Partie : Mail / Android / SSL-TLS

Table des matières

[I. Question 1: Java mail 3](#_Toc155896270)

[A. Décrire en les comparant les trois protocoles de messagerie électronique SMTP, POP3 et IMAP. 3](#_Toc155896271)

[B. Quelles différences IMAP apporte-t-il par rapport à POP3 et expliquer en termes d'agents le cheminement d'un message électronique depuis sa rédaction par l'auteur jusqu'à sa lecture par le destinataire. 4](#_Toc155896272)

[C. Expliquer comment ces agents laissent des traces dans les headers des messages et comment on peut les retrouver en programmation Java pour un message reçu. 6](#_Toc155896273)

[II. Question 2: Java mail 7](#_Toc155896274)

[A. Dans les librairies javax.mail, décrire et expliquer les rôles des interfaces, classes abstraites et classes instanciables nécessaires à l'envoi d'un message sans pièces attachées. 7](#_Toc155896275)

[B. Dresser le tableau synoptique de description synthétique des protocoles classiques SMTP, POP3, IMAP, HTTP. 8](#_Toc155896276)

[III. Question 3: Java mail 9](#_Toc155896277)

[A. Dans les librairies javax.activation, décrire et expliquer les rôles des interfaces, classes abstraites et classes instanciables nécessaires à l'envoi d'un message avec pièces attachées. 9](#_Toc155896278)

[B. Donnez un exemple de programmation pour la récupération d’un mail contenant plusieurs pièces attachées (fichiers texte, image) 10](#_Toc155896279)

[C. Expliquer en quoi l'examen des types MIME et de la "disposition" des pièces attachées intervient dans la programmation de cette réception de messages composites. 11](#_Toc155896280)

[IV. Question 4: Mobile 14](#_Toc155896281)

[A. Décrire et expliquer le contenu d'un fichier apk. Quels sont les outils qui interviennent dans sa construction ? En particulier, expliquer le contenu et le rôle de la classe R.java. 14](#_Toc155896282)

[V. Question 5: Mobile 15](#_Toc155896283)

[A. Décrire l'architecture Android à 5(6) couches en en décrivant les éléments principaux et décrire le bytecode particulier utilisé par les JVMs d'Android. Expliquer les caractéristiques des JVM Dalvik et ART et ce qui les différencie de la JVM classique. 15](#_Toc155896284)

[VI. Question 6: Mobile 19](#_Toc155896285)

[A. Décrire la structure des répertoires (et leur principal contenu) qui correspondent à une application Android comportant une activité classique avec images, couleurs, menus, textes personnalisés selon la langue et la manière d'en faire usage dans le code Java d'une activité. 19](#_Toc155896286)

[VII. Question 7: Mobile 21](#_Toc155896287)

[A. Sous Android, décrire à partir d'un exemple d'application ne comportant qu'une seule activité, les diverses manières de programmer la réponse à un événement graphique et la manière de gérer un composant comme une ListView selon le modèle MVC. 21](#_Toc155896288)

[VIII. Question 8: Mobile 23](#_Toc155896289)

[A. Sous Android, décrire à partir d'un exemple d'application ne comportant qu'une seule activité l'utilisation d'un AsyncTask qui accède à un serveur vendant des actions. Expliquer le fonctionnement du modèle MVC pour les composants graphiques d'Android à prtir de l'affichage par l'activité des actions achetées. 23](#_Toc155896290)

[IX. Question 9: Crypto 24](#_Toc155896291)

[A. Expliquer comment une communication réseau est sécurisée par SSL/TLS (SSL-Record), une fois le SSL-Handshake réalisé (donc sans le décrire). Pourquoi parle-t-on de "sous-protocole(s)" ? Qu'entend-on par "variables de session" et "variable de connexion". 24](#_Toc155896292)

[X. Question 10: Crypto 28](#_Toc155896293)

[A. Décrire en détails le SSL-Handshake. En particulier, expliquer en quoi on peut considérer qu'il est adaptable à différentes situations. Pourquoi parle-t-on de "sous-protocoles" ? 28](#_Toc155896294)

[XI. Question 11: Crypto 32](#_Toc155896295)

[A. Expliquer le code Java utilisant SSL à substituer au code classique d'un serveur TCP/IP dont on désire sécuriser les communications. En particulier, décrire les keystores nécessaires au bon fonctionnement de ce code et leur utilisation, côté serveur et cté client. Pourquoi voit-on intervenir dans javax.net.\* un si grand nombre d'interfaces et de classes abstraites ? 32](#_Toc155896296)

[XII. Question 12 (Android): 35](#_Toc155896297)

[Expliquer ce qu'est un « intent » et les différents mécanismes qui permettent de les mettre en œuvre. 35](#_Toc155896298)

[XIII. Question 13: 37](#_Toc155896299)

[A. Expliquez et illustrez les différences (améliorations, mécanisme, …) entre les versions TLS précédentes la 1.3 et TLS 1.3. 37](#_Toc155896300)

[XIV. Question 14: 40](#_Toc155896301)

[A. Termes à définir brièvement, remettre dans son contexte, acronyme de … , utilité(s), rôle(s), exemple d’applications. 40](#_Toc155896302)

# Question 1: Java mail

## Décrire en comparant les trois protocoles de messagerie électronique SMTP, POP3 et IMAP.

**SMTP :**

**SMTP est principalement utilisé pour l'envoi d'e-mails depuis le client de messagerie de l'expéditeur vers le serveur de messagerie du destinataire. Il facilite la communication entre les serveurs de messagerie sortants.**

**Le client SMTP envoie les e-mails au serveur SMTP, qui les transfère ensuite au serveur de messagerie du destinataire. SMTP ne stocke pas les e-mails, il les transfère uniquement. Il suppose que le serveur de messagerie du destinataire est toujours accessible.**

**POP3 :**

**POP3 est conçu pour récupérer les e-mails du serveur de messagerie et les transférer vers le client de messagerie de l'utilisateur. Les e-mails sont téléchargés sur le client, retirés du serveur et stockés localement.**

**Cela signifie que les e-mails ne sont généralement accessibles que depuis le périphérique sur lequel ils ont été téléchargés. POP3 est simple et direct. Une fois les e-mails téléchargés, ils ne résident généralement plus sur le serveur, à moins d'une configuration spécifique.**

**IMAP :**

**IMAP est utilisé pour synchroniser les e-mails entre le serveur de messagerie et le client de messagerie de l'utilisateur. Les e-mails restent sur le serveur même après avoir été consultés. Cela permet un accès depuis plusieurs appareils, car tous les clients se synchronisent avec le serveur.**

**IMAP prend en charge la hiérarchie des dossiers sur le serveur, permettant une meilleure organisation des e-mails. Les modifications apportées sur un appareil sont reflétées sur tous les autres.**

## Quelles différences IMAP apporte-t-il par rapport à POP3 et expliquer en termes d'agents le cheminement d'un message électronique depuis sa rédaction par l'auteur jusqu'à sa lecture par le destinataire.

**IMAP offre une approche plus flexible, permettant une gestion centralisée des e-mails sur le serveur, tandis que POP3 télécharge les e-mails sur un seul appareil.**

**Une image contenant diagramme, dessin, croquis, Plan

Description générée automatiquement**

**Lors du cheminement de notre mail, on va passer par toute une série d’agents. Ces derniers ne représentent pas une machine, mais une app client, ou une app serveur. Lorsque je veux envoyer un mail, je vais utiliser gmail, outlook, ou autre.**

**Ce sont des SMUA (Sending Mail User Agent). Une fois que j’ai rédigé mon message et que cette application client ajoute tous les headers nécessaires. Elle se chargera d’envoyer notre mail vers un serveur de messagerie.**

**Le protocole qui permet cet envoi s’appelle le SMTP (Simple Mail Transfer Protocol). Une fois arrivé sur notre serveur de mail, celui-ci fait au moins tourner 2 applications serveur. D’une part un MTA (Mail Transfer Agent) et d’autre part un MDA (Mail Delivery Agent).**

**Quand notre serveur mail reçoit quelque chose, le MTA commence par inspecter ce mail et regarde si c’est fabriqué correctement, il vérifie que les headers sont de vrais headers de messagerie, il vérifie qu’il n’y a pas de crasses, etc…**

**Et s’il s’agit d’un message convenable, le MTA va simplement taper notre mail dans une queue et faire un appel système pour prévenir le MDA qu’il peut aller pomper dans la queue.**

**Ce couple MDA/MTA va regarder si le message est destiné à l’environnement dans lequel il est (hepl.be !?) Si oui, il va aller taper le message dans une boite aux lettres(le RMTA). Cependant, s’il ne s’agit pas du bon domaine, il contacte un autre serveur mail(un FMTA) qui saura peut-être quoi faire.**

**Pour lire le mail, on fait appel à un RMUA qui va contacter un serveur de messagerie( vers un MRA) avec le protocole IMAP ou POP3. Le MRA ira directement dans la boite mail rechercher le message souhaité.**

**MTA => MDA => RMTA/FMTA => RMUA => MRA**

## Expliquer comment ces agents laissent des traces dans les headers des messages et comment on peut les retrouver en programmation Java pour un message reçu.

**Les messages reçus et envoyés peuvent passer par différents serveurs et donc laissent des traces dans les headers à chaque passage sur un serveur comme la version do format utilisé pour le message, le type de contenu du message ou encore le champ "Received" qui contient une trace(Nom, ip, etc…) de chaque serveur par lequel est passé ce message.  
  
Comment les récupérer ?**

**try {**

**// Récupérer les en-têtes du message**

**Enumeration<Header> headers = ((MimeMessage) message).getAllHeaders();**

**// Afficher les en-têtes**

**while (headers.hasMoreElements()) {**

**Header header = headers.nextElement();**

**System.out.println(header.getName() + ": " + header.getValue());**

**}**

**}**

# Question 2: Java mail

## Dans les librairies javax.mail, décrire et expliquer les rôles des interfaces, classes abstraites et classes instanciables nécessaires à l'envoi d'un message sans pièces attachées.

**La classe "Session" qui représente une session de messagerie. Elle contient des propriétés de configuration telles que les informations d'authentification, les propriétés du serveur de messagerie, etc.**

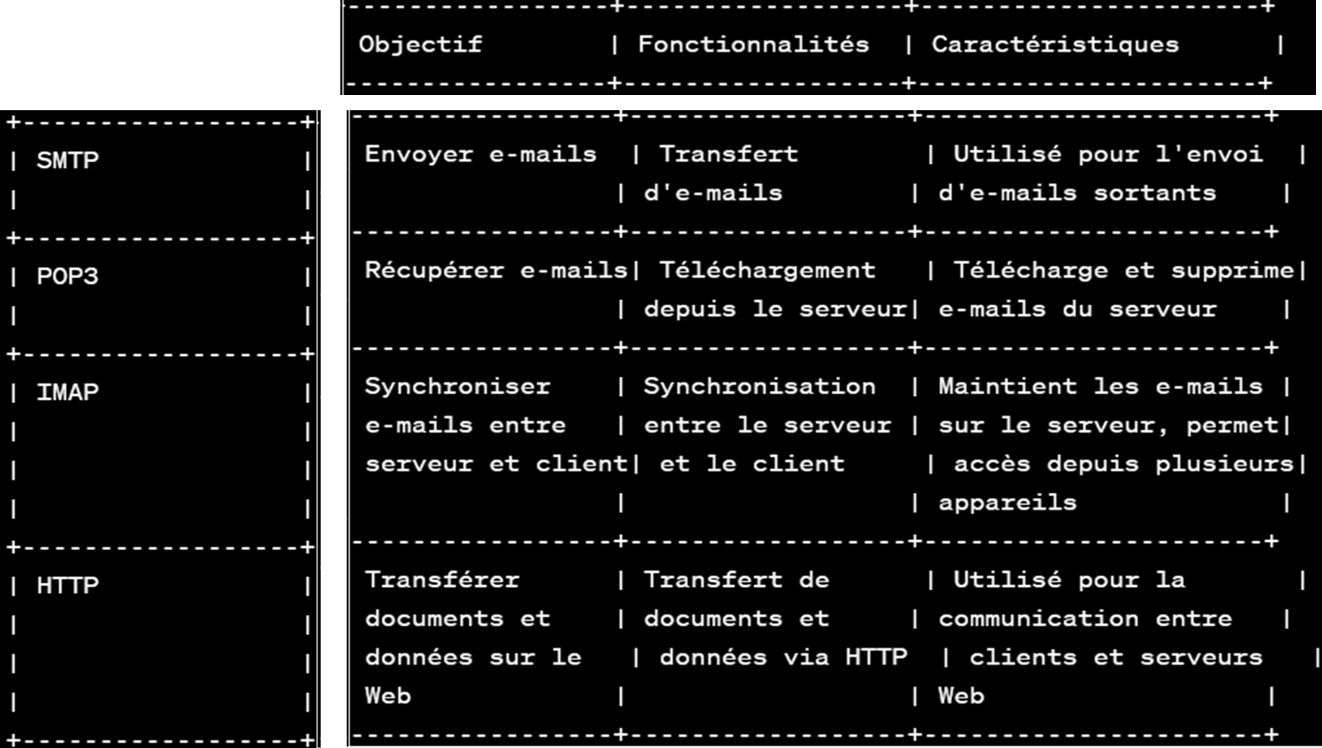
**Les classes abstraites :**

* **La classe abstraite "Authenticator" est une classe abstraite utilisée pour fournir des informations d'authentification lors de l'envoi de messages. Cet objet contient les informations d'identification (nom d'utilisateur et mot de passe) et est passé à la méthode Session.getInstance() lors de la création de la session.**
* **La classe abstraite "Message" représente un message électronique. On l’étend à l’aide de classe instanciable comme "MimeMessage" pour créer des messages standard.**

**Les classes instanciables :**

* **La classe "Transport" représente la couche de transport pour l'envoi de messages.**
* **La classe "MimeMessage" représente un message MIME standard, Elle permet de spécifier les destinataires, le sujet, le corps et les pièces jointes du message.**

## Dresser le tableau synoptique de description synthétique des protocoles classiques SMTP, POP3, IMAP, HTTP.



# Question 3: Java mail

## Dans les librairies javax.activation, décrire et expliquer les rôles des interfaces, classes abstraites et classes instanciables nécessaires à l'envoi d'un message avec pièces attachées.

**DataHandler (Interface) :**

**Rôle : La classe DataHandler est utilisée pour encapsuler les données à envoyer dans le corps du message, y compris les pièces jointes. Elle fournit une interface uniforme pour accéder à différentes sources de données.**

**DataSource (Interface) :**

**Rôle : L'interface DataSource est mise en œuvre par des classes qui représentent différentes sources de données.**

**MimeMessage (Classe instanciable) :**

**Rôle : La classe MimeMessage est utilisée pour représenter un message MIME standard (Multipurpose Internet Mail Extensions).**

**Elle permet de spécifier les destinataires, le sujet, le corps et les pièces jointes du message .**

**Session (Classe instanciable) :**

**Rôle : La classe Session représente une session de messagerie et est utilisée pour configurer divers paramètres liés à l'envoi de messages, tels que les propriétés du serveur SMTP, l'authentification, etc.**

**MimeMultiPart (Classe instanciable) :**

**Rôle : La classe MimeMultiPart est utilisée pour regrouper plusieurs parties du message en une seule entité. Cela est utile lorsque vous avez besoin d'inclure des pièces jointes ou du contenu multipart dans votre message.**

**MimeBodyPart (Classe instanciable) :**

**Rôle : La classe MimeBodyPart représente une partie individuelle du message, telle que le corps du message principal ou une pièce jointe. Elle peut être attachée à un objet MimeMultiPart pour composer le message global.**

## Donnez un exemple de programmation pour la récupération d’un mail contenant plusieurs pièces attachées (fichiers texte, image)

***for (int j = 0; j < multipart.getCount(); j++) {***

***BodyPart bodyPart = multipart.getBodyPart(j);***

***if (bodyPart instanceof MimeBodyPart) {***

***MimeBodyPart mimeBodyPart = (MimeBodyPart) bodyPart;***

***System.out.println("Attachment: " + mimeBodyPart.getFileName());***

***if (mimeBodyPart.isMimeType("text/plain")) {***

***// Traitement pour les fichiers texte***

***String textContent = (String) mimeBodyPart.getContent();***

***System.out.println("Text content: " + textContent);***

***} else if (mimeBodyPart.isMimeType("image/\*")) {***

***// Traitement pour les fichiers image***

***InputStream imageStream = (InputStream) mimeBodyPart.getContent();***

***String fileName = mimeBodyPart.getFileName();***

***String savePath = "/chemin/vers/enregistrement/" + fileName;***

***saveFile(imageStream, savePath);***

***System.out.println("Image saved at: " + savePath);***

***}***

***}***

***}***

## Expliquer en quoi l'examen des types MIME et de la "disposition" des pièces attachées intervient dans la programmation de cette réception de messages composites.

**Types MIME (Multipurpose Internet Mail Extensions) :**

**Les types MIME décrivent le type de média et le sous-type d'un contenu dans un message électronique. Cela inclut des types tels que text/plain, image/jpeg, application/pdf, etc. En Java, pour traiter les pièces jointes, je dois examiner le type MIME de chaque partie du message.**

**Cela permet de déterminer comment la pièce jointe doit être traitée, qu'il s'agisse d'un texte brut, d'une image, d'un document PDF, etc.**

**Je peux utiliser la classe javax.mail.Part pour obtenir des informations sur les pièces jointes, y compris leur type MIME. Ensuite, je peux prendre des décisions basées sur ce type pour traiter la pièce jointe correctement dans mon application.**

**Disposition des pièces jointes :**

**La disposition d'une pièce jointe dans un message électronique indique comment elle doit être traitée lors de son affichage ou de son enregistrement. La disposition peut être en ligne (inline) ou en pièce jointe (attachment).**

**En Java, je peux accéder à la disposition d'une pièce jointe en utilisant la classe javax.mail.Part. Cela permet de déterminer si la pièce jointe doit être affichée directement dans le corps du message (inline) ou si elle doit être enregistrée en tant que fichier attaché (attachment).**

**En fonction de la disposition, je peux décider de l'afficher directement dans mon application (par exemple, pour les images en ligne) ou de l'enregistrer sur le disque pour un traitement ultérieur.**

# Question 4: Mobile

## Décrire et expliquer le contenu d'un fichier apk. Quels sont les outils qui interviennent dans sa construction ? En particulier, expliquer le contenu et le rôle de la classe R.java.

**Le fichier .apk qui remplace notre bon vieux .jar, et contient comme d’habitude le dossier META-inf, mais également Le fichier .dex (unique) et l’ AndroidManifest.xml qui remplace notre fichier manifest de l’époque. Dans cet AndroidManifest, on déclare les différentes activités que notre application va utiliser, on y trouve aussi les permissions (les intentions) : faire du réseau, du Bluetooth, notifications, …**

**L’apk contient également les répertoires res et arsc, res est un zip qui contient les fichiers (données en .xml par exemple) mais en clair. L’arsc est la version binaire des ressources. On parle de ressources ‘compilées’.**

**On le sait, une application android est composée d’activités, mais il n’existe pas de main(). C’est simplement écrit dans l’AndroidManifest.xml. Il faut aussi savoir qu’une application tourne sur le kernel linux du téléphone et possède son propre id et est exécutée dans sa propre JVM. Chaque activité est un objet instance de la classe Activity, et donne lui a un écran, un GUI. Cet interface utilisateur est défini dans un fichier .xml.**

**Une application android peut également être composée de service (une activité sans interface graphique). Le service est partageable entre plusieurs applications.**

**Lorsqu’on va démarrer notre nouvelle activité, on va en réalité appeler automatiquement sa méthode onCreate, méthode que l’on peut surcharger. A l’intérieur on y retrouve super.onCreate() pour appeler la méthode de démarrage de la classe mère Activity, mais également un setContentView(R.layout.main).**

**Le main qu’on lui donne fait en fait référence au fichier main.xml dans lequel on a défini l’interface graphique qu’on souhaite incomber à notre activité.**

**Le setContentView() utilise le polymorphisme et est disponible en 2 versions : Le premier qui prend une View, view étant la classe mère de tous les composants graphiques (un peu comme la JComponent). Le second prend un entier layoutResID.**

**D’un côté j’ai mon activité act.java qui hérite de Activity, de l’autre j’ai mon fichier ressource main.xml : Tous ces fichiers xml (GUI) vont se trouver dans un le rep /res/layout/name.xml.**

**Lorsque je veux fabriquer l’apk, le boulot commence par transformer le répertoire res (zippé) et il faudra bien un jour où l’autre aller trouver le fichier xml stocké dans ce répertoire pour savoir quoi afficher sur le téléphone.**

**Il y a en fait un outil (l’AAPT) qui parcourt le res, tout ce qu’il y a dedans et fabrique un seul fichier, l’arsc.** **Qui est en réalité une version binaire de toutes nos ressources. L’AAPT ‘parse’ et ‘combine’ tous nos fichiers du répertoire res, dans un seul fichier binaire l’arsc. Il suffit de connaitre la position dans le fichier binaire (indexé) et on retrouve directement la ressource souhaitée. En réalité, lorsque l’AAPT fait la conversion. Il fabrique une classe R.java, R comme référence. Par l’intermédiaire de la classe R, on pourra référencer n’importe quoi dans le fichier binaire arsc.**

**La classe R est constituée de plein de classes imbriquées. Une de ces classes imbriquée s’appelle layout, et on voit qu’elle contient un « static final int » qui s’appelle main. Ben cette valeur est la position dans l’arsc pour retrouver la description de l’interface graphique.**

# Question 5: Mobile

## Décrire l'architecture Android à 5(6) couches en décrivant les éléments principaux et décrire le bytecode particulier utilisé par les JVMs d'Android. Expliquer les caractéristiques des JVM Dalvik et ART et ce qui les différencie de la JVM classique.

**La couche KERNEL :**

**L’idée est de fournir au monde du développement les accès au hardware qui se trouve en dessous. Encore une fois, on parle ici de Kernel Linux, car certaines choses ont été retirée d’un OS linux normal, mais d’autres choses propres au monde du téléphone y ont été ajouté : Alarm (réveil du téléphone après une mise en veille), Ashmem (gestion de la mémoire réduite d’un téléphone) : avec si peu d’espace, on part du principe qu’on se partage un maximum de chose intra et inter applications.**

**Les libraires : (bas niveau)**

**Ce sont toutes les librairies C/C++ qui seront utilisées par les couches supérieurs. On parle de Bionic LibC, librairies compilées qui tient compte de l’environnement réduit. Le Webkit (moteur de rendu) qui permet de lire l’HTML et le faire apparaitre. SQLite, permet de mémoriser les données en local (sauvegardé en binaire).**

**Le Hardware Abstraction Layer :**

**Se nomme aussi le HAL, contient des interfaces communes aux différents kernel existant et des librairies spécifiques.**

**L’android runtime :**

**Ce sont les librairies développées en Java. Evidemment on a une JVM qui va faire le pont entre ce qu’on écrit en Java et la machine elle-même. On y retrouve donc les librairies de base en Java et les JVM. Petit plus, on peut toujours à ce niveau habillé nos méthodes natives en C par du Java. Cela s’appelle le JNI, Java Native Interface, comme les méthodes de la classe Thread en application classique par exemple.**

**L’Application Manager :**

**Il s’agit du toolkit que les applications utiliseront. Ce sont tous les services qui vont tourner en arrière-plan et qui sont nécessaire au fonctionnement de nos applis sur le téléphone. On y retrouve l’Activity Manager (un écran = une application = une activité).**

**Le Package Manager qui gère les fichiers .apk (le .jar, .zip du monde android) qui contient le manifeste mais surtout un fichier XML qui s’appelle l’AndroidManifest qui est le tableau de bord de l’application et qui contient, entre autres, le nom de l’activité de base de notre application. On trouve également le Ressource Manager(gestion ressource) et le Content Provider(pour les données).**

***Et on y retrouve également les Hardware Services, qui sont les API permettant d’accéder au matériel :* son, Bluetooth, Wifi, USB, GPS, …**

**Caractéristiques des JVMS Dalvik/Art comparé à la JVM classique :**

**Qu’on parle de Dalvik ou d’ART, ces JVM se différencient des JVM ‘normales’ par leur architecture : une JVM classique est orientée pile (on économise les adresses, mais instructions de bas niveau beaucoup plus nombreuses), tandis que Dalvik et ART sont orientées registres (elles utilisent des adresses, ça prend moins de place).**

**Pour commencer, il faut savoir que la machine virtuelle Dalvik ne bouffe pas du bytecode, autrement dit, elle ne bouffe pas des fichiers .class. On a vu quand Java un fichier .java donnait lieu à un fichier .class, l’idée ici est que, tous ces fichiers .class on va les regrouper en un seul fichier. On va écrire notre code en Java, et on va compiler (javac) : il génère les .class, et ensuite on a un outil DX (Dalvik Executable) qui rassemble tous les fichiers en seul fichier .dex.**

* **Chaque application android tourne sur une JVM dalvik individuelle. Cela permet que si l’application plante, elle ne plante pas, ni les autres, ni le téléphone en lui-même. Pour cela, lorsque le téléphone boot, le kernel Linux lui-même démarre le processus ‘Zygote’. Premièrement Zygote va lancer la JVM Dalvik primaire, celle de départ pour faire tourner le java utilisé par le téléphone indépendamment des applications. Ensuite, à chaque apk démarré, on va créer une nouvelle instance de la machine virtuelle. Et c’est sur cette nouvelle VM que l’application va tourner.**
* **Il existe également une seconde machine virtuelle, l’Android RunTime (ART), qui, au niveau du partage des ressources, fonctionne de la même manière que Dalvik .java (javac) .class (dx) .dex.**

**La différence entre les deux se situent au niveau de l’exécution du code. Dalvik est ‘*Just-In-Time’*, alors que ART est ‘*Ahead-Of-Time’*.**

**L’idée du *just-in-time* est qu’une fois qu’on a traduit les instructions du bytecode en langage machine, on le met en cache. Ainsi, avec Dalvik si on souhaite réexécuter une instruction, on reprend directement la traduction en cache, et on l’exécute immédiatement.**

**Avec ART c’est différent, je développe mon .apk, je le mets sur le téléphone, au moment de son installation, la JVM ART voit arriver l’application et la traduit directement. ART traduit entièrement au moment de l’installation et garde la version traduite. L’intérêt est de réduire le temps d’attente d’un utilisateur, l’installation sera effectivement plus longue, mais lorsqu’on l’exécute, tout est déjà traduit.**

**Une image contenant texte, capture d’écran, Police, logiciel

Description générée automatiquement**

# Question 6: Mobile

## Décrire la structure des répertoires (et leur principal contenu) qui correspondent à une application Android comportant une activité classique avec images, couleurs, menus, textes personnalisés selon la langue et la manière d'en faire usage dans le code Java d'une activité.

**Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement**

**Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement**

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

# Question 7: Mobile

## Sous Android, décrire à partir d'un exemple d'application ne comportant qu'une seule activité, les diverses manières de programmer la réponse à un événement graphique et la manière de gérer un composant comme une ListView selon le modèle MVC.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

**Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquementUne image contenant texte, capture d’écran, Police

Description générée automatiquement**

**This** représentant le contexte dans lequel on se trouve(donc la page)

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

# Question 8: Mobile

## Sous Android, décrire à partir d'un exemple d'application ne comportant qu'une seule activité l'utilisation d'un AsyncTask qui accède à un serveur vendant des actions. Expliquer le fonctionnement du modèle MVC pour les composants graphiques d'Android à partir de l'affichage par l'activité des actions achetées.

Cette classe générique est un "helper", une classe fournie pour faciliter la manipulation de threads au comportement stéréotypé. On peut lui spécifier les paramètres à lui passer en entrée et sortie ainsi que ce qu’elle fera une fois lancée.

Elle permet de créer une tâche en fond qui va s’effectuer en arrière-plan et qui sera capable de communiquer sur son état d’avancement. Les tâches à effectuées seront principalement des communications réseaux, appels base de données et l’avantage de celle-ci est qu’elle prend en charge toute les contraintes possibles entre le réseau et l’UI (comme le wait, notify, synchronized, moniteurs, etc).

Elle possède 4 fonctions mais une seule nous importera ici car c’est la seule qui est threadée : doInBackground(), qui va contenir le gros de la tâche à faire et pouvoir suivre l’avancement avec une méthode : publishProgress(). Elle possède aussi onPreExecute() et onPostExecute() assez explicites sur leur rôle dans cette méthode.

Pour récupérer les infos sur l’UI, on utilise onProgressUpdate().

Dans l’UI on veut un composant comme un bouton pour démarrer notre traitement et une barre de progression. On met sur écoute notre bouton (onClickListener), lors du clic on va instancier un objet de la classe qui fera le traitement et qui héritera d’AsyncTask.

Donc on lance la méthode doInBackground() et on réalise la tâche, appel réseau, requête bd, ou des traitements lourds en somme. On peut appeler la méthode publishProgress() en précisant un entier(pourcentage) afin de mettre à jour l’UI.

Visualisation :

****

# Question 9: Crypto

## Expliquer comment une communication réseau est sécurisée par SSL/TLS (SSL-Record), une fois le SSL-Handshake réalisé (donc sans le décrire). Pourquoi parle-t-on de "sous-protocole(s)" ? Qu'entend-on par "variables de session" et "variable de connexion".

Le protocole SSL (Secure Socket Layer) est bien connu, avec des algorithmes publics, mais la clé reste secrète. Il permet une communication sécurisée entre serveurs et clients, avec authentification mutuelle et chiffrement. Le processus commence par un handshake asymétrique, suivi d'une communication symétrique authentifiée avec HMAC. SSL s'intercale entre la couche applicative (comme HTTP) et la couche de transport (TCP/IP), reconnaissable par le port utilisé (443 pour HTTPS).

Le stack TCP/IP client classique comprend une trame du protocole applicatif découpée, puis le chiffrement SSL est inséré entre HTTP et TCP. Du côté serveur, la couche Ethernet vérifie l'intégrité avec un checksum. ICMP gère les erreurs au niveau IP, tandis que TCP surveille les ACK. SSL, en tant que sous-protocole, utilise SSL Alert pour gérer ses erreurs.

Le processus SSL comprend plusieurs étapes : découpage, compression, authentification avec HMAC, chiffrement avec DES, et ajout d'un header SSL. Le handshake asymétrique initie l'échange de clés secrètes. Du côté serveur, la réception implique la vérification des headers, le décryptage avec la clé partagée, la vérification HMAC, la décompression, et la reconstitution de la trame pour le protocole applicatif.

Le protocole SSL-Alert gère les erreurs avec un code d'erreur, signalant si c'est une warning ou une erreur fatale. Le serveur répond en utilisant une troisième clé (Kh') pour authentifier les messages envoyés au client, ajoutant une complexité avec trois clés échangées lors du handshake entre le client et le serveur (Kh' pour le serveur, Kh pour le client).

# Question 10: Crypto

## Décrire en détails le SSL-Handshake. En particulier, expliquer en quoi on peut considérer qu'il est adaptable à différentes situations. Pourquoi parle-t-on de "sous-protocoles" ?

**On l’a vu, lors du SSL-Record, on utilise 3 clés. Ces 3 clés sont partagées donc connues à la fois par le client et par le serveur.**

**Les 3 clés sont composées :**

* **Clé Publique du Serveur -> permet au client de crypté à destination du Serveur**
* **Clé Publique du client -> permet au serveur de crypté à destination du client**
* **Clé Secrète (session) -> Permet de crypté en asymétrique**

Comme dit plus haut, on peut dire que le SSL-Handshake est un sous-protocole, car au même titre que SSL-Record et SSL-Alert que les deux utilisent, ce sont tous des composants de SSL, on parle bien de « sous-protocole ».

Lors du SSL-Record, trois clés secrètes partagées entre le client et le serveur sont utilisées. Le SSL-Handshake, en tant que sous-protocole, permet l'authentification, la négociation des algorithmes de chiffrement et HMAC, ainsi que la négociation des clés symétriques. Les phases du SSL-Handshake sont les suivantes :

Phase 1 : Les paramètres de sécurité Le client envoie un message client\_hello avec des informations telles que la version de SSL, des nombres aléatoires, la date-heure, etc. Le serveur répond avec server\_hello en choisissant la stratégie optimale. Les nombres aléatoires et algorithmes sont en mémoire, rien n'est écrit sur disque.

Phase 2 : Authentification du serveur & génération des clés Le serveur envoie son certificat (Certificate) pour authentification. Si le certificat est valide, le serveur est authentifié. Dans certains cas, le serveur peut demander un certificat au client (certificate\_request). Cette phase peut contenir plusieurs messages : certificate, server\_key\_exchange, et certificate\_request. Le server\_hello\_done indique la fin de cette phase.

Phase 3 : La génération du « master secret » + certificat client Le client reçoit le certificat du serveur, le vérifie, et génère un premaster secret. Ce premaster est crypté avec la clé publique du serveur et envoyé. La génération du master secret implique des échanges basés sur les nombres de session, les nombres aléatoires client et serveur, permettant la dérivation des clés Kh, Kh’, et Kc.

Phase 4 : Enregistrement et validation des spécifications négociées Le message change\_cipher\_spec enregistre les algorithmes et les clés sur le disque côté client et côté serveur. Une confirmation supplémentaire se fait par l'échange de messages finished, qui contiennent des hashages de tous les éléments échangés depuis le début. Si les hashages correspondent, le handshake est terminé, et la communication peut commencer avec le SSL-Record utilisant les clés générées.

# Question 11: Crypto

## Expliquer le code Java utilisant SSL à substituer au code classique d'un serveur TCP/IP dont on désire sécuriser les communications. En particulier, décrire les keystores nécessaires au bon fonctionnement de ce code et leur utilisation, côté serveur et côté client. Pourquoi voit-on intervenir dans javax.net.\* un si grand nombre d'interfaces et de classes abstraites ?

Au niveau de la programmation SSL, on utilise des SSLSocket et des SSLServerSocket en remplacement des Socket et ServerSocket habituels. Le processus de handshake doit être programmé, et pour ce faire, le serveur doit posséder un certificat avec une paire de clés publique et privée, nécessitant l'utilisation de KeyStore. On distingue deux types de tiroirs dans KeyStore : les tiroirs simples contenant uniquement un certificat, et les Key Entry, contenant les clés privées associées aux certificats. Deux KeyStores sont créés, un pour le client et un pour le serveur, chacun contenant une KeyEntry et une Trusted Certificate Entry.

Les SSLSocket et SSLServerSocket, héritant de Socket et ServerSocket respectivement, sont des classes abstraites. Des factorys sont nécessaires pour créer ces sockets. Pour ce faire, des classes dérivées de SSLSocketFactory et SSLServerSocketFactory, qui sont à leur tour abstraites, doivent être créées en implémentant le handshake. Ces classes mettent à disposition les méthodes CreateSocket() et CreateServerSocket(), permettant la construction des sockets SSL.

Pour obtenir ces SSLSocketFactory, le SSLContext est utilisé. Cet objet, dépositaire des opérations du handshake, possède les méthodes getSocketFactory() et getServerSocketFactory(). Le SSLContext doit être rempli avec une KeyStore, pour cela, une instance de KeyStore est obtenue avec getInstance(). Ensuite, le KeyStore est rempli avec le contenu d'un fichier .keystore créé à l'aide de l'outil KeyTool.

En résumé, le processus de programmation SSL implique la création de SSLSocket et SSLServerSocket en remplacement des sockets classiques, l'utilisation de KeyStore pour gérer les certificats et clés, et la mise en place du handshake avec SSLContext, KeyManager, TrustManager, KeyManagerFactory, et TrustManagerFactory.

# Question 12 (Android):

Expliquer ce qu'est un « intent » et les différents mécanismes qui permettent de les mettre en œuvre.

**Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement**

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

**Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement**

# Question 13:

## Expliquez et illustrez les différences (améliorations, mécanisme, …) entre les versions TLS précédentes la 1.3 et TLS 1.3.

****

**Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement**

**Une image contenant texte, capture d’écran, Police, Parallèle

Description générée automatiquement**

**Echanges pour TLSv1.3:**

**Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement**

**Une image contenant texte, capture d’écran, Police

Description générée automatiquement**

**Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement**

# Question 14:

## Termes à définir brièvement, remettre dans son contexte, acronyme de … , utilité(s), rôle(s), exemple d’applications.

**1) MFS**

***Message Format Standard*** **=** **Utilisé en Java mail pour l’envoi de mail sans pièce attachée, tout en spécifiant la nature des données transmises.**

**Application : Détermine un message simple (SimpleMessage) et sans pièce attachée.**

**2) RFC822**

**C’est une norme qui définit le format des messages en Java mail, cela spécifie comment les entêtes, contenu et adresses doivent être formatés. Mais elle ne comprend pas la spécification MIME.**

**3) SMTP**

***Simple Mail Transfer Protocol* = Utilisé en Java mail, c’est un protocole utilisé pour l’envoi d’email vers le serveur de messagerie du destinataire.**

**Application : envoyer des e-mails**

**4) POP3**

***Post Office Protocol version 3* = Conçu pour récupérer les e-mails du serveur de messagerie et les transférer vers le client de messagerie de l'utilisateur (Le pop3 va supprimé les emails du serveur de messagerie, une fois envoyé au client).**

**Application : récupérer des e-mails**

**5) IMAP**

***Internet Message Access Protocol* = Utilisé pour synchroniser les e-mails entre le serveur de messagerie et le client de messagerie de l'utilisateur (les emails restent tous sur le serveur).**

**Application : recevoir des e-mails**

**6) APK**

***Android Package* = Un package d’installation pour les applications mobiles. Il contient le code, les configurations, les ressources pour exécuter les applications Android. C’est une archive qui regroupe l'ensemble des composants de l'application.**

**7) RMTA** **couple(MDA/MTA)**

**Receiving Mail Transport Agent = Reçois les emails qui proviennent d’autres domaines. Si le message est destiné à son domaine il utilise un MDA pour les mettre dans la bonne Mailbox, si non, il retransfère le mail vers un autre domaine.**

**8) FMTA couple(MDA/MTA)**

**Forwarding Mail Transport Agent = Le FMTA lit le domaine du destinataire. Si c’est un autre domaine il utilise le MDA pour transférer le mail en dehors de son domaine, sinon, le MDA se contente de transférer le mail dans la mailbox du destinataire. Des headers sont ajoutés au fur et à mesure du transfert du message par les MTAs.**

**9) MDA**

**Mail Delivery Agent = Va rechercher les mails dans une file d’attente et les envois au meilleurs MTAs possible / ou dans la mailbox si elle est gérée par le serveur.**

**Application : Son rôle est d’envoyé ses courriers à des MTAs ou au MailBox.**

**10) MRA**

**Mail Retreival Agent = Va récupérer les messages depuis la boîte aux lettres (côté serveur) de l'utilisateur lorsque celle-ci n'est pas accessible directement par le RMUA.**

**Application : S’utilise quand le RMUA ne peux pas accéder à sa boite mail directement. Alors il envoie des requêtes pop3/imap au MRA, qui va les lui renvoyé.**

**11) RMUA**

**Receiving Mail User Agent = Lit les messages reçus via les protocoles POP3 ou IMAP. (Requête POP3/IMAP envoyé à un MRA).**

**Application : Via une app, récupère via le protocole POP3/IMAP les emails reçus dans la "mailbox" du Serveur et les transfère.**

**12) SMUA**

**Sending Mail User Agent = Application(Gmail/Outlook) chargée d’envoyer des mails via un serveur de messagerie.**

**13) R.java**

**Un fichier "R.java" est automatiquement généré lors de la compilation d’un projet Android. Ce fichier contient des classes Java avec des identifiants statiques correspondant aux ressources définies dans le dossier "res“.**

**Application : On peut utiliser ces références à ces ressources définies dans le code de l’application.**

**14) .arsc**

**Un fichier binaire contenant des ressources compilées inclut dans l’APK, utilisé par le système d'exploitation pour accéder rapidement aux ressources nécessaires sans avoir à traiter les fichiers XML originaux, ce qui peut améliorer les performances.**

**Application : Compilation et exécution d’applications Android.**

**15) JVM/ART**

***Java Virtual Machine* = C’est une machine virtuelle conçue pour exécuter des programmes Java.**

**Application : Permet aux programmes Java de s'exécuter de manière indépendante de l'architecture matérielle sous-jacente.**

***Android RunTime* = C’est un environnement d’exécution utilisé par Android, il a remplacé la Dalvik virtual machine (DVM).**

**Application : Agit au moment de l’exécution d’une application, réduit la consommation de mémoire mais rallonge le temps lors de l’installation de celle-ci.**

**16) FindViewById**

**Méthode utilisée en Android pour rechercher un composant via son id dans la page .xml de notre activité ou fragment et ainsi pouvoir l’utiliser dans la logique du code de notre activité.**

**Application : Pouvoir changer la valeur, le chaine ou encore l’image affichée à l’écran et/ou les utilisées pour le bon fonctionnement de l’application.**

**17) Intent**

**En Android, cet objet est principalement utilisé pour démarrer des composants de l’application (activités, fragments, services).**

**18) AsyncTask**

**Utilisé en Android pour effectuer des tâches en fonds tout en utilisant le thread UI. Elles ont pour but de simplifié la création de petites tâches asynchrone.**

**Application :** **S’applique à toutes actions pouvant bloquer ou stopper le programme principal (Requête réseau (TCP / http), lecture de fichier …).**

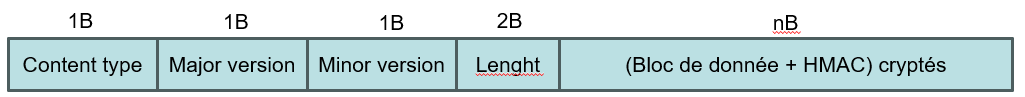
**19) SSL-Record :**

***Secure Sockets Layer – Record : La couche Record utilise la clé secrète (de session) pour chiffrer les blocs de données et créer un HMAC pour chaque bloc chiffré.***

***Sa structure comprend un header SSL composé de :   
- Content Type (change\_cipher\_spec, alert, handshake, prot applicatif (http, ftp…)***

**- Major SSL Version**

**- Minor SSL Version**

**- length (longueur du bloc chiffré + hmac)  
  
**

**20) SSL-Handshake**

***Secure Sockets Layer - Handshake* = Processus par lequel un client et un serveur établissent une connexion sécurisée en utilisant le protocole SSL ou son successeur, TLS (Transport Layer Security). Le processus comprend l’échange des algorithmes utilisées, de 2 clés publiques (1 client et 1 serveur), ainsi que la création d’1 clé de session (utilisée ensuite pour crypter les données). Sa création se fait sur base d’un preMaster échangé entre le client et le serveur.**

**Application : Sécurisé une connexion pour pouvoir par la suite crypter les données envoyées sur le réseau.**

**21) Keystore**

**Emplacement sécurisé utilisé pour stocker et gérer des couples clés privées/publiques(+ certificats), des clés secrètes (cryptage synchrone), et des certificats/clé publique de confiances.**