Table of contents

Authentification

Authentification de l'origine des données et d'entités

Méthodes d'authentification

L'authentification de l'origine garantit qu'un message provient bien de l'entité prétendument émettrice.

Méthodes symétriques :

- **MAC seul** : `A → B: X, MACk(X)` - B vérifie avec la clé partagée k
- **MDC + cryptage** : `A → B: X, Ek(MDC(X))` ou `A → B: Ek(X, MDC(X))`

Méthode asymétrique :

- **MDC + signature** : `A → B: X, Sigpriv-A(MDC(X))` - Offre en plus la non-répudiation

Limitations : Ces protocoles simples ne protègent ni contre les replay attacks ni ne garantissent l'actualité des messages. Des mécanismes tenant compte du temps ou du contexte sont nécessaires.

Texte original

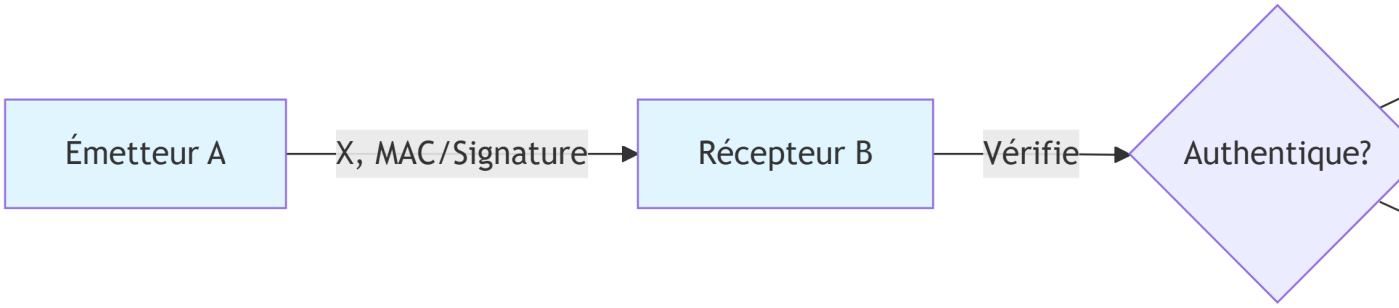
Authentification de l'origine des données
1) MAC avec une clé symétrique k connue de A et B : `A → B: X, MACk(X)` Si B calcule de son côté `MACk(X)` et obtient la même valeur le message provient de A.
2) MDC + cryptage symétrique (clé k connue de A et B) `A → B: X, Ek(MDC(X))` B calcule `MDC(X)` puis `Ek(MDC(X))`. Si égal message vient de A.
3) Comme 2) avec confidentialité de X en plus : `A → B: Ek(X,MDC(X))`
4) MDC + signature digitale : `A → B: X, Sigpriv-A(MDC(X))` B calcule `MDC(X)` et vérifie `Sigpriv-A(MDC(X))` avec une copie authentique de pub-A. Si égalité A est à l'origine du message. Cette solution offre en plus la **non-répudiation d'origine**.
Ces protocoles simples n'offrent aucun support sur l'**unicité** ni sur l'**actualité (timeliness)** des messages reçus et sont exposés à des **replay attacks**. Ils nécessitent des mécanismes tenant compte du temps ou du contexte de la transaction.

Révision rapide

4 méthodes

- MAC seul
- MDC+cryptage
- MDC+cryptage confidentiel
- MDC+signature

Attention : Vulnérable aux replay attacks sans mécanisme temporel



Authentification d'Entités - Introduction

Objectifs d'un protocole robuste

L'authentification d'entités (ou identification) vise à prouver l'identité d'une entité en temps réel.

Propriétés requises :

1. Si A et B sont honnêtes et A s'authentifie, B doit accepter l'identité de A
2. B ne peut pas réutiliser l'information de A pour se faire passer pour A auprès de C
3. Probabilité négligeable qu'une entité C réussisse à usurper l'identité de A
4. La propriété 3 reste vraie même si C a observé ou participé à des instances précédentes

Éléments de base :

- **Something known** : passwords, PINs, clés
- **Something possessed** : carte à puce, générateur de passwords
- **Something inherent** : biométrie (empreintes, rétine, ADN)

Classification :

- **Authentification faible** : Révélation du secret (userid/password)
- **Authentification forte** : Preuve de possession du secret sans le révéler
- **Zero-knowledge** : Authentification forte sans révéler aucune information sur le secret

Texte original

Authentification d'entités (entity authentication), aussi appelé **identification**
Objectifs d'un protocole d'identification robuste :

1. Si A et B sont "honnêtes" : si A est capable de s'authentifier auprès de B, B doit accepter l'identité de A.
2. B ne peut pas réutiliser l'information remise par A pour s'identifier en tant que A auprès de C.
3. La probabilité qu'une tierce entité C réussisse à se faire passer par A auprès de B est négligeable.
4. Le point 3) reste vrai même si :
    - C a observé un grand nombre (polynomial) d'instances du protocole d'identification entre A et B
    - C a participé (éventuellement en se faisant passer par quelqu'un d'autre) à des exécutions précédentes du protocole d'identification auprès de A ou B
    - Plusieurs instances du protocole (éventuellement initiées par C) peuvent s'exécuter simultanément sans compromettre le processus d'identification

Terminologie : L'utilisateur (A) est appelé **claimant** (celui qui prétend être A), le système (B) est le **verifier** (celui qui vérifie l'identité).
Éléments de base pour l'authentification :

- **something known** : passwords, PINs, clés privées ou secrètes, etc.
- **something possessed** : passeport, carte à puces, générateurs de passwords, etc.
- **something inherent to the human individual** : propriétés biométriques comme les empreintes digitales, la rétine, le code ADN, etc.

Authentification faible (weak authentication) : L'utilisateur présente un couple (userid, password) au système. Le userid est l'identité prétendue et le password l'évidence corroborant.
Authentification forte (strong authentication) : Le secret permettant de corroborer l'identité n'est pas révélé explicitement. L'utilisateur fournit au système une preuve de possession de ce secret.
Authentification par zero knowledge : Protocoles d'authentification forte qui ont en plus la caractéristique de prouver l'identité sans dévoiler aucune information (ni même une piste) sur le secret lui-même. Il s'agit de donner une preuve d'une assertion sans en révéler le moindre détail.
Les protocoles d'**authentification faible** satisfont les points 1) et 3). Les protocoles d'**authentification forte** satisfont (au moins partiellement) les points 2) et 4) en plus.

Révision rapide

3 niveaux : Faible (révèle secret) < Forte (preuve de possession) < Zero-knowledge (aucune info révélée)
4 objectifs

- Acceptation si honnête
- non-réutilisation
- résistance usurpation
- résistance observation

Attaques et contre-mesures

Attaques Dictionnaire

Principe et contre-mesures

Une attaque dictionnaire teste systématiquement des mots de passe probables contre un système cryptographique.

Méthodes d'attaque :

- **Offline** : L'attaquant obtient la base de données hashées des mots de passe ou capture des échanges
- **Online** : Tentatives directes contre le système (généralement limitées par le système)

Exemple de vulnérabilité :

- A $\rightarrow$ B: A
- A $\leftarrow$ B: R (challenge aléatoire)
- A $\rightarrow$ B: E_p(R)

Le couple (R, E_p(R)) permet une attaque dictionnaire offline.

Contre-mesures :

- Limitation des tentatives online
- Salting (ajout d'un élément aléatoire)
- Utilisation de fonctions de dérivation lentes
- Authentification forte évitant la transmission du password

Texte original

Attaques Dictionnaire (Dictionary Attacks)
Une attaque dictionnaire consiste à utiliser une base de données contenant des mots de dictionnaire d'une ou plusieurs langues (ainsi que des variantes) comme entrée à un système d'encryption ou de hachage afin d'obtenir des clés secrètes ou des passwords.
Cette attaque est très efficace pour obtenir des mots de passe de mauvaise qualité même si dès nos jours il existent des bases de données de très grande taille contenant des variations de mots ainsi que des règles mnémotechniques complexes permettant de "casser" des mots de passe de plus forte entropie.
Une attaque dictionnaire peut être montée :

- En obtenant la base de données des mots de passe (encryptée ou hashée) du système d'authentification
- À partir d'un ou plusieurs échanges d'une instance d'authentification, suite à une attaque passive (observation de paquets réseau). Par exemple :
  - A $\rightarrow$ B: A (A envoie son identité)
  - A $\leftarrow$ B: R (R = un nombre aléatoire, challenge)
  - A $\rightarrow$ B: E_p(R) (A encrypte R avec son password)

  Le couple (R, E_p(R)) permet de monter une attaque dictionnaire **offline**.

Les attaques dictionnaire sont normalement moins efficaces **online** car les systèmes d'exploitation limitent le nombre d'essais infructueux d'authentification.

Révision rapide

Offline (via BdD ou capture) > **Online** (limitée par le système)
Protection : salting, limitation tentatives, authentification forte

---

Plaintext-Equivalence

Concept et risques

Une chaîne est **plaintext-equivalent** à un password si elle permet d'obtenir le même accès que le password lui-même.

Exemple de vulnérabilité :

Si le système stocke `H(p)` et que le protocole est : `A → B: H(p)`

Alors `H(p)` est plaintext-equivalent à `p` car l'attaquant peut l'utiliser directement.

Contre-exemple (UNIX classique) :

Le système stocke `H(p)` mais le protocole transmet `p`. Le hash stocké n'est donc pas plaintext-equivalent.

Principe de sécurité : Les informations stockées par le serveur ne doivent être ni plaintext-equivalent aux passwords ni exposées à des attaques dictionnaire offline.

Texte original

Equivalence Plaintext (Plaintext-Equivalence)
Une chaîne de données est dite **plaintext-equivalent** à un mot de passe si elle peut être utilisée pour obtenir le même niveau d'accès correspondant à l'utilisation du password.
Exemple : Si le système B stocke une liste de tous les mots de passe hashés dans le procédé d'authentification suivant : `A → B: H(p)` (A envoie à B le hash du password)
La chaîne d'information `H(p)` est **plaintext-equivalent** au mot de passe `p`.
Ceci est équivalent à dire que l'application d'une fonction de hachage pour le stockage des passwords ne constitue pas une sécurité supplémentaire pour le système.
Contre-exemple : Dans le système d'authentification classique d'UNIX, le hash du password stocké dans le fichier `/etc/passwd` n'est **pas** plaintext-equivalent au mot de passe car c'est `p` et non pas `H(p)` qui est échangé entre le client et le serveur.

Cette propriété est essentielle car les bases de données des mots de passe sont normalement protégées par des mécanismes logiques qui sont souvent mis en évidence par des failles du système d'exploitation du serveur.
Si ces bases de données centrales contiennent des mots de passe en clair ou des informations plaintext-equivalent à ces derniers, les conséquences en cas d'attaque sont dévastatrices.
Le cas idéal est que les informations stockées par le serveur ne soient ni plaintext-équivalent aux passwords ni exposées à des attaques dictionnaire offline.

Révision rapide

Plaintext-equivalent : Donnée utilisable comme le password original
Danger : Si le système transmet `H(p)` et stocke `H(p)` → `H(p)` est plaintext-equivalent
Bon design : Système transmet `p`, stocke `H(p)` → pas plaintext-equivalent

---

Authentification Faible

Passwords Fixes

Stockage et protection

Les systèmes à password fixe présentent des vulnérabilités importantes.

Techniques de stockage :

- **En clair** : Protection par contrôle d'accès OS (vulnérable aux failles OS, backups)
- **Encrypté ou hashé** : Vulnérable aux attaques offline (guessing, dictionary, collisions)

Problème majeur : Le password peut être rejoué après observation sur un réseau non protégé.

Techniques de protection :

- Règles strictes de création (entropie minimale)
- Ralentissement et limitation du nombre de tentatives
- **Salting** : Ajout d'un élément aléatoire avant hachage
- Restriction de diffusion des fichiers de passwords

Entropie typique des passwords : Faible (~40 bits pour un password de 8 caractères aléatoires, beaucoup moins pour des mots courants).

Texte original

Authentification Faible - Password fixe
Les systèmes d'authentification faible sont divisés en deux catégories principales :

- **Password fixe** : Le password ne dépend pas du temps ni du nombre de fois que le protocole d'identification a été exécuté. Cette catégorie inclut les systèmes où le password est changé par décision de l'utilisateur ou par mesure de sécurité du système.
- **Password variable** : La modification du password en fonction du temps et/ou du nombre d'exécutions fait partie du protocole d'identification.

Techniques de stockage propres aux systèmes à password fixe :

- **Stockage du password en clair** dans un fichier protégé par les mécanismes de contrôle d'accès propres au système d'exploitation.
  - Problèmes : failles dans le OS, privilèges du "super-user", backups, etc.
- **Stockage du password encrypté ou après l'application d'une one-way function** (éventuellement en rendant publique l'accès à ce fichier, cf. exemple UNIX).
  - Problèmes : attaques off-line, i.e. guessing attacks, brute-force dictionary attacks, identification de collisions, etc.

Problème le plus grave du password fixe : il peut être rejoué après avoir écouté une instance d'identification sur un réseau non protégé.
Techniques de protection des systèmes de password fixe :

- Règles strictes de comportement concernant la création, le maintien et la mise à jour des passwords en tenant compte de la faible entropie des passwords choisis habituellement par les utilisateurs
- Ralentir le processus d'identification ainsi que limiter le nombre d'essais infructueux afin de contrer les "on-line brute force attacks"
- **Salting** (cf. exemple UNIX)
- Restreindre ou même éviter la diffusion des fichiers de mots de passe, même encryptés

Révision rapide

2 types : Password fixe (statique) vs Password variable (change à chaque instance)
Stockage : Clair (très vulnérable) vs Encrypté/Hashé (attaques offline)
Protections : Règles strictes, limitation tentatives, salting, non-diffusion

---

Passwords Variables

One-time passwords et générateurs

Les passwords variables changent à chaque authentification, réduisant le risque de replay.

Schéma de Lamport (S-Key) :

```
Initialisation :
  A génère secret w, choisit t
  A → B: wt = Ht(w)
  B stocke: wstored := wt, n := t-1

Identification (t-n)ème :
  A → B: A, n, wn = Hn(w)
  B teste: H(wn) == wstored
  Si OK: n := n-1, wstored := wn
```

Attaques si B non authentifié :

- **Pre-play attack** : C obtient `wn` avant A et le rejoue
- **Small `n` attack** : C demande un `n` < ncourant

Générateurs hardware (SecureID) :

- Carte générant un code toutes les 30-60 secondes
- Basé sur une clé secrète partagée avec le système
- Vulnérable au pre-play mais fenêtre temporelle limitée

> Texte original
>
> **Authentification Faible : Password Variable**
> Les deux techniques les plus connues d'identification par password variable sont les **one-time passwords** et les **générateurs (hardware) de nombres aléatoires**.
> **One-time passwords - Schéma de Lamport (S-Key) :**
> **Initialisation :**
>
> - A génère un secret w
> - Une constante t (= nb. d'identifications ~1000) et une OWF H sont choisies
> - `A → B: wt = Ht(w)` (H appliqué t fois à w)
> - B stocke : `wstored := wt`, `n := t-1`

Messages correspondants à l'identification (t-n)ème :

- `A → B: A` (identité de A)
- `A → B: n` (itération courante pour A)
- `A → B: wn = Hn(w)`
- B teste : `H(wn) == wstored`. Si OK `n := n - 1` et `wstored := wn`

Fin : Quand n == 0, A choisit un nouveau w et on recommence…
Attaques : Authentification de B nécessaire! Sinon : C se fait passer par B et :

- obtient le mot de passe courant wn et peut le rejouer (**pre-play attack**)
- fournit un n < ncourant et peut ainsi générer tous les Hm>n(wn) (**small n attack**)

Générateurs (hardware) de nombres aléatoires :

- Il s'agit de cartes à puces qui génèrent périodiquement (~ tous les 30 ou 60 secs) des nombres différents servant à identifier (avec en plus, un PIN et des informations sur l'identité de la personne) le détenteur de la carte.
- La génération se fait à partir d'une clé secrète présente sur la carte et connue du système.
- Le plus connu est **SecureId** fabriquée par RSA Security.
- Il a été adopté par de nombreuses banques comme support d'authentification du tele-banking sur Internet.
- Il est également exposé au pre-play attack mais le délai pour rejouer le password se limite à la fréquence de changement (30 ou 60 secs).

Conclusions authentification faible :

- Les password fixes offrent un niveau de sécurité très réduit.
- Les password variables constituent un pas important vers l'authentification forte mais nécessitent des précautions supplémentaires.

Révision rapide

Lamport : wn+1 = H(wn), authentification par vérification de la chaîne de hash
Hardware : Générateur synchronisé (30-60s), limité au pre-play
Attention : Nécessite authentification de B pour éviter pre-play et small-n attacks

---

Authentification Forte

Symétrique

Protocoles de Base

Challenge-Response

L'authentification forte utilise la cryptographie pour prouver la possession d'un secret sans le révéler.

Authentification unilatérale basique :

```
A → B: A
A ← B: R (challenge aléatoire)
A → B: Ek-AB(R)
B vérifie en décryptant
```

Clé de session : `K := R`

Améliorations :

- Ajouter identité de B : `Es(B, ra)` pour key confirmation
- Ajouter timestamp : `Es(B, ta, ra)` pour freshness (nécessite horloges synchronisées)
- Utiliser MAC au lieu d'encryption : `Hk-AB(R)` (plus rapide)

Vulnérabilités :

- Man-in-the-Middle si pas d'authentification mutuelle
- Chosen-plaintext attacks possibles
- Replay si challenges mal gérés

> Texte original
>
> **Authentification Forte : Solutions Symétriques**
> Les protocoles d'authentification forte utilisent des techniques cryptographiques symétriques ou asymétriques.
> **Authentification unilatérale à clé symétrique partagée :**
>
> ```
> A → B: A (A envoie son identité)
> A ← B: R (R = un nombre aléatoire, challenge)
> A → B: Ek-AB(R) (A encrypte R avec la clé partagée)
> ```
>
> B décrypte `Ek-AB(R)` et identifie A s'il trouve R
> **Remarques :**

- B doit s'assurer que le challenge R est aléatoire et ne doit pas le répéter.
- Ce protocole constitue une amélioration remarquable par rapport à l'authentification par password car la variation des challenges empêche Eve de rejouer des parties du protocole.
- Eve peut essayer un **off-line known-plaintext attack** à partir d'un nombre (qui reste normalement réduit) de couples (R, Ek-AB(R)) mais la plupart des systèmes de cryptage sont sûrs à cet égard (DES est vulnérable seulement à partir de 247 paires).
- C peut se faire passer par B et choisir ses challenges R pour monter un **chosen-plaintext attack** (la vulnérabilité de DES à cet égard est aussi de 247 mais d'autres systèmes de cryptage sont plus sensibles à ces attaques).
- C pourrait monter une attaque **Active Man-in-the-Middle** en se faisant passer par B puisque B n'est pas authentifié, mais il doit convaincre A pour commencer le protocole.
- Un **MDC** : `H(k-AB,R)` ou un **MAC** : `Hk-AB(R)` peuvent remplacer `Ek-AB(R)` et accélérer l'identification.
- Après l'identification initiale, un canal sûr (au moins authentifié) doit être établi à l'aide d'une protection cryptographique pour éviter que C puisse injecter des paquets en se faisant passer par A.

Les protocoles de ce type où une entité doit répondre en tenant compte d'un challenge proposé par l'autre s'appellent **challenge and response protocols** et sont la forme la plus répandue d'authentification forte.
Authentification unilatérale à clé symétrique partagée, 2ème variante :

```
A → B: A, Ek-AB(timestamp)
```

Horloges synchronisées entre A et B nécessaires.
Avantage : un message en moins et protocole stateless
Mais :

- La synchronisation d'horloges est difficile à obtenir dans la réalité et des "flottements" peuvent être exploités par un adversaire.
- De plus, si on arrive à convaincre B "d'avancer sa montre", certaines instances d'identification passées peuvent redevenir valables.

Révision rapide

Challenge-Response : B envoie challenge R, A répond avec E_k(R)
Alternative : MAC au lieu d'encryption (plus rapide)
Avec timestamp : Un message en moins mais nécessite synchronisation horloges

---

Authentification Mutuelle

Protocoles robustes et reflection attacks

L'authentification bilatérale nécessite des précautions contre les reflection attacks.

Protocole vulnérable (naïf) :

```
A → B: A, R2
A ← B: R1, Ek-AB(R2)
A → B: Ek-AB(R1)
```

Attaque par réflexion : C peut démarrer deux instances et utiliser la réponse de B à sa propre requête pour compléter l'authentification.

Protocole robuste :

```
(1) A → B: A, R2
(2) A ← B: Ek-AB(R1, R2, A)
(3) A → B: Ek-AB(R2, R1)
```

Protections :

- Inclusion de l'identité A dans (2) contre reflection attacks
- Asymétrie dans l'ordre des challenges (R1,R2) vs (R2,R1)
- Inclusion des challenges dans le message encrypté

> Texte original
>
> **Authentification Forte : Solutions Symétriques (Authentification mutuelle)**
> **Authentification bilatérale à clé symétrique partagée (solution intuitive) :**
>
> ```
> A → B: A, R2
> A ← B: R1, Ek-AB(R2)
> A → B: Ek-AB(R1)
> ```
>
> À première vue le protocole semble robuste mais observons ce qu'un adversaire C peut faire en démarrant deux processus d'identification :
>
> ```
> C → B: A, R2 (C prétend être A)
> C ← B: R1, Ek-AB(R2) (B répond)
> ```

À ce moment, C démarre une deuxième instance :

```
C → B: A, R1
C ← B: R3, Ek-AB(R1) (C ne peut plus poursuivre mais...)
```

Complète avec succès la première instance d'identification avec :

```
C → B: Ek-AB(R1) (et c'est fait !)
```

Du fait que C renvoie à B le même R qu'il a reçu de lui, ce genre d'attaques s'appellent **reflection attacks**.
Comme la clé est partagée, C aurait pu obtenir le même résultat (même plus discrètement) en exécutant la deuxième instance auprès de A (en prétendant être B).
Authentification bilatérale avec clé symétrique partagée (solution robuste) :

```
(1) A → B: A, R2
(2) A ← B: Ek-AB(R1, R2, A)
(3) A → B: Ek-AB(R2, R1)
```

La présence de **A** dans (2) rajoute une sécurité supplémentaire au cas où les reflection attacks évidents ne sont pas détectés par le protocole. Autrement, si A lance une authentification avec celui qu'il croit B mais qui est en réalité C :

```
A → C: A, R2 (*)
```

Alors C commence une nouvelle instance d'authentification avec A avec le même R2 :

```
C → A: B, R2
```

Si A ne voit pas R2 comme réflexion évidente, alors il répond :

```
C ← A: Ek-AB(R1, R2) (Comme dans (2) mais sans le 'A')
```

Ce qui est utilisé par C pour compléter son protocole (*). Cependant, si A répond avec B à l'intérieur du paquet comme recommandé dans le protocole :

```
A → C: Ek-AB(R1, R2, B)
```

Ceci ne sera plus utilisable par C pour continuer (*) car il faudrait A à la place de B.
À noter également que le fait d'inclure R1 dans la partie encryptée protège également des dangers de **chosen plaintext attacks** de la solution précédente.

Révision rapide

Reflection attack : Utiliser la réponse d'une session pour en authentifier une autre
Protection : Inclure identités + asymétrie dans challenges (R1,R2) vs (R2,R1)

Asymétrique

Protocoles à clés publiques

L'asymétrie permet d'éviter le partage de secrets mais nécessite des précautions contre les chosen-ciphertext attacks.

Protocole vulnérable :

```
A → B: A
A ← B: Epub-A(R)
A → B: R
```

Problème : B peut faire décrypter n'importe quoi à A.

Protocole robuste :

```
A → B: A
A ← B: H(R), B, Epub-A(B, R)
A → B: R (après vérification de H(R) et B)
```

Protection : Structurer le texte encrypté et prouver la connaissance du plaintext via `H(R)`.

Authentification mutuelle (Needham-Schroeder) :

```
(1) A → B: Epub-B(r1, A)
(2) A ← B: Epub-A(r1, r2)
(3) A → B: Epub-B(r2)
```

La présence de A dans (1) empêche les chosen-ciphertext attacks.

Texte original

Authentification Forte : Solutions Asymétriques
Authentification unilatérale à clé asymétrique (solution intuitive...) :

```
A → B: A
A ← B: Epub-A(R) (B encrypte avec la clé publique de A)
A → B: R (A retourne R après décryptage)
```

Remarques :

- B doit connaître la clé authentique de A pour éviter des man-in-the-middle attacks.
- Mais surtout : B peut monter des **chosen-ciphertext attacks** (i.e. B peut faire décrypter n'importe quoi à A!).

Authentification unilatérale avec clé asymétrique (solution robuste) :
Idée : structurer le texte encrypté avec pub-A et montrer que B connaît le plaintext :

```
A → B: A
A ← B: H(R), B, Epub-A(B, R) (H(R) témoigne du fait que B connaît R)
```

A décrypte `Epub-A(B, R)` et obtient B' et R'. A suspend le protocole si `h(R')` `h(R)` ou `B'` `B`, sinon :

```
A → B: R
```

B identifie A si coïncidence avec le R initial.
Un protocole dual peut être imaginé en utilisant la signature de A avec priv-A (au lieu de l'encryption avec pub-A), mais les mêmes précautions concernant la structure s'appliquent pour éviter que A signe un message "mal intentionné" généré par B.
Authentification bilatérale à clé asymétrique. Solution robuste due à Needham et Schroeder :

```
(1) A → B: Epub-B(r1, A)
(2) A ← B: Epub-A(r1, r2)
(3) A → B: Epub-B(r2)
```

À noter que la présence de A dans (1) démonte les chosen ciphertext attacks.
Le protocole peut être renforcé en rajoutant un "témoin" `H(r1)` dans (1).
Remarques finales sur authentification classique :

- L'authentification d'entités est un processus très complexe rempli de pièges inespérés.
- Certains protocoles comme celui proposé par l'ISO en 1988 pour l'authentification dans les répertoires distribués ont des failles très semblables à celles que nous avons mis en évidence ici.
- Lorsque l'identification se fait dans le cadre d'une session, il est impératif que tous les paquets propres à la session soient authentifiés (p.ex. moyennant l'établissement d'un canal sûr avec l'établissement de clés de session).

Révision rapide

Vulnérabilité : Chosen-ciphertext attacks si pas de structure
Protection : Inclure H(R), identité B dans le message encrypté, A vérifie avant de révéler R
Needham-Schroeder : 3 messages avec inclusion identités pour éviter chosen-ciphertext

---

Zero-Knowledge Proofs

Concepts

Définitions et principes

Les preuves à divulgation nulle de connaissance permettent de prouver la possession d'un secret sans révéler aucune information sur celui-ci.

Propriétés requises :

- **Consistance (completeness)** : Si A et B sont honnêtes, B accepte la preuve de A
- **Significativité (soundness)** : Si C réussit à tromper B, alors C détient le secret de A (ou équivalent)
- **Zero-knowledge** : B ne peut rien apprendre sur le secret de A

Structure générique :

```
(1) A → B: témoin (witness)
(2) A ← B: défi (challenge)
(3) A → B: réponse (response)
```

Types de ZKIP :

- **Computational ZKIP** : Un observateur en temps polynomial ne peut distinguer une vraie preuve d'une simulation
- **Perfect ZKIP** : Aucune différence probabiliste entre vraie preuve et simulation (garantie par théorie de l'information)

Principe :

- A s'engage sur une classe de questions (1)
- B choisit une question dans cette classe (2)
- A répond en utilisant son secret (3)

- Répétition pour réduire la probabilité de deviner.

Texte original

Zero Knowledge Proofs : Définitions
Problème avec les méthodes d'authentification "classiques" : B (ou même un observateur) est en mesure d'obtenir des informations sur le secret détenu par A :

- Dans les méthodes d'authentification faible (par password) c'est le secret dans son intégrité qui est dévoilé.
- Dans les méthodes challenge and response classiques, B peut obtenir des couples [plaintext / ciphertext] pouvant servir à la cryptanalyse.

Définition : Un protocole interactif est une **preuve de connaissance (proof of knowledge)** lorsqu'il a les deux caractéristiques suivantes :

- **Consistance (completeness)** : si A et B sont deux entités "honnêtes", B accepte la preuve fournie par A.
- **Significativité (soundness)** : Si une entité "malhonnête" C est capable de "tromper" B alors C détient le secret de A (ou une information polynomialement équivalente au secret). Ceci équivaut à exiger la possession du secret pour la réussite de la preuve.

Une preuve de connaissance interactive est dite **"sans apport d'information" (zero knowledge interactive proof ou ZKIP)** si elle a, en plus, la propriété que A est capable de convaincre B sur un fait sans ne révéler aucune information sur le secret qu'elle possède.
Un protocole est une **ZKIP calculatoire (computational ZKIP)** si un observateur capable d'effectuer des tests probabilistes en temps polynomial n'est pas capable de distinguer une preuve authentique (où A répond) d'une preuve simulée (p.ex. par un générateur aléatoire).
Un protocole est une **ZKIP parfait (perfect ZKIP)** s'il n'existe aucune différence (au sens probabiliste) entre la vraie preuve et la preuve simulée. L'absence d'information dans la preuve est garantie par la théorie de l'information de Shannon et non pas par des critères calculatoires.
Structure générique d'une ZKIP :

```
(1) A → B: témoin (witness)
(2) A ← B: défi (challenge)
(3) A → B: réponse (response)
```

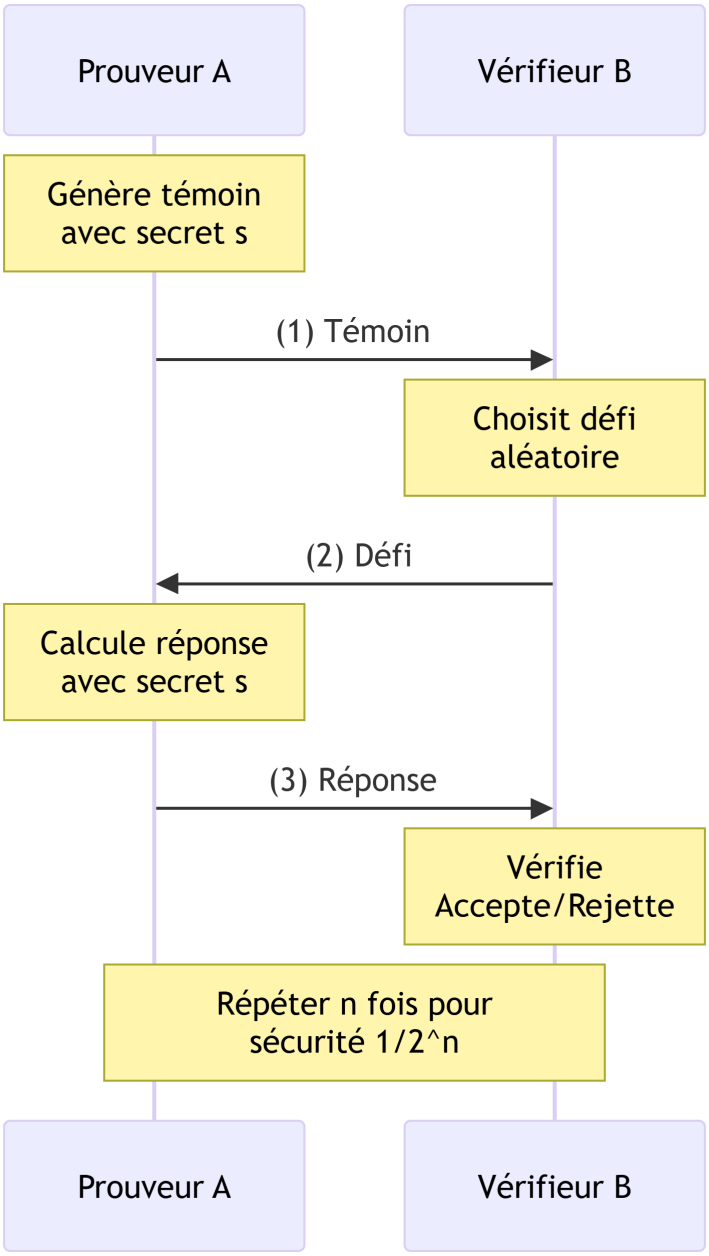
- **(1)** A choisit un nombre aléatoire secret et envoie à B une preuve de possession de ce secret. Ceci constitue un engagement de la part de A et définit une classe de questions à laquelle A prétend savoir répondre.

- **(2)** Le défi envoyé par B choisit (aléatoirement) une question dans cette classe.
- **(3)** A répond (en utilisant son secret).

Si nécessaire, le protocole est répété afin de réduire au maximum la probabilité qu'un "imposteur" devine "par chance" les réponses correctes.

Révision rapide

3 propriétés : Consistance (accepte si honnête), Significativité (nécessite secret), Zero-knowledge (aucune info révélée)
Structure : Témoin → Défi → Réponse (répéter n fois)
Perfect ZK : Indistinguable d'une simulation même avec ressources infinies



ZKIP - Exemple Intuitif (Caverne d'Ali Baba)

Illustration du concept

Cet exemple illustre intuitivement le principe de zero-knowledge.

Scénario :

- A connaît le passage secret entre y et z dans une caverne
- B veut vérifier cette connaissance sans apprendre comment traverser

Protocole :

1. B se tient à l'entrée E
2. A choisit d'aller vers y ou z (témoin)
3. B entre et s'arrête au point x
4. B demande à A de revenir par la droite ou la gauche (défi)
5. A utilise le secret pour obéir (si nécessaire)

Répétition : n fois. Si A ne connaît pas le secret : probabilité de succès = 2^{-n}

Propriétés :

- B constate que A peut traverser mais n'apprend pas comment
- B ne peut convaincre une tierce partie B' (A et B auraient pu convenir des séquences)
- Inspiré de la technique du "cut and choose"

Texte original

ZKIP : Exemple Intuitif
Cet exemple est décrit dans [Qui89] (Quisquater et al., "How to Explain Zero-Knowledge Protocols to Your Children", Crypto'89). Admettons que A connaît un passage entre y et z (le secret).

(1) B se tient à l'entrée de la caverne au point E.
(2) A choisit une direction et se dirige vers les points y ou z (choix de témoin).
(3) Une fois A à l'intérieur de la caverne, B entre à son tour mais s'arrête au point x.
(4) B demande à A de se rendre au point x par la droite ou par la gauche (le défi).
(5) En utilisant le secret pour passer de y à z (ou réciproquement) si nécessaire, A obéit aux instructions de B.
Répéter les points 1 à 5 n fois. Si A ne connaît pas le secret, il a une probabilité de 2^{-n} de réussir à tromper B (de deviner "juste").
Dans cet exemple, B constate que A peut traverser à volonté le passage yz mais n'obtient aucune information sur la manière de le faire même si le protocole est exécuté des millions

de fois.
Par ailleurs, B ne peut pas convaincre B' du fait que A connaît le secret (comme il aurait été le cas si A encryptait une information en utilisant une clé privée, p.ex.). B' pourrait suspecter A et B d'avoir convenu les séquences (droite/gauche).
Ce genre de protocoles sont inspirés de la technique du **"cut and choose"** où A et B partagent équitablement une tarte en suivant les étapes suivantes : - A coupe la tarte. - B choisit un morceau. - A prend le morceau restant.
Le premier ZKIP a été publié en 1985 par S. Goldwasser [Gol85]. L'application du paradigme du cut and choose aux protocoles cryptographiques est due à Rabin [Rab78].

Révision rapide

Caverne : A entre aléatoirement (y ou z), B demande sortie (gauche/droite)
Probabilité triche : 2^{-n} après n répétitions
ZK : B vérifie connaissance mais n'apprend pas le secret, ne peut convaincre tierce partie

---

ZKIP - Isomorphisme de Graphes

ZKIP - Isomorphisme de Graphes

Protocole formel

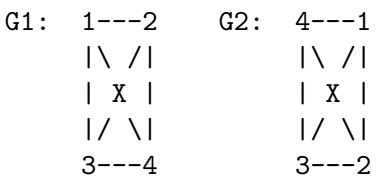
Les preuves zero-knowledge peuvent être construites sur des problèmes mathématiques difficiles.
Contexte : Deux graphes G_1 et G_2 sont isomorphes s'il existe une permutation π telle que pour tout arc $\{u, v\} \in E_1$, on a $\{\pi(u), \pi(v)\} \in E_2$.
Propriété : Trouver la permutation π entre deux graphes de ~1000 sommets est calculatoirement difficile (pas d'algorithme polynomial connu).
Protocole :

```
Init: A choisit G1 et crée G2 =  (G1) avec   secret
(1) A → B: H (A crée H =  (G2) aléatoire)
(2) A ← B: i   {1,2}
(3) A → B:   tel que H =  (Gi)
    Si i=2:   :=
    Si i=1:   :=
(4) B vérifie H =  (Gi)
(5) Répéter n fois
```

Vérification zero-knowledge parfait : Les transcriptions du protocole sont indistinguables (distribution probabiliste) de celles produites par un simulateur.

Texte original

ZKIP : Isomorphisme de Graphes
Deux graphes $G_1 = (V_1, E_1)$ et $G_2 = (V_2, E_2)$ sont **isomorphes** s'il existe une permutation π t.q. $\{u, v\} \in E_1$ ssi $\{\pi(u), \pi(v)\} \in E_2$.
Exemple : $G_1 = (V, E_1)$ et $G_2 = (V, E_2)$ avec $V = \{1, 2, 3, 4\}$, $E_1 = \{12, 13, 23, 24\}$, et $E_2 = \{12, 13, 14, 34\}$ sont isomorphes avec la permutation $G_1 \to G_2 : \{4, 1, 3, 2\}$:



À partir d'un graphe G_1, on peut facilement (en temps polynomial) trouver une permutation π t.q. $G_2 = \pi(G_1)$.
Cependant, aucun algorithme polynomial n'est connu pour déterminer si deux graphes suffisamment grands (~1000 sommets) sont isomorphes (c.à.d. trouver la permutation π à partir des G_1 et G_2).
ZKIP sur la base de l'isomorphisme des graphes :
(Initialisation) A choisit un graphe G_1 suffisamment grand et invente une permutation π (le secret) lui permettant de calculer un deuxième graphe $G_2 = \pi(G_1)$. G_1 et G_2 sont rendus publiques.

`(1) A → B: H`

A choisit une permutation aléatoire ϕ telle que $H = \phi(G_2)$ et envoie H à B (le témoin)

`(2) A ← B: i`

B choisit un entier $i \in \{1, 2\}$ et l'envoie à A (le défi)

`(3) A → B:`

A calcule ψ telle que $H = \psi(G_i)$: - Si $i = 2$: $\psi := \phi$ - Si $i = 1$: $\psi := \phi \circ \pi$
(4) B contrôle si $H = \psi(G_i)$ et accepte l'étape comme juste.
(5) Répéter (1) à (4) un nombre de fois assez grand pour minimiser les risques de "deviner juste".
Vérification des propriétés :

- **Consistance** : Le protocole est accepté si A connaît le secret (i.e. la permutation π entre les deux graphes).
- **Significativité** : Si C essaye de se faire passer par A sans connaître π, il pourra fixer un j et fournir une permutation correcte $\psi(G_j)$ mais ne pourra pas trouver une permutation correcte pour les deux graphes. Il devra se contenter de deviner le défi fourni par B.
- **Zero-Knowledge** : A réussit à convaincre B du fait que les deux graphes sont isomorphes mais n'apprend rien sur π. B ne voit qu'un graphe aléatoire H isomorphe à G_1 et G_2 ainsi qu'une permutation entre H et G_1 ou entre H et G_2.
- **Zéro-Knowledge parfait** : Ceci équivaut à dire que B pourrait générer de telles informations tout seul (à l'aide d'un générateur aléatoire et des calculs polynomiaux). On peut prouver que les transcriptions fournies par le protocole ne peuvent se distinguer (d'un point de vue de distribution probabiliste) de celles produites par un simulateur (même en admettant que B "triche").

L'utilisation du paradigme de l'isomorphisme de graphes dans les protocoles d'authentification reste relativement marginale dû à des problèmes d'efficacité d'implantation.

Révision rapide

Problème : Trouver permutation entre 2 graphes isomorphes = difficile
Protocole : A crée H aléatoire, B demande permutation vers G1 ou G2, A répond
Perfect ZK : Transcriptions indistinguables d'un simulateur

ZKIP - Algorithme de Fiat-Shamir

Protocole pratique

Fiat-Shamir est un protocole ZKIP efficace et pratique basé sur le problème de la racine carrée modulo un composite.

Initialisation :

- Tierce de confiance T choisit $n = pq$ (garde p,q secrets)
- A choisit secret s avec $\gcd(s, n) = 1$
- A calcule $v = s^2 \bmod n$ et distribue v (clé publique certifiée)

Protocole :

```
(1) A → B: x = r² mod n
    (A choisit r aléatoire, témoin)

(2) A ← B: e  {0,1}
    (B envoie défi)

(3) A → B: y = r·s  mod n
    (A calcule réponse avec secret s)

B rejette si y = 0
B accepte si y²  x·v  (mod n)
```

Répétition : Plusieurs fois pour sécurité 2^{-nk}

Propriétés :

- **Significativité** : Un imposteur peut répondre à e=0 facilement, mais pour e=1 il devrait calculer $\sqrt{x}$ mod n (difficile par SQROOTP)
- **Zero-knowledge parfait** : Les paires (x,y) peuvent être simulées par B en choisissant y aléatoire et calculant $x = y^2$ ou y^2/v
- B ne peut se faire passer pour A car il ne peut prédire les défis

Texte original

ZKIP : Algorithme de Fiat-Shamir
But : Permettre à A de s'identifier en prouvant la connaissance d'un secret s (associé à A au moyen d'informations publiques authentiques) auprès de B sans lui révéler des informations sur s.
Il s'agit d'un protocole qui sert comme base à des implantations réelles et efficaces.
Algorithme :
(Initialisation) :

(a) Un tierce de confiance, T choisit et publie un n t.q. $n = pq$ et garde p et q secrets.

(b) A choisit un secret s avec $1 \leq s \leq n-1$ et $\gcd(s, n) = 1$, calcule $v = s^2$ mod n et distribue v comme clé publique certifiée par T.

```
(1) A → B: x = r² mod n
```

A choisit un r aléatoire et envoie un témoin r^2

```
(2) A ← B: e  {0,1}
```

B envoie son défi

```
(3) A → B: y = r·s  mod n
```

A calcule la réponse en utilisant le secret s.
B rejette la preuve si $y = 0$ (un imposteur pourrait fausser la preuve avec $r = 0$) et accepte la preuve si $y^2 \equiv x \cdot v^e \pmod{n}$.
Les étapes (1) à (3) sont répétées jusqu'à atteindre une marge de confiance suffisante.
Vérification des propriétés :

- **Consistance** : Si A connaît s, le protocole accepte la preuve d'identification.
- **Significativité** : Dans le cas simple, un imposteur pourrait seulement répondre à $e = 0$. Sinon, il pourrait choisir un r aléatoire et envoyer $x = r^2/v$ dans (1) et répondre au défi $e = 1$ avec une réponse correcte $y = r$. Dans le cas où $e = 0$, il devrait calculer la racine carrée de x mod n (n composite de factorisation inconnue) ce qui est difficile par SQROOTP. La réussite de la preuve nécessite, donc, la possession du secret.
- **Zéro Knowledge** : B ne peut obtenir aucune information sur s car lorsque $e = 1$, il est caché par un nombre aléatoire (blinding factor).
- **Zéro-Knowledge parfait** : Les paires (x,y) obtenues de A peuvent également être simulées par B en choisissant un y aléatoire et un $x = y^2$ ou $y^2/v \bmod n$. On peut prouver que ces paires ont une distribution probabiliste identique à celles fournies par A (qui les calcule différemment!).

À noter que, malgré cette dernière propriété, B est incapable de se faire passer par A auprès de B' car il ne peut pas prédire les valeurs des défis e.

Révision rapide

Secret : s tel que $v = s^2 \bmod n$ (clé publique)
Protocole : Témoin r^2, défi $e \in \{0, 1\}$, réponse $y = r \cdot s^e$
Vérification : $y^2 \equiv x \cdot v^e \pmod{n}$
Perfect ZK : Paires (x,y) simulables par B

ZKIP - Implantations Pratiques

Protocoles efficaces

Les implantations pratiques améliorent l'efficacité de Fiat-Shamir.

Feige-Fiat-Shamir (FSS) :

- Utilise des témoins et défis multiples (k valeurs) par itération
- Probabilité de tricher : 2^{-nk} pour n itérations
- Réduit le nombre d'échanges nécessaires

Guillou-Quisquater (GQ) :

- Basé sur Fiat-Shamir mais avec domaine de défis élargi
- Diminue la probabilité de deviner sans augmenter les échanges
- Meilleur compromis efficacité/sécurité

Schnorr :

- Basé sur la difficulté des logarithmes discrets (DLP)
- Domaine très grand de défis possibles
- **Identification en 3 échanges seulement**
- Sacrifie parfois la propriété perfect zero-knowledge pour l'efficacité

Avantages : Plus efficaces que RSA, implantables sur supports à capacité réduite (cartes à puces).

Texte original

ZKIP : Implantations Courantes
Feige-Fiat-Shamir (FSS) :

- Basé sur le protocole de Fiat-Shamir mais en utilisant des témoins et des défis multiples (des ensembles de k valeurs) à chaque itération; ce qui pour n itérations nous donne une probabilité de 2^{-nk} de deviner toutes les réponses.

Guillou-Quisquater (GQ) :

- Également basé sur Fiat-Shamir mais en augmentant le choix des défis ce qui diminue la probabilité de deviner sans augmenter le nombre d'instances transférées et d'étapes du protocole.

Schnorr :

- Basé sur la difficulté de calculer des logarithmes discrets (DLP)
- Il utilise également un domaine très grand de défis possibles ce qui lui permet de réaliser une identification en **3 échanges de messages seulement**.

Ces protocoles sont nettement plus efficaces que RSA et peuvent être implantés sur des supports à capacité de calcul réduite (smart cards).

Ils satisfont les propriétés de consistance, significativité mais la propriété zero-knowledge est parfois sacrifiée (comme dans le cas de Schnorr) pour augmenter l'efficacité.
Pour une description détaillée de ces protocoles se référer à [Men97] ou à [Sti95].

Révision rapide

FSS : Témoins/défis multiples → probabilité 2^{-nk}
GQ : Domaine de défis élargi → moins d'échanges
Schnorr : DLP + grands défis → **3 échanges seulement**
Tous : Plus efficaces que RSA, adaptés aux cartes à puces

ZKIP - Attaque Mafia et Remarques Finales

Vulnérabilités et contre-mesures

Même les protocoles ZKIP robustes peuvent être vulnérables à certaines attaques sophistiquées.

Attaque Mafia (1989, Adi Shamir) :

Scénario : C (attaquant) et D (complice) collaborent pour que D se fasse passer pour A auprès de B.

```
A   C: Instance ZKIP     D   B: Instance ZKIP
```

C relaie les messages de A vers D (complice), qui les utilise pour s'authentifier auprès de B. L'attaque est transparente pour A et B.

Contre-mesures :

- **Chambres blindées** (cage de Faraday) empêchant les communications radio
- **Synchronisation forte** pour éviter les échanges annexes
- **Distance bounding protocols** limitant le délai de réponse

Recommandations générales :

- Choisir une solution prouvée plutôt qu'inventer
- Vérifier que les objectifs sont atteints
- Analyser pratiquement (reflection attacks, redondance, etc.)
- Analyser formellement (logique BAN, model checking)

Texte original

ZKIP : Remarques Finales
Les ZKIP offrent un très bon niveau de sécurité cryptographique. Ils permettent de procéder à des identifications en minimisant les chances d'un imposteur hypothétique et, surtout, en protégeant les informations sécrètes des utilisateurs "honnêtes".
En 1989 (SECURICOM'89) Adi Shamir disait à propos des ZKIP : "I could go to a Mafia owned store a million successive times and they will still not be able to misrepresent themselves as me"…
Et pourtant : A participe à une ZKIP avec C Mafia; en même temps, D (complice de C) participe à une autre ZKIP où il prétend se faire passer par A auprès de B (un vérificateur "honnête").

```
(1) A → C: t1 (témoin que C fait suivre par liaison radio à D)
(1') D → B: t1
(2') D ← B: d1 (B envoie le défi à D; D le fait suivre à C...)
(2) A ← C: d1 (C reprend le défi dans son dialogue avec A)
(3) A → C: r1 (la réponse en utilisant son secret, que C envoie à D)
(3') D → B: r1 (B accepte r1 et ainsi de suite!)
```

Solutions :

- Procéder à des identifications dans des chambres blindées (cage de Faraday)…
- Utiliser des algorithmes de synchronisation forte pour éviter des échanges annexes.

Authentification : Récapitulation - Attaques et Protections

Attaque	Description	Protection
replay	rejouer une instance d'identification précédente	zero-knowledge, challenge and response, one-time password (attention aux pre-play !)
known/chosen-plaintext	obtenir des couples plaintext/ciphertext	zero-knowledge
chosen-ciphertext	faire décrypter (ou signer) à A des informations soigneusement choisies	zero-knowledge, ch. & resp. + témoin de connaissance + structure (redondance !)
reflection	répondre le même nombre qui a été reçu	inclure l'entité cible dans les messages, asymétrie dans les messages
interleaving	utiliser des messages appartenant à plusieurs instances de protocoles simultanées	inclure l'entité cible dans les messages, introduire un chaînage cryptographique entre les messages d'une même instance d'identification
collusion	connivence entre les intervenants	cage de Faraday, synchronisation forte

Révision rapide

Attaque Mafia : Relais des messages via complice → authentification frauduleuse transparente
Protections : Cage Faraday, synchronisation forte, distance bounding
Tableau attaques : replay, chosen-plaintext/ciphertext, reflection, interleaving, collusion

Récapitulation - Attaques et Protections

Attaque	Description	Protection
replay	rejouer une instance d'identification précédente	zero-knowledge, challenge and response, one-time password (attention aux pre-play !)
known/chosen-plaintext	obtenir des couples plaintext/ciphertext	zero-knowledge

Attaque	Description	Protection
chosen-ciphertext	faire décrypter (ou signer) à A des informations soigneusement choisies	zero-knowledge, ch. & resp. + témoin de connaissance + structure (redondance !)
reflection	répondre le même nombre qui a été reçu	inclure l'entité cible dans les messages, asymétrie dans les messages
interleaving	utiliser des messages appartenant à plusieurs instances de protocoles simultanées	inclure l'entité cible dans les messages, introduire un chaînage cryptographique entre les messages d'une même instance d'identification
collusion	connivence entre les intervenants	cage de Faraday, synchronisation forte