IOS SECURITY

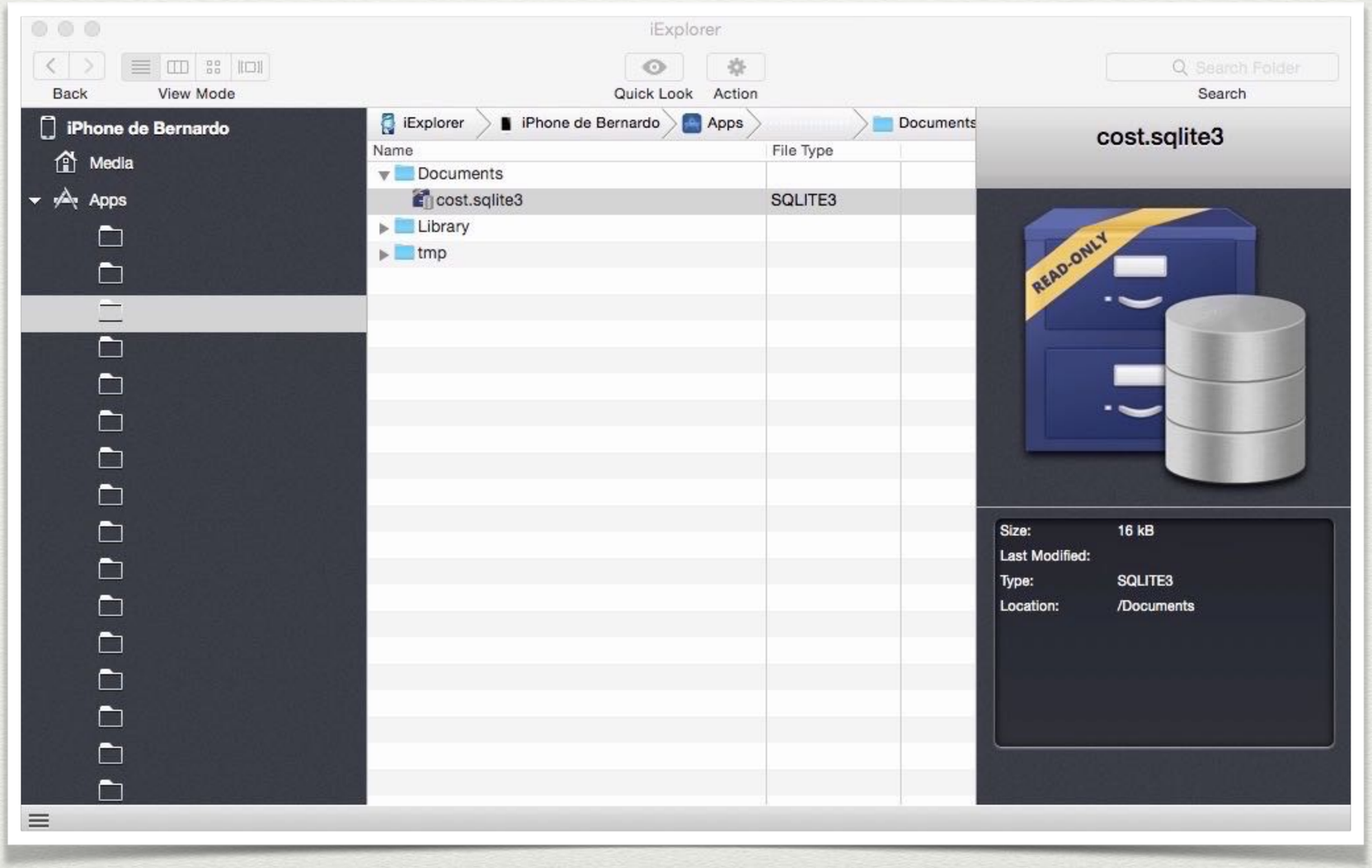
Bernardo Breder

Agenda

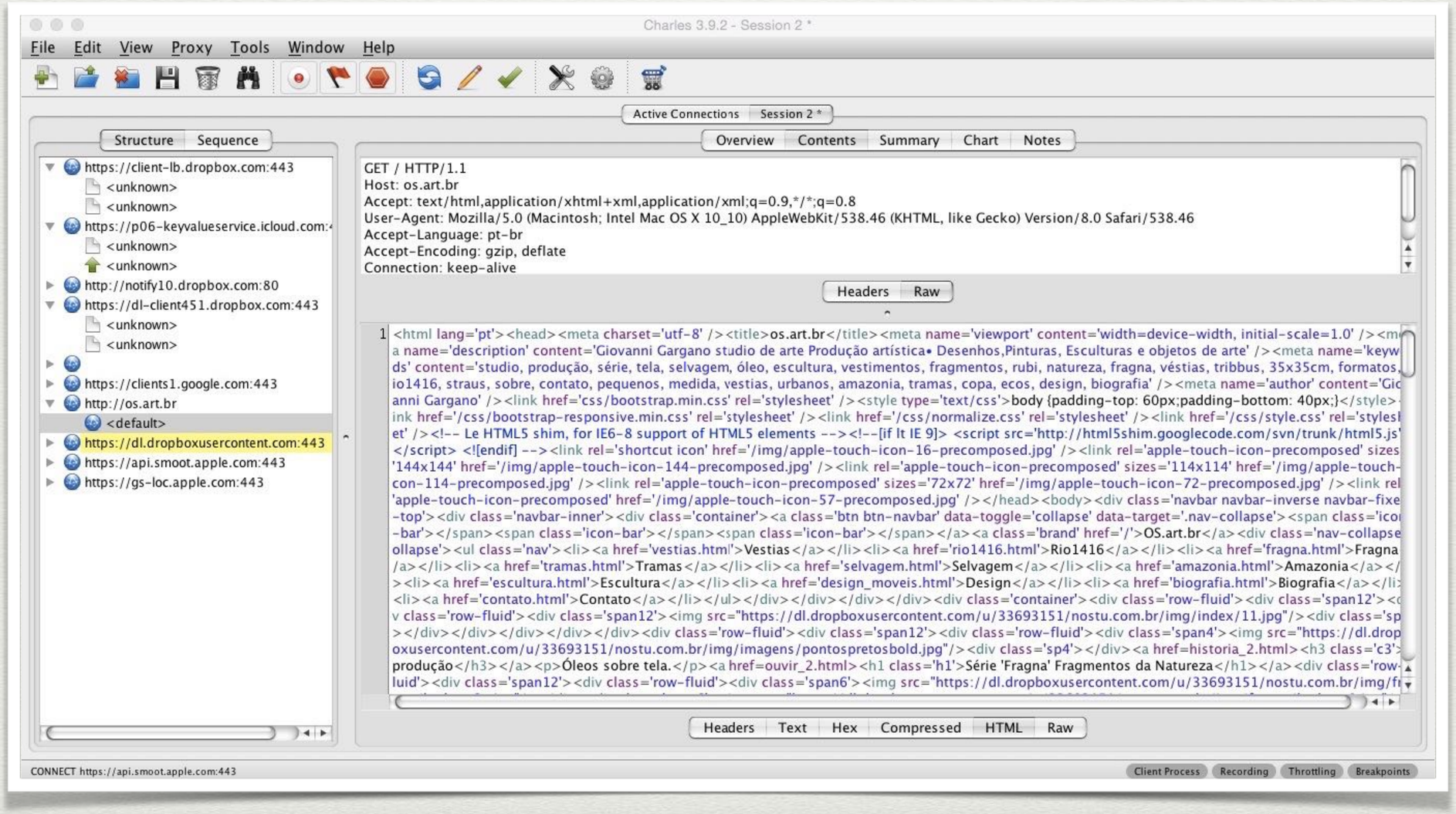
- Visão Geral
- Resumo de Mensagem
- Criptografia AES
- Criptografia RSA
- Keychain

VISÃO GERAL

Motivação


iExplorer
Back
View Mode
Quick Look
Action
Search Folder
Search
iPhone de Bernardo
Media
Apps
iExplorer
iPhone de Bernardo
Apps
Documents
Name
File Type
Documents
cost.sqlite3
SQLITE3
Library
tmp
cost.sqlite3
READ-ONLY
Size:
16 kB
Last Modified:
Type:
SQLITE3
Location:
/Documents

Motivação


Charles 3.9.2 - Session 2 *
File Edit View Proxy Tools Window Help
Active Connections Session 2 *
Structure Sequence
Overview Contents Summary Chart Notes
https://client-lb.dropbox.com:443
<unknown>
<unknown>
https://p06-keyvalueservice.icloud.com:
<unknown>
<unknown>
http://notify10.dropbox.com:80
https://dl-client451.dropbox.com:443
<unknown>
<unknown>
https://clients1.google.com:443
http://os.art.br
<default>
https://dl.dropboxusercontent.com:443
https://api.smoot.apple.com:443
https://gs-loc.apple.com:443
GET / HTTP/1.1
Host: os.art.br
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_10) AppleWebKit/538.46 (KHTML, like Gecko) Version/8.0 Safari/538.46
Accept-Language: pt-br
Accept-Encoding: gzip, deflate
Connection: keep-alive
Headers Raw
Headers Text Hex Compressed HTML Raw
CONNECT https://api.smoot.apple.com:443
Client Process Recording Throttling Breakpoints

Conceitos Básicos

- O que é Criptografia ?
- Como iOS lida com AES 256 (32 bytes)
- Como iOS lida com SHA 256 (32 bytes)
- O que é Device ID
- Jailbreak

Device ID

```
#import <Security/Security.h>
#import <CommonCrypto/CommonDigest.h>
#import <CommonCrypto/CommonCryptor.h>
#import <CommonCrypto/CommonHMAC.h>

+ (NSString*)deviceId
{
    return [[[UIDevice currentDevice] identifierForVendor] UUIDString];
}
```

Jailbreak

```
- (BOOL)checkJailbreak
{
    FILE *f = fopen("/bin/ssh", "r");
    if (f) {
        fclose(f);
        return true;
    }
    return false;
}
```

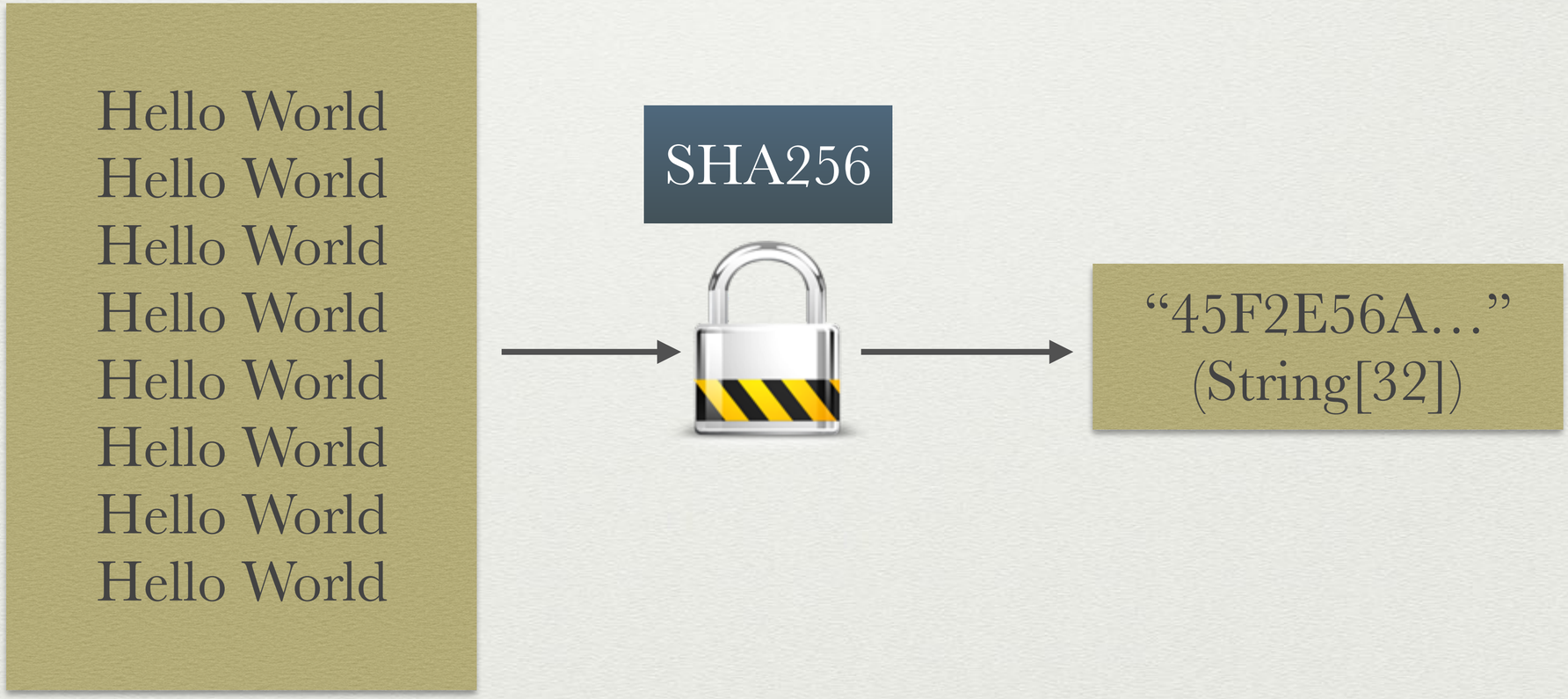
RESUMO DE MENSAGEM

MESSAGE DIGEST

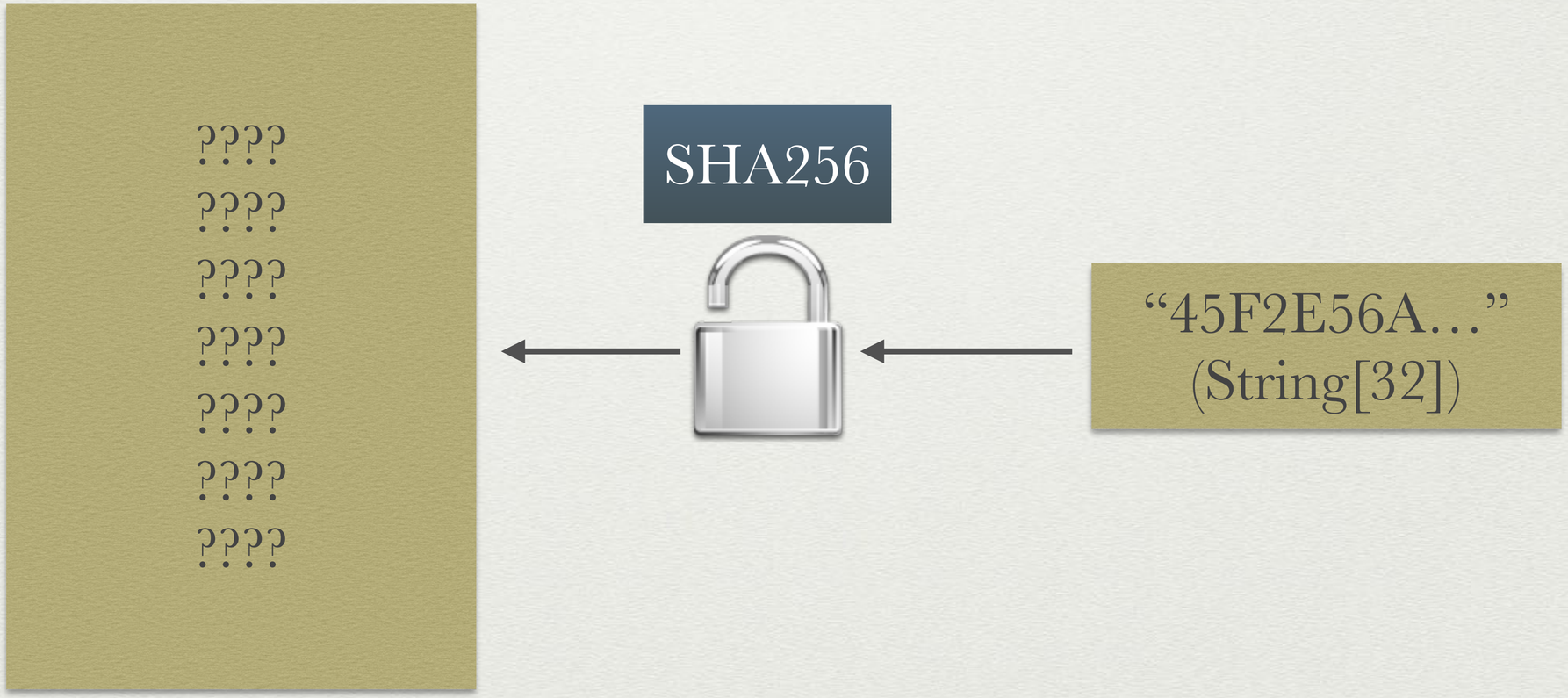
Resumo de Mensagem

- Transformar um conteúdo grande ou pequeno em uma String de 32 caracteres
- Objetivo é converte um conteúdo para um resumo
- Conversão é uni-direcional
- Operação rápida

Resumo de Mensagem



Resumo de Mensagem



SHA 256

```
#import <Security/Security.h>
#import <CommonCrypto/CommonDigest.h>
#import <CommonCrypto/CommonCryptor.h>
#import <CommonCrypto/CommonHMAC.h>

- (NSData *) SHA256Hash
{
    unsigned char hash[CC_SHA256_DIGEST_LENGTH];
    (void) CC_SHA256( [self bytes], (CC_LONG)[self length], hash );
    return ( [NSData dataWithBytes: hash length: CC_SHA256_DIGEST_LENGTH] );
}
```

MD5

```
#import <Security/Security.h>
#import <CommonCrypto/CommonDigest.h>
#import <CommonCrypto/CommonCryptor.h>
#import <CommonCrypto/CommonHMAC.h>

- (NSData *) MD5Sum
{
    unsigned char hash[CC_MD5_DIGEST_LENGTH];
    (void) CC_MD5( [self bytes], (CC_LONG)[self length], hash );
    return ( [NSData dataWithBytes: hash length: CC_MD5_DIGEST_LENGTH] );
}
```

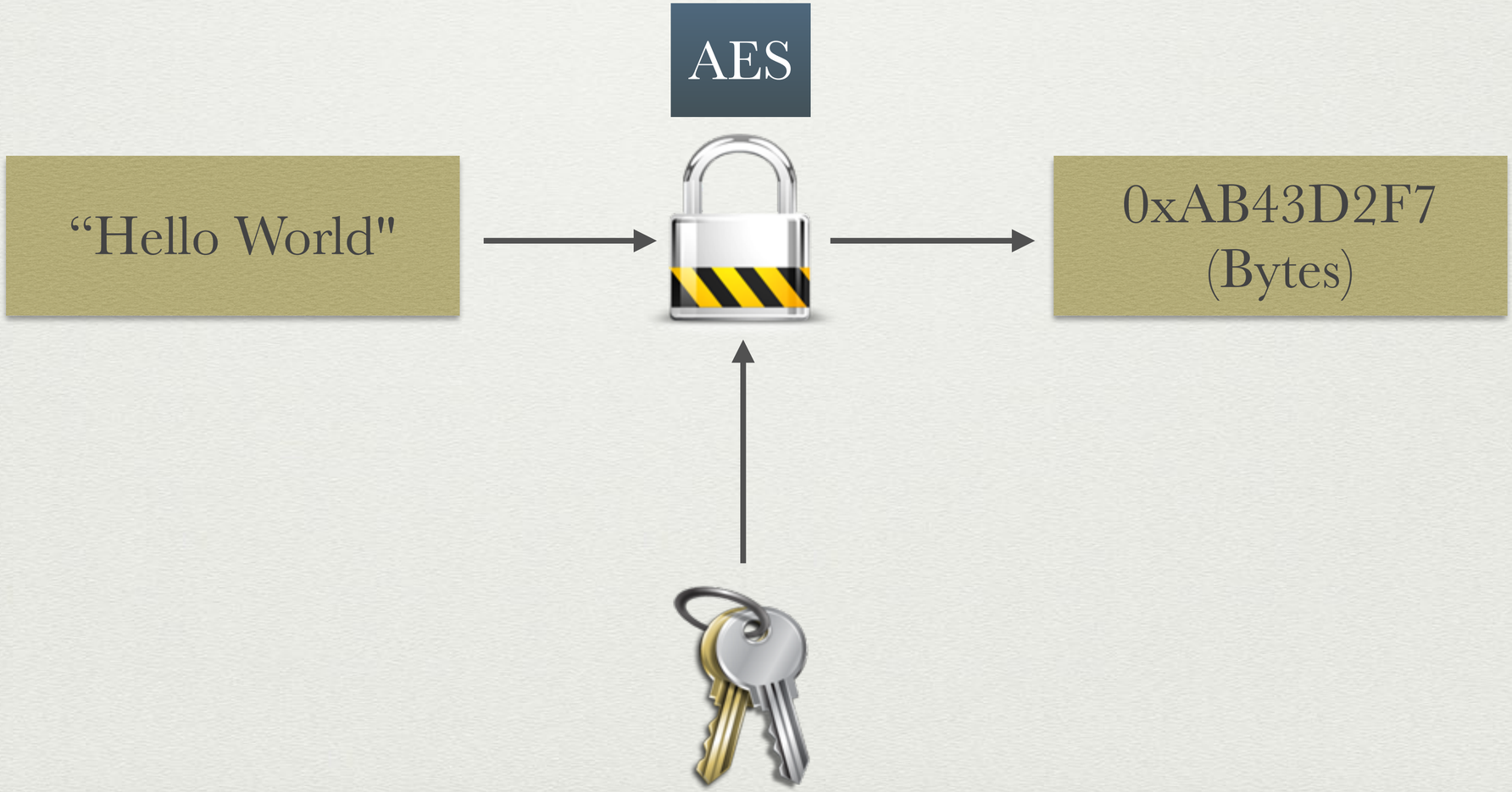
CRIPTOGRAFIA AES

AES CRYPTOGRAPHY

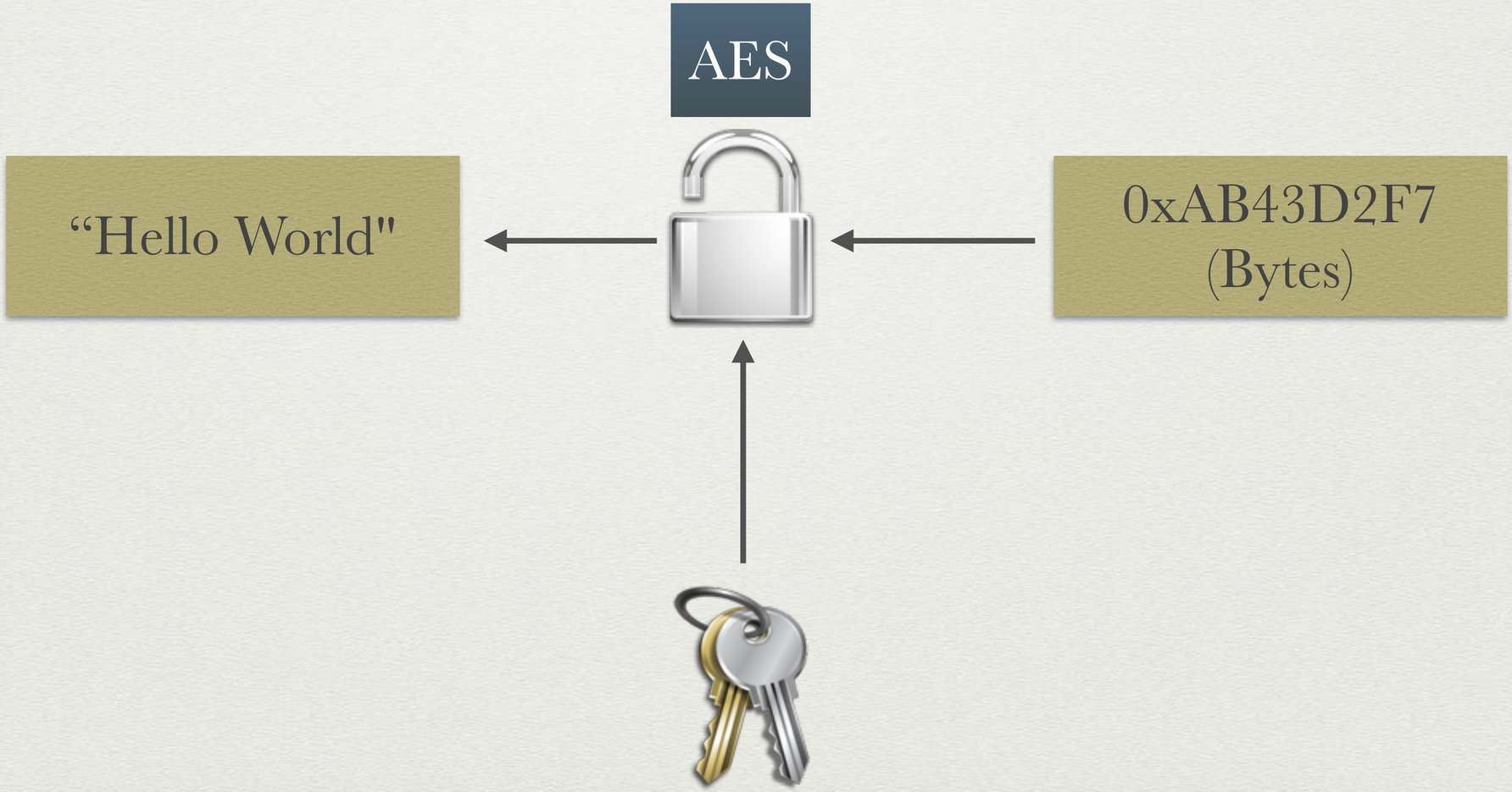
Criptografia AES

- Transformar um conteúdo grande ou pequeno em um conteúdo criptografado
- É necessário uma chave para converte o conteúdo
- É necessário da mesma chave para voltar ao conteúdo original
- Conversão é bi-direcional se a chave for a mesma
- Operação rápida

Chave Simétrica



Chave Simétrica



Criptografar

```
- (NSData*)dataEncrypted:(NSMutableData*)keyData iv:(NSMutableData*)ivData
{
    if (keyData.length <= 16) [keyData setLength: 16];
    else if (keyData.length < 24) [keyData setLength: 24];
    else [keyData setLength: 32];
    [ivData setLength: [keyData length]];
    CCCryptorRef cryptor;
    if ( CCCryptorCreate( kCCEncrypt, kCCAlgorithmAES128, kCCOptionPKCS7Padding, [keyData bytes], [keyData length]
        , [ivData bytes], &cryptor) != kCCSuccess ) return nil;
    size_t bufsize = CCCryptorGetOutputLength(cryptor, (size_t)self.length, true);
    void * buf = malloc(bufsize);
    size_t bufused = 0;
    size_t bytesTotal = 0;
    if (CCCryptorUpdate( cryptor, [self bytes], (size_t)[self length], buf, bufsize, &bufused) != kCCSuccess) {
        free(buf);
        return nil;
    }
    bytesTotal += bufused;
    if (CCCryptorFinal(cryptor, buf + bufused, bufsize - bufused, &bufused) != kCCSuccess) {
        free(buf);
        return ( nil );
    }
    bytesTotal += bufused;
    CCCryptorRelease(cryptor);
    return [NSData dataWithBytesNoCopy:buf length: bytesTotal];
}
```

Decriptografar

```
- (NSData*)dataDecrypted:(NSMutableData*)keyData iv:(NSMutableData*)ivData
{
    if (keyData.length <= 16) [keyData setLength: 16];
    else if (keyData.length < 24) [keyData setLength: 24];
    else [keyData setLength: 32];
    [ivData setLength: [keyData length]];
    CCCryptorRef cryptor;
    if ( CCCryptorCreate( kCCDecrypt, kCCAlgorithmAES128, kCCOptionPKCS7Padding, [keyData bytes], [keyData length]
        , [ivData bytes], &cryptor) != kCCSuccess ) return nil;
    size_t bufsize = CCCryptorGetOutputLength(cryptor, (size_t)self.length, true);
    void * buf = malloc(bufsize);
    size_t bufused = 0;
    size_t bytesTotal = 0;
    if (CCCryptorUpdate(cryptor, [self bytes], (size_t)[self length], buf, bufsize, &bufused) != kCCSuccess) {
        free(buf);
        CCCryptorRelease(cryptor);
        return nil;
    }
    bytesTotal += bufused;
    if (CCCryptorFinal(cryptor, buf + bufused, bufsize - bufused, &bufused) != kCCSuccess) {
        free(buf);
        CCCryptorRelease(cryptor);
        return nil;
    }
    bytesTotal += bufused;
    CCCryptorRelease(cryptor);
    return [NSData dataWithBytesNoCopy:buf length: bytesTotal];
}
```

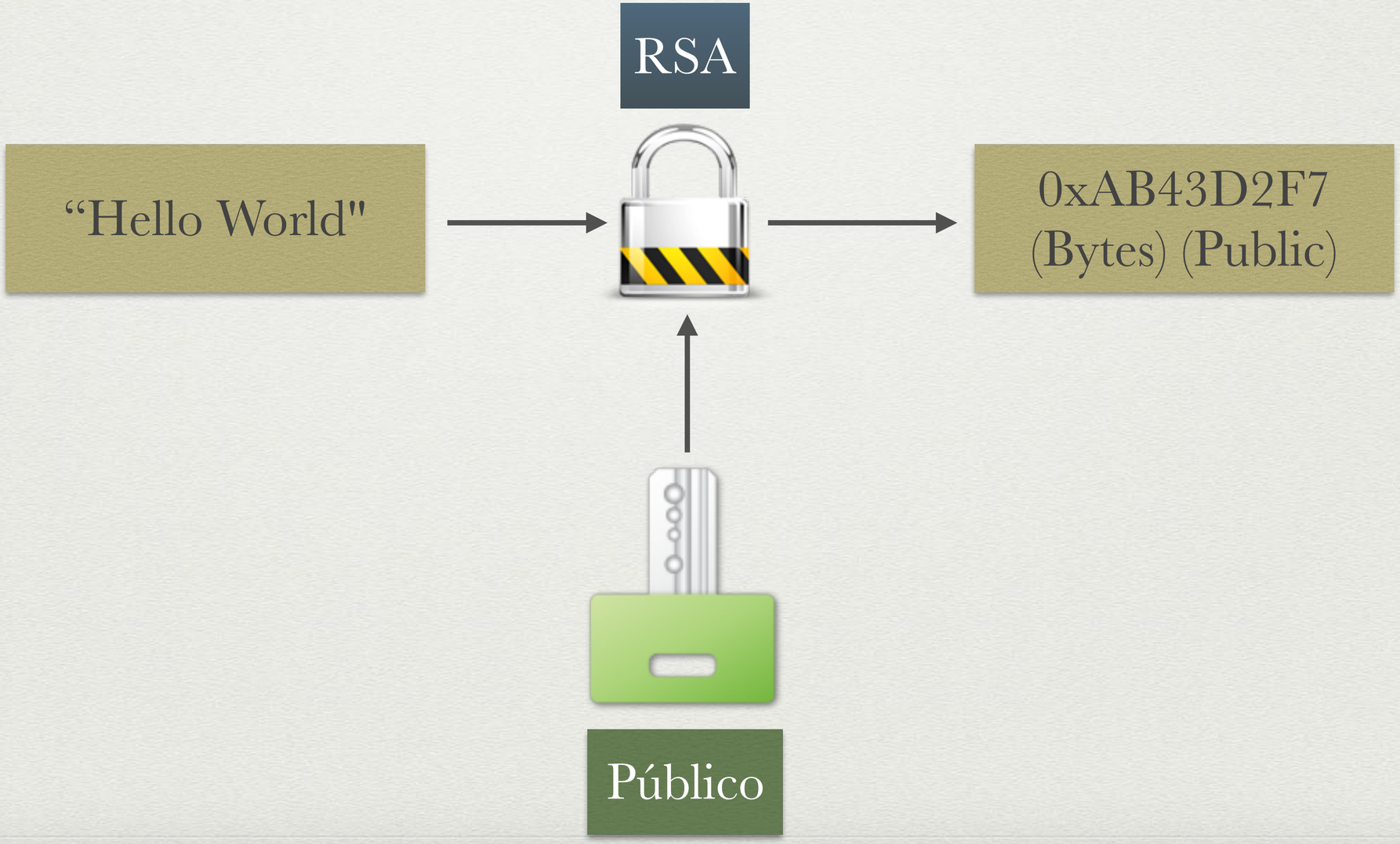
CRIPTOGRAFIA RSA

RSA CRYPTOGRAPHY

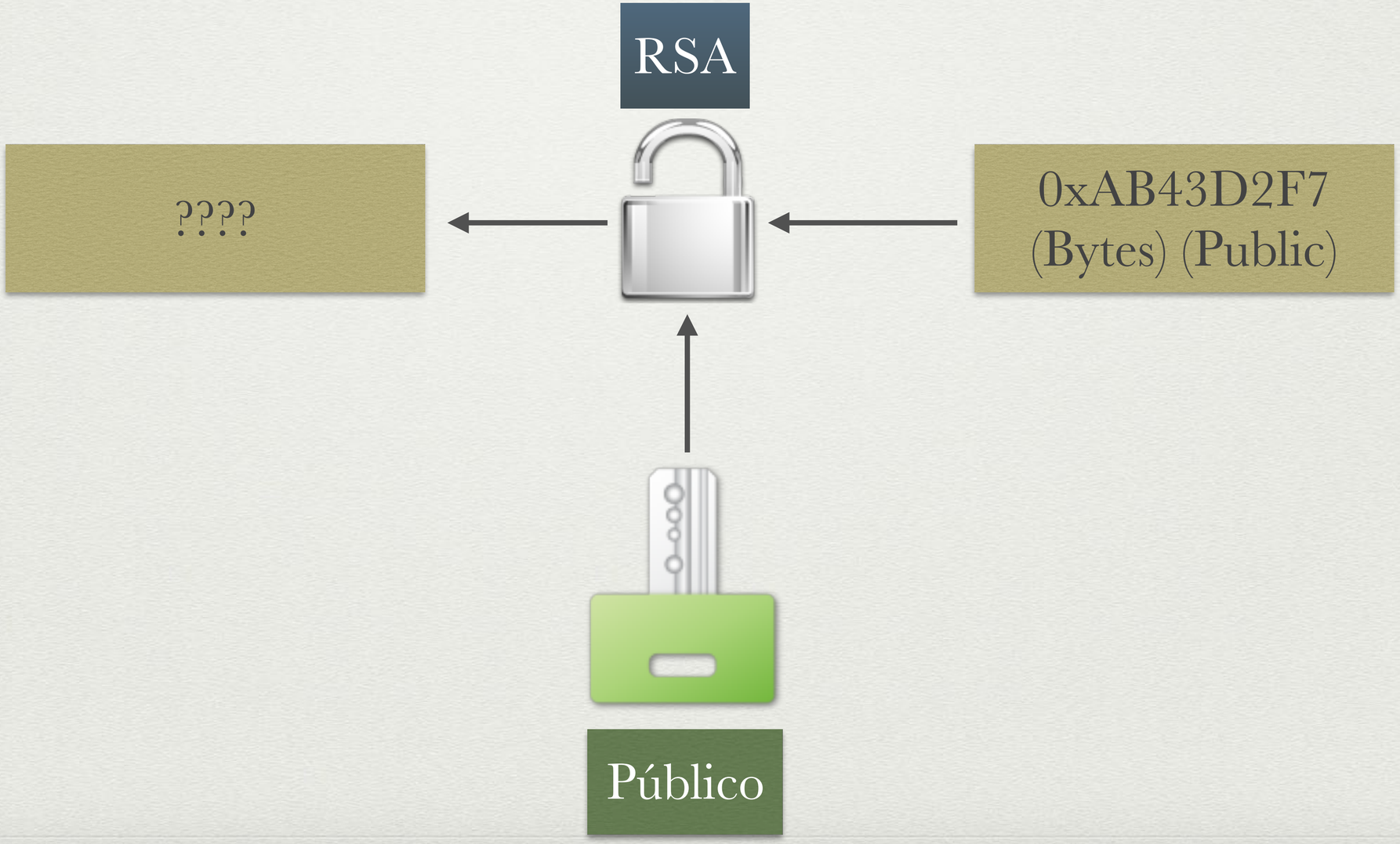
Criptografia RSA

- Transformar um conteúdo grande ou pequeno em um conteúdo criptografado
- Chave publica é usado para criptar a mensagem
- Chave privada é usada para decriptar o conteúdo criptado
- Conversão é bi-direcional se o par chave pública e privada forem compatíveis
- Operação lenta

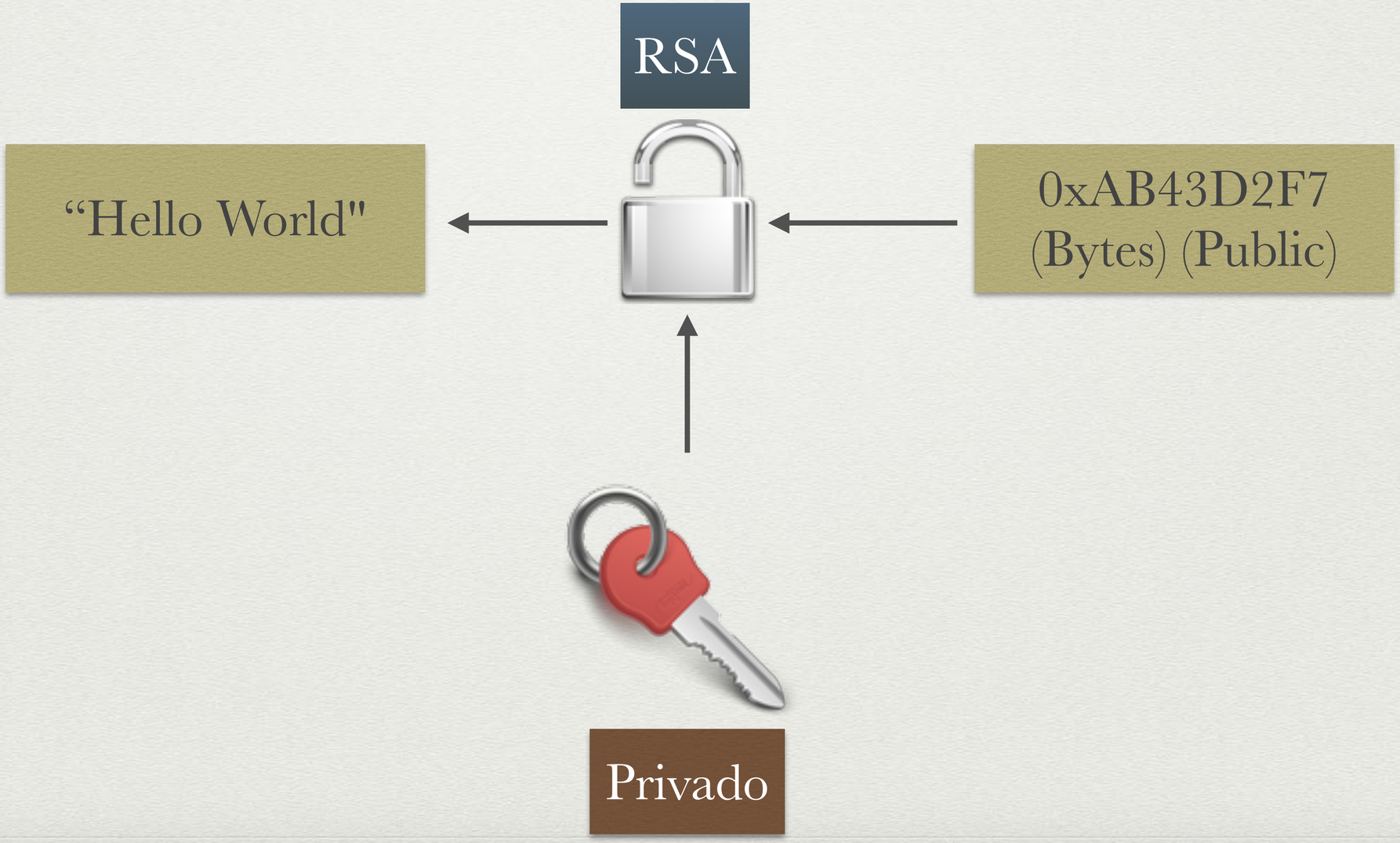
Criptografia RSA



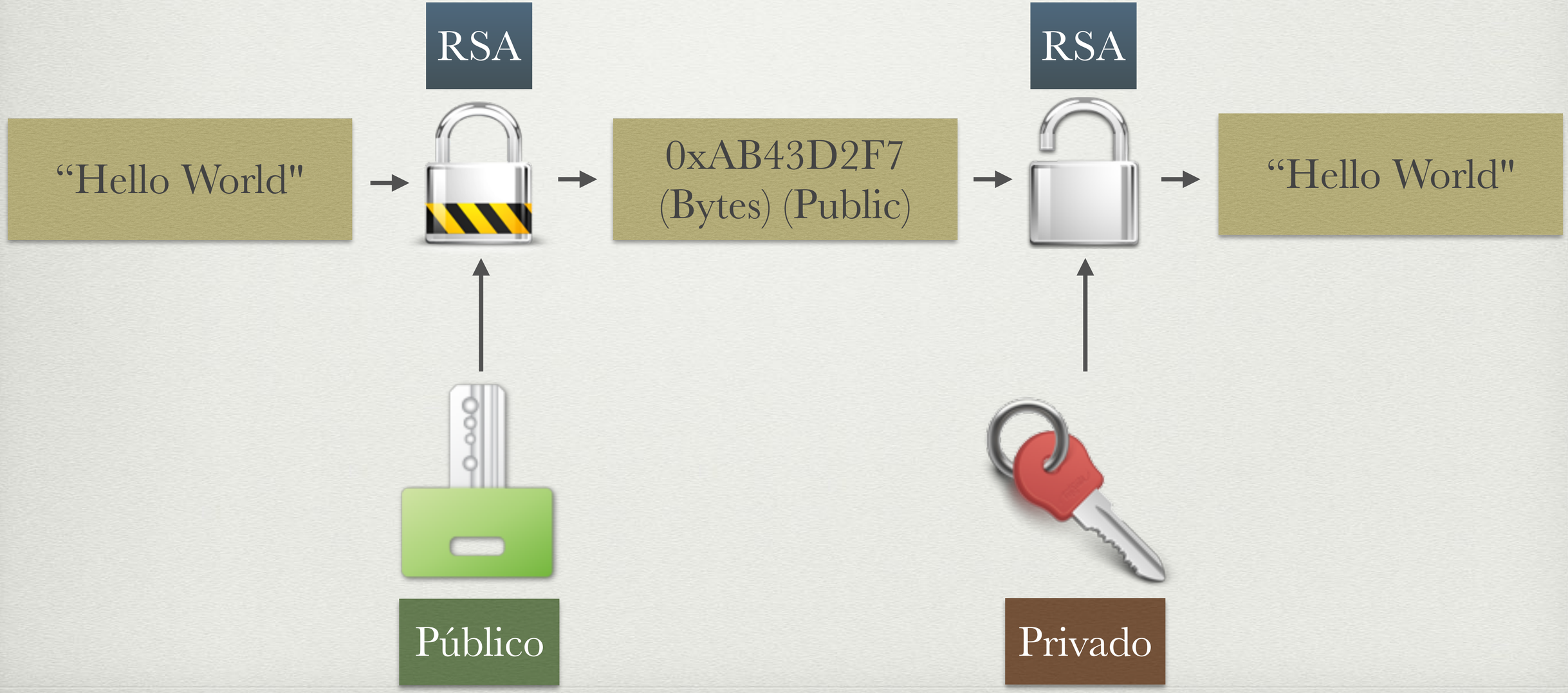
Criptografia RSA



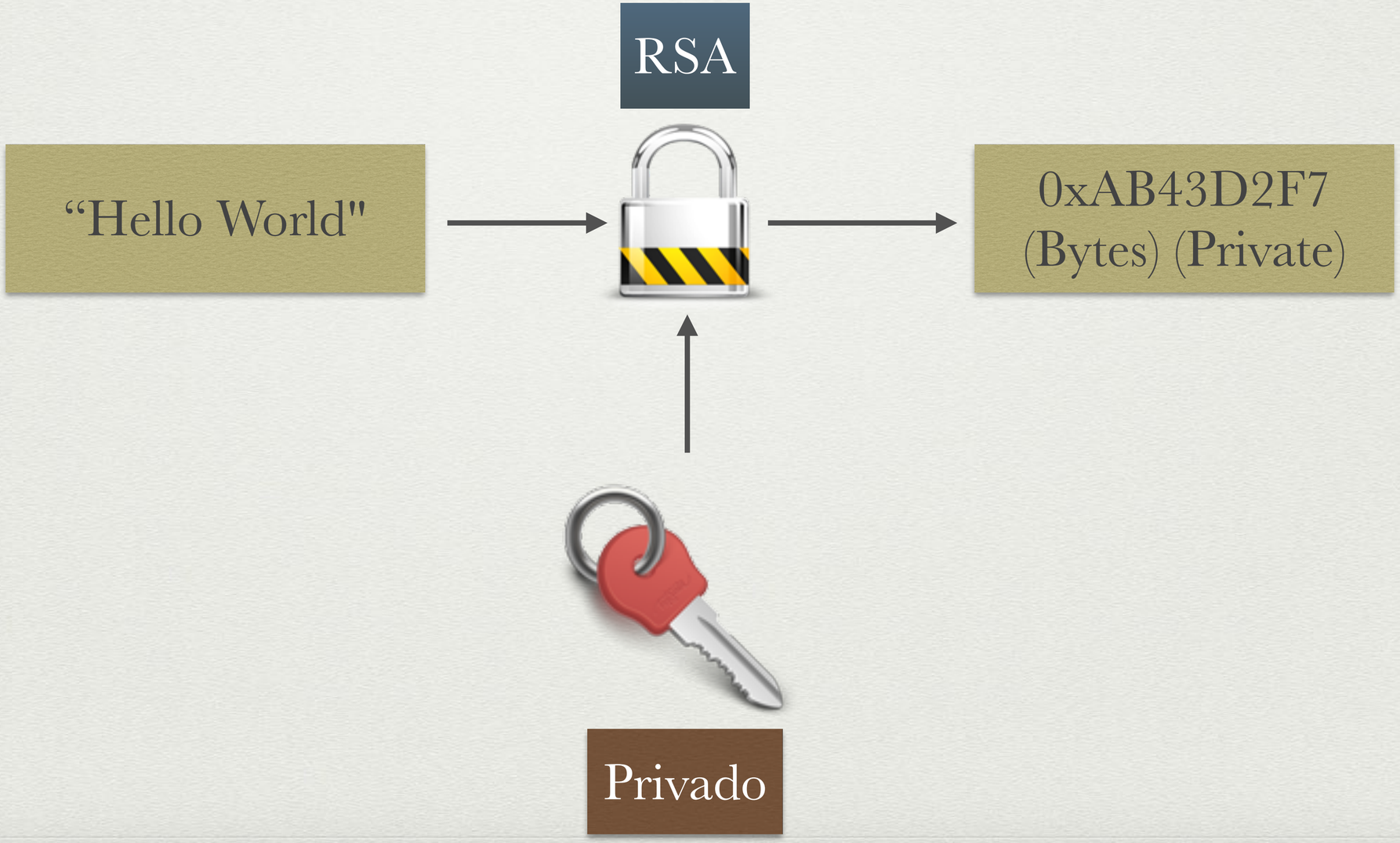
Criptografia RSA



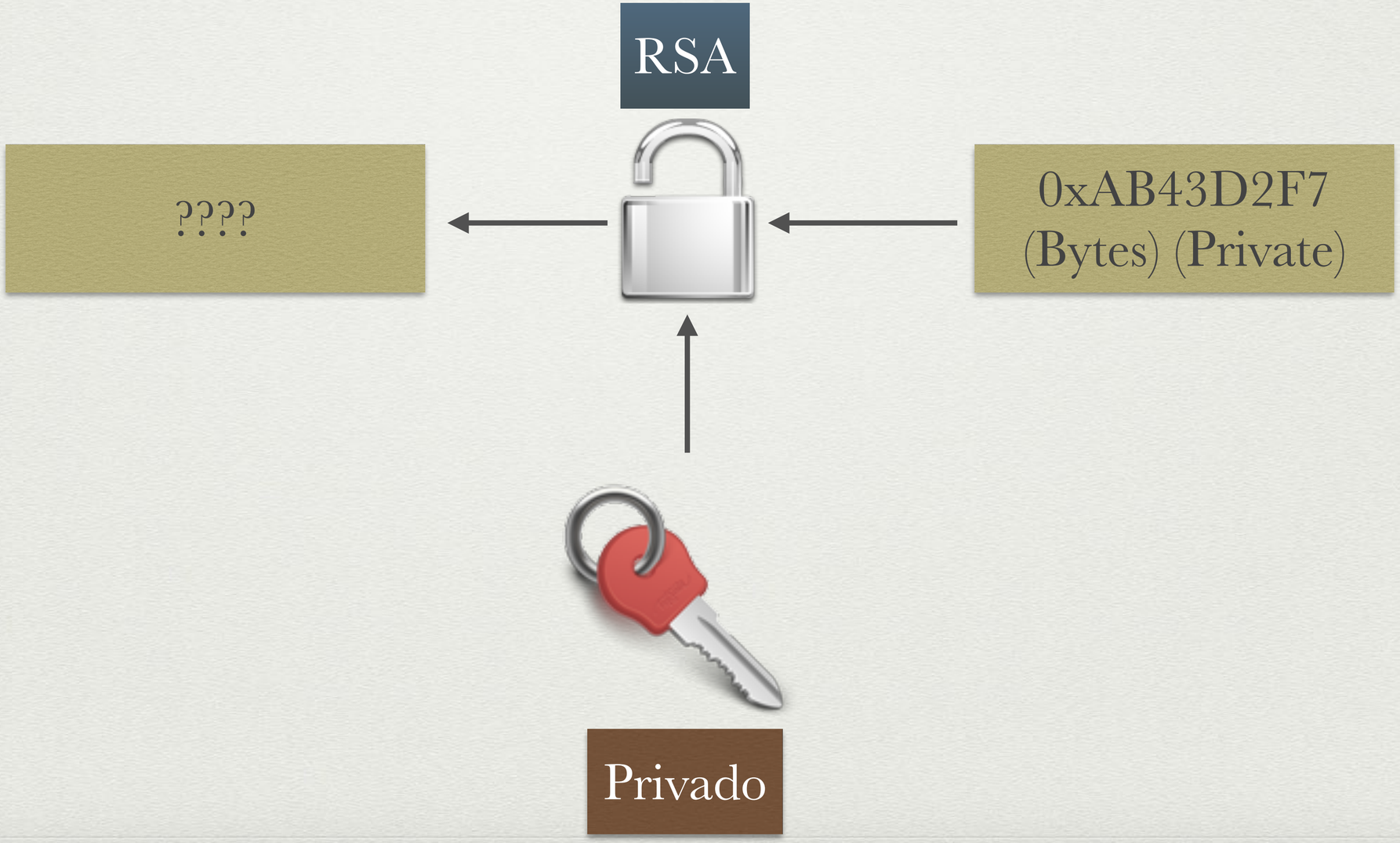
Criptografia RSA



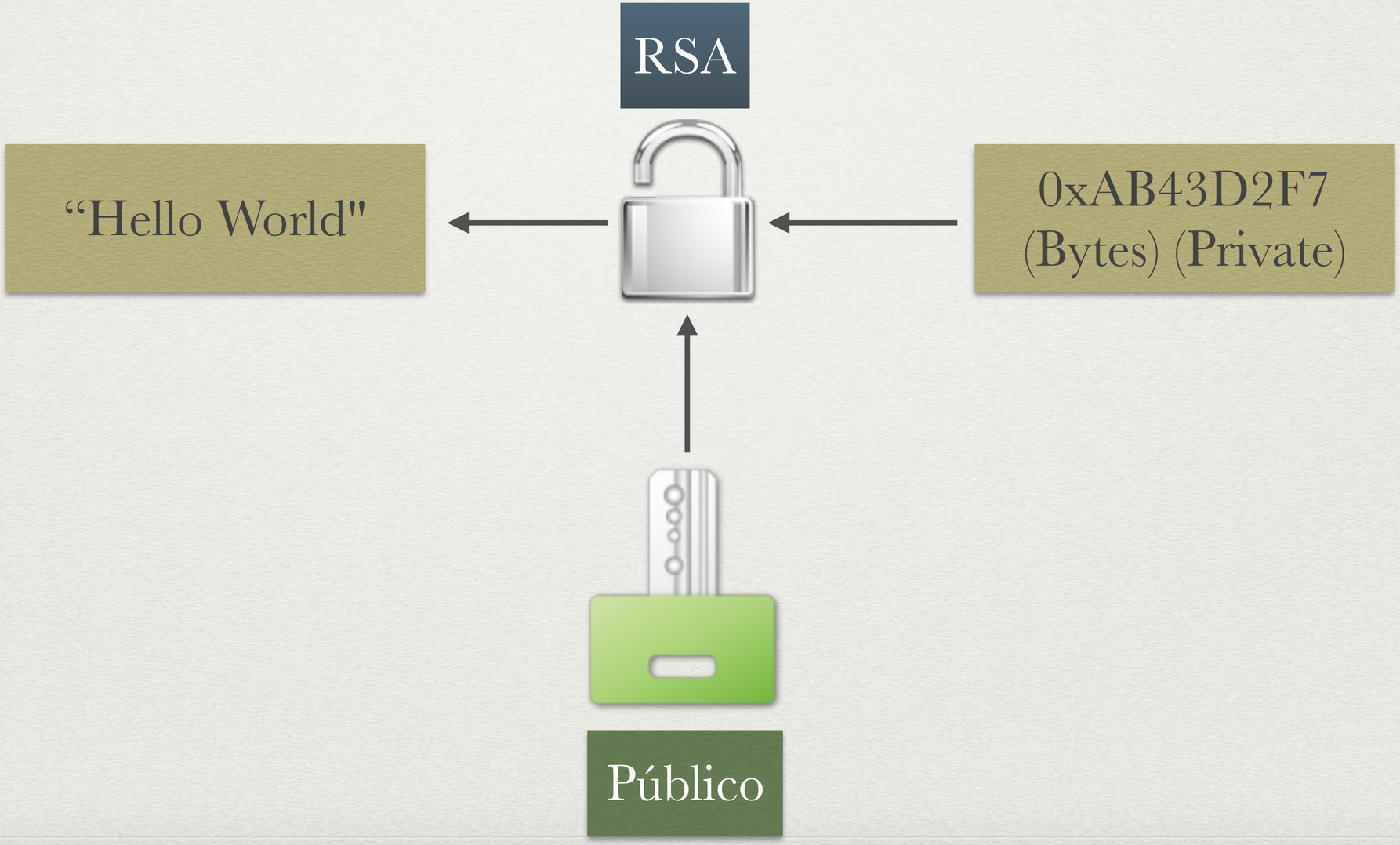
Criptografia RSA



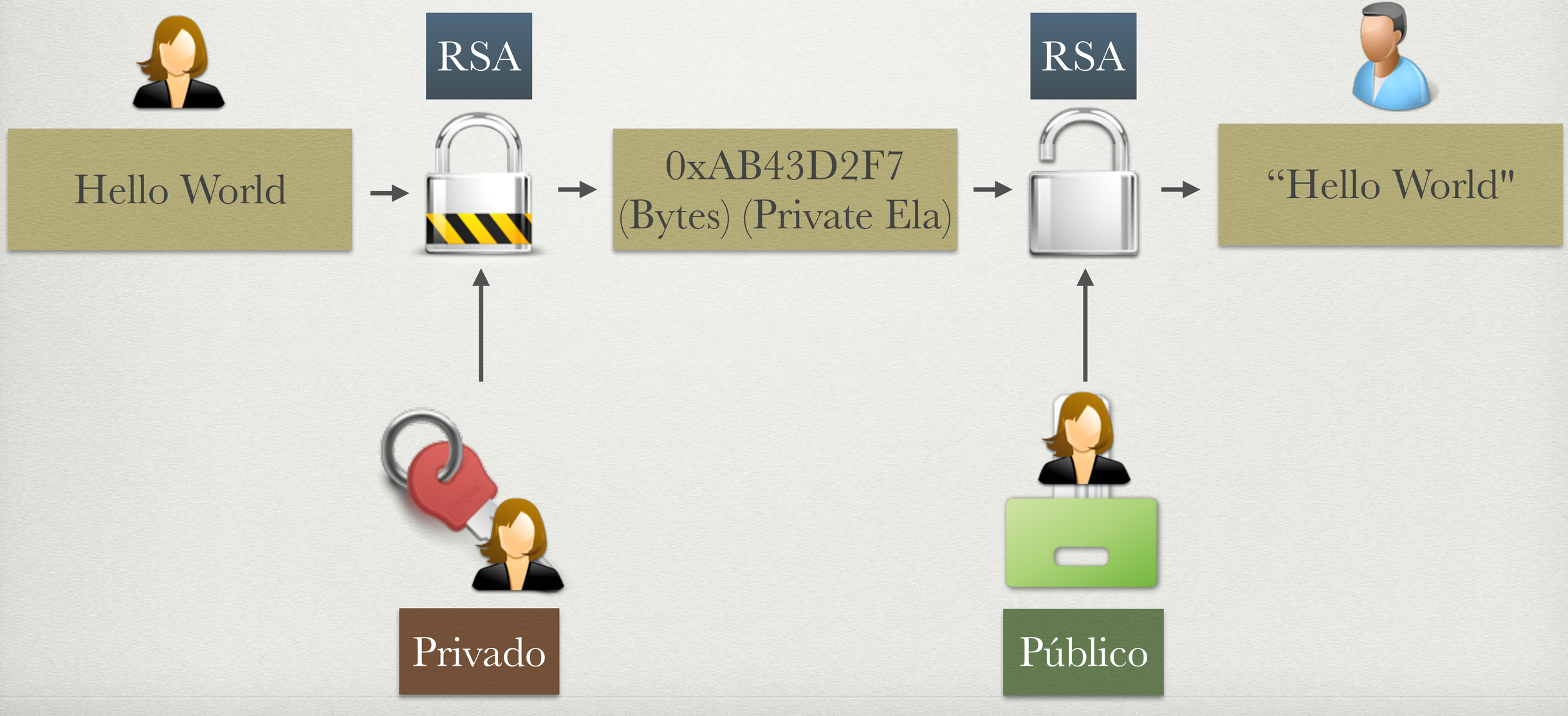
Criptografia RSA



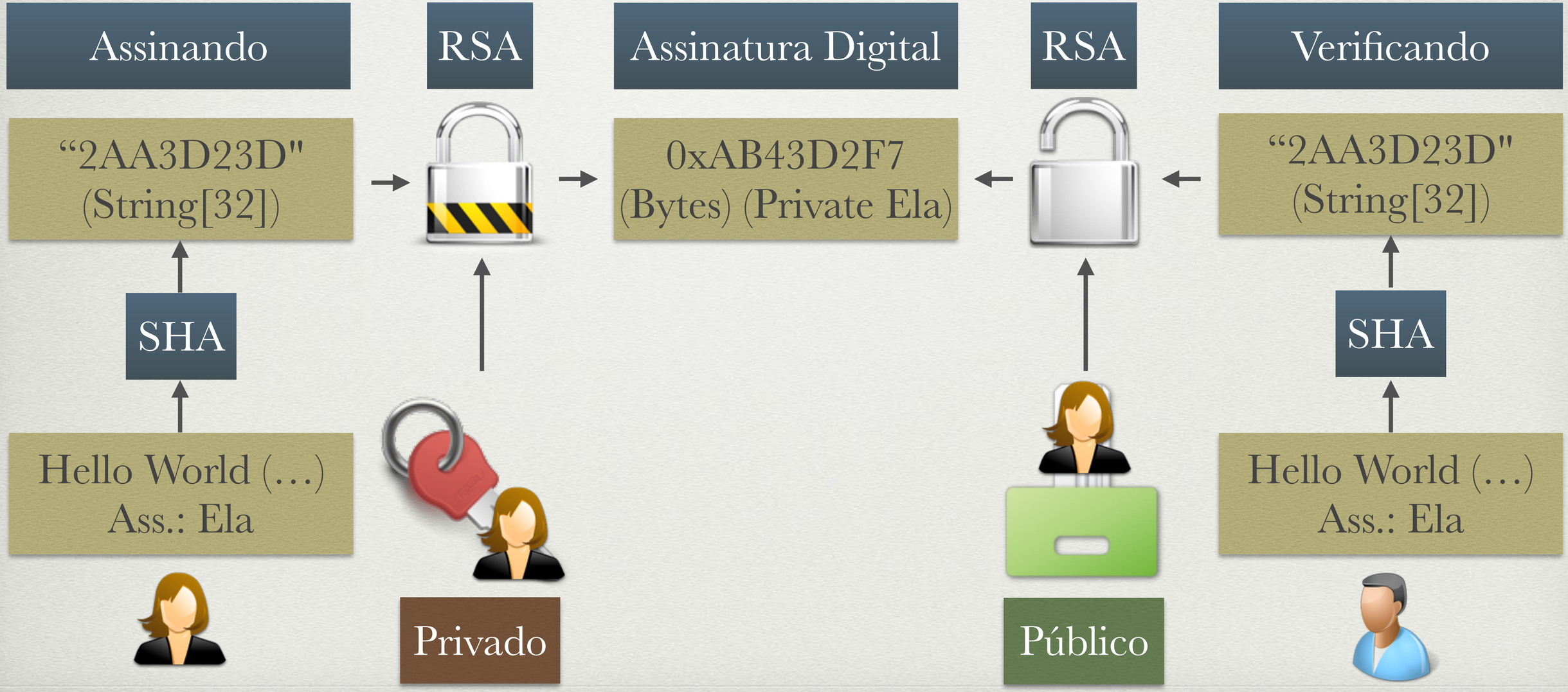
Criptografia RSA



Criptografia RSA - Troca de Mensagem



Criptografia RSA - Assinatura Digital



Gerador de Par de Chaves

```
+ (NSRSAKeyPair*)generateKeyPairWithPublicKey:(NSData*)publicKey withPrivateKey:(NSData*)privateKey
{
    NSRSAKey *publicKeyRef = [[NSRSAKey alloc] init];
    NSRSAKey *privateKeyRef = [[NSRSAKey alloc] init];
    NSMutableDictionary * privateKeyAttr = [NSMutableDictionary dictionaryWithCapacity:0];
    NSMutableDictionary * publicKeyAttr = [NSMutableDictionary dictionaryWithCapacity:0];
    NSMutableDictionary * keyPairAttr = [NSMutableDictionary dictionaryWithCapacity:0];
    [keyPairAttr setObject:(__bridge id)kSecAttrKeyTypeRSA forKey:(__bridge id)kSecAttrKeyType];
    [keyPairAttr setObject:[NSNumber numberWithUnsignedInteger:kSecAttrKeySizeInBitsLength] forKey:(__bridge id)kSecAttrKeySizeInBits];
    [privateKeyAttr setObject:[NSNumber numberWithBool:YES] forKey:(__bridge id)kSecAttrIsPermanent];
    [privateKeyAttr setObject:privateKey forKey:(__bridge id)kSecAttrApplicationTag];
    [publicKeyAttr setObject:[NSNumber numberWithBool:YES] forKey:(__bridge id)kSecAttrIsPermanent];
    [publicKeyAttr setObject:publicKey forKey:(__bridge id)kSecAttrApplicationTag];
    [keyPairAttr setObject:privateKeyAttr forKey:(__bridge id)kSecPrivateKeyAttrs];
    [keyPairAttr setObject:publicKeyAttr forKey:(__bridge id)kSecPublicKeyAttrs];
    SecKeyRef _publicKeyRef = publicKeyRef.keyRef;
    SecKeyRef _privateKeyRef = privateKeyRef.keyRef;
    if (SecKeyGeneratePair((__bridge CFDictionaryRef)keyPairAttr, &_publicKeyRef, &_privateKeyRef) != noErr) return nil;
    publicKeyRef.keyRef = _publicKeyRef;
    privateKeyRef.keyRef = _privateKeyRef;
    NSRSAKeyPair *keyPair = [[NSRSAKeyPair alloc] init];
    keyPair.publicKey = publicKeyRef;
    keyPair.privateKey = privateKeyRef;
    return keyPair;
}
```

Criptografar com RSA

```
+ (NSData*)encryptData:(NSData*)plainData withKey:(NSRSAKey*)key
{
    if (!plainData || !key) return nil;
    size_t cipherBufferSize = SecKeyGetBlockSize(key.keyRef);
    uint8_t *cipherBuffer = malloc(cipherBufferSize * sizeof(uint8_t));
    memset((void *)cipherBuffer, 0*0, cipherBufferSize);
    NSData *plainTextBytes = plainData;
    size_t blockSize = cipherBufferSize - 11;
    size_t blockCount = (size_t)ceil([plainTextBytes length] / (double)blockSize);
    NSMutableData *encryptedData = [NSMutableData dataWithCapacity:0];
    for (NSUInteger i = 0 ; i < blockCount ; i++) {
        NSUInteger bufferSize = MIN(blockSize,[plainTextBytes length] - i * blockSize);
        NSData *buffer = [plainTextBytes subdataWithRange:NSMakeRange(i * blockSize, bufferSize)];
        if (SecKeyEncrypt(key.keyRef, kSecPaddingPKCS1, (const uint8_t *)[buffer bytes], [buffer length],
            cipherBuffer, &cipherBufferSize) != noErr){
            if (cipherBuffer) free(cipherBuffer);
            return nil;
        }
        NSData *encryptedBytes = [NSData dataWithBytes:(const void *)cipherBuffer length:cipherBufferSize];
        [encryptedData appendData:encryptedBytes];
    }
    if (cipherBuffer) free(cipherBuffer);
    return encryptedData;
}
```

Decriptografar com RSA

```
+ (NSData*)decryptData:(NSData*)encryptData withKey:(NSRSAKey*)key
{
    if (!encryptData || !key) return nil;
    NSData *wrappedSymmetricKey = encryptData;
    size_t cipherBufferSize = SecKeyGetBlockSize(key.keyRef);
    size_t keyBufferSize = [wrappedSymmetricKey length];
    NSMutableData *bits = [NSMutableData dataWithLength:keyBufferSize];
    if (SecKeyDecrypt(key.keyRef, kSecPaddingPKCS1, (const uint8_t *) [wrappedSymmetricKey bytes],
        cipherBufferSize, [bits mutableBytes], &keyBufferSize) != noErr) {
        return nil;
    }
    [bits setLength:keyBufferSize];
    return bits;
}
```

KEYCHAIN

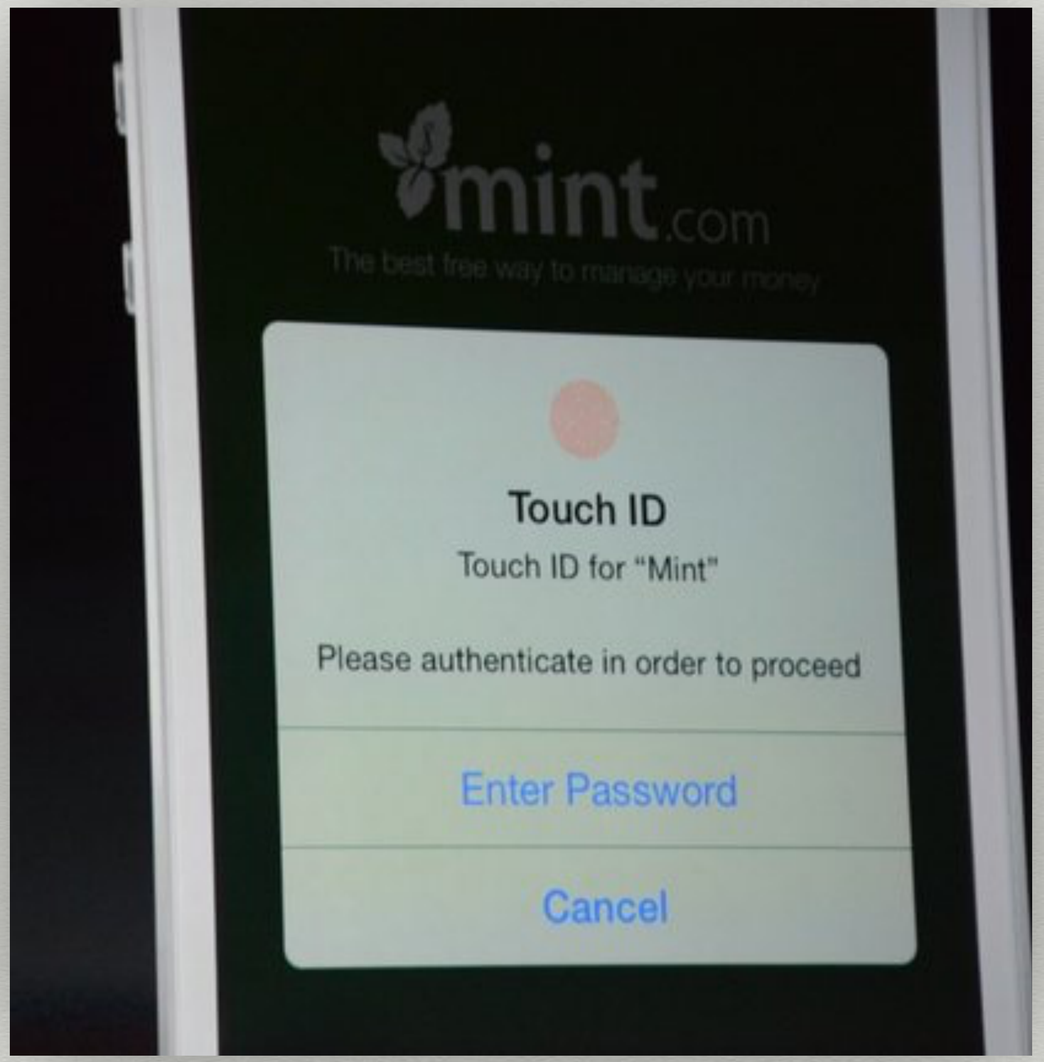
Keychain

- Local para armazenar Chaves
- Protegido contra acesso
- Somente pode ser lido quando estiver desbloqueado

Keychain

```
static NSString *serviceName = @"com.mycompany.myAppServiceName";

+ (NSMutableDictionary *)newSearchDictionary:(NSString *)identifier {
    NSMutableDictionary *searchDictionary = [[NSMutableDictionary alloc] init];
    [searchDictionary setObject:(__bridge id)kSecClassGenericPassword forKey:(__bridge id)kSecClass];
    NSData *encodedIdentifier = [identifier dataUsingEncoding:NSUTF8StringEncoding];
    [searchDictionary setObject:encodedIdentifier forKey:(__bridge id)kSecAttrGeneric];
    [searchDictionary setObject:encodedIdentifier forKey:(__bridge id)kSecAttrAccount];
    [searchDictionary setObject:serviceName forKey:(__bridge id)kSecAttrService];
    return searchDictionary;
}

+ (NSData*)searchKeychainCopyMatching:(NSString *)identifier {
    NSMutableDictionary *searchDictionary = [self newSearchDictionary:identifier];
    [searchDictionary setObject:(__bridge id)kSecMatchLimitOne forKey:(__bridge id)kSecMatchLimit];
    [searchDictionary setObject:(id)kCFBooleanTrue forKey:(__bridge id)kSecReturnData];
    CFDataRef result;
    if (SecItemCopyMatching((__bridge CFDictionaryRef)searchDictionary, (CFTypeRef *)&result) != noErr) return nil
        ;
    return (__bridge_transfer NSData*)result;
}

+ (BOOL)createKeychainValue:(NSData*)data forIdentifier:(NSString *)identifier {
    NSMutableDictionary *dictionary = [self newSearchDictionary:identifier];
    [dictionary setObject:data forKey:(__bridge id)kSecValueData];
    OSStatus status = SecItemAdd((__bridge CFDictionaryRef)dictionary, NULL);
    return status == errSecSuccess;
}

+ (BOOL)updateKeychainValue:(NSData*)data forIdentifier:(NSString *)identifier {
    NSMutableDictionary *searchDictionary = [self newSearchDictionary:identifier];
    NSMutableDictionary *updateDictionary = [[NSMutableDictionary alloc] init];
    [updateDictionary setObject:data forKey:(__bridge id)kSecValueData];
    OSStatus status = SecItemUpdate((__bridge CFDictionaryRef)searchDictionary, (__bridge CFDictionaryRef)
        updateDictionary);
    return status == errSecSuccess;
}

+ (void)deleteKeychainValue:(NSString *)identifier {
    NSMutableDictionary *searchDictionary = [self newSearchDictionary:identifier];
    SecItemDelete((__bridge CFDictionaryRef)searchDictionary);
}
```

Keychain


mint.com
The best free way to manage your money
Touch ID
Touch ID for "Mint"
Please authenticate in order to proceed
Enter Password
Cancel

OBRIGADO

BERNARDOBREDER@GMAIL.COM