistent version of the key to system keychain
163     [publicKey setObject:data forKey:(__bridge id)kSecValueData];
164     [publicKey setObject:(__bridge id) kSecAttrKeyClassPublic forKey:(__bridge id)
165     kSecAttrKeyClass];
166     [publicKey setObject:[NSNumber numberWithInt:YES] forKey:(__bridge id)
167     kSecReturnPersistentRef];
168
169     CTypeRef persistKey = nil;
170     OSStatus status = SecItemAdd((__bridge CFDictionaryRef)publicKey, &persistKey);
171     if (persistKey != nil){
```

```

172         CFRelease(persistKey);
173     }
174     if ((status != noErr) && (status != errSecDuplicateItem)) {
175         return nil;
176     }
177
178     [publicKey removeObjectForKey:(__bridge id)kSecValueData];
179     [publicKey removeObjectForKey:(__bridge id)kSecReturnPersistentRef];
180     [publicKey setObject:[NSNumber numberWithInt:YES] forKey:(__bridge id)kSecReturnRe
181     [publicKey setObject:(__bridge id) kSecAttrKeyTypeRSA forKey:(__bridge id)kSecAttrk
182
183     // Now fetch the SecKeyRef version of the key
184     SecKeyRef keyRef = nil;
185     status = SecItemCopyMatching((__bridge CFDictionaryRef)publicKey, (CFTyperef *)&key
186     if(status != noErr){
187         return nil;
188     }
189     return keyRef;
190 }
191
192 + (NSData *)stripPublicKeyHeader:(NSData *)d_key{
193     // Skip ASN.1 public key header
194     if (d_key == nil) return(nil);
195
196     unsigned long len = [d_key length];
197     if (!len) return(nil);
198
199     unsigned char *c_key = (unsigned char *)[d_key bytes];
200     unsigned int idx = 0;
201
202     if (c_key[idx++] != 0x30) return(nil);
203
204     if (c_key[idx] > 0x80) idx += c_key[idx] - 0x80 + 1;
205     else idx++;
206
207     // PKCS #1 rsaEncryption szOID_RSA_RSA
208     static unsigned char seqiod[] =
209     { 0x30, 0x0d, 0x06, 0x09, 0x2a, 0x86, 0x48, 0x86, 0xf7, 0x0d, 0x01, 0x01,
210       0x01, 0x05, 0x00 };
211     if (memcmp(&c_key[idx], seqiod, 15)) return(nil);
212
213     idx += 15;
214
215     if (c_key[idx++] != 0x03) return(nil);
216
217     if (c_key[idx] > 0x80) idx += c_key[idx] - 0x80 + 1;
218     else idx++;
219
220     if (c_key[idx++] != '\0') return(nil);
221
222     // Now make a new NSData from this buffer
223     return ([NSData dataWithBytes:&c_key[idx] length:len - idx]);
224 }
225
226 + (NSData *)encryptData:(NSData *)data withKeyRef:(SecKeyRef) keyRef{
227     const uint8_t *srcbuf = (const uint8_t *)[data bytes];
228     size_t srclen = (size_t)data.length;
229
230     size_t block_size = SecKeyGetBlockSize(keyRef) * sizeof(uint8_t);
231     void *outbuf = malloc(block_size);
232     size_t src_block_size = block_size - 11;
233
234     NSMutableData *ret = [[NSMutableData alloc] init];
235     for(int idx=0; idx<srclen; idx+=src_block_size){
236         //NSLog(@"%d/%d block_size: %d", idx, (int)srclen, (int)block_size);
237         size_t data_len = srclen - idx;
238         if(data_len > src_block_size){

```



```

239         data_len = src_block_size;
240     }
241
242     size_t outlen = block_size;
243     OSStatus status = noErr;
244     status = SecKeyEncrypt(keyRef,
245                             kSecPaddingPKCS1,
246                             srcbuf + idx,
247                             data_len,
248                             outbuf,
249                             &outlen
250                             );
251     if (status != 0) {
252         NSLog(@"SecKeyEncrypt fail. Error Code: %d", status);
253         ret = nil;
254         break;
255     }else{
256         [ret appendBytes:outbuf length:outlen];
257     }
258 }
259
260 free(outbuf);
261 CFRelease(keyRef);
262 return ret;
263 }
264
265 /* END: Encryption with RSA public key */
266
267 #pragma mark - 使用私钥字符串解密
268
269 /* START: Decryption with RSA private key */
270
271 //使用私钥字符串解密
272 + (NSString *)decryptString:(NSString *)str privateKey:(NSString *)privKey{
273     if (!str) return nil;
274     NSData *data = [[NSData alloc] initWithBase64EncodedString:str options:NSDataBase64Decoding];
275     data = [self decryptData:data privateKey:privKey];
276     NSString *ret = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
277     return ret;
278 }
279
280 + (NSData *)decryptData:(NSData *)data privateKey:(NSString *)privKey{
281     if(!data || !privKey){
282         return nil;
283     }
284     SecKeyRef keyRef = [self addPrivateKey:privKey];
285     if(!keyRef){
286         return nil;
287     }
288     return [self decryptData:data withKeyRef:keyRef];
289 }
290
291 + (SecKeyRef)addPrivateKey:(NSString *)key{
292     NSRange spos = [key rangeOfString:@"-----BEGIN RSA PRIVATE KEY-----"];
293     NSRange epos = [key rangeOfString:@"-----END RSA PRIVATE KEY-----"];
294     if(spos.location != NSNotFound && epos.location != NSNotFound){
295         NSUInteger s = spos.location + spos.length;
296         NSUInteger e = epos.location;
297         NSRange range = NSMakeRange(s, e-s);
298         key = [key substringWithRange:range];
299     }
300     key = [key stringByReplacingOccurrencesOfString:@"\\r" withString:@""];
301     key = [key stringByReplacingOccurrencesOfString:@"\\n" withString:@""];
302     key = [key stringByReplacingOccurrencesOfString:@"\\t" withString:@""];
303     key = [key stringByReplacingOccurrencesOfString:@" " withString:@""];
304 }

```



```

305 // This will be base64 encoded, decode it.306 | NSData *data = base64_decode(key
307 data = [self stripPrivateKeyHeader:data];
308 if(!data){
309     return nil;
310 }
311
312 //a tag to read/write keychain storage
313 NSString *tag = @"RSAUtil_PrivKey";
314 NSData *d_tag = [NSData dataWithBytes:[tag UTF8String] length:[tag length]];
315
316 // Delete any old lingering key with the same tag
317 NSMutableDictionary *privateKey = [[NSMutableDictionary alloc] init];
318 [privateKey setObject:(__bridge id) kSecClassKey forKey:(__bridge id)kSecClass];
319 [privateKey setObject:(__bridge id) kSecAttrKeyTypeRSA forKey:(__bridge id)kSecAttr
320 [privateKey setObject:d_tag forKey:(__bridge id)kSecAttrApplicationTag];
321 SecItemDelete((__bridge CFDictionaryRef)privateKey);
322
323 // Add persistent version of the key to system keychain
324 [privateKey setObject:data forKey:(__bridge id)kSecValueData];
325 [privateKey setObject:(__bridge id) kSecAttrKeyClassPrivate forKey:(__bridge id)
326     kSecAttrKeyClass];
327 [privateKey setObject:[NSNumber numberWithInt:YES] forKey:(__bridge id)
328     kSecReturnPersistentRef];
329
330 CTypeRef persistKey = nil;
331 OSStatus status = SecItemAdd((__bridge CFDictionaryRef)privateKey, &persistKey);
332 if (persistKey != nil){
333     CFRelease(persistKey);
334 }
335 if ((status != noErr) && (status != errSecDuplicateItem)) {
336     return nil;
337 }
338
339 [privateKey removeObjectForKey:(__bridge id)kSecValueData];
340 [privateKey removeObjectForKey:(__bridge id)kSecReturnPersistentRef];
341 [privateKey setObject:[NSNumber numberWithInt:YES] forKey:(__bridge id)kSecReturnF
342 [privateKey setObject:(__bridge id) kSecAttrKeyTypeRSA forKey:(__bridge id)kSecAttr
343
344 // Now fetch the SecKeyRef version of the key
345 SecKeyRef keyRef = nil;
346 status = SecItemCopyMatching((__bridge CFDictionaryRef)privateKey, (CTypeRef *)&ke
347 if(status != noErr){
348     return nil;
349 }
350 return keyRef;
351 }
352
353 + (NSData *)stripPrivateKeyHeader:(NSData *)d_key{
354     // Skip ASN.1 private key header
355     if (d_key == nil) return(nil);
356
357     unsigned long len = [d_key length];
358     if (!len) return(nil);
359
360     unsigned char *c_key = (unsigned char *)[d_key bytes];
361     unsigned int idx = 22; //magic byte at offset 22
362
363     if (0x04 != c_key[idx++]) return nil;
364
365     //calculate length of the key
366     unsigned int c_len = c_key[idx++];
367     int det = c_len & 0x80;
368     if (!det) {
369         c_len = c_len & 0x7f;
370     } else {
371         int byteCount = c_len & 0x7f;

```

```

372         if (byteCount + idx > len) {373|           //rsa length field longer than buf
373             return nil;
374         }
375         unsigned int accum = 0;
376         unsigned char *ptr = &c_key[idx];
377         idx += byteCount;
378         while (byteCount) {
379             accum = (accum << 8) + *ptr;
380             ptr++;
381             byteCount--;
382         }
383         c_len = accum;
384     }
385 }
386
387 // Now make a new NSData from this buffer
388 return [d_key subdataWithRange:NSMakeRange(idx, c_len)];
389 }
390
391 + (NSData *)decryptData:(NSData *)data withKeyRef:(SecKeyRef) keyRef{
392     const uint8_t *srcbuf = (const uint8_t *)[data bytes];
393     size_t srclen = (size_t)data.length;
394
395     size_t block_size = SecKeyGetBlockSize(keyRef) * sizeof(uint8_t);
396     UInt8 *outbuf = malloc(block_size);
397     size_t src_block_size = block_size;
398
399     NSMutableData *ret = [[NSMutableData alloc] init];
400     for(int idx=0; idx<srclen; idx+=src_block_size){
401         //NSLog(@"%d/%d block_size: %d", idx, (int)srclen, (int)block_size);
402         size_t data_len = srclen - idx;
403         if(data_len > src_block_size){
404             data_len = src_block_size;
405         }
406
407         size_t outlen = block_size;
408         OSStatus status = noErr;
409         status = SecKeyDecrypt(keyRef,
410                               kSecPaddingNone,
411                               srcbuf + idx,
412                               data_len,
413                               outbuf,
414                               &outlen
415                               );
416         if (status != 0) {
417             NSLog(@"SecKeyEncrypt fail. Error Code: %d", status);
418             ret = nil;
419             break;
420         }else{
421             //the actual decrypted data is in the middle, locate it!
422             int idxFirstZero = -1;
423             int idxNextZero = (int)outlen;
424             for ( int i = 0; i < outlen; i++ ) {
425                 if ( outbuf[i] == 0 ) {
426                     if ( idxFirstZero < 0 ) {
427                         idxFirstZero = i;
428                     } else {
429                         idxNextZero = i;
430                         break;
431                     }
432                 }
433             }
434
435             [ret appendBytes:&outbuf[idxFirstZero+1] length:idxNextZero-idxFirstZero-1]
436         }
437     }
438 }

```

```

439         free(outbuf);
440         CFRelease(keyRef);
441         return ret;
442     }
443
444     /* END: Decryption with RSA private key */
445
446 @end

```

4. 测试加密、解密

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

```

1  #import "ViewController.h"
2  #import "RSAEncryptor.h"
3
4  @interface ViewController ()
5
6  @end
7
8  @implementation ViewController
9
10 - (void)viewDidLoad {
11     [super viewDidLoad];
12
13     // 原始数据
14     NSString *originalString = @"这是一段将要使用'.der'文件加密的字符串!";
15
16     // 使用.der和.p12中的公钥私钥加密解密
17     NSString *public_key_path = [[NSBundle mainBundle] pathForResource:@"public_key.der"
18     NSString *private_key_path = [[NSBundle mainBundle] pathForResource:@"private_key.p"
19
20     NSString *encryptStr = [RSAEncryptor encryptString:originalString publicKeyWithConte
21     NSLog(@"加密前:%@", originalString);
22     NSLog(@"加密后:%@", encryptStr);
23     NSLog(@"解密后:%@", [RSAEncryptor decryptString:encryptStr privateKeyWithContentsOff
24
25 }
26
27 - (void)didReceiveMemoryWarning {
28     [super didReceiveMemoryWarning];
29     // Dispose of any resources that can be recreated.
30 }
31
32 @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/ELAcnQRSglei10PAwSL6z
NFr5S2kCtZW5tkzvFLnk/ZbiVMkXm+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RSAEncrypt方法加密, ViewController.m中代码如下:

```

1  #import "ViewController.h"
2  #import "RSAEncryptor.h"
3
4  @interface ViewController ()
5

```

```
6 | @end 7 |
8 | @implementation ViewController
9
10 - (void)viewDidLoad {
11     [super viewDidLoad];
12
13     // 原始数据
14     NSString *originalString = @"这是一段将要使用'秘钥字符串'进行加密的字符串!";
15
16     // 使用字符串格式的公钥私钥加密解密
17     NSString *encryptStr = [RSAEncryptor encryptString:originalString publicKey:@"MIGfMA0CAQYKQgY0hazAvRMYYx/
18
19     NSLog(@"加密前:%@", originalString);
20     NSLog(@"加密后:%@", encryptStr);
21     NSLog(@"解密后:%@", [RSAEncryptor decryptString:encryptStr privateKey:@"MIICeAIBADANI
22
23 }
24
25 - (void)didReceiveMemoryWarning {
26     [super didReceiveMemoryWarning];
27     // Dispose of any resources that can be recreated.
28 }
29
30 @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:hQ1TVLy20R00HNdRU9C3RwM4r4teesRc0UIMnc6Q0RxVH2FKgY0hazAvRMYYx/
mc1dphJRbsUUT6imYygt3w404Q1D9Udd1HtWANDs1y+Z1VYyG6gcqdzHvTBK+FttAigmkFSiW8GvjzrrdDZdSLAtw+qtmn117P1TdHBNN7BM=
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>

著作权归作者所有, 转载请联系作者获得授权, 并标注“简书作者”。

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?

我来说两句

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。

月薪40K+! 区块链以太坊开发八周学会

区块链DApp开发学习路线图, 月薪4万很轻松

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/43f7fc8d8e14>iOS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需

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....

记着冒死拍摄收藏市场！爆出惊人内情！

众慧商贸 · 熯熯

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 ...

博主推荐



Defonds

关注

449篇文章



小宝鸽

关注

97篇文章



orangleliu

关注

538

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？

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 - (NSData

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

“人喝茶三年，茶养人一辈子”已被科学证实！

三亿茶叶 · 熯熯

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 加密解密工具类

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



解密单片机

百度广告

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...

远离假传奇！这游戏爆率+999倍，有充值有VIP算我输！

贪玩游戏 · 顶新

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

为了方便自己使用，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 -*-..



ios开发怎么入门

百度广告

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

《开国大典》纪念币，限量发售，机遇难求，收官在即。

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 and 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...



人脸识别算法

百度广告

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1

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1

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 Shami

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 ...

记着冒死拍摄收藏市场！爆出惊人内情！

众慧商贸 · 熾熾

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,产生的字符串实际上是hex后和密钥串 你可以将他们当成静态字符串存在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



stc单片机解密

百度广告

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](#)