



Android Cryptography and Network Security

Lecture 5

Security of Mobile Devices

2020



SMD

Cryptographic Providers

SSL/TLS

JSSE

Android JSSE Providers

Bibliography

Cryptographic Providers

SSL/TLS

JSSE

Android JSSE Providers

Bibliography

- ▶ Java Cryptography Architecture (JCA)
 - ▶ Extensible cryptographic provider framework
 - ▶ Set of APIs - major cryptographic primitives
 - ▶ Applications specify an algorithm, do not depend on particular provider implementation
- ▶ Cryptographic Service Provider (CSP)
 - ▶ Package with implementation of cryptographic services
 - ▶ Advertises the implemented services and algorithms
 - ▶ JCA maintains a registry of providers and their algorithms
 - ▶ Providers in a order of preference
- ▶ Service Provider Interface (SPI)
 - ▶ Common interface for implementations of a specific algorithm
 - ▶ Abstract class implemented by provider

- ▶ JCA engines provide:
 - ▶ Cryptographic operations (encrypt/decrypt, sign/verify, hash)
 - ▶ Generation or conversion of cryptographic material (keys, parameters)
 - ▶ Management and storage of cryptographic objects (keys, certificates)
- ▶ Decouple client code from algorithm implementation
- ▶ Static factory method `getInstance()`
- ▶ Request implementation indirectly

```
static EngineClassName getInstance(String algorithm)
    throws NoSuchAlgorithmException
static EngineClassName getInstance(String algorithm, String provider)
    throws NoSuchAlgorithmException, NoSuchProviderException
static EngineClassName getInstance(String algorithm, Provider provider)
    throws NoSuchAlgorithmException
```

- ▶ Hash function

```
MessageDigest md = MessageDigest.getInstance("SHA-256");  
byte [] data = getMessage();  
byte [] hash = md.digest(data);
```

- ▶ Data provided in chunks using `update()` then call `digest()`
- ▶ If data is short and fixed - hashed in one step using `digest()`

- ▶ Digital signature algorithms based on asymmetric encryption
- ▶ Algorithm name: <digest>with<encryption>
- ▶ Sign:

```
byte [] data = "message to be signed".getBytes(" ASCII" );  
  
Signature s = Signature.getInstance(" SHA256withRSA" );  
s.initSign(privKey);  
s.update(data);  
byte [] signature = s.sign();
```

- ▶ Verify:

```
Signature s = Signature.getInstance(" SHA256withRSA" );  
s.initVerify(pubKey);  
s.update(data);  
boolean valid = s.verify(signature);
```

- ▶ Encryption and decryption operations
- ▶ Encryption:

```
Secret key = getSecretKey();  
Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");  
  
byte[] iv = new byte[c.getBlockSize()];  
SecureRandom sr = new SecureRandom();  
sr.nextBytes(iv);  
IvParameterSpec ivp = new IvParameterSpec(iv);  
c.init(Cipher.ENCRYPT_MODE, key, ivp);  
  
byte[] data = "Message to encrypt".getBytes("UTF-8");  
byte[] ciphertext = c.doFinal(data);
```


► Decryption:

```
Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");
c.init(Cipher.DECRYPT_MODE, key, ivp);

byte [] data = c.doFinal(ciphertext);
```

► Message Authentication Code algorithms

```
SecretKey key = getSecretKey();  
Mac m = Mac.getInstance("HmacSha256");  
m.init(key);  
byte [] data = "Message".getBytes("UTF-8");  
byte [] hmac = m.doFinal(data);
```

- ▶ Generates symmetric keys
- ▶ Additional checks for weak keys
- ▶ Set key parity when necessary
- ▶ Takes advantage of the cryptographic hardware

```
KeyGenerator kg = KeyGenerator.getInstance("HmacSha256");  
SecretKey key = kg.generateKey();
```

```
KeyGenerator kg = KeyGenerator.getInstance("AES");  
kg.init(256);  
SecretKey key = kg.generateKey();
```

- ▶ Generates public and private keys

```
KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");  
kpg.initialize(1024);  
KeyPair pair = kpg.generateKeyPair();  
PrivateKey priv = pair.getPrivate();  
PublicKey pub = pair.getPublic();
```

- ▶ Harmony's Crypto Provider
 - ▶ Limited JCA provider part of the Java runtime library
 - ▶ SecureRandom (SHA1PRNG), KeyFactory (DSA)
 - ▶ MessageDigest (SHA-1), Signature (SHA1withDSA)
- ▶ Android's Bouncy Castle Provider
 - ▶ Full-featured JCA provider
 - ▶ Part of the Bouncy Castle Crypto API
 - ▶ Cipher, KeyGenerator, Mac, MessageDigest, SecretKeyFactory, Signature, CertificateFactory
 - ▶ Large number of algorithms
- ▶ AndroidOpenSSL Provider
 - ▶ Native code, performance reasons
 - ▶ Covers most functionality of Bouncy Castle
 - ▶ Preferred provider
 - ▶ Implementation uses JNI to access OpenSSL's native code

Cryptographic Providers

SSL/TLS

JSSE

Android JSSE Providers

Bibliography

- ▶ Cryptographic providers for securing communication
- ▶ Recommendation: standardized security protocols
- ▶ Secure Sockets Layer (SSL) and Transport Layer Security (TLS)
- ▶ SSL is the predecessor of TLS

- ▶ Secure point-to-point communication protocols
- ▶ Authentication, message confidentiality and integrity for communication over TCP/IP
- ▶ Combination of symmetric and asymmetric encryption for confidentiality and integrity
- ▶ Public key certificates for authentication

- ▶ Cipher suite = set of algorithms for key agreement, authentication, integrity protection and encryption
- ▶ Client sends SSL version and list of cipher suites
- ▶ Client and server negotiate common cipher suite

- ▶ Authenticate through certificates
 - ▶ Usually only server authentication
 - ▶ Client authentication is also supported
- ▶ Compute shared symmetric key
- ▶ Secure communication using symmetric encryption & key

- ▶ Binding an identity to a public key
- ▶ X.509 certificates
- ▶ Signature algorithm
- ▶ Validity
- ▶ Subject DN
- ▶ Issuer DN

| | |
|----------------------------|------------------------------|
| Subject Name | _____ |
| Country | US |
| State/Province | California |
| Locality | Mountain View |
| Organization | Google Inc |
| Common Name | *.google.com |
| Issuer Name | _____ |
| Country | US |
| Organization | Google Inc |
| Common Name | Google Internet Authority G2 |
| Serial Number | 9085461370495713600 |
| Version | 3 |
| Signature Algorithm | SHA-256 with RSA Encryption |

- ▶ SSL client communicates with a small number of servers
- ▶ Set of trusted server certificates = trust anchors
- ▶ Can be self-signed
- ▶ A server is trusted if it's certificate is part of the set
- ▶ Good control over trusted server
- ▶ Hard to update server key and certificate

- ▶ Private Certificate Authority (CA) -> trust anchor
- ▶ Signs server certificate
- ▶ Client trusts any certificate issued by the CA
- ▶ Easy to update server key and certificate
- ▶ Single point of failure

- ▶ Public Certificate Authorities (CAs) -> trust anchors
- ▶ Client configured with a set of trust anchors
- ▶ Web browsers - more than 100 CAs

Cryptographic Providers

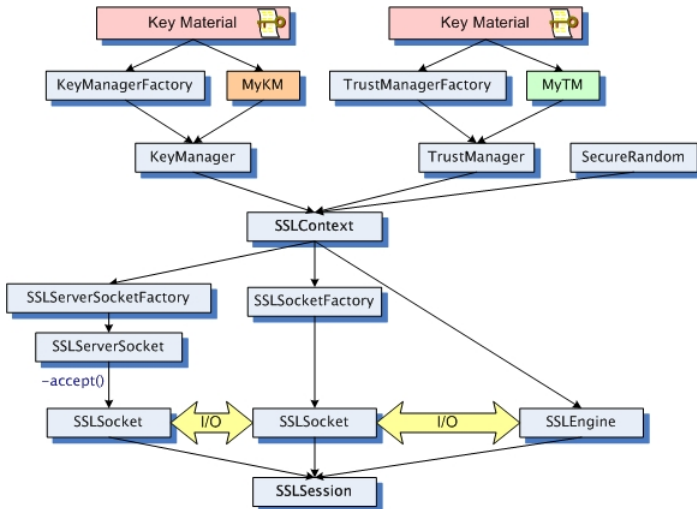
SSL/TLS

JSSE

Android JSSE Providers

Bibliography

- ▶ Android provides support for SSL/TLS through JSSE
- ▶ `javax.net` and `javax.net.ssl`
- ▶ Provides:
 - ▶ SSL client and server sockets
 - ▶ Socket factories
 - ▶ `SSLEngine` - producing and consuming SSL streams
 - ▶ `SSLContext` - creates socket factories and engines
 - ▶ `KeyManager`, `TrustManager` and factories
 - ▶ `HttpsURLConnection`



Source: <http://docs.oracle.com/javase/7/docs/technotes/guides/security/jsse/JSSERefGuide.html>

- ▶ SSLSocket is created:
 - ▶ Through SSLSocketFactory
 - ▶ Accepting a connection on a SSLServerSocket
- ▶ SSLServerSocket created through SSLServerSocketFactory
- ▶ SSLEngine:
 - ▶ Created by SSLContext
 - ▶ I/O operations handled by the application

- ▶ SSL Context obtained in two ways
- ▶ 1. `getDefault()` method of `SSLServerSocketFactory` or `SSLSocketFactory`
 - ▶ Default context initialized with default `KeyManager`, default `TrustManager` and secure random generator
 - ▶ Key material from system properties

- ▶ 2. `getInstance()` static method of `SSLContext`
 - ▶ Context initialized with array of `KeyManager`, array of `TrustManager`, secure random generator
 - ▶ `KeyManager` obtained from `KeyManagerFactory`
 - ▶ `TrustManager` obtained from `TrustManagerFactory`
 - ▶ Factories initialized with `KeyStore` (key material)

- ▶ Established SSL connection -> `SSLSession` object
- ▶ Includes identities, cipher suites, etc.
- ▶ `SSLSession` used in multiple connections between same entities

- ▶ JSSE delegates trust decisions to TrustManager
- ▶ Delegates authentication key selection to KeyManager
- ▶ Each SSLSocket has access to them through SSLContext
- ▶ TrustManager has a set of trusted CA certificates (trust anchors)
 - ▶ A certificate issued by a trusted CA is considered trustworthy

- ▶ Default JSSE TrustManager initialized using the system trust store
 - ▶ Major commercial and government CA certificates
 - ▶ /system/etc/security/cacerts.bks

```
TrustManagerFactory tmf = TrustManagerFactory
    .getInstance(TrustManagerFactory.getDefaultAlgorithm());
tmf.init((KeyStore) null);

X509TrustManager xtm = (X509TrustManager) tmf
    .getTrustManagers()[0];

for (X509Certificate cert : xtm.getAcceptedIssuers()) {
    String certStr = "S:" + cert.getSubjectDN().getName()
        + "\nI:" + cert.getIssuerDN().getName();
    Log.d(TAG, certStr);
}
```

- ▶ Until Android 4.0,
 - ▶ A single file: `/system/etc/security/cacerts.bks`
 - ▶ Read-only partition
- ▶ From Android 4.0
 - ▶ In addition, two directories
 - ▶ `/data/misc/keychain/cacerts-added`
 - ▶ `/data/misc/keychain/cacerts-removed`
 - ▶ Modified only by the system user
 - ▶ Add trust anchors through `TrustedCertificateStore` class


```
TrustManagerFactory tmf = TrustManagerFactory
                                .getInstance("X509");
tmf.init((KeyStore) null);

TrustManager[] tms = tmf.getTrustManagers();
X509TrustManager xtm = (X509TrustManager) tms[0];

X509Certificate[] certChain = {serverCert};
xtm.checkServerTrusted(certChain, "RSA");
```

- ▶ SSLSocket and HttpURLConnection perform similar validations

- ▶ Preferred method for connecting to a HTTPS server
- ▶ Uses default `SSLConnectionFactory` to create secure sockets
- ▶ Custom trust store or authentication keys
 - ▶ `setDefaultSSLConnectionFactory()` or `setSSLConnectionFactory()` of `HttpsURLConnection`

- ▶ Use your own trust store instead of the system trust store
 - ▶ Load trust store in a `KeyStore` object
 - ▶ Obtain `TrustManagerFactory` and initialize it with trust store
 - ▶ Load key material in `KeyStore` object (for client authentication)
 - ▶ Obtain `KeyManagerFactory` and initialize it with key store
 - ▶ Obtain `SSLContext` and initialize it with `TrustManager` and `KeyManager`
 - ▶ Create URL and `HttpsURLConnection`
 - ▶ Associate `SSLConnectionFactory` of `SSLContext` to `HttpsURLConnection`

```
KeyStore trustStore = loadTrustStore();
KeyStore keyStore = loadKeyStore();

TrustManagerFactory tmf = TrustManagerFactory
    .getInstance(TrustManagerFactory.getDefaultAlgorithm());
tmf.init(trustStore);

KeyManagerFactory kmf = KeyManagerFactory
    .getInstance(KeyManagerFactory.getDefaultAlgorithm());
kmf.init(keyStore, KEYSTORE.PASSWORD.toCharArray());

SSLContext sslCtx = SSLContext.getInstance("TLS");
sslCtx.init(kmf.getKeyManagers(), tmf.getTrustManagers(), null);

URL url = new URL("https://myserver.com");
HttpsURLConnection urlConnection = (HttpsURLConnection) url
    .openConnection();
urlConnection.setSSLSocketFactory(sslCtx.getSocketFactory());
```



- ▶ Generate your trust store using Bouncy Castle and openssl in comand line
- ▶ Trust store file in res/raw/

```
KeyStore localTrustStore = KeyStore.getInstance("BKS");
InputStream in = getResources().openRawResource(
    R.raw.mytruststore);
localTrustStore.load(in, TRUSTSTORE.PASSWORD.toCharArray());

TrustManagerFactory tmf = TrustManagerFactory
    .getInstance(TrustManagerFactory.getDefaultAlgorithm());
tmf.init(localTrustStore);

SSLContext sslCtx = SSLContext.getInstance("TLS");
sslCtx.init(null, tmf.getTrustManagers(), null);

URL url = new URL("https://myserver.com");
HttpsURLConnection urlConnection =
    (HttpsURLConnection) url.openConnection();
urlConnection.setSSLSocketFactory(sslCtx.getSocketFactory());
```

Cryptographic Providers

SSL/TLS

JSSE

Android JSSE Providers

Bibliography

- ▶ JSSE providers implement functionality for engine classes
 - ▶ Trust managers, key managers, secure sockets, etc.
 - ▶ Developers work with engine classes
- ▶ Android includes two JSSE providers:
 - ▶ HarmonyJSSE
 - ▶ AndroidOpenSSL

- ▶ Implemented in Java
- ▶ Java sockets, JCA cryptographic classes
- ▶ SSLv3, TLSv1
- ▶ Deprecated, not actively maintained

- ▶ Calls to OpenSSL native library (JNI)
- ▶ TLSv1.1, TLSv1.2
- ▶ Server Name Indication (SNI)
 - ▶ SSL clients specify target hostname
 - ▶ Server with multiple virtual hosts
 - ▶ Used by default by `HttpsURLConnection`
- ▶ Both providers share `KeyManager` and `TrustManager` code
- ▶ Different SSL socket implementation

Cryptographic Providers

SSL/TLS

JSSE

Android JSSE Providers

Bibliography

- ▶ Android Security Internals, Nikolay Elenkov
- ▶ <http://nelenkov.blogspot.ro/2011/12/using-custom-certificate-trust-store-on.html>
- ▶ <https://github.com/nelenkov/custom-cert-https>
- ▶ <http://docs.oracle.com/javase/7/docs/technotes/guides/security/jsse/JSSERefGuide.html>

- ▶ Java Cryptography Architecture
- ▶ Cryptographic Service Provider
- ▶ Engine classes
- ▶ SSL/TLS
- ▶ Trust anchors
- ▶ Certificate Authority
- ▶ Trust store
- ▶ Java Secure Socket Extension
- ▶ SSLSocket
- ▶ HttpsURLConnection
- ▶ AndroidOpenSSL