Android Cryptography and Network Security

Lecture 5

Security of Mobile Devices

2022

Outline

JCA Provider Architecture

- 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 Engine Classes

- 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
```

Message Digest

- Hash function

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

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

MAC

- Message Authentication Code algorithms

```
byte[] message = ...;
SecretKey key = ...;
Mac m = Mac.getInstance("HmacSha256");
m.init(key);
byte[] hmac = m.doFinal(message);
```

Signature

- Digital signature algorithms based on asymmetric encryption
- Algorithm name: `<digest>with<encryption>`
- Create the signature:

```
byte[] message = ...;
PrivateKey key = ...;
Signature s = Signature.getInstance("SHA256withECDSA");
s.initSign(key);
s.update(message);
byte[] signature = s.sign();
```

Signature

- Verify the signature:

```
byte[] message = ...;
byte[] signature = ...;
PublicKey key = ...;
Signature s = Signature.getInstance("SHA256withECDSA");
s.initVerify(key);
s.update(message);
boolean valid = s.verify(signature);
```

Cipher

- Encryption and decryption operations
- Encrypt a message:

```
byte[] plaintext = ...;
KeyGenerator keygen = KeyGenerator.getInstance("AES");
keygen.init(256);
SecretKey key = keygen.generateKey();
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
cipher.init(Cipher.ENCRYPT_MODE, key);
byte[] ciphertext = cipher.doFinal(plaintext);
byte[] iv = cipher.getIV();
```

Cipher

- Decrypt the message:

```
Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");
c.init(Cipher.DECRYPT_MODE, key, iv);
byte[] plaintext = c.doFinal(ciphertext);
```

KeyGenerator

- 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();
```

KeyPairGenerator

- 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();
```

Recommended Algorithms

Class	Recommendation
Cipher	AES in either CBC or GCM mode with 256-bit keys (such as `AES/GCM/NoPadding`)
MessageDigest	SHA-2 family (eg, `SHA-256`)
Mac	SHA-2 family HMAC (eg, `HMACSHA256`)
Signature	SHA-2 family with ECDSA (eg, `SHA256withECDSA`)

Source: `https://developer.android.com/guide/topics/security/cryptography`

SMD

Outline

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
- Many algorithm implementations are deprecated

AndroidOpenSSL Provider

- (aka Conscrypt)
- Native code, performance reasons
- Covers most functionality of Bouncy Castle
- Preferred provider in Android
- Implementation uses JNI to access OpenSSL's native code

Outline

Secure Communication

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

SSL/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

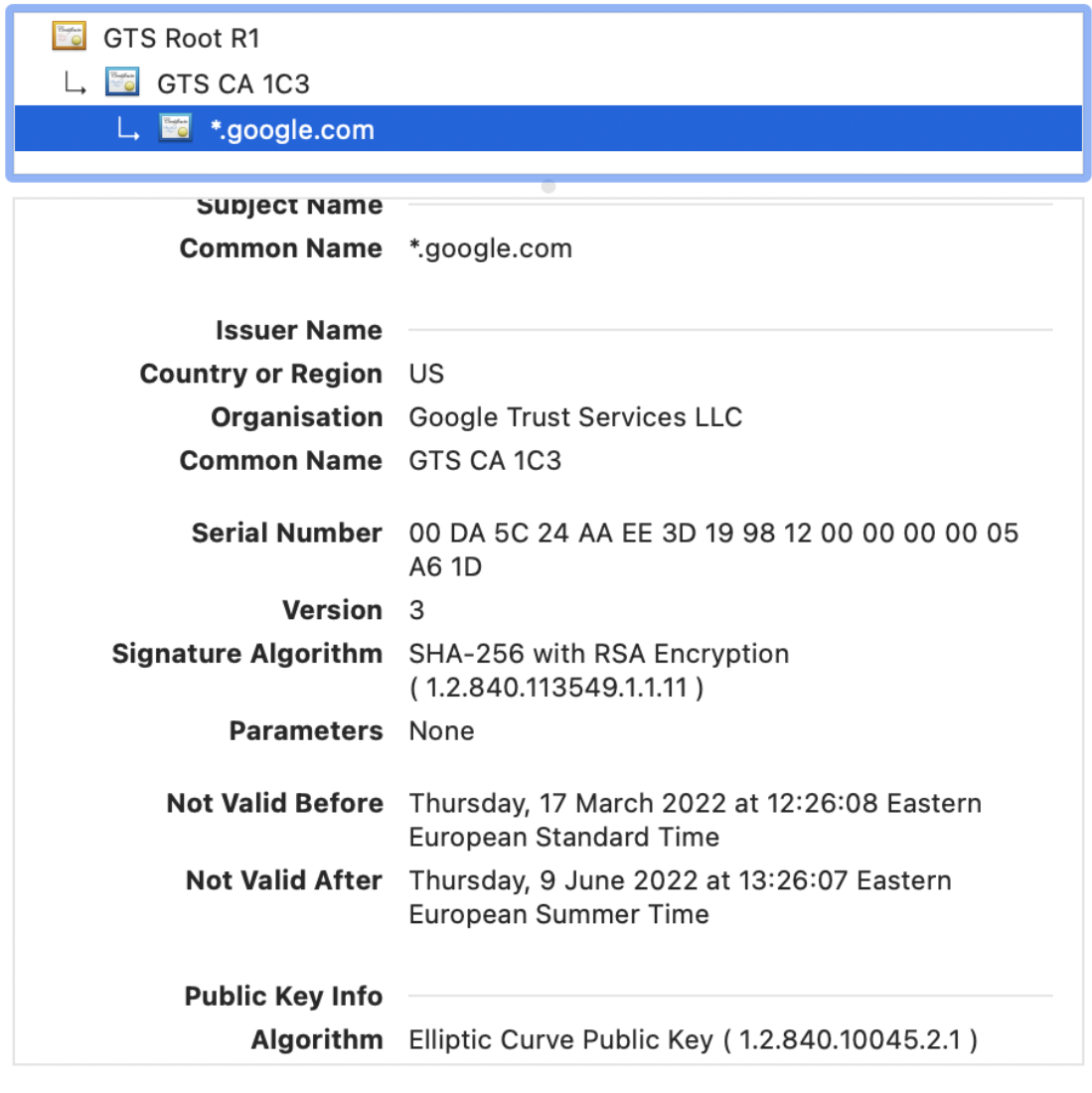
- Cipher suite = set of algorithms for key agreement, authentication, integrity protection and encryption
- Client sends SSL version and list of cipher suites
- Client and server negotiate common cipher suite

Authentication and Encryption

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

Public Key Certificates

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



Direct Trust

- 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 CAs

- 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 CAs

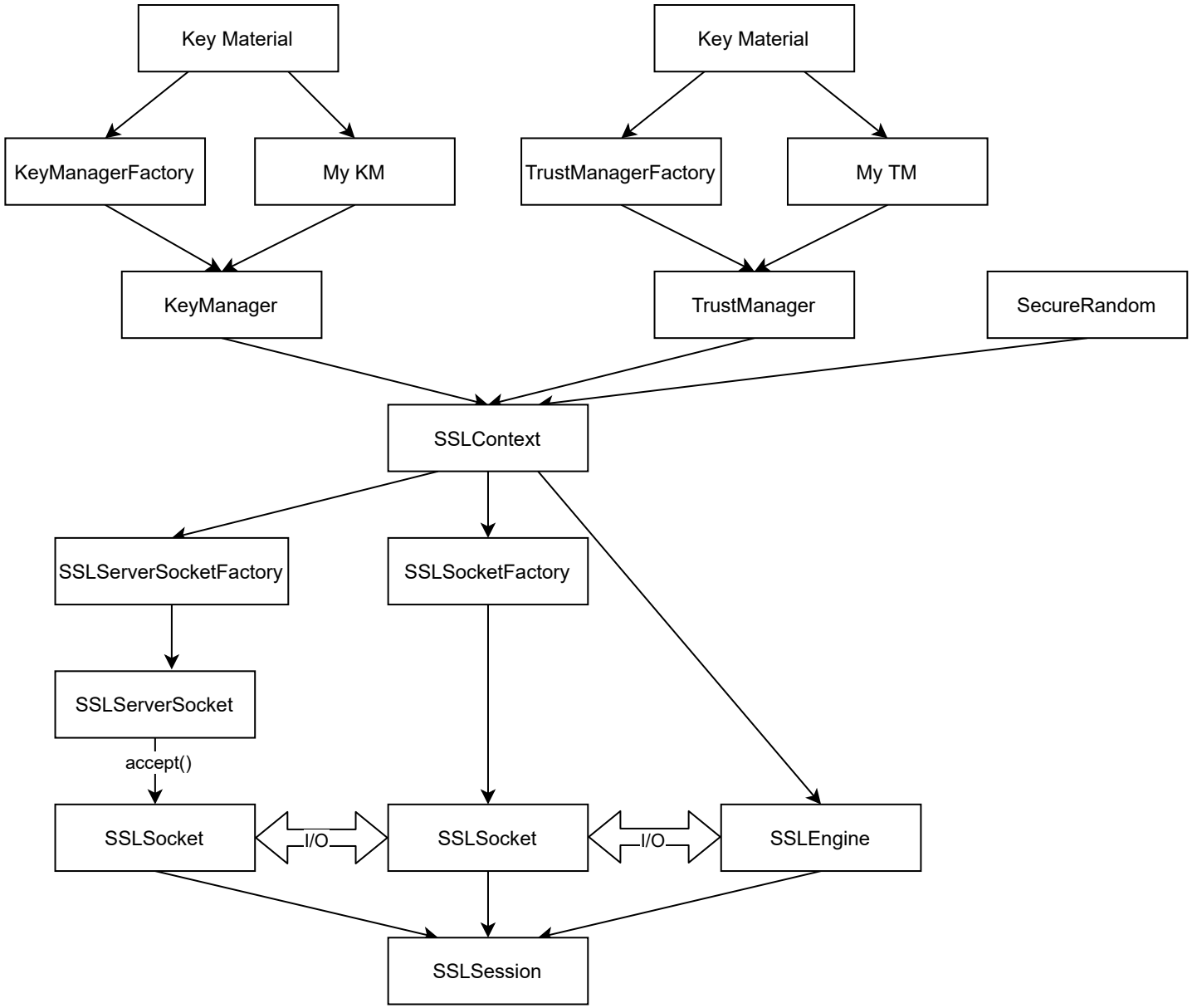
- Public Certificate Authorities (CAs) -> trust anchors
- Client configured with a set of trust anchors
- Web browsers - more than 100 CAs

Outline

Java Secure Socket Extension (JSSE)

- 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

JSSE Classes



SSLSocket

- `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

SSLContext

- 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

SSLContext

- 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)

SSLSession

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

TrustManager and KeyManager

- 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

System Trust Store

- 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);
}
```

System Trust Store

- 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

Validate Server Certificate

```
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 HttpsURLConnection perform similar validations

HttpsURLConnection

- Preferred method for connecting to a HTTPS server
- Uses default `SSLSocketFactory` to create secure sockets

```
URL url = new URL("http://www.android.com/");
HttpsURLConnection connection = null;
try {
    connection = (HttpsURLConnection) url.openConnection();
    InputStream in = new BufferedInputStream(
                                    connection.getInputStream());
    readStream(in);
} catch (IOException e) {
    Log.e(TAG, e.toString());
} finally {
    if(connection != null) connection.disconnect();
}
```

Network Security Configuration

- Configure security settings without modifying the code
- Add a custom set of trusted CAs
- Restrict the public trusted CAs
- Prevent the application from sending cleartext traffic
- Restrict the secure connections to a set of certificates

Network Security Configuration File

- Specify in the Manifest the network security configuration file
- XML format

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application android:networkSecurityConfig="@xml/
network_security_config" ... >
        ...
    </application>
</manifest>
```

Customize trusted CAs

- Use a custom set of trusted CAs
- Self-signed or private signed certificate
- Limit to a set of trusted CAs
- Trust a CA that is not in the default trust store
- TLS & HTTPS use the default trust store
- Customize using the Network Security Settings

Configure a custom CA

- Self-signed certificate
- Certificate signed by a private CA
- Add the certificate in `my_ca` file
- `res/xml/network_security_config.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">example.com</domain>
        <trust-anchors>
            <certificates src="@raw/my_ca"/>
        </trust-anchors>
    </domain-config>
</network-security-config>
```

Limit the set of trusted CAs

- Limited set of trusted CAs
- For specific domains
- Multiple certificates can be included in the file

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <domain-config>
        <domain includeSubdomains="true">sec.example.com</domain>
        <domain includeSubdomains="true">cdn.example.com</domain>
        <trust-anchors>
            <certificates src="@raw/trusted_roots"/>
        </trust-anchors>
    </domain-config>
</network-security-config>
```

Trust additional CAs

- Trust CAs that are not included in the default trust store
- `res/xml/network_security_config.xml`

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
    <base-config>
        <trust-anchors>
            <certificates src="@raw/extracas"/>
            <certificates src="system"/>
        </trust-anchors>
    </base-config>
</network-security-config>
```

Outline

Android JSSE Providers

- 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 (deprecated)
  - AndroidOpenSSL

AndroidOpenSSL

- Calls to OpenSSL native library (JNI)
- TLSv1.1, TLSv1.2, TLSv1.3 (from Android 10)
- Server Name Indication (SNI)
  - SSL clients specify target hostname
  - Server with multiple virtual hosts
  - Used by default by `HttpsURLConnection`
- Default JSSE provider

Outline

Bibliography

- https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html
- http://docs.oracle.com/javase/7/docs/technotes/guides/security/jsse/JSSERefGuide.html
- https://developer.android.com/training/articles/keystore
- https://developer.android.com/guide/topics/security/cryptography
- https://developer.android.com/training/articles/security-ssl
- https://developer.android.com/training/articles/security-config
- Android Security Internals, Nikolay Elenkov

Keywords

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