# Programmation Internet

# Cours 9

kn@lri.fr
http://www.lri.fr/~kn

# Plan

# *Disclaimer*

- Aborde juste quelques aspects de sécurité
- Essaye de montrer quelques principes fondamentaux
- Uniquement axé sur le Web

⇒ Ça ne va pas faire de vous des *hackers*, juste vous sensibiliser aux problèmes de sécurité...

# Élements de cryptographie (1)

Alice et Bob veulent échanger des données confidentielles.

1. Chiffrement **symétrique**:
   - Ils se mettent d'accord sur une **clé commune**
   - Alice **chiffre** son message avec la clé et l'envoie à Bob
   - Bob déchiffre le message avec **la clé**

   **Non sûr** (Alice et Bob doivent se mettre d'accord sur une clé en « clair », par email par exemple) ou **non pratique** (ils doivent se rencontrer physiquement pour échanger la clé).
   **Efficace**: on peut implanter plusieurs algorithmes de chiffrements en utilisant uniquement des opérations logiques de bases (AND, OR, XOR). Il est facile de les implanter sur des puces dédiées (cartes de crédit, processeurs mobiles). Ils sont « sûrs » tant que la clé reste secrète.

# Élements de cryptographie (2)

Alice et Bob veulent échanger des données confidentielles.

2. Chiffrement **assymétrique**:
   - Bob crée une **clé publique** $K^B_{pub}$ et une **clé secrète** $K^B_{priv}$, telle que

     $$\forall msg,\ K^B_{priv}(K^B_{pub}(msg)) = K^B_{pub}(K^B_{priv}(msg)) = msg$$

     Bob **diffuse** sa clé publique (sur sa page Web par exemple, ou dans un annuaire de clé) et garde sa clé privée **secrète**.
   - Alice **chiffre** son message **m** avec la **clé publique** de Bob ($K^B_{pub}(m)$) et l'envoie à Bob
   - Bob déchiffre le message avec sa clé privée: $K^B_{priv}(K^B_{pub}(m))=m$

     **Sûr et pratique** (Bob a généré une paire de clé, et a déposé la clé publique sur une page Web)
     **Peu efficace**: repose sur des problèmes mathématiques difficiles (factorisation de grands entiers, courbes eliptiques sur les corps finis). Chiffrer et déchiffrer un message n'est pas réaliste pour des grands messages (vidéo en streaming, requêtes Web, ...).

# Élements de cryptographie (3)

On combine les deux méthodes. (Alice envoie un message à Bob)

- Alice choisit une clé symétrique secrète *s*
- Elle l'envoie à Bob en utilisant la clé publique de ce dernier ($K^B_{pub}(s)$)
- Bob décrypte le message et obtient $s=K^B_{priv}(K^B_{pub}(s))$
- Bob et Alice se sont mis d'accord de manière sûre sur une clé commune **s**! Ils peuvent utiliser un algorithme de chiffrement symétrique pour le reste de la conversation

⇒ Ceci est à la base de protocoles tels que HTTPS

# Éléments de cryptographie (4)

Le chiffrement assymétrique permet aussi d'avoir la preuve que quelqu'un est bien Bob!

- Alice choisit un message secret aléatoire m, sans le divulguer (appelé *challenge*)
- Alice calcule $K^B_{pub}(s)$ et l'envoie à la personne qui prétend être Bob
- Seule la personne qui possède la clé privée de Bob (donc Bob ...) peut déchiffrer le message et renvoyer l'original à Alice.

⇒ Comment garantir que la personne qui a généré les clés au départ est bien Bob ?

# HTTP: protocole texte « en clair »

HTTP est un protocole **texte**, les données ne sont pas chiffrées (cf. TP3) et **sans identification**

- **Confidentialité** : n'importe qui (avec les privilèges nécessaires) peut lire ce qui transite entre un client et un serveur Web
- **Authenticité** : n'importe qui peut se faire passer pour un serveur Web (attaque *man in the middle*)

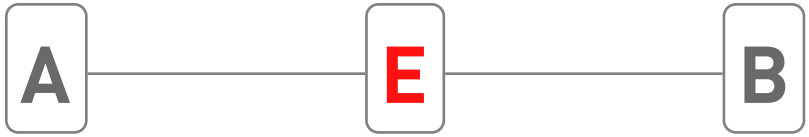
# Espionnage de connexion

Alice représente le client, Bob le serveur et Eve (*Eavesdropper*) l'attaquante

On suppose que **Eve** est **root** sur la machine. Il suffit de lire les paquets qui transitent par la carte réseau (`tcpdump` sous Linux).

- Eve et Alice sont sur la même machine (démo):


- Fonctionne aussi si Eve est sur une machine se trouvant sur la route entre Alice et Bob:




Ce problème touche tous les protocoles en clair: HTTP, POP, IMAP, FTP, .... Il peut être résolu grace au **chiffrement** de toute la connexion.

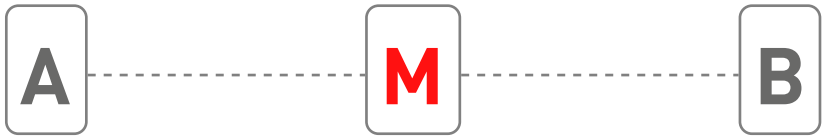
# Attaque *Man in the middle*

**Mallory** se place entre Alice (cliente) et Bob (banque), par exemple au moyen d'un **e-mail** frauduleux en HTML:

1. L'émail contient:

```
<html>
 <body>
     Bonjour,
        veuillez vous connecter à votre banque en cliquant ici:
         <a href='mallory.com' >www.bob.com</a>
  </body>
</html>
```

2. Alice, insouciante, clique sur le lien




3. Mallory peut retransmettre les requêtes entre Bob et Alice, en les modifiant au passage. Le problème est causé par un manque d'authentification (Mallory n'a pas a prouver à Alice qu'il est Bob)

# Solution: HTTPS

HTTP Secure

1. Respose sur de la cryptographie assymétrique (pour l'authentification et le partage de clé) et symétrique (pour le chiffrement de connexion)
2. Permet d'authentifier les correspondants et de protéger les données
3. Suppose l'existence de tiers de confiance Alice et Bob font confiance à Trent (*Trusted Third Party*)

Bob possède des clés publiques et privées ($K^B_{pub}$ et $K^B_{priv}$), Trent possède des clés publiques et privées ($K^T_{pub}$ et $K^T_{priv}$)

# HTTPS (détail du protocole)

Bob et Trent se rencontrent. Trent signe la clé publique de Bob en calculant

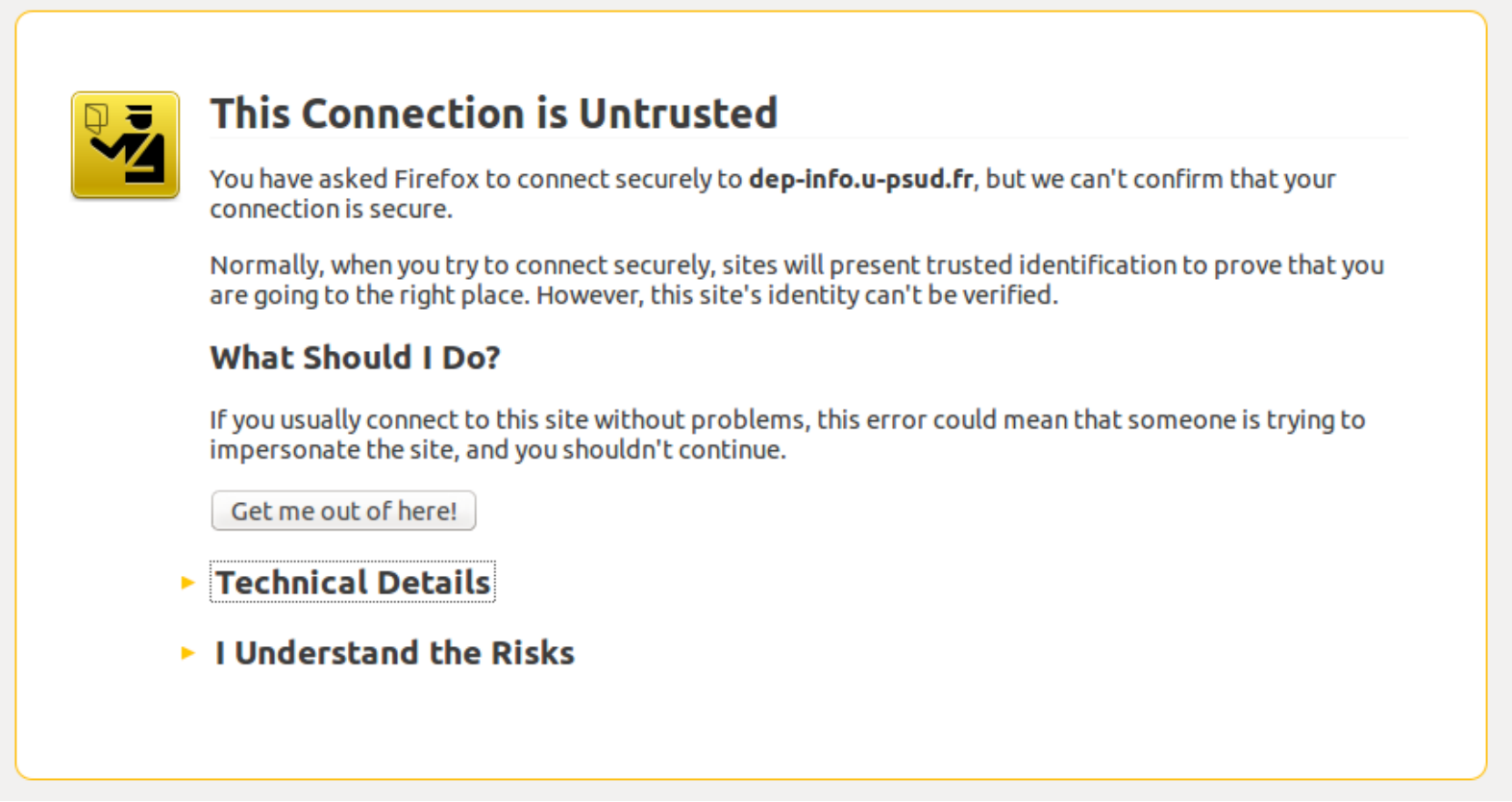
$$S^B = K^T_{priv}(K^B_{pub})$$

Comme Trent utilise sa clé privée on sait que seul Trent a pu générer cette signature. De plus, Trent a rencontré Bob donc il certifie que la clé $K^B_{pub}$ appartient bien à quelqu'un nommé Bob.

1. Alice (client) veut se connecter à Bob. Bob fournit sa clé publique $K^B_{pub}$ et la signature $S^B$
2. Alice contacte Trent (en qui elle a confiance) et récupère sa clé publique $K^T_{pub}$. Elle déchiffre la signature: $K^T_{pub}(S^B)$ et vérifie qu'elle retombe bien sur la clé publique de Bob.
3. Elle peut alors choisir une clé symétrique, la chiffrer avec $K^B_{pub}$ et entammer une communication chiffrée et authentifiée avec Bob.

# Tiers de confiance

Les tiers de confiance sont des entités (états, associations, compagnies privées) qui se chargent de vérifier les clés publiques d'autres entitées. C'est une vérification physique (documents administratifs, ...).




Cette erreur s'affiche quand une signature n'est pas conforme ou n'a pas pu être vérifiée

# Tiers de confiance

Attaques contre les **authorités de certifications** (tiers de confiance): difficiles, mais pas impossible. Certains tiers de confiance sont douteux (états voyous, compagnie piratées dont les clées **privées** sont compromises,...)

**Attaques d'implémentation** (plus probables) : on exploite un **bug** dans le code des serveurs web ou des navigateurs

**Autres faiblesses**: HTTPS est en « haut » dans la pile IP (application). On peut donc avoir connaissance du nombre de paquet échangés, des adresses IP des participants, la taille et la fréquence des paquets... (même si on n'en connait pas le contenu). Cela permet certaines attaques statisties ou de deni de service.

# Plan

# Traçage par cookies

*Normalement*, un cookie ne peut être lu que que par le site émetteur (cf. cours 8). But:

1. Empêcher un tiers de lire des données personnelles (ok)
2. Empêcher un tiers de savoir quels sites ont été visités (pas ok)

1. Un site B utilise des publicités pour se rémunérer. Le site marchand **M** fournit du code HTML:
   ```
   <script src="http://marchand.com/pub.js"/>
   ```
2. A visite le site B. Le code `pub.js` peut alors faire les choses suivantes:
   1. Scanner le contenu de la page de B. Possible car le script est « inclus » dans une page fournie par B
   2. Se connecter à`http://marchand.com/collecte.php` et passer en paramètre `post` ou `get` le contenu de la récolte
   3. `http://marchand.com` peut alors stocker un cookie valide pour son domaine avec le contenu de la récolte d'information
3. Lorsque A visite le site marchand **M**, ce dernier relit son cookie et fait des propositions ciblées.

# Solutions

- Désactiver les cookies de « tierce partie » (cookie dont l'origine n'est pas celle de la page visitée)
- Effacer par défaut tous les cookies quand on quite le navigateur
- Rajouter des exceptions pour certains sites au cas par cas

Nouveau standard du W3C en préparation pour signifier à un site qu'on ne souhaite pas être suivi (méthode « volontariste » qui suppose que les sites commerciaux sont gentils et respectent le protocole)

# Sécurité des cookies de session

On a vu que les sessions PHP (vrai aussi pour les autres langages côté serveur) stockent dans un cookie un identifiant unique. Que se passe-t-il si on vole ce cookie ? (démo)

Pas d'autre solution que de faire confiance au root (solutions partielles basées sur le chiffrement des disques dur)

# Plan

# Injection de code Javascript/HTML

**Vulnérabilité: on exploite le fait qu'un site utilise directement les entrées fournies par l'utilisateur.**
**Exemple: commentaires sur un blog.**

1. **Une page Web utilise un formulaire pour permettre de poster des commentaires sur un blog:**

```
<form action="comment.php" method="post">
  Commentaire: <br/>
  <textarea rows="20" cols="60"  name="zonetexte"/>
 <br/>
 <button type="submit">Envoyer</button>
</form>
```

2. **Un bout de code PHP écrit le commentaire sur la page:**

```
echo "Commentaire #$i: <p>";
 echo $_POST["zonetexte"];
 echo "</p>";
```

# Injection de code Javascript/HTML

**Problème tout ce qui est dans la zone de texte est copié dans la page HTML de chaque client qui consulte la page et interprété par son navigateur:**

```
Debut du commentaire
        <script type="text/javascript">
          ... //code javascript malicieux
        </script>
        Fin du commentaire
```

# Injection de code PHP

**Problème lié à l'utilisation de la fonction**

`eval(command)`

`command` **est une chaîne de caractères considérée comme étant du code PHP et** `eval` **exécute cette chaîne:**

```
echo eval ("1 + 2 * 3");  // affiche 7
echo eval ('$x = 4;');    // définit la variable $x
echo $x;                  // affiche 4
```

**Il ne faut jamais donner une chaine de caractère de l'utilisateur comme argument à** `eval` **(sauf durant le TP 9)**

# Solutions

- **Ne jamais utiliser** `eval`
- **Utiliser la fonction** `htmlspecialchars` **qui échappe les caractères** <, >, &, ', "
- **Utiliser la fonction** `striptags` **qui supprime tout ce qui est une balise**
- **Toujours valider les entrées d'un utilisateur**

# Injection de code SQL

**SQL: language de requête permettant d'interroger des bases de données. Utilisation classique depuis PHP (on suppose un formulaire qui met dans le champ** `"nom"` **le nom d'un étudiant):**

```
$Q = "SELECT * FROM STUDENTS WHERE ";
$Q = $Q . "(NAME = '" . $_POST["nom"] . "');";
mysql_query($Q);
```

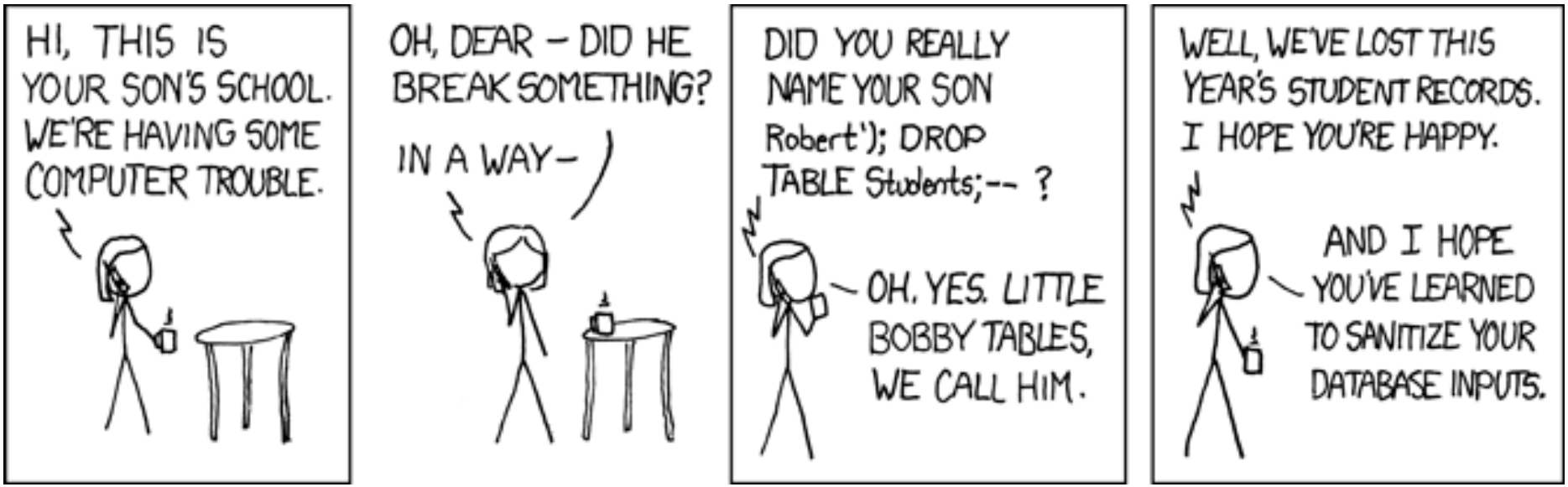
**Si l'utilisateur donne comme nom « Toto », la requête envoyée à la base est:**

```
SELECT * FROM STUDENTS WHERE (NAME = 'Toto');
```

**Affiche toutes les lignes de la table** `STUDENTS` **pour lesquel le nom est** `Toto`

# Jusqu'au jour où ...

©xkcd




```
SELECT * FROM STUDENTS WHERE (NAME ='Robert');
DROP TABLE STUDENTS; --');'
```

# Ou bien...