Introduction à la sécurité – Cours 8
Anonymat

Catalin Dima

Sources de données

- Financières (transfert bancaire, salaires, investissements, achats par carte de crédit, impôts payés,...)
- Programme de voyage (frequent flyer), programmes de fidélité, clubs,...
- Données téléphoniques : source, temps, conenu (si mis sur écoute...), source d'appels de téléphones portables.
- Données médicales.
- Films regardés dans un hôtel (même sous 2 minutes...)

Sources informatiques

- Tout paquet IP identifie source et destination.
- HTTP inclut informations supplémentaires : browser, hostname,...
- Toutes les activités externes peuvent être enregistrées dans les passerelles (gateways) – bcp. d'employeurs font cela d'ailleurs !
- Moteurs de recherche
- Archives DejaNews – les messages ne seront jamais effacés !

Définitions

- **Intimité** : Les données collectées peuvent être utilisées seulement à des fins très limités et connus à l'avance.
  - E.g. adresse client utilisée seulement pour l'envoi de produits soft.
- Questions éthiques :
  - Quelles propriétés devraient être respectées par une banque ?
  - Hot-line pour les suicides ?
- **Anonymat** : les parténaires d'une communication restent inconnus *à l'intérieur d'un groupe*.
  - Inclut l'anonymat de l'expéditeur, du destinataire et de la *relation entre les deux*.
  - Anonyme *dans un groupe* : ses propres actions ne peuvent pas être distinguées des actions d'autres membres du groupe.
  - Plus grand est le groupe, plus l'anonymat est "meilleur".
  - Toutes les technologies d'*anonymisation* possèdent des surcoûts de performance ou de fiabilité.

Anonymat de recipient : broadcast

- Renvoi du message à tous les membres d'un groupe. :
  - Multicast, transmission radio, token ring (LAN),...
- Le destinataire devrait être identifiable par un attribut qui serait visible pour lui mais invisible pour les autres :
  - Chiffrer attribut avec clé publique du destinataire.
- Désavantage : deni de service, non-scalable (limité).

Anonymat de l'expéditeur : proxies

- Les paquets (requêtes HTTP) anonymisées par un proxy.
- Exemple : `www.anonymizer.com`.
- Désavantages :
  - Pas d'anonymat jusqu'au proxy.
  - Le proxy connaît tout – qui le gère ?
- Généralisation : cascade de proxies avec chiffrement :
  - Supposons que le client partage une clé avec le proxy ;
  - Alors il renvoie $\{M, S\}_{K_P}$ ($S =$ adresse du serveur).
  - On généralise à une chaîne de proxies, avec chiffrement en cascade.
  - Un seul proxy non-compromis serait suffisant.

Réseaux mix

- Mécanisme pour développer un canal anonyme.
- Conçu pour oeuvrer dans un environnement où l'attaquant peut
  - Apprendre l'origine, la destination et la représentation de tous les messages qui transitent le système de communication.
  - Injecter, enlever ou modifier les messages.
  - Mais il ne lui est pas possible de déterminer la correspondence entre un ensemble de messages chiffrés et un autre ensemble de messages non-chiffrés.
  - Et il ne peut pas non plus forger des nouveaux messages.
- Proposée à l'origine pour des e-mails anonymes.
- A donné naissance à toute une série de re-mailers (Cypherpunk, Mixmaster).

Un seul mix

- Un *mix* est un serveur qui traite des items (e-mails, par exemple).
- Représente une variante de proxy, avec chiffrement et remplissage (*padding*).
- Client veut envoyer de manière anonyme message M à l'adresse A.
  - Le client envoie au mix $\{R_1, \{R_0, M\}_{K_A}, A\}_{K_1}$.
  - Le mix renvoie $\{R_0, M\}_{K_A}, A$.
- Caractéristiques :
  - Les R_i sont des bits de remplissage – pour avoir des messages de longueur égale !
  - L'information répétée doit être bloquée – pour éviter des attaques de rejeu de la part des participants malhonnêts.
  - L'ordre d'arrivée peut être cachée en envoyant les items (messages) par groupes.
  - On doit assurer un traffic suffisament élevé, pour avoir un degré suffisament grand d'anonymat.
  - *Solution* : les clients envoient régulièrement des messages factices.

Génération de reçus

- Si nécessaire, le mix peut renvoyer à l'émetteur un reçu du message :

$$Y = \{C, \{R_1, \{R_0, M\}_{K_A}, A\}_{K_1}\}_{K_1^{-1}}$$

  où C est une constante de grande taille, connue par tout le monde.
- L'émetteur peut ultérieurement prouver que c'est lui qui a envoyé le message en fournissant $\{R_0, M\}_{K_A}, A$ et en retenant la séquence de remplissage R_1.
- Le mix peut ensuite vérifier la réclamation :

$$\{Y\}_{K_1} = C, \{R_1, X\}_{K_1}$$

Réseaux de mix

- Le groupe d'anonymat dispose de plusieurs mix.
- L'expéditeur encapsule son message dans une cascade de chiffrements, chacun avec la clé d'un mix :

$$\{R_n, \{\ldots\{R_1, \{R_0, M\}_{K_A}, A\}_{K_1}, \ldots A_{n-1}\}_{K_n}$$

- Chaque mix "pèle" le niveau le plus à l'extérieur et revoie le résultat.
- Comme pour le cas d'une cascade de proxies, il suffit d'avoir un seul mix non-compromis pour assurer l'anonymat.

Adresses de retour non-traçables

- Pour répondre à un expéditeur anonyme x avec un message de retour M – le cas d'un seul mix (de clé K_1) :
- L'expéditeur inclut l'adresse de retour :

$$\{R_1, A_x\}_{K_1}, K_x$$

  - R_1 est une nonce qui sera utilisée en tant que clé partagée avec le mix.
  - K_x est une clé publique fraîche, créée spécialement pour cela.
  - A_x est l'adresse de l'expéditeur.
- Le destinataire renvoie sa réponse au mix :

$$\{R_1, A_x\}_{K_1}, \{R_0, M\}_{K_x}$$

- Le mix renvoie alors à A_x le message

$$\{\{R_0, M\}_{K_x}\}_{R_1}$$

- Chiffrement avec R_1 permet de masquer la correlation entre message initial et réponse.
- Seulement l'expéditeur A_x peut déchiffrer le message car c'est lui qui a créé K_x et R_1.

Adresses de retour non-traçables – cas général

- Généralisation des adresses de retour du cas d'un seul mix :

$$\{R_1, \{R_2, \ldots \{R_n, A_x\}_{K_n}, \ldots\}_{K_2}\}_{K_1}, K_x$$

- Le destinataire renvoie cette réponse au premier mix :

$$\{R_1, \{R_2, \ldots \{R_n, A_x\}_{K_n}, \ldots\}_{K_2}\}_{K_1}, \{R_0, M\}_{K_x}$$

- Résultat du premier mix :

$$\{R_2, \ldots \{R_n, A_x\}_{K_n}, \ldots\}_{K_2}, \{\{R_0, M\}_{K_x}\}_{R_1}$$

- Ce que le dernier mix renvoie à A :

$$\{A_x, \{R_1, \{\ldots \{\{R_0, M\}_{K_x}\}_{R_1} \ldots\}_{R_n}$$

Signature aveugle

- Authentification d'un message sans avoir à révéler le contenu de celui-ci
  - Le demandeur de signature cache son message dans "une enveloppe".
  - Le signataure reçoit l'enveloppe et la signe avec sa clé privée.
  - ... sans ouvrir l'enveloppe !
- Utile dans des protocôles à plusieurs participants :
  - La source du message n'est pas le même que le signataire.
- Services de sécurité assurés : intimité et anonymat.
  - *Non-traçabilité* : le signataire pourrait même pas être capable de certifier, a posteriori, que le contenu (divulgué) d'une enveloppe est bien le contenu de l'enveloppe qu'il a signé *à un certain moment !*.
- Implémentation : utilisation d'algorithmes à clés publiques, mais avec des facteurs "aveuglants".

RSA (Rivest, Shamir & Adleman)

1. Alice choisit p et q nbrs. premiers grands et calcule $N = p \times q$.
2. Alice choisit e aléatoire sans diviseur commun avec $(p-1)$ ou $(q-1)$:

$$\gcd\big(e, (p-1)(q-1)\big) = 1$$

3. Alice publie la paire (N, e) – clé publique.
4. La paire (p, q) sert à Alice à calculer l'unique d – clé privée d'Alice – avec

$$e \times d \equiv 1 \mod (p-1)(q-1) = \phi(pq)$$

5. Bob chiffre son message M et envoie à Alice le message chiffré C :

$$C \equiv M^e \mod N$$

6. Alice déchiffre le message avec sa clé privée :

$$M \equiv C^d \mod N$$

♦ Propriété de correctitude :

$$(M^e)^d \equiv M^{e \cdot d} \equiv M \mod N \text{ car } de \equiv 1 \mod \phi(N)$$

Signature aveugle avec RSA

D. Chaum (1982)

- Le propriétaire multiplie le message avec un facteur aléatoire, pour "brouiller" celui-ci :
  - Choisir r t.q. $\gcd(r, N) = 1$.
  - Envoyer $B \equiv (r^e \cdot M) \mod N$.
- Le signataire reçoit le message brouillé et le signe comme d'habitude :

$$S \equiv B^d \equiv (r^{ed} \cdot M^d) \equiv (r \cdot M^d) \mod N$$

- Le propriétaire retrouve le message initial, signé par le signataire :

$$O \equiv r \cdot M^d \cdot r^{-1} \mod N$$

Signature aveugle en BouncyCastle

- Classe `RSABlindingEngine` – utilisée pour brouiller le message et pour enlever le brouillage.
  - Constructeur sans arguments.
  - `void init(boolean forEncryption, CipherParameters param)` – initialisation.
  - `byte[] processBlock(byte[] in, int inOff, int inLen)` – pour créer le message brouillé.
- Classe `RSABlindingParameters` – création de paramètres de brouillage.
  - Constructeur `RSABlindingParameters(RSAKeyParameters publicKey, BigInteger blindingFactor)`
  - Objet à employer dans la `RSABlindingEngine`.
- Classe `RSABlindingFactorGenerator`
  - Constructeur sans arguments.
  - `void init(CipherParameters param)` – paramètres = clé du `RSABlindingParameters`.
  - `BigInteger generateBlindingFactor()`.

Mise en place de la signature aveugle

- Combinaison avec un algorithme de hachage
  - Moins de surcharge de calcul.
  - Plus de sécurité – message plus court, plus difficile à faire de la cryptanalise.
  - Évite les calculs de remplissage.
- ♦ Exemple : combinaison avec des algorithmes d'association message-signature :
  - Utiliser un algorithme de signature qui construit des blocs compatibles avec RSA.

Exemple pour BouncyCastle

RSA PSS = Probabilistic Signature Scheme.

- Utiliser un `PSSSigner` pour le remplissage correcte des données.
  - Constructeur `PSSSigner(AsymmetricBlockCipher cipher, Digest digest, int sLen)`.
  - `cipher` = `RSABlindedEngine` pour faire le bourrage.
  - `digest` = un algorithme de type Message Digest (ex. comme implémenté dans la classe `SHA1Digest`)
  - `slen` = longueur du “sel” à utiliser – la même que le nombre d’octets utilisés pour le Digest.
  - `void init(boolean forSigning, CipherParameters param)` – pour initialiser, `params` = le même que celui utilisé lors du dé-brouillage !
  - `byte[] generateSignature()` – et on retrouve le message brouillé.

Exemple pour BouncyCastle (cont.)

- Lors de la création de la signature, utiliser un `RSAEngine`, qui implémente strictement l'algo RSA (sans autres fioritures...)
  - Constructeur `RSAEngine()`.
  - Initialiser avec `signerEngine.init(boolean, CipherParameters ciph)`, où `ciph` est la clé privée du signataire.
  - Construire la signature aveugle avec `byte[] processBlock(byte[] data, int start, int length)`.
- Lors de la vérification, utiliser toujours un `PSSSigner`,
  - Initialisé avec la clé publique du signataire (donc à la place de `params` !)
  - Verifie signature avec `boolean verifySignature(byte[] signature)`.

Conclusions

- Réseaux mix = grand dégré d'anonymat :
  - Pas de correlation entre entrée et sortie dans un mix.
  - Seulement une fraction des mix doivent être honnêts.
  - Avec un traffic factice suffisament grand, l'anonymat est assurée pour le réseau entier.
- La cryptographie est utilisée dans un nouveau contexte :
  - Avec les réponses anonymes, l'expéditeur est anonyme même pour le destinataire.
  - Peut also utilisé pour l'envoi de messages "anonymes certifiés", où le reçu est signé à la fois par le destinataire et par chaque mix sur le chemin d'accès.
- Désavantages : délais réseau, grand surcoût de communication, chiffrement multiple.
  - Dans un réseau avec communication substantielle, les messages factices peuvent être reduits.

Application : vote électronique

Conditions de sécurité dans les systèmes de vote électronique :

– Exactitude :
  1. Il n'est pas possible d'altérer un vote valide.
  2. Il n'est pas possible d'éliminer un vote valide du contrôle (comptage) final.
  3. Il nést pas possible de compter un vote invalide.

– Démocratie :
  1. Il est permis seulement aux électeurs inscrits (validés) de voter.
  2. Tout électeur inscrit peut voter une seule fois.

– Intimité :
  1. Personne (même l'authorité contrôlant l'élection !) ne peut pas faire de lien entre un vote et un votant.
  2. Aucun votant ne peut prouver la façon dont il a voté.
  3. Tous les votes restent sécrets tant que l'élection n'est pas terminée.

– Vérifiabilité :
  1. Les votants peuvent vérifier si leurs votes ont bien pris en compte dans le décompte final.

Protocôle SENSUS

- Utilisation de trois autorités :
  1. Autorité d'inscription.
  2. Autorité de légitimation (AL).
  3. Autorité de décompte (AD).

♦ Trois phases :

1. Phase d'inscription des votants.
2. Phase de vote.
   - Étape de validation du bulletin de vote
   - Étape de collection des votes.
3. Phase de décompte.

Phase d’inscription

Votant se fait authentifier auprès de l’autorité d’inscription.

– Votant crée paires de clés publique/privée.
– Votant se fait authentifier sa clé publique auprès de l’autorité d’inscription.
– Votant garde sa clé privée.
– Tout ce que votant va signer avec sa clé privée sera considéré comme provenant de manière authentique du votant.
– AL récupère toutes les clés des votants.
– Votant récupère clé publique de l’AD.

Peut se faire aussi en utilisant des clés secrètes.

Phase de vote

1. Votant crée une clé secrète pour son vote.
2. Votant prépare son bulletin de vote.
3. Votant hache le bulletin de vote
4. Votant commence sa phase dans le protocôle de signature aveugle :
   – Ex. avec RSA PSS : brouillage du bulletin de vote avec `PSSSigner`.
5. Votant crée aussi une signature du bulletin brouillé avec sa clé privée (authentifiée auprès de l'autorité d'inscription).
6. Votant envoie bulletin + signature à l'AL.
7. AL vérifie la signature du votant avec la clé publique correspondante.
8. AL marque le votant comme ayant voté ou réfuse son vote s'il a déjà voté.
9. AL signe en aveugle le bulletin brouillé (validation !).
10. AL renvoie le bulletin signé au votant.

Phase de vote (2)

11. Votant dé-brouille son bulletin.
12. Votant chiffre son bulletin de vote avec la clé secrète.
13. Votant envoie à l'AD son bulletin de vote haché et signé, ainsi que son bulletin de vote chiffré.
14. AD vérifie la signature du bulletin haché et place le bulletin crypté dans la liste des votes à compter.
15. AD renvoie le bulletin haché, signé par l'AD même.
16. Votant vérifie signature de l'AD, la garde comme accusé de réception, et renvoie sa clé secrète à l'AD.
17. AD décrypte les bulletins avec les clés secrètes reçues et fait les décomptes.

Avantages et faiblesses du protocôle SENSUS

- Avantages :
  - Prise en compte seulement des votes valides, non-altération des votes.
  - Anonymat et vérifiabilité.
  - Vote en une seule session (excl. inscription).
- Faiblesses :
  - Pas de possibilité de contrer l'achat des votes.
  - Unique AL – cible d'attaques.