4.

- In the last lecture we saw that Android provides several cryptographic providers for hashing, message authentication codes, encryption, and so on. They can be used in order to ensure secure communication.
- However, in order to avoid vulnerabilities, it is better to use standardized security protocols, that have been specified and used for securing network communication.
- The most used security protocols are TLS and its predecessor SSL.

5.

- These are protocols for point-to-point secure communication that provide authentication, confidentiality and integrity to the messages sent between these two entities that communicate over TCP/IP.
- They use a combination between symmetrical and asymmetrical encryption in order to provide confidentiality and integrity, and use certificates for authentication.

6.

- Usually, a client that wants to communicate with a server through SSL, initiates the communication by sending the supported version of SSL and a list of cipher suites.
- A cipher suite is a set of algorithms used for key agreement, authentication, integrity protection and encryption.
- The client and server will negotiate a cipher suite that is supported by both of them.

7.

- After that, they will authenticate (verify each other's identity) through certificates.
- In general, only the server authenticates itself to the client. However, client authentication is also supported by SSL.
- If authentication is successful, they will compute a shared symmetric key that is used for securing the communication.
- (Authentication is implemented using public key cryptography and certificates. Each entity will offer its certificate and the other entity, if it trusts the certificate, will negotiate a shared key for encrypting communications by using public and private keys.)
- The subsequent communication will be secured by using a symmetric encryption algorithm and the negotiated key.

8.

- A public key certificate is used for associating an identity to a public key.
- SSL uses a X.509 certificates for authentication
- X.509 certificates include a large number of fields: Signature Algorithm, Issuer, Validity, Subject, etc.
- The subject is represented by a set of attributes including the common name (CN), location and organization, which form the distinguished name (DN)
- The issuer is described by similar attributes
- This is a part of Google's certificate

9.

- If the SSL client communicates with a small number of servers, it can be configured with the set of trusted server certificates (trust anchors).
- These certificates can be self-signed.
- A server is trusted if its certificate is part of that set.
- The advantage is that you have a good control over the trusted servers
- The disadvantage is that is harder to update the server key and certificate, you have to modify/reconfigure the client.

10.

- Another option is to use a private Certificate Authority (CA) to sign the certificate of the server. The private CA is used as trust anchor. The client will trust any certificate that is issued by this CA.
- The advantage is that you can easily upgrade the server key and certificate, without updating the client.
- The disadvantage is that the CA becomes a single point of failure. If the CA is compromised, an attacker can generate certificates that will be automatically trusted by clients.

11.

- An SSL client (like a web browser, email client) that does not know in advance which are going to be the target servers, is usually configured with a set of trust anchors, which are well-known, public CAs.
- Most web browsers include a set of more 100 CA certificates as trust anchors.

13.

- Android provides support for SSL/TLS through the implementation of Java Secure Sockets Extension (JSSE).
- The JSSE API is available in the javax.net and javax.net.ssl packages
- And provides:
    - SSL client sockets and SSL server sockets
    - Socket factories
    - SSLEngine - producing and consuming SSL streams
    - Secure socket context - SSLContext - creates socket factories and engines
    - Key managers and factories to create them
    - Trust managers and factories to create them
    - HttpsURLConnection - for HTTPS connections

14.

- This image includes the JSSE classes and the interaction between them.
- In JSSE, the endpoint classes of a connection are SSLSocket and SSLEngine.
- The figure shows which are the main classes that are used for creating SSLSocket/SSLEngine.

15.

- An SSLSocket is created either through SSLSocketFactory, or by accepting a connection on an SSLServerSocket.
- An SSLServerSocket is created through SSLServerSocketFactory.
- An SSLEngine is created directly by the SSLContext and relies on the application to handle the I/O operations.

16.

- An SSLContext can be obtained in two ways:
    - Calling getDefault() method of SSLServerSocketFactory or SSLSocketFactory - this provides a default context initialized with the default KeyManager, the default TrustManager and a secure random generator. The key material found in the default keystore and truststore is obtained from the system properties.

17.

    - Obtain an SSLContext instance by calling the static method getInstance() of SSLContext. Then, the context is initialized by giving as parameters: an array of KeyManager objects, an array of TrustManager objects and a SecureRandom. The KeyManager and TrustManager objects can be obtained through KeyManagerFactory and TrustManagerFactory. These factories can be initialized with KeyStore containing the key material. We will see a code example later.

18.

- After a connection is established between an SSL client and server, an SSLSession object is created.
- SSLSession includes information like the endpoint identities, the negotiated cipher suite, etc.
- Each SSL connection has an associated SSLSession, but an SSLSession can be used in multiple connections between the same entities.

19.

- JSSE will delegate the decisions regarding trust in certificates to the TrustManager class and the key selection for authentication to the KeyManager class.
- Each instance of SSLSocket created through JSSE will have access to those classes through the associated SSLContext instance.
- TrustManager has a set of trusted certificates generated by certification authorities, and will make decisions based on them. If a certificate is issued by a trusted certification authority, then that certificate will be considered trustworthy.

20.

- The default TrustManager is initialized with the system trust store, which includes a set of major commercial and government CA certificates.
- The system trust store is found for example in /system/etc/security/cacerts.bks
- Here you have an example on how to obtain all certificates from the system trust store.
- First of all, we obtain an instance of TrustManagerFactory, which is initialized (if we give it null, it will take the default/system trust store automatically).

- Then we obtain the first TrustManager (which is the default one) and cast it to X509TrustManager.
- Then through the method getAcceptedIssuers() we obtain the list of CA certificates from the system trust store.
- For each certificate, some information is displayed (the subject DN and the issuer DN).

21.

- Until Android 4.0, the system trust store was represented by a single file /system/etc/security/cacerts.bks. However, the system partition is read-only, so the file cannot be modified (not even by system applications).
- From Android 4.0, in addition to the /system/etc/security/cacerts.bks file, we have two additional directories: /data/misc/keychain/cacerts-added and /data/misc/keychain/cacerts-removed. The first one includes CA certificates that are added to the system trust store, and the second one includes CA certificates that are removed from the system trust store.
- Only the system user can add or remove CA certificates from the system trust store.
- The trust anchors can be added through the TrustCertificateStore class, which is accessible through the JCA KeyStore API.

22.

- This is an example on how to manually validate a server certificate using the system trust store.
- First of all, we obtain an instance of the TrustManagerFactory. We initialize it with the system trust store by passing null to the init method.
- Then we obtain the array of TrustManagers. We take the first one and cast it to X509TrustManager.
- We build a certificate chain including the server certificate and any intermediate issuers.
- Finally, we validate the certificate chain using the method checkServerTrusted() of the X509TrustManager object.
- SSLSocket and HttpsURLConnection perform such validation automatically.

23.

- HttpURLConnection is the preferred method for connecting to a HTTPS server. It uses the default SSLSocketFactory in order to create secure SSL sockets.
- When we need to use a custom trust store or authentication keys, the default SSLSocketFactory can be replaced through the static method setDefaultSSLSocketFactory() of HttpsURLConnection.
- Another method is to configure the socket factory for the current HttpsURLConnection though the method setSSLSocketFactory().

24-25.

- Sometimes you may want to use your own trust store instead of the system trust store.

- For this you need to load your own trust store (containing trust anchors) in a KeyStore object. Then you obtain a TrustManagerFactory using the static method getInstance(). Then initialize the TrustManagerFactory using the trust store.
- If you need to perform client authentication, you have to load the key material in a KeyStore object. Then you can obtain a KeyManagerFactory using the static method getInstance(). Then initialize KeyManagerFactory using the key store.
- Then you obtain the SSLContext using the static method getInstance(). Then initialize the context based on TrustManager and KeyManager obtained from the factories.
- Finally, create an URL object, obtain the HttpsURLConnection and associate the SSLSocketFactory of the SSLContext to the HttpsURLConnection.
- (An API very used for HTTPS is HttpsURLConnection, that connects to HTTPS through JSSE. In order to use our own trust store, we need to create and initialize a SSLContext (practically this is what SSLSocketFactory makes in the back).)

26.

- Here you have a more specific example in which we load our own trust store.
- We can generate the trust store from the command line using Bouncy Castle and openSSL.
- We place the trust store file in /res/raw.
- First of all, we obtain an instance of the KeyStore in Bouncy Castle format, in which we can load our trust store (we need to specify the password).
- Then obtain an instance of the TrustManagerFactory and initialize it with our trust store.
- Then obtain an instance of SSLContext and initialize it with the TrustManager obtained from the factory. We observe that the KeyManager is null because we do not perform client authentication in this example.

28.

- Similar to JCA cryptographic providers, we have JSSE providers that implement the functionality for the engine classes defined in the API that was described in the previous slides.
- The functionality implemented by providers include: secure sockets, trust managers, key managers, etc. The application developer will not work directly with the implementation classes, but with the engine classes.
- On Android, we have two JSSE providers: Harmony JSSE, that is implemented in Java, and AndroidOpenSSL, which is implemented in native code, and accessed through JNI.

29.

- HarmonyJSSE is based on Java sockets and uses JCA cryptographic classes for implementing SSL.
- It provides only SSLv3 and TLSv1 support
- It is considered deprecated and not actively maintained.

30.

- AndroidOpenSSL implements most functionality by making calls into the OpenSSL native library (through JNI).
- It supports TLSv1.1 and TLSv1.2
- It also supports the TLS extension called Server Name Indication (SNI) - this means that the SSL clients can specify the target hostname, in case the server has multiple virtual hosts.
- SNI is used by default when establishing a connection with HttpsURLConnection
- Both providers share the same TrustManager and KeyManager code, but the SSL socket implementation is different