Introduction

1. Protocole FTP
2. Generalite
3. Historique
4. Modele FTP
5. Fonctionnement du FTP
6. Presentation des donnees
7. Etablissement du canal
8. Mode de transmission
9. Recuperation des erreurs et retransmission
10. Fonction de transfert des fichiers
11. Les commandes FTP
12. Les reponses FTP
13. Installation et configuration du FTP
14. Installation
15. Configuration
16. Les limites du FTP
17. Protocole SSH
18. Generalite sur le SSH
19. Methode de chiffrement
20. Chiffrement symetrique
21. Chiffrement asymetrique
22. Fonctionnement du SSH
23. Installation et configuration du ssh
24. Sécurisation du transfert des fichiers avec le SSH
25. Client
26. MySecureShell
27. Protocole FTP
28. Généralité
29. Historique

La première proposition de mécanisme de transfert de fichiers a été faite 1971 et avait été développée pour une application sur les hôtes du MIT (*Massachusetts Institute of Technology*). MIT-Project MAC, en juin 1971 proposait un protocole de niveau utilisateur pour le transfert de fichiers entre ordinateurs (y compris des terminaux IMPs). Une révision de celui-ci redonnait un état du FTP pour évolution ultérieure, tandis que la BBN en décembre 71 suggérait encore d'autres modifications. L'usage d'une transaction "Set Data Type" a été proposée par MIT-Project MAC en Janvier 1982. Le File Transfer Protocol était désormais défini comme un protocole de transfert de fichiers entre des hôtes d'un ARPANET, et dont la fonction première était définie comme le transfert efficace et fiable entre des hôtes pour profiter de l'utilisation d'une capacité de stockage de données distante. Certains correctifs à certaines erreurs, certains points, et notions au protocole ont été ajoute dans les années 72.

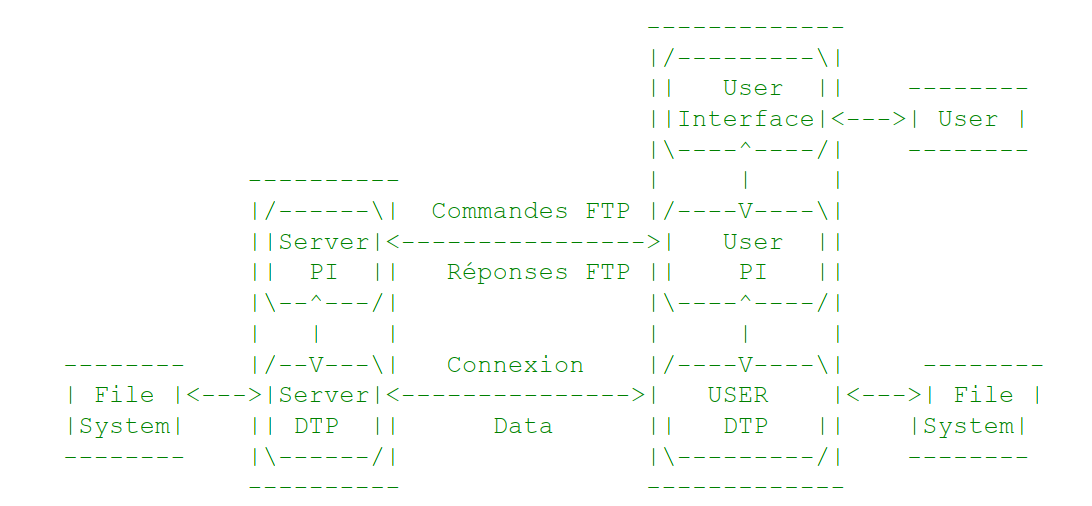
Depuis Juillet 1973, des changements considérables sont intervenus, mais la structure globale est restée la même. BBN a été publié comme une nouvelle spécification "officielle" pour refléter certains changements. Cependant, de nombreuses implémentations basées sur l'ancienne spécification n'étaient pas remises à jour. En 1974, les BBN et UCSB ont apporté de nouveaux commentaires à propos de FTP. Motivée par le passage du NCP (Network Communication Protocol) à TCP comme protocole sous-jacent, un phœnix est né de nouveau à partir de tous les efforts ci-dessus en 1980 comme une nouvelle spécification de FTP basée sur le protocole réseau TCP. Cette édition de la spécification FTP est écrite pour corriger quelques erreurs par la suite, tout en étendant les explications de certaines fonctionnalités du protocole, et enfin en ajoutant la définition de quelques commandes supplémentaires.

1. Modèles FTP

Les objectifs de FTP sont multiples parmi lesquels on :

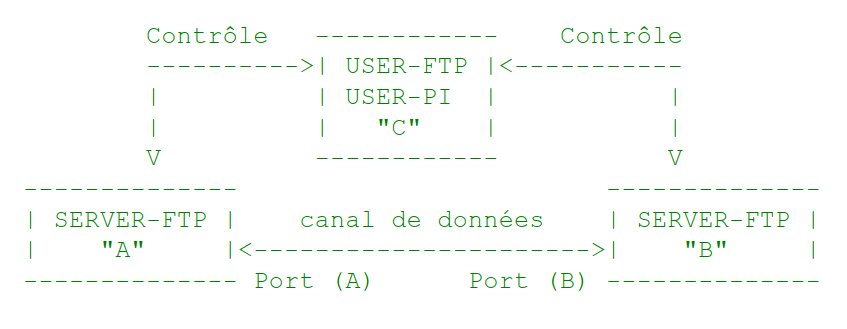
* Promouvoir le partage de fichiers (programmes informatiques et/ou données)
* Faciliter l'utilisation indirecte ou implicite (via des programmes) d'ordinateurs distants
* Prémunir l'utilisateur contre les variations de formats de stockage de données entre les différents hôtes
* Transférer les données d'une façon efficace et fiable. FTP, bien que directement utilisable par un utilisateur depuis un terminal, est néanmoins conçu essentiellement pour être utilisé par des programmes.

FTP, bien que directement utilisable par un utilisateur depuis un terminal, est néanmoins conçu essentiellement pour être utilisé par des programmes. Dans le modèle décrit à la figure 1, l'interpréteur de protocole utilisateur (USER-PI) instaure le canal de contrôle. Ce circuit de communication utilise le protocole Telnet. A l'instauration de cette connexion, des commandes FTP standard sont générées par le USER-PI et transmises au processus serveur via le canal de contrôle. (L'utilisateur pourra néanmoins établir une liaison de contrôle directe avec le SERVER-FTP, à partir d'un terminal TAC par exemple, et générer les commandes standard indépendamment, en se substituant au processus USER-FTP). Des réponses standardisées sont émises en retour par le SERVER-PI au USER-PI via le canal de contrôle alors



*Fig1: Modèle d'usage de FTP*

établie. Les commandes FTP spécifient les paramètres du canal de données (port de données, mode de transfert, type pour la représentation, et structure des données) ainsi que la nature du fonctionnement des systèmes de fichiers (enregistrement, lecture, ajout, suppression, etc.). Le USER-DTP ou son délégué se mettra en "écoute" sur le port de données spécifié, et le serveur instaurera le canal de données et effectuera le transfert de fichiers selon les paramètres spécifiés. Il doit être noté que le port de  
données n'est pas nécessairement sur le même hôte que celui qui a émis les premières commandes FTP par son canal de contrôle, bien que l'utilisateur ou le USER-FTP doive continuer à assurer "l'écoute" sur le port spécifié. Il doit être ici signalé en outre que le canal de données mis en place peut servir simultanément à la lecture et à l'écriture de données. Une autre situation peut consister en un utilisateur qui souhaite transférer des fichiers entre deux hôtes, les deux étant des hôtes distants différents de celui de l'utilisateur. L'utilisateur établit alors un canal de contrôle vers chacun des deux serveurs et utilise ces canaux pour créer un canal de données entre ces deux hôtes. De cette façon, les informations de contrôle passent par le USER-PI bien que les données soient transmis ente deux processus serveurs de transfert. Ce qui suit est un modèle de cette interaction entre serveurs.



FTP s'appuie sur le protocole Telnet pour le dialogue du canal de contrôle. Ceci est  
effectif en deux sens: premièrement, le USER-PI ou le SERVER-PI devront suivre les  
règles du protocole Telnet directement dans leur propres procédures; ou bien, le USERPI ou le SERVER-PI peuvent faire appel à un module Telnet existant et disponible dans le système d'exploitation. La facilité d'implémentation, les principes de réutilisabilité, et la programmation modulaire font pencher en faveur de la deuxième solution. L'efficacité et l'indépendance vis à vis de la plate-forme sont des arguments en faveur de la première. En pratique, FTP n'utilise qu'un tout petit sous ensemble du protocole Telnet, et de ce fait, la première approche n'induit pas un travail de programmation insurmontable.

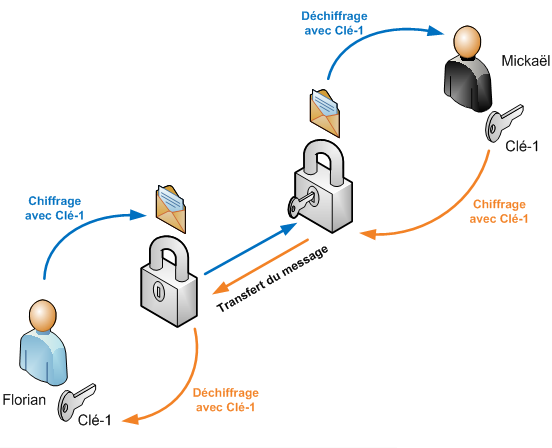
1. Fonctionnement
2. Protocole ssh
3. Généralités sur ssh

**SSH** (Secure Shell) est un protocole de communication sécurisé. Apres que l’université de technologie d’Helsinki ait été victime d’une attaque par renfilage de mot de passe (cela consiste à créer un faux réseau wifi se faisant passer pour un vrai réseau wifi public, pour écouter les communications afin de récupérer le contenu transmis ). La première version de SSH a été développé en 1995 par TATUN YLONEN, un professeur Finlandais qui, à l’époque souhaitait sécuriser les connexions distantes vers un serveur Unix. Dans cette version, le serveur écoute sur le port 22 et attend que le client initie la connexion. Ce client génère une clé qui va être envoyée au serveur et qui va permettre le chiffrement des données. Grâce à ce protocole, on peut ainsi accéder à des machines à distance et interagir avec elles en ligne de commande de manière sécurisée. Cette version est cependant obsolète à cause des problèmes de sécurité qui subsistent dans la **vérification de l’intégrité des données**, C’est ainsi qu’est née la version 2.0 de ssh, en Janvier 2006 a **l’IETF** (Internet Engineering Task Force). Grace à cette nouvelle version, de nouveaux algorithmes et des services de transfert de fichiers (SFTP, le Tunneling, le port forwarding, l’auhtentification via des clés privées sécurisées et bientôt l’utilisation des certificats x.509) ont été intégrés.

**2) Méthodes de chiffrement**

**a) Chiffrement symétrique**

Son principe est le suivant : si **A** veut envoyer un message confidentiel à **B**, **A** et **B** doivent d'abord posséder une même clé secrète. **A** crypte le message avec la clé secrète et l'envoie à **B** sur un canal qui n'est pas forcément sécurisé. **B** décrypte le message grâce à la clé secrète. Toute autre personne en possession de la clé secrète peut décrypter le message.



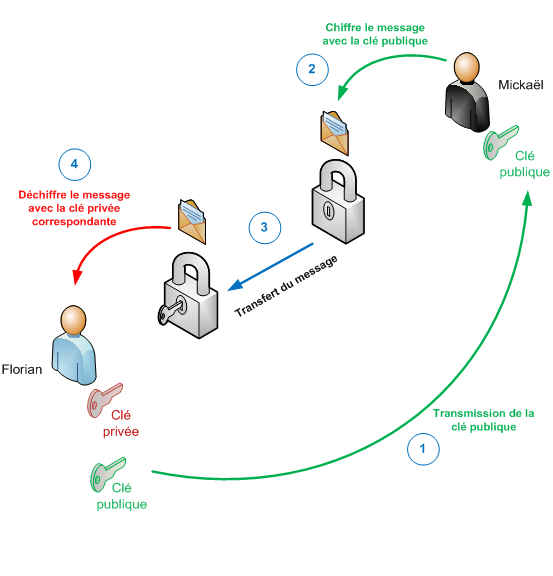
A travers ce schéma on voit bien que Florian chiffre un message avec la clé « **Clé-1** » puis le transfert à Mickaël qui le déchiffre avec la clé « **Clé-1** » qui est la même que celle dont dispose Florian. Il en est de même dans le sens inverse c'est-à-dire de Mickaël vers Florian. Ce qui montre bien que lors de l’utilisation d’une clé symétrique, la même clé permet de chiffrer et déchiffrer.

**Avantage et inconvénient**

L’avantage d’utiliser ce type de chiffrement est la simplicité de fonctionnement ce qui rend le [système](https://www.it-connect.fr/cours-tutoriels/administration-systemes/windows-server/systeme/) rapide et performant. Toutefois, si vous perdez ou distribuez malencontreusement votre clé, la personne qui l’a récupère pourra chiffrer des messages mais surtout, et c’est plus inquiétant, déchiffrer les messages chiffrés à partir de cette clé.

1. **Chiffrement asymétrique**

SSH utilise la cryptographie asymétrique RSA ou DSA. En cryptographie asymétrique, chaque personne dispose d'un couple de clé : une clé publique et une clé privée. La clé publique peut être librement publiée tandis que la clé privée doit rester secrète. La connaissance de la clé publique ne permet pas d'en déduire la clé privée. Si la personne **A** veut envoyer un message confidentiel à la personne **B**, **A** crypte le message avec la clé publique de **B** et l'envoie à **B** sur un canal qui n'est pas forcément sécurisé. Seul **B** pourra décrypter le message en utilisant sa clé privée.

****

A travers ce schéma on voit que Florian possède une paire de clés asymétriques dont une clé privée et une clé publique, ce sera lui qui recevra les messages et qui sera les seuls à pouvoir les déchiffrer. Dans un premier temps, Florian transmet sa clé publique à Mickaël pour qu’il puisse lui transmette des messages chiffrés. Ensuite, Mickaël chiffre un message avec la clé publique reçu puis transfert le message à Florian. Pour finir, Florian déchiffre le message grâce à sa clé privée qui est la seule à pouvoir déchiffrer le message.

**Avantage et inconvénient**

L’avantage de ce type de chiffrement c’est qu’on peut distribuer la clé publique sans risquer que les messages soient déchiffrés avec, étant donné que seul la clé privée permet de déchiffrer ses messages. L’inconvénient c’est que ça fonctionne en mode unidirectionnel uniquement à moins d’avoir deux paires de clés, et que le mode de fonctionnement est plus complexe que l’utilisation de clés symétriques et donc moins performant. De plus, il faut bien prendre conscience qu’il est impossible de retrouver une clé privée à partir d’une clé publique.

* + 1. **Fonctionnement du ssh**

Un serveur SSH dispose d'un couple de clés RSA stocké dans le répertoire **/etc/ssh/** et généré lors de l'installation du serveur. Le fichier **ssh\_host\_rsa\_key**contient la clé **privée** et a les permissions **600**. Le fichier **ssh\_host\_rsa\_key.pub** contient la clé **publique** et a les permissions **644**. Nous allons suivre par étapes l'établissement d'une connexion SSH :

1. Le serveur envoie sa clé publique au client.
2. Le client génère une clé secrète et l'envoie au serveur, en cryptant l'échange avec la clé publique du serveur (cryptographique asymétrique). Le serveur décrypte la clé secrète en utilisant sa clé privée, ce qui prouve qu'il est bien le vrai serveur.
3. Pour le prouver au client, il crypte un message standard avec la clé secrète et l'envoie au client. Si le client retrouve le message standard en utilisant la clé secrète, il a la preuve que le serveur est bien le vrai serveur.
4. Une fois la clé secrète échangée, le client et le serveur peuvent alors établir un canal sécurisé grâce à la clé secrète commune (cryptographie symétrique).
5. Une fois que le canal sécurisé est en place, le client va pouvoir envoyer au serveur le login et le mot de passe de l'utilisateur pour vérification. Le canal sécurisé reste en place jusqu'à ce que l'utilisateur se déloge.

La seule contrainte est de s'assurer que la clé publique présentée par le serveur est bien sa clé publique... sinon le client risque de se connecter à un faux serveur qui aurait pris l'adresse IP du vrai serveur (ou toute autre magouille). Une bonne méthode est par exemple de demander à l'administrateur du serveur quelle est le *fingerprint* de la clé publique du serveur avant de s'y connecter pour la première fois.  **NB : Le** **fingerprint** d'une clé publique est une chaîne de 32 caractères hexadécimaux unique pour chaque clé ; il s'obtient grâce à la commande **ssh-keygen -l**.

**4) Installation et configuration du ssh**

**4.1) Installation sur linux**

## -Mise a jour du gestionnaire de paquets :

## Sudo apt update