Chiffrement et authentification

(*Q2hpZmZyZW1lbnQgZXQgYXV0aGVudGlmaWNhdGlvbgo=*)

Éléments de cryptographie
Connexion sécurisée SSL
Les certificats X509
Les utilitaires SSH
Les réseaux privés virtuels

Fabrice Harrouet
École Nationale d'Ingénieurs de Brest
harrouet@enib.fr
http://www.enib.fr/~harrouet/

Propos

- **Limiter les risques liés à la communication**
  - Confidentialité : données lisibles par quiconque ?
    - Utiliser des algorithmes de chiffrement
  - Intégrité : données modifiées pendant le transport ?
    - Utiliser des algorithmes de hachage (condensé)
  - Authentification : dialogue avec l'entité attendue ?
    - Obtenir un document officiel identifiant l'interlocuteur
  - Non-répudiation : producteur d'un document ?
    - La signature d'un document atteste de son origine
- **Mise en œuvre de *SSL* présentée dans le module *RX***
  - Seuls les principes sont vus ici, pas le code

Éléments de cryptographie

▷ Vocabulaire

Chiffrer **:** transformer à l'aide d'une clef de chiffrement un message en clair en un message incompréhensible sans la clef de déchiffrement

Déchiffrer **:** retrouver à l'aide de la clef de déchiffrement le message en clair à l'origine du message chiffré

Chiffre **:** algorithme utilisé pour le chiffrement

Cryptogramme **:** message chiffré

Décrypter **:** retrouver le message en clair correspondant à un cryptogramme sans connaître la clef de déchiffrement (*"casser"* le code secret)

Cryptographie **:** science de la protection des messages par des clefs

Cryptanalyse **:** science du décryptage

Cryptologie **:** cryptographie et cryptanalyse

~~*Crypter*~~ **:** incorrect, aucune signification !

Éléments de cryptographie

▷ **Symboles usuellement utilisés**

- ◇ Message en clair (*plain-text*) : P
- ◇ Cryptogramme (*cipher-text*) : C
- ◇ Clef de chiffrement (*encryption-key*) : k_e
- ◇ Clef de déchiffrement (*decryption-key*) : k_d
- ◇ Fonction de chiffrement (*encrypt*) : $E(k_e, P) = C$
- ◇ Fonction de déchiffrement (*decryp*) : $D(k_d, C) = P$
- ◇ L'ensemble doit respecter : $D(k_d, E(k_e, P)) = P$
- ◇ Fonction de hachage/condensat (*hash/message-digest*) : $Md(P) = H$

Éléments de cryptographie

- **Robustesse du procédé de chiffrement**
  - Reposant sur le secret des algorithmes utilisés ?
    - Aucune idée de sa robustesse intrinsèque (peu étudié)
    - La découverte fortuite de l'algorithme remet tout en cause
    - Solution à éviter !
  - Reposant sur des algorithmes publiés
    - De nombreux cryptanalistes peuvent en étudier la robustesse
    - Déterminer la probabilité de trouver la clef de (dé)chiffrement
      → selon les proriétés de l'algorithme
      → selon la longueur de la clef
      → par recherche exhaustive (*brute-force*)
      → par corrélation entre cryptogramme et texte clair
      → ...
    - Solutions éprouvées, à privilégier

Éléments de cryptographie

▷ **Exemple : le chiffre de César**

◇ Décaler les lettres de `N` positions dans l'alphabet

◇ À chaque caractère on fait correspondre un autre

○ Clefs de chiffrement/déchiffrement : $k_e = k_d = N$

○ Fonction de chiffrement : $E(k_e, c) = (c + k_e)\%26$

○ Fonction de déchiffrement : $D(k_d, c) = (c - k_d)\%26$

◇ Facile à *décrypter*

○ Par une approche exhaustive (*brute-force*), faible combinatoire

○ Par l'analyse des statistiques d'occurence de chaque caractères

◇ ex : `PETIT CURIEUX !` $\overset{rot13}{\longleftrightarrow}$ `CRGVG PHEVRHK !`

Éléments de cryptographie

- **Chiffrement symétrique ou** *"à clef secrète"*
  - Une clef unique sert au chiffrement au déchiffrement ($k_e = k_d$)
    - Une chaîne de *bits* de longueur choisie
    - Seuls les détenteurs de cette clef k_{sec} peuvent se comprendre ($D(k_{sec}, E(k_{sec}, P)) = P$)
  - Le rapport temps-de-(dé)chiffrement/robustesse est satisfaisant
    - Plus une clef est longue plus elle est robuste
    - Traitement rapide des messages même avec une bonne robustesse
  - Ex : *DES*, *Rijndael*, *Blowfish* ...
  - Problème essentiel : comment transmettre la clef à un interlocuteur ?
    - Une autre personne ne doit pas l'intercepter !
    - Si c'est le cas, elle peut espionner les échanges et y participer

Éléments de cryptographie

▷ **Chiffrement asymétrique ou** *"à clef publique"*

◇ Une paire de clefs complémentaires est générée ($k_e \neq k_d$)

○ Une clef *publique* (k_{pub}) : elle doit être largement publiée

○ Une clef *privée* (k_{priv}) : elle doit être gardée très secrètement

○ Ce qui est chiffré par l'une ne peut être déchiffré que par l'autre
($D(k_{priv}, E(k_{pub}, P)) = P$ et $D(k_{pub}, E(k_{priv}, P)) = P$)

○ Une clef ne peut pas déchiffrer ce qu'elle a chiffré
(l'autre est nécessaire pour cette opération)

○ nb : k_{pub} peut se déduire de k_{priv} mais pas l'inverse (heureusement !)

◇ Le rapport temps-de-(dé)chiffrement/robustesse est peu satisfaisant

○ Plus les clefs sont longues plus elles sont robustes

○ Traitement lent des messages même avec une faible robustesse

◇ Ex : *DSA*, *RSA*, *ElGamal* ...

Éléments de cryptographie

- **Exemple : génération de clefs *RSA* avec l'outil `openssl`**
  - Générer une *clef privée* ***RSA*** de 1024 bits

    `$ openssl genrsa -out key.pem 1024`
  - Générer la *clef publique* associée à la *clef privée* précédente

    `$ openssl rsa -in key.pem -pubout -out pubkey.pem`

Éléments de cryptographie

▷ **Exemple : (dé)chiffrement asymétrique avec l'outil `openssl`**

◇ `-encrypt` (`-decrypt`) chiffre (resp. déchiffre) avec la clef publique (resp. privé)

◇ `-sign` (`-verify`) chiffre (resp. déchiffre) avec la clef privée (resp. publique)

◇ nb : `-pubin -inkey pubkey.pem` peut être remplacé par `-inkey key.pem` (puisque `pubkey.pem` se déduit de `key.pem`, mais pas l'inverse !!!)

```
$ echo "C'EST CLAIR" | openssl rsautl -pubin -inkey pubkey.pem -encrypt > cryptogram
$ cat cryptogram
...A=J..#l#..?..}...7G{h..zMt..H..md.&>....1.."×...;$
$ cat cryptogram | openssl rsautl -inkey key.pem -decrypt
C'EST CLAIR

$ cat cryptogram | openssl rsautl -inkey another_private_key.pem -decrypt
RSA operation error
$ cat cryptogram | openssl rsautl -pubin -inkey pubkey.pem -decrypt
A private key is needed for this operation

$ echo "C'EST CLAIR" | openssl rsautl -inkey key.pem -sign > cryptogram
$ cat cryptogram
..Z...2./...m.D..C....%...÷(,N.okT.e....Iu....\..)y&....O..j.. – L..q1D$
$ cat cryptogram | openssl rsautl -pubin -inkey pubkey.pem -verify
C'EST CLAIR
```

Éléments de cryptographie

- **Chiffrement asymétrique ou** *"à clef publique"*
  - Exemple : *Alice* envoie un message confidentiel à *Bob*
    - *Alice* chiffre son message avec la clef publique de *Bob*
      $C = E(k_{pubB}, P)$ (cette clef est librement accessible)
    - Elle transmet le message chiffré à *Bob*
      (l'interception de ce message est inexploitable)
    - *Bob* utilise sa propre clef privée pour déchiffrer le message reçu
      $D(k_{privB}, C) = P$ (c'est le seul à pouvoir le faire)
  - Pour répondre : démarche réciproque
    - *Bob* chiffre son message avec la clef publique d'*Alice* k_{pubA}
    - *Alice* utilise sa clef privée k_{privA} pour déchiffrer le message reçu
  - Intéret du procédé : pas besoin de transmettre une clef secrète
    - Les oreilles indiscrètes ne comprennent rien aux échanges

Éléments de cryptographie

▷ **Performances des clefs symétriques et asymétriques**

◇ Asymétrique beaucoup plus lent que symétrique (à robustesse équivalente)

- Facteur $\simeq 100$ entre *RSA* logiciel et *DES* logiciel
- Facteur $\simeq 1000$ entre *RSA* logiciel et *DES* matériel

◇ Longueurs de clefs pour une résistance équivalente à la *brute-force* (approximatif, cf `http://www.keylength.com/`)

de 2007 à :	2010	2016	2026	2036	
symétrique :	80	96	112	128	bits
asymétrique :	1248	1776	2432	3248	bits

Dépend fortement des moyens (budget) dont on dispose pour *décrypter*

◇ Alors . . . quel procédé utiliser ? $\rightarrow$ les deux à la fois !

- Chiffrement asymétrique pour s'échanger une clef symétrique temporaire
- Puis chiffrement symétrique pour (dé)chiffrer les messages

Éléments de cryptographie

▷ **Intégrité des messages et calcul de condensat (hachage, *hash*)**

◇ Produire un motif de longueur fixe caractérisant une message en clair

- ○ Calcul très rapide (beaucoup plus que le chiffrement)
- ○ Irréversible : impossible de retrouver P depuis $Md(P)$
- ○ Déterministe : $P = P' \Rightarrow Md(P) = Md(P')$
- ○ Infiniment improbable d'avoir $Md(P) = Md(P')$ avec $P \neq P'$
- ○ Messages très proches $\rightarrow$ condensats très différents
- ○ ex : *SHA-1*, *MD5* ...

◇ Le condensat accompagnant un message permet d'en vérifier l'intégrité

- ○ Envoi de $(P, H)/H = Md(P) \rightarrow$ réception de (P', H)
- ○ Si $Md(P') = H$ le message P n'a pas été altéré

◇ Problème : H n'a-t-il pas lui-même été altéré ? (aucune signature ici !)

- ○ Un intermédiaire a pu remplacer (P, H) par $(P', H')/H' = Md(P')$

Éléments de cryptographie

▷ Signature numérique

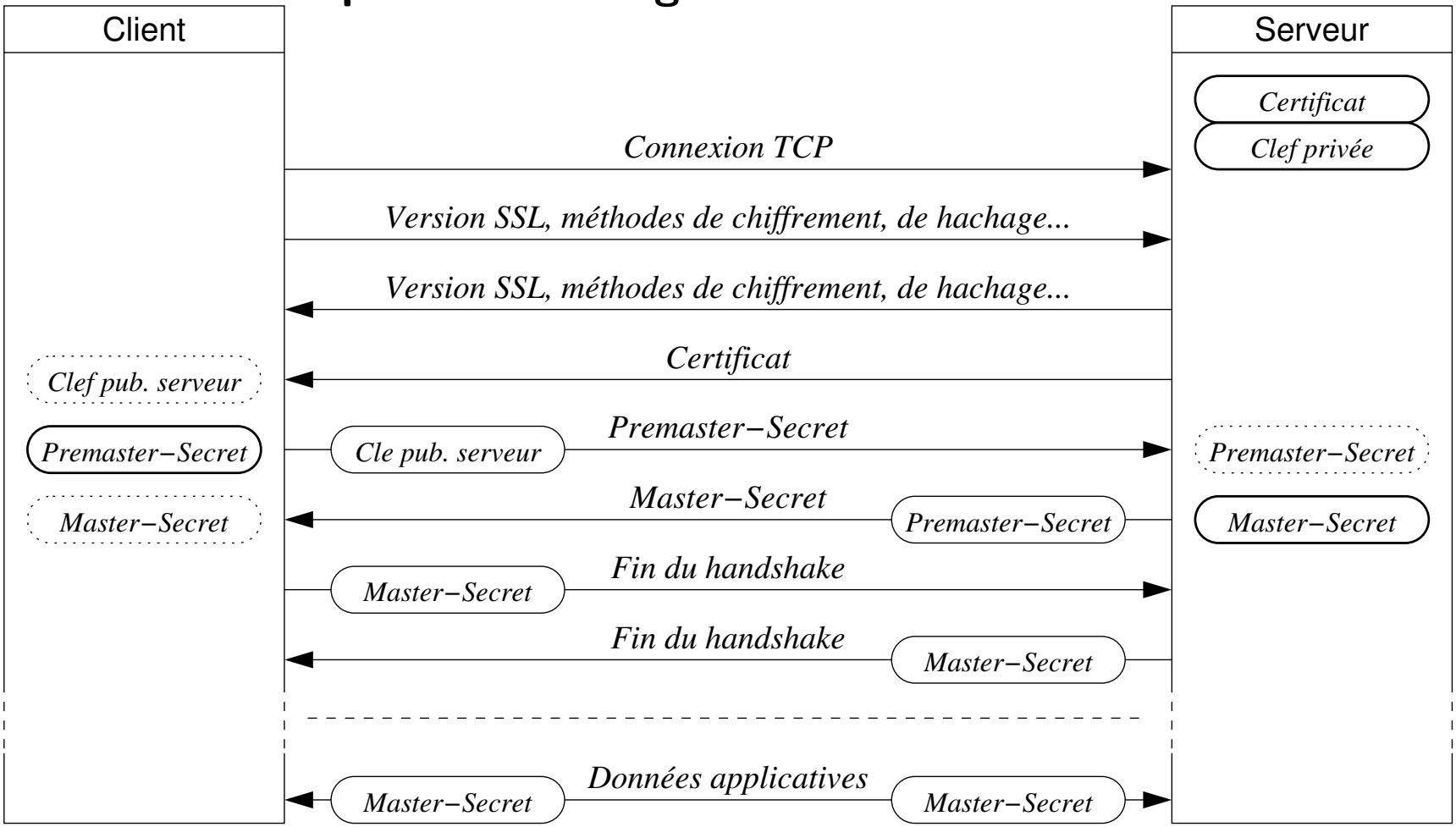
- ◇ S'assurer que le message n'est pas modifié durant son transport
- ◇ S'assurer que seul l'émetteur déclaré a pu le produire
- ◇ Chiffrement avec la clef privée → déchiffrement avec la clef publique
- ◇ Exemple : *Alice* envoie un message signé à *Bob*
  - ○ *Alice* envoie $(P, E(k_{privA}, Md(P)))$ à *Bob*
  - ○ *Bob* reçoit (P', H')
  - ○ $D(k_{pubA}, H') = Md(P') \Rightarrow P' = P$ (le message n'est pas altéré)
  - ○ Ceci indique à *Bob* que c'est bien *Alice* qui a produit le condensat (sa clef publique ne déchiffre que ce qui provient de sa clef privée)
- ◇ nb : le message n'est pas confidentiel ici
  - ○ Si besoin, *Alice* transmet $P' = E(k_{pubB}, E(k_{privA}, P))$
  - ○ *Bob* calcule alors $D(k_{pubA}, D(k_{privB}, P')) = P$

Connexion sécurisée *SSL*

- **Négociation de la connexion sécurisée (*handshake*)**
  - Chiffrement asymétrique par échange de *clefs publiques* (*RSA*, *DH* ...)
    → Contrainte d'authentification
  - Choix d'une méthode de chiffrement symétrique (*DES*, *RC4*, *AES* ...)
    → Contrainte de confidentialité
  - Choix d'une méthode de hachage (*MD5*, *SHA* ...)
    → Contrainte d'intégrité
  - Choix d'une éventuelle méthode de compression
- **Utilisation de la connexion sécurisée**
  - Chiffrement symétrique, hachage, compression selon ce qui a été négocié
  - Procédé transparent pour l'application

Connexion sécurisée *SSL*

▷ Vision simplifiée de la négociation de la connexion



Connexion sécurisée *SSL*

▷ **Vision simplifiée de la négociation de la connexion**

- ◇ Le client se connecte au serveur
- ◇ Il lui envoie sa version *SSL* et les paramètres de connexion
- ◇ Le serveur répond avec des informations équivalentes
- ◇ Il envoie ensuite son *certificat* contenant entre autre sa *clef publique*
  - ○ Utilisation d'une paire *certificat/clef privée* préparée à l'avance
- ◇ Le client produit un *premaster secret* de 48 octets
  - ○ Permet de générer les clefs temporaires pour les algorithmes négociés
- ◇ Il le chiffre avec la *clef publique* du serveur et envoie le *premaster-secret*
  - ○ Le serveur le déchiffre avec sa *clef privée*
- ◇ Il produit le *master-secret* définitif, et l'envoie selon le *premaster-secret*
- ◇ Les deux envoient un message de fin du *handshake* selon le *master-secret*
- ◇ Les échanges applicatifs utilisent le chiffrement symétrique (*master-secret*)

Connexion sécurisée *SSL*

- **La connexion en elle-même est relativement sécurisée**
  - Le *premaster-secret* du client n'est lisible que par le serveur
  - Le *master-secret* du serveur n'est lisible qu'avec le *premaster-secret*
  - Les données applicatives ne sont lisibles qu'avec le *master-secret*
  - Des segments *TCP* capturés renferment des données incompréhensibles
  - Leur rejeu est inefficace (utilisation de numéros de séquence)

Connexion sécurisée *SSL*

▷ **Est-on certain de s'adresser au bon serveur ?**

- ◇ Procédé sensible aux attaques *man-in-the-middle* !
  (corruption du *DNS* par de fausses informations par exemple)
- ◇ Le client se connecte à un *"faux"* serveur et utilise son *certificat*
- ◇ Le *"faux"* serveur peut se connecter au *"vrai"* serveur
  - ○ Relayer le dialogue en espionnant/modifiant les données ($MITM$)
- ◇ Le *"faux"* serveur peut émuler le comportement du *"vrai"* serveur
- ◇ nb : le problème est le même dans le cas de la signature numérique

Connexion sécurisée *SSL*

▷ **Déroulement d'une attaque** *man-in-the-middle*

- ◇ *Alice* veut effectuer des échanges confidentiels avec *Bob*
- ◇ Elle tente de se connecter à *Bob* pour obtenir sa clef publique k_{pubB}
- ◇ *Charles*, par un procédé technique, détourne la connexion vers lui
- ◇ *Charles* transmet sa clef k_{pubC} à *Alice* en se faisant passer pour *Bob*
- ◇ *Alice* chiffre avec k_{pubC} les messages qu'elle destinait initialement à *Bob*
- ◇ *Charles* déchiffre alors ces cryptogrammes avec sa propre clef k_{privC}
- ◇ La connexion est bien sécurisée, mais entre *Alice* et *Charles* !
  - ◦ *Alice* ne sait pas qu'elle échange avec un inconnu
  - ◦ *Bob* ne sait pas qu'*Alice* devait échanger avec lui

⇒ **Il faut authentifier celui qui propose une clef publique**

Les certificats *X509*

- **La *"carte d'identité"* du serveur**
  - Le champ *TBS* (*to be signed*)
    - Le numéro de version du certificat (v1, v2 ou v3) + extensions
    - Le numéro de série du certificat
    - Le *DN* (*distinguish name*) du certificat signataire (*issuer*)
    - La période de validité du certificat et de sa clef publique
    - Le *DN* du titulaire de la clef publique (*subject*)
    - La clef publique du titulaire et l'algorithme associé
  - L'algorithme asymétrique et la fonction de condensat de la signature
  - La signature de l'empreinte numérique du *TBS*

Les certificats *X509*

▷ **Création d'un certificat**

- ◇ On s'adresse à un autorité de certification (*CA*, *Certificate Authority*)
  - ○ Tiers de confiance (*Verisign*, *Thawte*, *Equifax*, *Entrust* ...)
  - ○ Structure hiérarchique (organisme certifié par un autre ...)
- ◇ On lui transmet notre identité et notre clef publique
- ◇ Elle vérifie l'ensemble (service payant)
- ◇ Elle renseigne le *TBS* et signe le certificat
  - ○ Calcul d'un condensat du *TBS*
  - ○ Chiffrement du condensat avec la clef privée du *CA*
  - ○ La clef publique du *CA* est largement diffusée (elle permettra de vérifier la signature)

Les certificats *X509*

- **Exemple : produire un certificat avec l'outil** `openssl`
  - Générer une demande de *certificat*

    `$ openssl req -newkey rsa:1024 -keyout` ***`key.pem`*** `-out` ***`req.pem`***
    - Demande des informations sur l'identité de l'objet du certificat (notamment le *common-name* du *subject*)
    - Le fichier ***`key.pem`*** contient une clef privée
    - Le fichier ***`req.pem`*** contient la clef publique associée à ***`key.pem`***
    - Il contient également les informations saisies
  - Transmettre ***`req.pem`*** à une autorité de certification (CA)
    - Elle le signe pour produire le certificat ***`cert.pem`***

      ```
      $ openssl ca -cert ca_cert.pem -keyfile ca_key.pem \
                   -in req.pem -out cert.pem
      ```

Les certificats *X509*

- **Vérification du certificat par le client**
  - Date courante dans la période de validité ?
  - *CA* dans la liste des *CA* ?
    - Le client doit avoir une liste de *CA* reconnus
  - Numéro de série dans la liste des certificats révoqués ?
  - Déchiffrer la signature avec la clef publique du *CA*
    - Calculer le condensat du champ *TBS* du certificat
    - Résultats identiques ?
  - Le nom de domaine du certificat est-il celui du serveur ?
    - Indiqué dans le *common-name* du *subject*
    - Ce dernier point n'est pas traité par *SSL*
    - C'est à la charge de l'application !
  - nb : procédé récursif car les *CA* sont hiérarchisés

Les certificats *X509*

- **Est-on certain de s'adresser au bon serveur ?**
  - Oui, mais seulement s'il dispose d'un certificat en bonne et due forme !
  - Envisageable pour les sites *"commerciaux"*
    - Le coût du certificat est amorti par ce que rapporte le service
  - Pour les autres sites (*webmail* ...) ou d'autres services (*SSH* ...)
    - Le coût d'un certificat n'est pas justifié
    - Utilisation d'un certificat auto-signé (voir `openssl req`)
      → Les clients doivent connaître **ce** certificat et l'accepter
      → Il doivent alerter si ce certificat change
    - Mise en place d'une *PKI* (*Public Key Infrastructure*)
      → Création de notre propre autorité de certification
      → Les clients doivent connaître **cette** autorité de certification (comme les *CA* officielles reconnues par défaut)

Les certificats *X509*

▷ **Exemple : produire un certificat auto-signé avec l'outil** `openssl`

◇ Générer un *certificat* auto-signé (`-x509`) sans pass-phrase (`-nodes`)

```
$ openssl req -x509 -nodes -newkey rsa:1024 \
              -keyout key.pem -out cert.pem
```

○ Demande des informations sur l'identité de l'objet du certificat (notamment le *common name* du *subject*)

○ Le fichier ***key.pem*** contient une clef privée

○ Le certificat ***cert.pem*** contient la clef publique associée à ***key.pem***

○ Il contient également les informations saisies

◇ Ce certificat n'est pas signé par une *CA* mais par lui-même

○ Précaution élémentaire pour des services peu sensibles

○ Permet également de faire des expériences ...

Les certificats *X509*

- **Vérification, par un client, d'un certificat auto-signé**
  - Le certificat est initialement rejetté (non signé par une CA reconnue)
    - Si ce *certificat* est vu pour la premiere fois
      $\rightarrow$ Avertir et mémoriser ce certificat pour ce serveur
    - S'il est différent de ce qui a été mémorisé pour ce serveur
      $\rightarrow$ Avertir et mettre fin à la connexion ($MITM$) !!!
  - On fait confiance à la première connexion au serveur
  - Une démarche semblable est utilisée dans *SSH*

Les certificats *X509*

▷ Première utilisation d'une clef publique inconnue

```
$ ssh somewhere
The authenticity of host 'somewhere (192.168.1.12)' can't be established.
RSA key fingerprint is 4a:f5:37:6c:3e:0d:cd:ee:87:ca:80:78:77:09:16:bb.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'somewhere,172.168.1.12' (RSA) to the list of known hosts.
user@somewhere's password: ********
{user@somewhere}$
```

▷ Accès au même serveur doté d'une nouvelle clef publique

```
$ ssh somewhere
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
bc:f5:3e:0d:80:cd:ee:4a:16:87:ca:78:77:37:09:6c.
Please contact your system administrator.
Add correct host key in /home/user/.ssh/known_hosts to get rid of this message.
Offending key in /home/user/.ssh/known_hosts:18
RSA host key for somewhere has changed and you have requested strict checking.
Host key verification failed.
$
```

Les certificats *X509*

- **Mise en place d'une *PKI* (*Public Key Infrastructure*)**
  - Reproduire localement le fonctionnement des *CA* officielles
    - Économique mais plus élaboré que les simples certificats auto-signés
    - Permet d'avoir un contrôle total de la procédure
  - Produire notre propre *CA* racine (certificat auto-signé)
  - Produire éventuellement des *CA* intermédiaires signées par notre racine
  - Nos *CA* signent les certificats de nos serveurs et clients (employés ...)
  - Nos clients et serveurs seront ammenés à vérifier ces certificats
    - Ils doivent avoir une connaîssance préalable de notre *CA* racine
    - La transmission de cette information est le point délicat !
  - Demande une étude très détaillée et une gestion quotidienne rigoureuse
    - Notamment en ce qui concerne la révocation des certificats (départ d'un employé, vol d'un ordinateur, perte d'une clef ...)

Les certificats *X509*

▷ Exemple : *PKI* minimale avec l'outil `openssl` (1/2)

◇ Notre propre *CA* pour signer nos propres certificats

```
#---- Creation d'une autorite de certification et de sa clef privee ----
#     Il s'agit d'un certificat auto-signe
#     Utilisation d'un mot de passe ou non lors des signatures (ici non : -nodes)
#     Un repertoire est necessaire a la signature (voir /etc/ssl/openssl.conf)
$ openssl req -x509 -newkey rsa:1024 -nodes -keyout ca_key.pem -out ca_cert.pem
...
Common Name (eg, YOUR name) []:MY_OWN_CA
...
$ mkdir demoCA demoCA/newcerts
$ touch demoCA/index.txt
$ echo 01 > demoCA/serial


#---- Un serveur genere une demande de certificat et sa clef privee ----
#     Un mot de passe est generalement inapproprie pour un serveur (-nodes)
#     Le common-name est souvent le nom de domaine du serveur
$ openssl req -newkey rsa:1024 -nodes -keyout srv_key.pem -out srv_req.pem
...
Common Name (eg, YOUR name) []:srv.example.net
...
#---- Le serveur transmet la demande a la CA qui signe alors le certificat ----
#     La CA n'a pas besoin de connaitre la clef privee du serveur !
$ openssl ca -cert ca_cert.pem -keyfile ca_key.pem -in srv_req.pem -out srv_cert.pem \
             -policy policy_anything
```

Les certificats *X509*

▷ Exemple : *PKI* minimale avec l'outil `openssl` (2/2)

```
#---- Un client/utilisateur genere une demande de certificat et sa clef privee ----
#     Un mot de passe est generalement souhaitable pour un client (pas -nodes)
$ openssl req -newkey rsa:1024 -keyout user_key.pem -out user_req.pem
...
Enter PEM pass phrase: ********
...
Common Name (eg, YOUR name) []:user_lambda
...
#---- Le client transmet la demande a la CA qui signe alors le certificat ----
#     La CA n'a pas besoin de connaitre la clef privee du client !
$ openssl ca -cert ca_cert.pem -keyfile ca_key.pem -in user_req.pem -out user_cert.pem \
             -policy policy_anything


#---- Verification d'un certificat ----
#     Chaque client/serveur doit disposer de sa propre paire certificat/clef-privee
#     Ils auront besoin du certificat de notre CA pour verifier les certificats recus
#     La verification a normalement lieu dans l'application finale
#     Ici il s'agit juste ici d'une verification prealable de nos certificats
$ openssl verify -CAfile ca_cert.pem user_cert.pem
user_cert.pem: OK
$ openssl verify -CAfile ca_cert.pem srv_cert.pem
srv_cert.pem: OK
```

SSL et les certificats *X509*

- **La connexion en elle-même est relativement sécurisée**
  - L'écoute du traffic est inutile
- **On peut être certain de s'adresser au bon serveur**
  - En n'acceptant que les certificats en bonne et due forme
  - En ayant une liste de *CA* à jour
  - En ayant une liste de certificats auto-signés bien tenue
  - nb : le serveur peut également demander un certificat au client
- **!!! Repose sur la vigilance de l'application et de l'utilisateur !!!**
  - Le certificat doit décrire ce que l'application cherche à joindre
  - Prise de risque à la première apparition d'un certificat auto-signé
  - Une seule fois peut suffire pour divulger des informations confidentielles
  - L'utilisateur doit prendre en compte les avertissements de l'application

Les utilitaires *SSH*

- **Propos**
  - Protocole et outils pour des communications sécurisées en réseaux
    - Les données sont confidentielles
    - L'authentification de l'utilisateur aussi !
  - Utilisation d'un *shell* sur une machine distante (*Secure Shell*)
  - Échanges de fichiers avec une machine distante
  - Mise en place de *tunnels TCP*
    - Permettre un trafic quelconque *via* une connexion sécurisée

Les utilitaires *SSH*

▷ **Principe de fonctionnement**

- ◇ Un serveur (`sshd`) est à l'écoute (port 22 *TCP*) sur un poste
- ◇ Un client (`ssh`) se connecte au serveur depuis un autre poste
  - ○ La connexion est sécurisée (*SSL*), tout ce qui suit est chiffré
- ◇ L'utilisateur (client) s'authentifie auprès du serveur
- ◇ Le serveur exécute un *shell* pour l'utilisateur
  ($\simeq$ *login* : utilisateur, groupe, répertoire ...)
- ◇ Les E/S du *shell* sont redirigées dans la connexion sécurisée
- ◇ Commandes saisies sur le poste client $\rightarrow$ exécutées par le poste serveur
- ◇ nb : l'usage final est similaire à `telnet`, `rlogin`, `rsh` ...
  - ○ Mais avec ceux-ci tout passe en clair, y compris les mots de passe !

Les utilitaires *SSH*

▷ Exemple : obtenir un *shell* distant

```
{user@host}$ ssh there.veryfar.net
user@there.veryfar.net's password: ********
{user@there}$ exit
logout
Connection to there.veryfar.net closed.

{user@host}$ ssh root@there.veryfar.net
root@there.veryfar.net's password: ********
{root@there}# exit
logout
Connection to there.veryfar closed.
```

▷ Exemple : exécuter une commande à distance

```
{user@host}$ ssh root@there.veryfar.net "du -hs /home/*"
root@there.veryfar.net's password: ********
84M     /home/user
24M     /home/dummy
16K     /home/lost+found

{user@host}$ tar cpf - SubDir | gzip -9 | \
             ssh there.veryfar.net "gzip -cd | ( cd Project ; umask 0 ; tar xpvf - )"
user@there.veryfar.net's password: ********
SubDir/
SubDir/readme.txt
SubDir/photo.jpg
```

Les utilitaires *SSH*

- **Transfert de fichiers**
  - Le client `scp` provoque l'exécution de `scp -t` sur le serveur
    - L'authentification est identique à celle de la commande `ssh`
    - Les deux processus communiquent par la connexion sécurisée
    - Selon la ligne de commande, des fichiers sont échangés
    - nb : semblable à `rcp` (mais celui-ci fait tout en clair !)
  - Le client `sftp` provoque l'exécution de `sftp-server` sur le serveur
    - L'authentification est identique à celle de la commande `ssh`
    - Les deux processus communiquent par la connexion sécurisée
    - Selon les commandes interactives saisies, des fichiers sont échangés
    - nb : semblable à `ftp` (mais celui-ci fait tout en clair !)
      (de plus `sftp` utilise une seule connexion, le filtrage est facilité)

Les utilitaires *SSH*

▷ Exemple : transfert de fichiers et de répertoires avec `scp`

```
{user$host}$ scp -r SubDir there.veryfar.net:Project
user@there.veryfar.net's password: ********
...

{user$host}$ scp there.veryfar.net:Project/SubDir/photo.jpg .
user@there.veryfar.net's password: ********
...

{user$host}$ scp data.tgz root@there.veryfar.net:
root@there.veryfar.net's password: ********
...
```

▷ Exemple : transfert de fichiers et de répertoires avec `sftp`

```
{user$host}$ sftp root@there.veryfar.net
Connecting to there.veryfar.net...
root@there.veryfar.net's password: ********
sftp> put data.tgz
Uploading data.tgz to /root/data.tgz
data.tgz                                      100%   18MB  17.7MB/s   00:01
sftp> get archive.tgz
Fetching /root/archive.tgz to archive.tgz
archive.tgz                                   100%   18MB   8.8MB/s   00:02
sftp> quit
{user$host}$
```

Les utilitaires *SSH*

▷ Authentification par clef publique

- ◇ Les utilitaires *SSH* effectuent systématiquement une authentification
- ◇ Par défaut, saisie du mot de passe de l'utilisateur distant ($\simeq$ *login*)
  - ○ Assez pénible si on doit le faire souvent
  - ○ Très inconfortable pour l'exécution de *scripts*
    (ex : toutes les nuits, transférer automatiquement une archive)
- ◇ Possibilité de préparer une paire de clefs pour l'authentification
  - ○ L'utilisateur client garde précieusement la clef privée k_{priv}
  - ○ La clef publique k_{pub} est transmise au compte utilisateur du serveur
- ◇ Les connexions suivantes ne nécessitent plus de mot de passe
  - ○ Le serveur choisit P aléatoirement, et envoie $E(k_{pub}, P) = C$ au client
  - ○ Le client reçoit C et répond $D(k_{priv}, C)$ au serveur
  - ○ Le serveur vérifie : $D(k_{priv}, C) = P \Rightarrow$ l'authentification est correcte

Les utilitaires *SSH*

▷ **Exemple : authentification par clef publique**

◇ L'utilisation des clefs peut être accompagnée d'un mot de passe

○ Nous n'utilisons pas cette possibilité ici (`-N ""`)
(Ça irait à l'encontre de notre objectif de confort !)

◇ nb : plusieurs clients peuvent le faire sur le même compte d'un serveur

```
{user$host}$ ssh-keygen -t rsa -N ""
Generating public/private rsa key pair.
Enter file in which to save the key (/home/user/.ssh/id_rsa):
Your identification has been saved in /home/user/.ssh/id_rsa.
Your public key has been saved in /home/user/.ssh/id_rsa.pub.
The key fingerprint is:
e3:35:86:dc:61:56:d0:24:76:3a:4a:bf:68:34:f0:d4 user@host

{user@host}$ cat .ssh/id_rsa.pub | ssh root@there.veryfar.net "cat >> .ssh/authorized_keys"
root@there.veryfar.net's password: ********

{user@host}$ ssh root@there.veryfar.net
{root@there}# cat .ssh/authorized_keys
ssh-rsa AAAAB3Nz2EAQEA+csp2RNz4fvdLmOK.........xtYx7CFvBJYjX== dummy@stuff
...
ssh-rsa AAAAB3NzaC1yc2EAmG6DFgXfElnBKD.........7nuL9E9uJMVew== user@host
{root@there}# exit
{user@host}$
```

Les utilitaires *SSH*

▷ **Redirection locale de port** *TCP*

◇ Relayer des connexions *TCP* dans un *tunnel SSH*

◇ Supposons que la connexion directe du *client* au *serveur* soit interdite

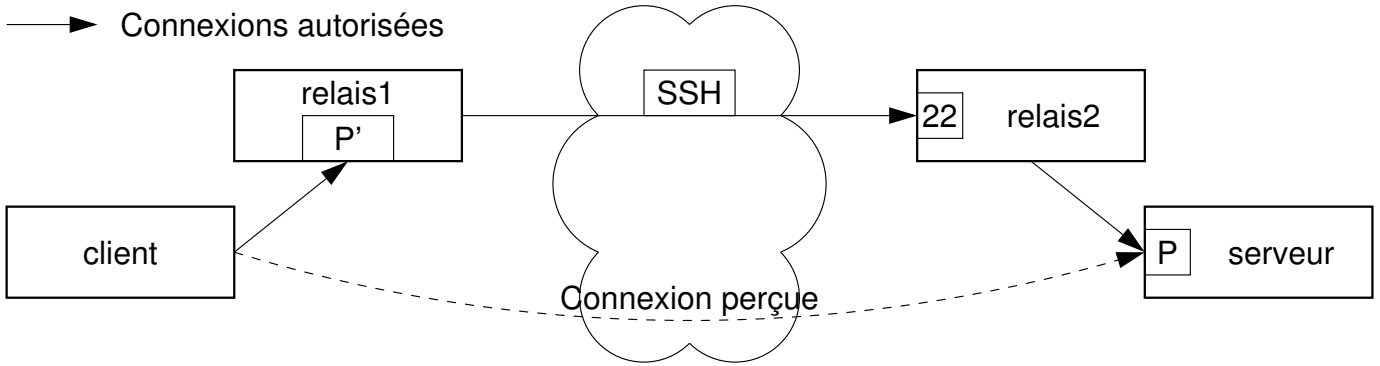
◇ *relais1* se connecte en *SSH* à *relais2* et écoute sur le port `P'`

◇ Le *client* se connecte au port `P'` sur *relais1*

○ Ceci provoque la connexion de *relais2* au port `P` du *serveur*

◇ Les données sont relayées de bout-en-bout (bidirectionnel)

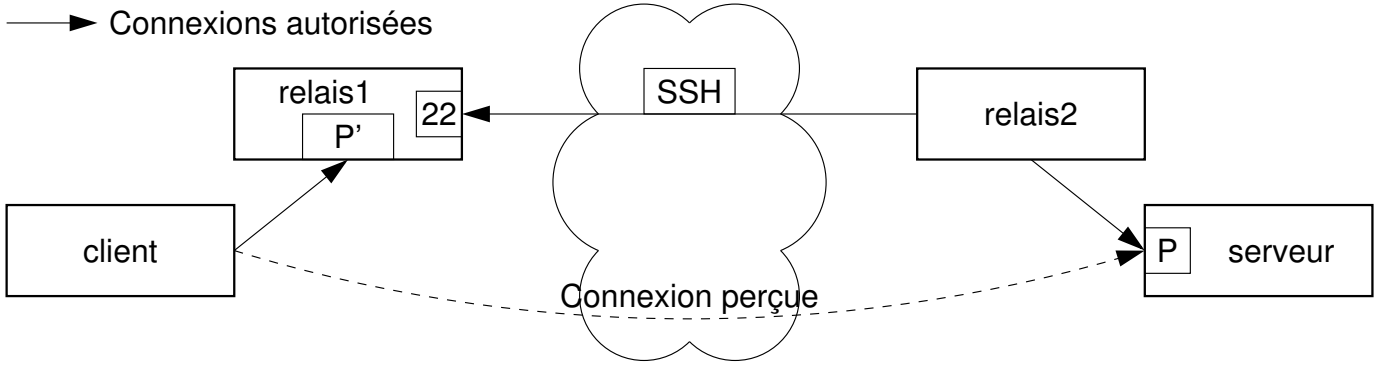
○ Le *client* dialogue ainsi avec le *serveur*



Les utilitaires *SSH*

▷ Redirection distante de port *TCP*

- ◇ La connexion sécurisée peut être établie dans la direction opposée
- ◇ C'est le site *serveur* qui *"invite"* le site *client*



▷ Exemple : commandes de redirections *SSH*

```
# redirection locale (-L) depuis relais1 avec P=80 et P'=8080
{someone@relais1}$ ssh -NgL 8080:serveur:80 relais2

# autre possibilite : redirection distante (-R) depuis relais2
{someone@relais2}$ ssh -NgR 8080:serveur:80 relais1

# le client consulte http://serveur/ via relais1:8080
{user@client}$ lynx http://relais1:8080/
```

Les utilitaires *SSH*

▷ **Accès sécurisé aux systèmes de fichiers partagés**

- ◇ Utiliser `sftp` de manière transparente pour fournir un disque virtuel
  - ○ *UNIX* : *"(dé)monter"* un système de fichiers dans répertoire
  - ○ *Window$* : *"(dé)connecter"* un lecteur (`A:`, `B:`, `C:`, ..., `Z:`)
- ◇ Utilisation du système de fichiers comme en local (disque, clef *USB*, ...)
- ◇ L'authentification et les transferts sont sécurisés (`sftp`)
  - ○ Solutions usuelles bien plus vulnérables (*NFS/NIS*, *SMB/NMB* ...)
- ◇ Rien à faire côté serveur `:-)` (utilisation de `sshd`/`sftp-server`)
- ◇ Très spécifique à l'*OS* côté client `:-(` (gestion des systèmes de fichier)
  - ○ *fuse*/*sshfs* sous *UNIX* : *Linux*, *NetBSD*, *FreeBSD*, *OpenSolaris*, *Hurd*
  - ○ *MacFuse*/*MacFusion* sous *MacOsX*
  - ○ *SftpDrive* (commercial) sous *Window$* (*WinFuse* ?)

Les utilitaires *SSH*

▷ Exemple : accès à un système de fichiers partagé avec `sshfs`

```
{user@host}$ mkdir THERE
{user@host}$ sshfs root@there.veryfar.net:SubDir THERE
root@there.veryfar.net's password: ********
{user@host}$ ls THERE
photo.jpg      readme.txt
{user@host}$ cat THERE/readme.txt
This file is very interesting !
{user@host}$ echo "Hi there !" > THERE/hello.txt
{user@host}$ ls THERE
hello.txt      photo.jpg      readme.txt
{user@host}$ mount | grep THERE
sshfs#root@there.veryfar.net:SubDir on /home/user/THERE type fuse (rw,nosuid,nodev, ...
{user@host}$ fusermount -u THERE
{user@host}$ mount | grep THERE
{user@host}$ ls THERE
{user@host}$ ssh root@there.veryfar.net "cat SubDir/hello.txt"
root@there.veryfar.net's password: ********
Hi there !
{user@host}$
```

Les utilitaires *SSH*

▷ `authpf` **:** *shell* **utilisateur pour les passerelles d'authentification**

- ◇ Ajustement d'un *firewall* en fonction de l'utilisateur (pas du poste)
  - ○ Conçu dans *PacketFilter* d'*OpenBSD* puis *NetBSD*, *FreeBSD* ...
- ◇ L'administrateur prépare des règles personnalisées sur le *firewall*
  - ○ Le *shell* des utilisateurs dans `/etc/passwd` est `/usr/sbin/authpf`
  - ○ Ne sert qu'à l'authentification des utilisateurs
- ◇ Un utilisateur se connecte en *SSH* sur le *firewall*
  - ○ Ses règles personnelles sont inserées dans le *firewall*
    (Elles sont adaptées au poste à l'origine de la connexion *SSH*)
  - ○ Elles sont retirées dès que la connexion *SSH* est coupée
- ◇ L'utilisateur peut utiliser le poste de son choix
  - ○ Il dispose des mêmes droits d'accès au réseau (∀ poste)
  - ○ Son activité personnelle peut être enregistrée dans un journal

Les utilitaires *SSH*

- **Outil de communication incontournable et très polyvalent**
  - Adapté à de très nombreux cas d'utilisation (exécutions distantes, transferts, systèmes de fichiers partagés ...)
  - Notamment grâce aux redirections de ports *TCP*
  - Interaction simple avec la politique de filtrage (unique port 22 *TCP*)
- **Les communications et l'authentification sont sécurisées**
  - Les services fournis sont sécurisés (propos essentiel, *OpenBSD* ...)
  - Permet de sécuriser des protocoles vulnérables en les encapsulant (toujours grâce aux redirections de ports *TCP*)
- **Disponible sur de nombreuses plateformes**
  - *OpenSSH* pour les *UNIX* : *MacOsX*, **BSD*, *Linux*, *Solaris*, *IRIX* ...
  - *Window$* : `putty.exe` (terminal), `winscp.exe` (`sftp` graphique) (ou *OpenSSH* sous *Cygwin*)

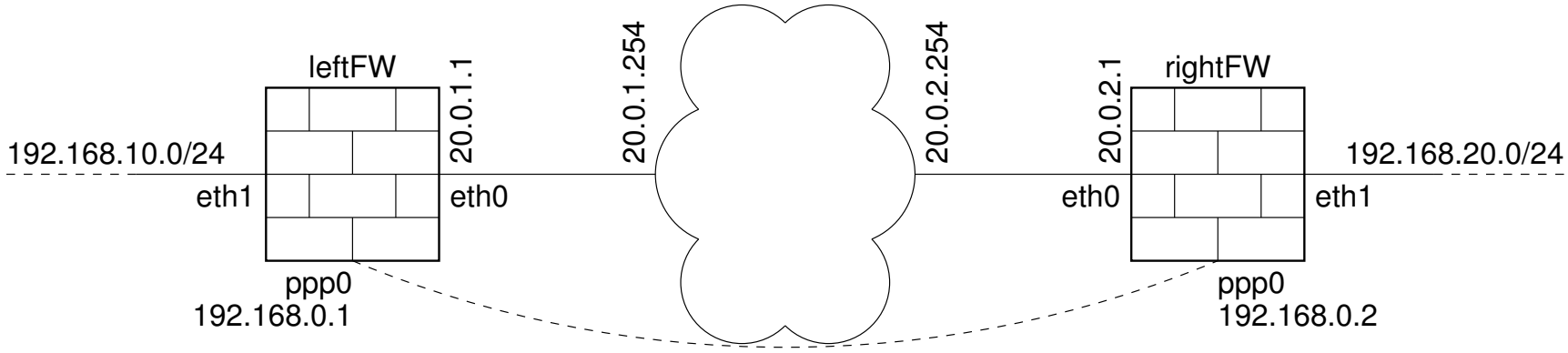
Les réseaux privés virtuels

- **Problématique des *VPN* (*Virtual Private Network*)**
  - Considérer plusieurs sites distants comme un même réseau local
    - Ils doivent travailler en commun sur les mêmes ressources
    - Comment des sites externes peuvent-ils accéder aux ressources locales ?
    - Sans autoriser tout ***Internet*** à le faire ! (filtrage !?!?)
  - Liaison spécialisée ? (fibre optique, cuivre, satellite, hertzienne ...)
    - (+) Bande passante dédiée, *a priori* toujours opérationnelle
    - (-) Très coûteux et peu souple (déménagement ?)
  - Solution ***VPN*** : communications sécurisées à travers ***Internet***
    - (-) De nombreux équipements intermédiaires (débit, disponibilité ?)
    - (+) Peu coûteux et très souple (uniquement logiciel)

Les réseaux privés virtuels

▷ Interconnexion de réseaux

- ◇ Relier les sous-réseaux locaux de plusieurs sites
  - ○ ex : plusieurs agences d'une même société
- ◇ Interconnecter les routeurs des sites de manière sécurisée
  - ○ Apparition d'interfaces *point-à-point*
- ◇ Problème classique de routage et de filtrage entre les sous-réseaux locaux
  - ○ Complètement transparent pour les postes de travail
  - ○ Entièrement géré par les routeurs filtrants



Les réseaux privés virtuels

▷ Exemple : interconnexion de réseaux (1/2)

◇ Solution minimale ici, juste à titre d'illustration (*PPP over SSH*)

```
#---- Liaison ppp securisee bidirectionnelle entre les routeurs filtrants ----
# (la transmission d'une clef publique d'authentification doit avoir lieu avant)
{root@leftFW}# pppd pty 'ssh -t -e none rightFW.veryfar.net pppd passive noauth' \
               noauth 192.168.0.1:192.168.0.2

#---- Ajustement du routage pour atteindre 192.168.20.0/24 ----
{root@leftFW}# route add -net 192.168.20.0/24 gw 192.168.0.2
{root@leftFW}# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.0.2     0.0.0.0         255.255.255.255 UH    0      0        0 ppp0
20.0.1.254      0.0.0.0         255.255.255.255 UH    0      0        0 eth0
192.168.20.0    192.168.0.2     255.255.255.0   UG    0      0        0 ppp0
192.168.10.0    0.0.0.0         255.255.255.0   U     0      0        0 eth1
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
0.0.0.0         20.0.1.254      0.0.0.0         UG    0      0        0 eth0
```

Les réseaux privés virtuels

▷ Exemple : interconnexion de réseaux (2/2)

```
#---- Ajustement du routage pour atteindre 192.168.10.0/24 ----
{root@rightFW}# route add -net 192.168.10.0/24 gw 192.168.0.1 ; route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.0.1     0.0.0.0         255.255.255.255 UH    0      0        0 ppp0
20.0.2.254      0.0.0.0         255.255.255.255 UH    0      0        0 eth0
192.168.10.0    192.168.0.1     255.255.255.0   UG    0      0        0 ppp0
192.168.20.0    0.0.0.0         255.255.255.0   U     0      0        0 eth1
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
0.0.0.0         20.0.2.254      0.0.0.0         UG    0      0        0 eth0

#---- Test de connectivite entre 192.168.10.0/24 et 192.168.20.0/24 ----
{user@host}# ping -Rn 192.168.20.8
PING 192.168.20.8 (192.168.20.8) 56(124) bytes of data.
64 bytes from 192.168.20.8: icmp_seq=1 ttl=63 time=0.818 ms
RR:     192.168.10.27
        192.168.0.1
        192.168.20.1
        192.168.20.8
        192.168.20.8
        192.168.0.2
        192.168.10.1
        192.168.10.27
64 bytes from 192.168.20.8: icmp_seq=2 ttl=63 time=0.921 ms     (same route)
```

Les réseaux privés virtuels

▷ ***VPN* d'accès**

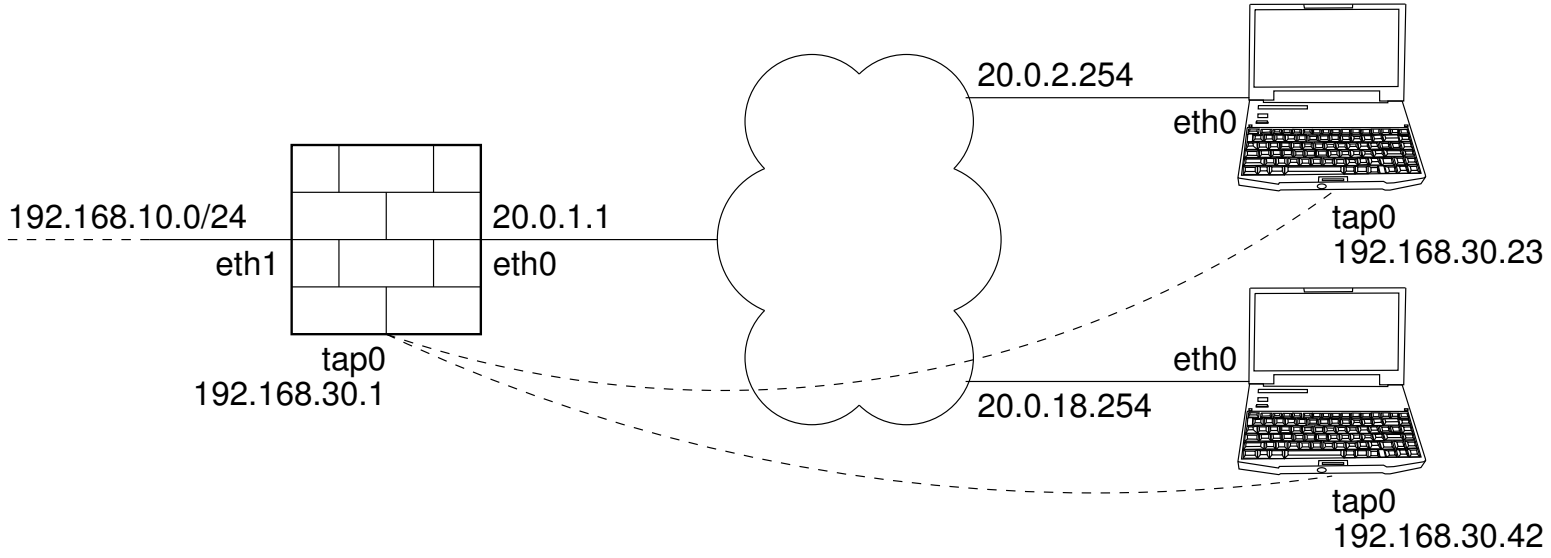
◇ Intégrer les postes des utilisateurs distants ou itinérants dans le réseau local

◇ Les postes en question doivent utiliser un dispositif spécifique

○ Apparition d'interfaces *"virtuelles"* (routage, filtrage . . .)

◇ Le dispositif reliant les interfaces virtuelles doit être sécurisé

○ Chiffrement, authentification de tous les postes (routeur et distants)



Les réseaux privés virtuels

▷ Exemple : *VPN* d'accès avec l'outil openvpn

```
{root@vpnserver}# cat vpnsrv.conf
mode server
tls-server
comp-lzo
dev tap
ifconfig 192.168.30.1 255.255.255.0
push "route 192.168.10.0 255.255.255.0 192.168.30.1"
client-config-dir VPN_CLIENTS
ccd-exclusive
client-to-client
dh dh1024.pem
; necessite une PKI minimale pour creer/verifier ces certificats
ca ca_cert.pem
; le common-name est verifie par les clients
cert vpnsrv_cert.pem
key vpnsrv_key.pem

{root@vpnserver}# cat VPN_CLIENTS/user_lambda
; nom de ce fichier de config = common-name du client (' ' devient '_' si besoin)
ifconfig-push 192.168.30.23 255.255.255.0

{root@vpnserver}# openvpn vpnsrv.conf
...
```

Les réseaux privés virtuels

▷ Exemple : *VPN* d'accès avec l'outil openvpn

```
{root@lambda}# cat user_lambda.conf
client
tls-client
comp-lzo
dev tap
remote 20.0.1.1 1194
; common-name attendu dans certificat du serveur
tls-remote vpn_srv_common_name
ca ca_cert.pem
; le common-name determine le fichier de config sur le serveur
cert user_lambda_cert.pem
key user_lambda_key.pem

{root@lambda}# openvpn user_lambda.conf
...

{root@lambda}# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
20.0.2.254      0.0.0.0         255.255.255.255 UH    0      0        0 eth0
192.168.10.0    192.168.30.1    255.255.255.0   UG    0      0        0 tap0
192.168.30.0    0.0.0.0         255.255.255.0   U     0      0        0 tap0
0.0.0.0         20.0.2.254      0.0.0.0         UG    0      0        0 eth0
```

Les réseaux privés virtuels

- **Quelques précautions pour l'interconnexion de réseaux**
  - Peu de risques en ce qui concerne la connexion des routeurs filtrants
    - Précautions d'authentification pour les administrateurs
  - Utilisateurs distants *a priori* aussi dangereux que les utilisateurs locaux
  - Peut-être plus dangereux !
    - Confiance en la rigueur de l'administration des sites distants ?
    - Envisager un filtrage plus rigoureux pour les sites distants (les quelques serveurs qui les concernent, rien de plus)

Les réseaux privés virtuels

- **Quelques précautions pour les *VPN* d'accès**
  - Les postes distants ont probablement un accès direct à ***Internet*** (filtrage ?)
    - Ils sont potentiellement plus *"infestés"* que les postes locaux
    - Envisager un filtrage très rigoureux pour les postes distants (les quelques serveurs qui les concernent, rien de plus)
  - Si l'ordinateur portable est perdu ou volé ?
    - L'authentification des utilisateurs distants est indispensable
    - Un certificat personnel (qui pourra être révoqué en cas de perte)
    - Saisie indispensable d'un mot de passe pour activer ce certificat (sa clef privée est nécessairement sur le même ordinateur !)

Les réseaux privés virtuels

- **Quelques solutions**
  - *PPTP* : solution propriétaire *Micro$oft*
    - (+) Logiciel client disponible par défaut dans *Window$*
    - (−) Jugé peu robuste par les cryptanalystes
    - (−) Peu interopérable
  - *IPSec* : solution standardisée
    - (+) Standardisée, très interopérable
    - (−) Difficilement portable (noyau) et jugé complexe à mettre en œuvre
  - *OpenVPN* : solution *open-source*
    - (+) Très interopérable, multi-plateformes (client et serveur)
    - (+) Robuste, repose sur des solutions éprouvées (*SSL*, certificats *X509*)
    - (+) Mise en œuvre aisée (*userspace*, peu de configuration, `1194` *UDP*)
    - (+) De nombreuses fonctionnalités optionnelles