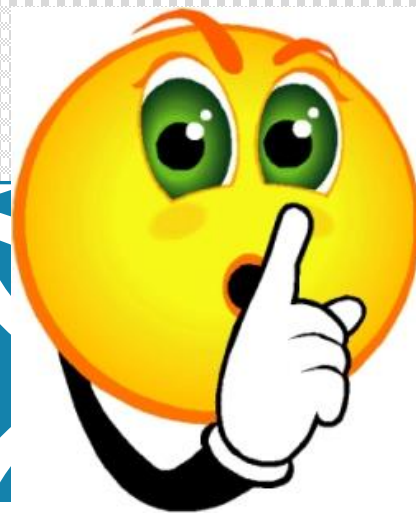


La sécurité prouvable



L'ÉCHANGE DE CLÉS

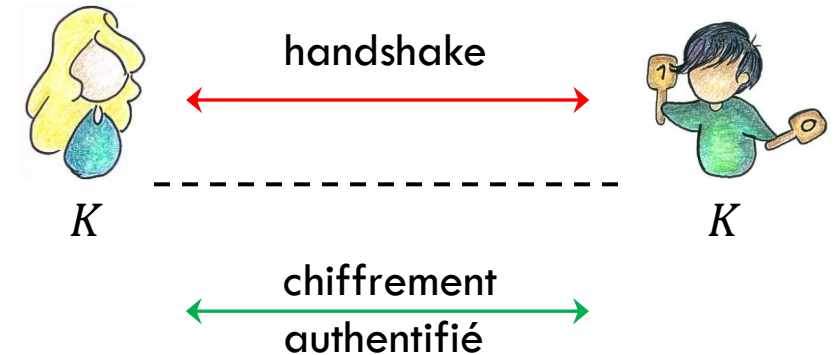
Cristina Onete  
cristina.onete@gmail.com

# MAC, AUTHENTIFICATION, ...

- Primitives/protocoles basés sur des clés symétriques
- Initialisation de chaque session :
  - ❖ **On suppose** que chaque paire de parties possède une clé secrète partagée
  - ❖ Au début du jeu de sécurité en MAC et authentification, une clé est choisie **aléatoirement** de l'espace  $K$
- D'où vient une telle clé ?
  - ❖ Les deux parties peuvent l'échanger hors ligne
  - ❖ En utilisant un protocole d'échange de clé

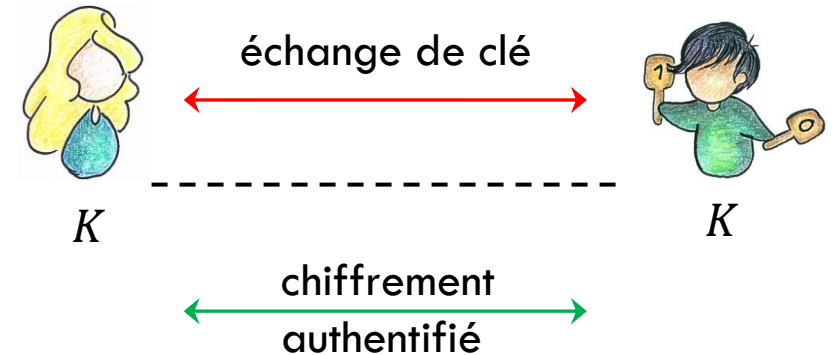
# L'ÉCHANGE DE CLÉS

- Typiquement un protocole à deux parties
- Deux parts :
  - ❖ **La poignée de main** : (handshake) les deux parties échangent des messages sur un canal non-sécurisé et finissent par calculer des clés
  - ❖ **Le chiffrement authentifié** : en utilisant les clés calculées, les deux parties échangent des messages sur un canal sécurisé
- Le chiffrement authentifié :
  - ❖ Aujourd'hui : typiquement AEAD
  - ❖ Également possible (moins sécurisé) : combinaison MAC et chiffrement



# SESSIONS, CLÉS, CLIENT-SERVEUR

- La communication sécurisée a lieu dans des **sessions**
  - ❖ Les sessions sont généralement de courte durée
  - ❖ Si besoin d'une durée plus longue, on doit pouvoir mettre à jour les clés
- Chaque session a ses clés :
  - ❖ Les clés de 2 sessions différentes doivent être indépendantes et aléatoires
  - ❖ Les clés de session peuvent évoluer pendant la session
- Souvent en mode "client-serveur", mais aussi entre deux "peers"
  - ❖ mode client-serveur hérité de l'environnement réseau



# POIGNÉE DE MAIN : ÉLÉMENTS TYPIQUES

➤ Fraîcheur : des éléments aléatoires (nonce = number once)

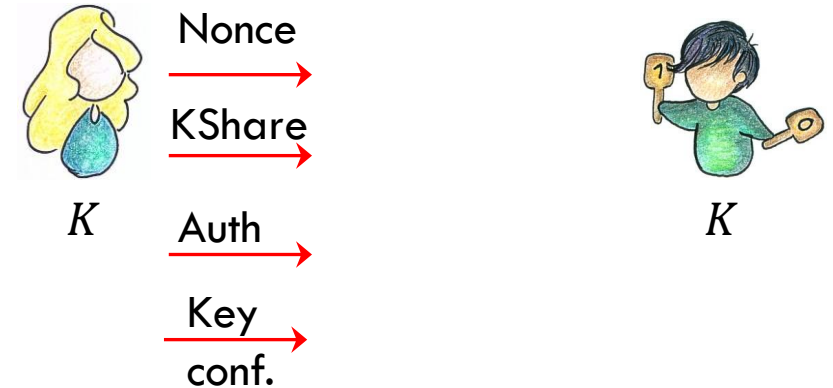
❖ Se prémunir contre des attaques par réjeu

➤ Contributions échange de clés : key shares

❖ Pour permettre le calcul d'une nouvelle clé de session

➤ Authentification

➤ Confirmation de clés : établir qu'on a calculé les mêmes clés



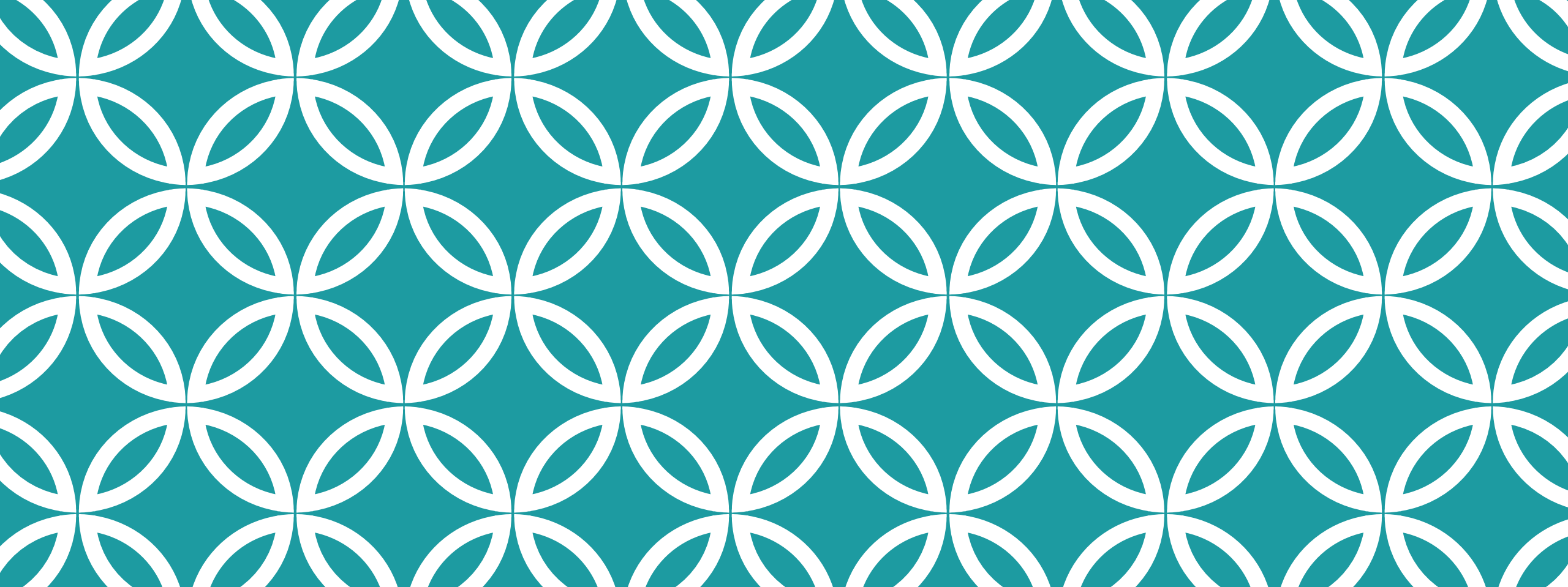
**En fonction de la présence et de l'ordre de ces éléments, une sécurité différente**

# CLÉ PUBLIQUE, CLÉ SECRETE...

- L'échange de clé existe à clé publique et à clé secrète
- A clé publique :
  - ❖ Échange de clé Diffie Hellman, Mécanisme d'encapsulation de clé (KEM), etc.
  - ❖ On transforme une ou deux clés publiques dans des clés de session secrètes
- A clé secrète :
  - On transforme des clés secrètes long-terme dans des clés court-terme

# L'ÉCHANGE DE CLÉ DANS LA VRAIE VIE

- Navigation Internet (https) : protocole TLS
  - ❖ Le même protocole est utilisé dans l'échange de mails (smtps) et d'autres applications (comme VoIP)
- Accès à distance à une machine : protocole ssh
- Réseaux de voitures connectées :
  - ❖ Deux voitures peuvent communiquer de façon sécurisée (pour indiquer des embouteillages, accidents, etc.)
- Réseaux de capteurs :
  - ❖ Souvent non-interactif, permet aux capteurs d'envoyer des messages à un terminal centralisé
- Réseaux mobiles : protocole AKA
  - ❖ Permet à un utilisateur de recevoir le signal mobile (pour appeler, envoyer des SMS, utiliser l'Internet...)
- Etc. ...



# L'AUTHENTIFICATION DANS L'ÉCHANGE DE CLÉS





# AUTHENTIFICATION ET ÉCHANGE DE CLÉS

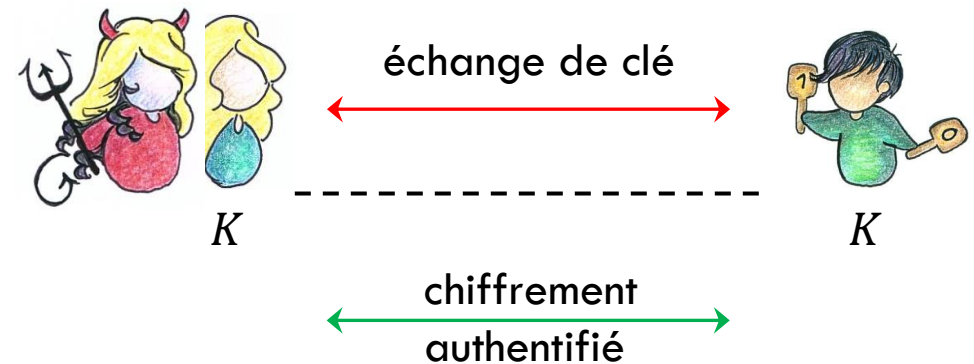
- Les poignées de main ont lieu sur un canal non-sécurisé :
  - ❖ Attaques actives => besoin **d'authentification**
  - ❖ Sinon, un attaquant pourrait se faire passer par les deux parties et trouver leurs clés
  
- Le type d'authentification influence la sécurité du protocole :
  - ❖ Authentification **unilatérale vs mutuelle**
  - ❖ Authentification **implicite vs. explicite**

# L'AUTHENTIFICATION UNILATÉRALE

- Deux participants, seulement un d'eux s'authentifie

## Quelles conséquences pour la sécurité de la clé ?

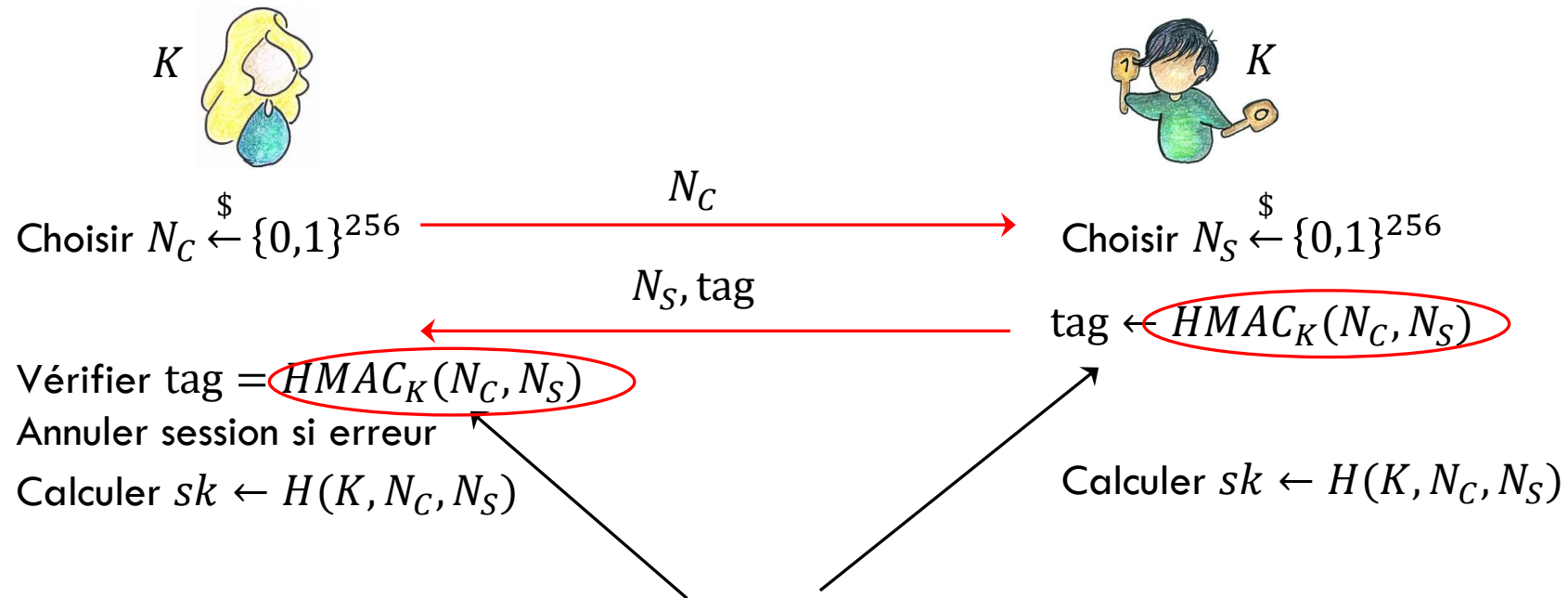
- La clé sera connue :
  - ❖ Soit par deux entités honnêtes (session client-serveur)
  - ❖ Soit par l'attaquant et l'entité honnête authentifiée



- Qui s'authentifie ?

- ❖ Dans le cas de TLS c'est le serveur, car la navigation Internet est ouverte à tout le monde
- ❖ Parfois le manque d'authentification permet dans un premier temps d'anonymiser l'échange de clés
- ❖ Ensuite, on peut avoir une authentification mutuelle dans le canal sécurisé, ainsi garantissant plus de privacy

# EXEMPLE DE PROTOCOLE

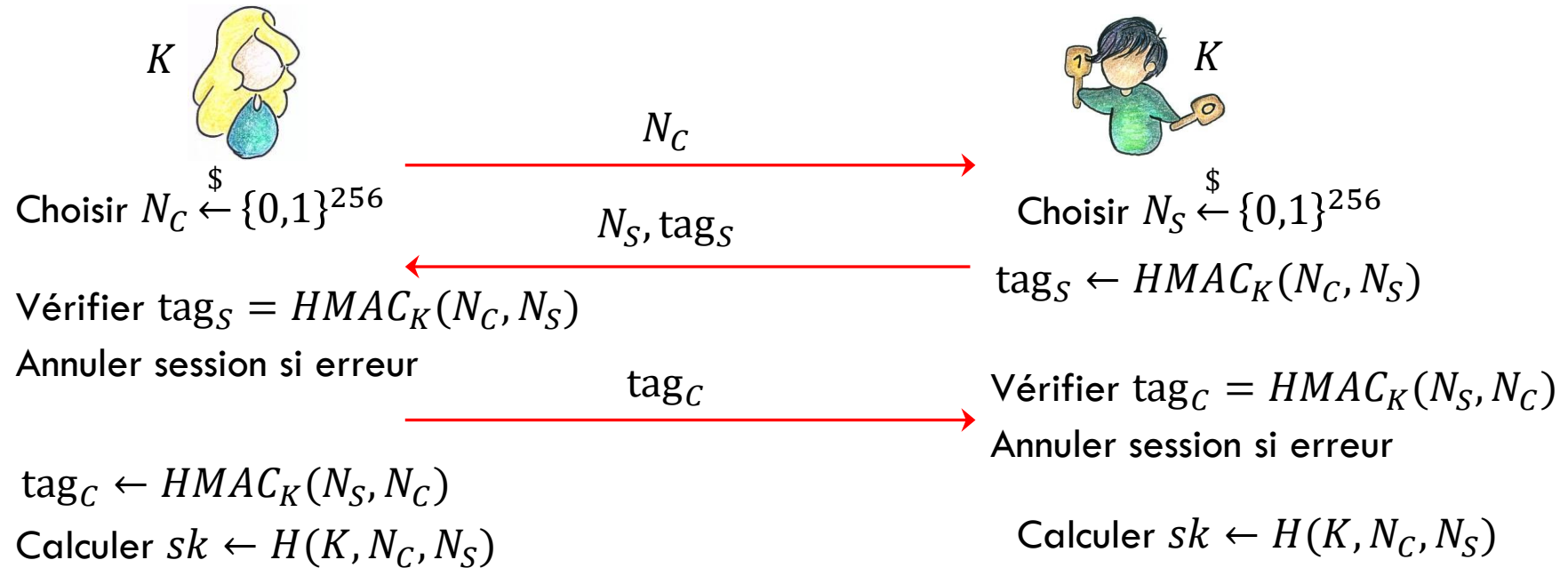


- Authentification du serveur par HMAC, avant le calcul de la clé
- Clé calculée par une fonction de hachage (qui devra se comporter comme un oracle aléatoire)

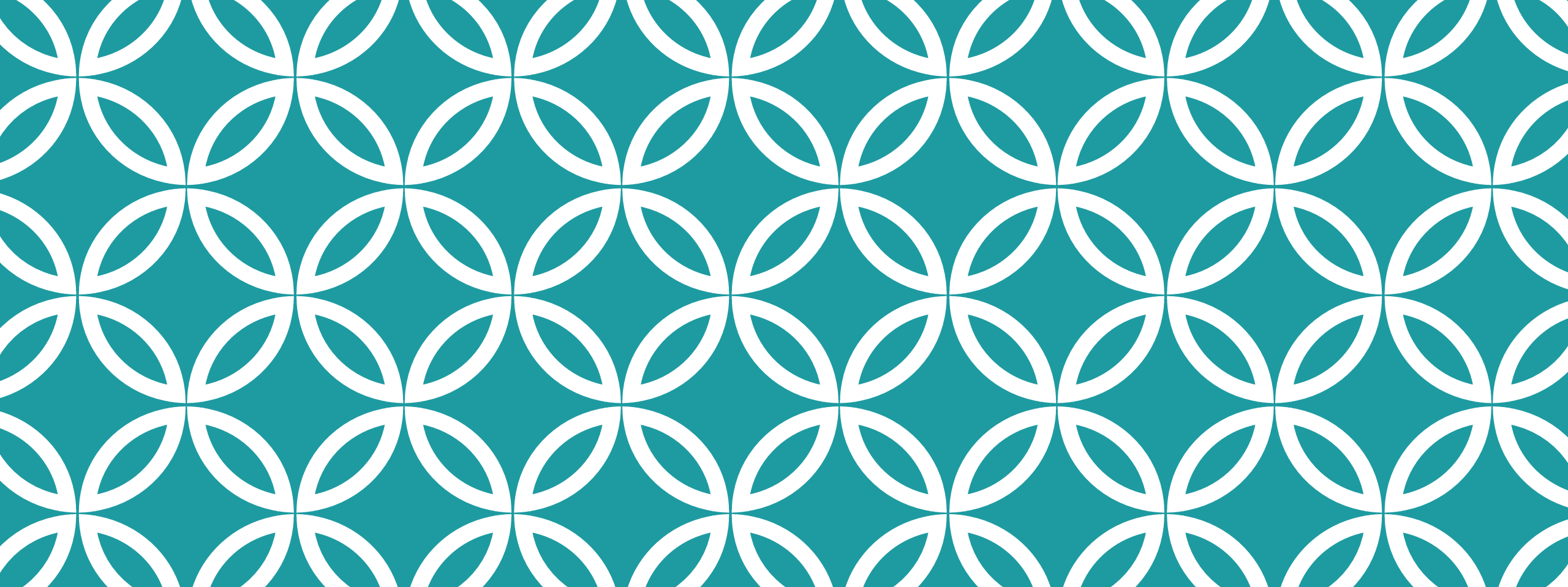
# L'AUTHENTIFICATION MUTUELLE

- Les deux entités s'authentifient réciproquement
- La clé sera connue par :
  - ❖ Les deux entités aux bouts du canal sécurisé
- Qui s'authentifie en premier ?
  - ❖ Le premier qui authentifie l'autre peut s'ouvrir à des attaques par deni de service
  - ❖ Mais il faut aussi prendre en compte la potentielle fuite d'informations sensibles lorsqu'on s'authentifie
  - ❖ Souvent par exemple dans les transactions bancaires, c'est le serveurs qui doit s'authentifier en premier

# EXEMPLE DE PROTOCOLE



➤ Le serveur s'authentifie en premier



# LA SÉCURITÉ DES HANDSHAKES (AKE)



# INTUITION DE LA SÉCURITÉ

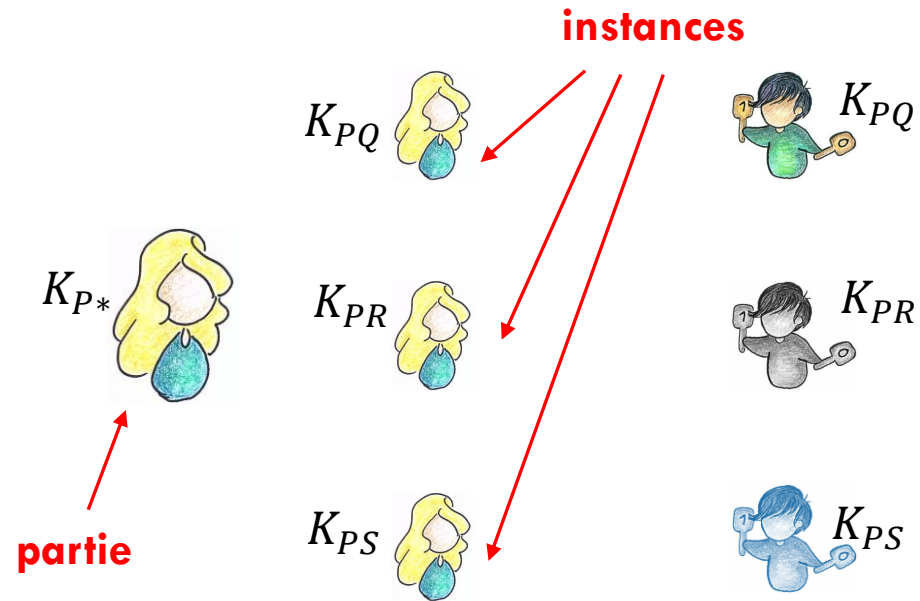
- Nous aurons un environnement avec beaucoup de parties
  - ❖ Chaque partie exécute un nombre de sessions avec des partenaires différents
  - ❖ Si une partie "accepte" une session, alors l'instance calcule une ou des clés de session
  
- Nous allons cibler les clés calculées par une instance d'une partie honnête, qui a accepté
  - ❖ ... dont le partenaire est honnête aussi (**Pourquoi ?**)
  - ❖ Session **correcte** : la clé est partagée par une instance partenaire
  - ❖ Cette clé doit être **indistinguable d'une clé aléatoirement choisie** de l'espace de toute clé possible
  - ❖ Elle doit être **indépendante** de la clé de toute autre session
  - ❖ De plus, il faut garantir la **sécurité de l'authentification**

# ENVIRONNEMENT ET PARTIES

- La sécurité AKE nécessite une modélisation de parties, instances et sessions
- Environnement d'exécution du protocole :
  - ❖ Ensemble de parties  $P$  – soit des clients, soit des serveurs
  - ❖ On suppose des clés  $K_{PQ} \stackrel{\$}{\leftarrow} \text{KEYS}$  aléatoires mais cohérentes pour chaque paire  $P \in \text{CLIENTS}, Q \in \text{SERVERS}$
  - ❖ Chaque instance de  $P$  instanciée avec  $K_{P*}$  ou  $K_{*Q}$  (en fonction du rôle de P)
- Il faut encore modéliser :
  - ❖ La notion de **partnering** entre deux instances
  - ❖ La notion **d'acceptance** d'une session par une instance d'une partie
  - ❖ Les **capacités** des attaquants : parties malveillantes, participer aux sessions, apprendre des clés...



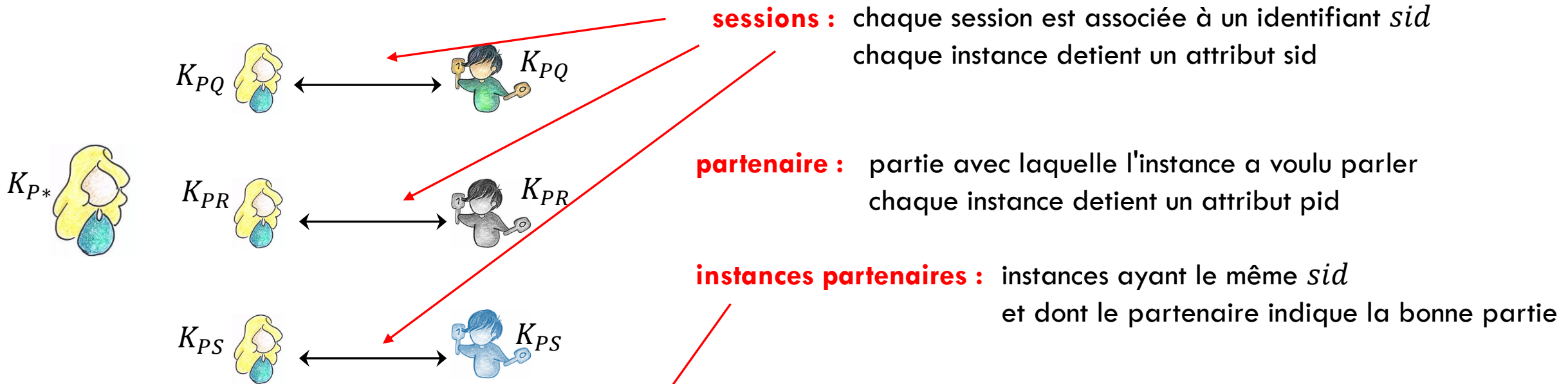
# ENVIRONNEMENT ET PARTIES



Il faut encore modéliser :

- ❖ La notion de partnering entre deux instances
- ❖ La notion d'acceptance d'une session par une instance d'une partie
- ❖ Les capacités des attaquants : parties malveillantes, participer aux sessions, apprendre des clés...

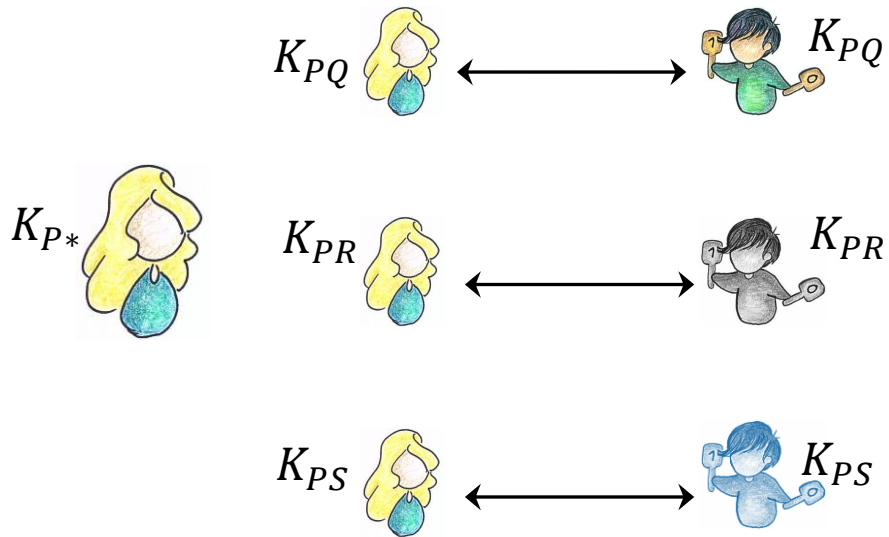
# ENVIRONNEMENT ET PARTIES



Il faut encore modéliser :

- ❖ La notion de partnering entre deux instances
- ❖ La notion d'acceptance d'une session par une instance d'une partie
- ❖ Les capacités des attaquants : parties malveillantes, participer aux sessions, apprendre des clés...

# ENVIRONNEMENT ET PARTIES

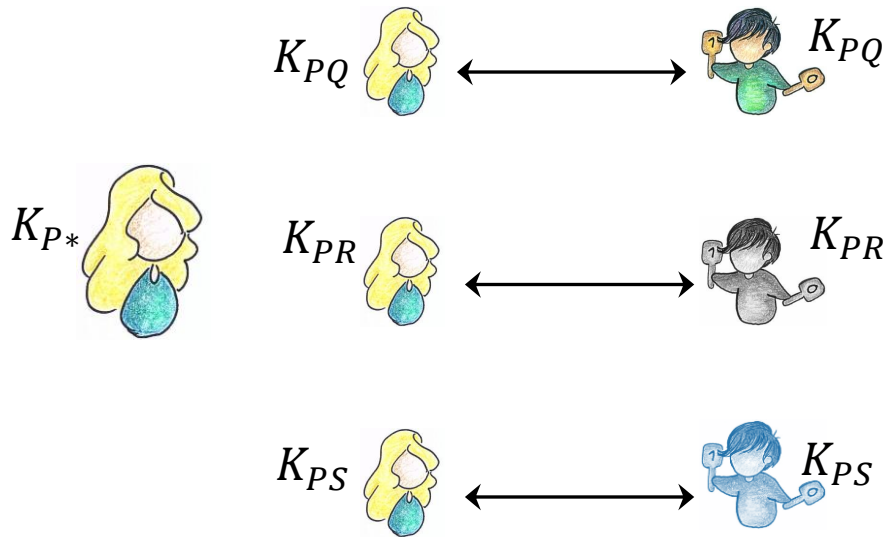


**acceptance :** une instance "accepte" une session si elle s'exécute normalement  
en pratique : toute vérification marche, authentification réussie, pas de message incorrect...  
chaque instance a un attribut  $\alpha \in \{0,1\}$  : reject/accept

Il faut encore modéliser :

- ❖ La notion de partnering entre deux instances
- ❖ La notion d'acceptance d'une session par une instance d'une partie
- ❖ Les capacités des attaquants : parties malveillantes, participer aux sessions, apprendre des clés...

# ENVIRONNEMENT ET PARTIES



Il faut encore modéliser :

- ❖ La notion de partnering entre deux instances
- ❖ La notion d'acceptance d'une session par une instance d'une partie
- ❖ Les capacités des attaquants : parties malveillantes, participer aux sessions, apprendre des clés...

L'attaquant peut :

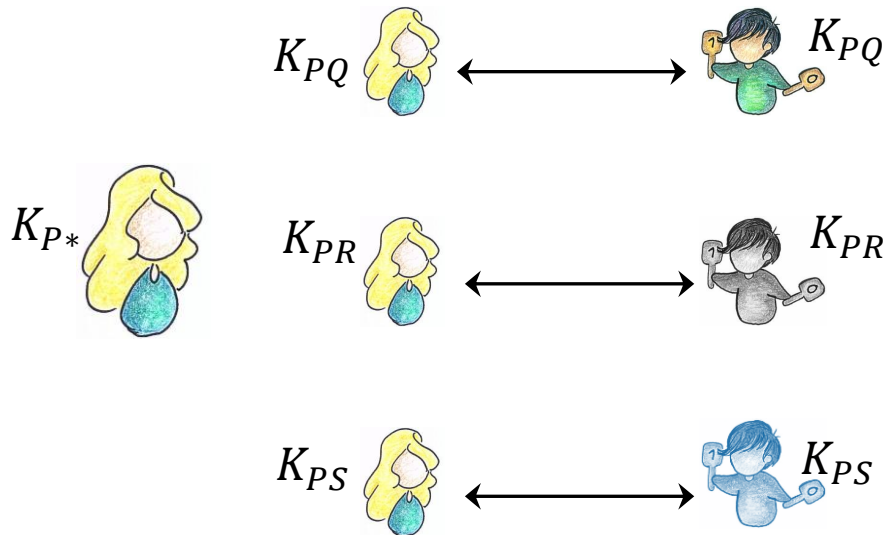
**créer des sessions :**  $\circ$ NewSession(P, Q)  
crée une instance  $\Pi_P^i$

**envoyer des messages :**  $\circ$ Send( $\Pi_P^i, m$ )  
envoie  $m$  à  $\Pi_P^i$ , qui retourne  $m'$   
message special "Start" pour démarrer la session

**corrompre des parties :**  $\circ$ Corrupt(P)  
retourne toutes les clés  $K_{P^*}$  ou  $K_{*P}$

**apprendre des clés de session :**  $\circ$ Reveal( $\Pi_P^i$ )  
retourne la clé  $\Pi_P^i.sk$  de  $\Pi_P^i$

# ENVIRONNEMENT ET PARTIES



## Résumé -- attributs de l'instance $\Pi_P^i$ :

- sid :**  $\Pi_P^i.sid$ , identifiant de la session  
collection de valeurs spécifiques à la session
- pid :**  $\Pi_P^i.pid$ , identifiant du partenaire  
cet identifiant se confirme si le partenaire s'authentifie
- bit d'acceptance :**  $\Pi_P^i.\alpha$ , avec  $\alpha \in \{0,1\}$ , mis à 1 si  $\Pi_P^i$  accepte
- bit de révélation :**  $\Pi_P^i.\rho \in \{0,1\}$ , mis à 1 si requête  $\text{oReveal}(\Pi_P^i)$
- bit de corruption :**  $\Pi_P^i.\gamma$ , avec  $\gamma \in \{0,1\}$ , activé si requête  $\text{oCorrupt}(P)$   
attribut uniforme sur tout  $P$ :  $\Pi_P^i.\gamma = \Pi_P^j.\gamma \quad \forall i, j$
- clé de long-terme :**  $\Pi_P^i.K \in \{K_{PQ}, K_{QP}\}$   
la clé utilisée pour cette session
- clé de session :**  $\Pi_P^i.sk$ , initialisée à  $\perp$

# UN PROTOCOLE CORRECT

➤ **Instances partenaires** : si  $\Pi_P^i$  et  $\Pi_Q^j$  deux instances de  $P$  et  $Q$  dans l'environnement d'exécution

On dit  $\Pi_P^i$  et  $\Pi_Q^j$  "matchent" ssi. :

$P \in \text{CLIENTS}$  et  $Q \in \text{SERVERS}$  (ou inversement)

$\Pi_P^i.\text{pid} = Q$  et  $\Pi_Q^j.\text{pid} = P$

$\Pi_P^i.\text{sid} = \Pi_Q^j.\text{sid}$

➤ **Protocole correct** : dans l'absence d'un attaquant si  $\Pi_P^i$  et  $\Pi_Q^j$  "matchent" et si  $\Pi_P^i.\alpha = \Pi_Q^j.\alpha = 1$  alors :

$\Pi_P^i.sk = \Pi_Q^j.sk$

# SÉCURITÉ AKE

$\Pi_P^i$  de  $P$  t.q.  $\Pi_P^i.\gamma = 0$

$\Pi_P^i.\alpha = 1$

➤ Cibler les clés calculées par **une instance d'une partie honnête, qui a accepté**

❖ ... dont le partenaire **est honnête aussi**

$\Pi_Q^i$  de  $Q$  t.q.  $\Pi_Q^i.\gamma = 0$

L'adversaire aura soit une vraie clé soit une clé aléatoire et devra les distinguer

❖ Cette clé doit être **indistinguable d'une clé aléatoirement choisie** de l'espace de toute clé possible

L'adversaire pourra faire  $\circ$ Reveal sur toute instance  $\Pi_P^i$  ou  $\Pi_*^*$  qui match

❖ Elle doit être **indépendante** de la clé de toute autre session

❖ De plus, il faut garantir la **sécurité de l'authentification**

On ne peut pas impersonifier aucune partie qui s'authentifie, sauf par corruption

# SÉCURITÉ DE L'AUTHENTIFICATION

- Échange de clés avec authentification mutuelle :
  - ❖ L'adversaire aura accès à `oNewSession`, `oSend`, `oReveal`, `oCorrupt`
  - ❖ Il gagnera s'il existe une instance  $\Pi_P^i$  tel que  $\Pi_P^i.\alpha = 1$  et  $\Pi_P^i.\gamma = 0$  et tel que :
    - ❖ Si  $\Pi_P^i.pid = Q$  alors  $\Pi_Q^*.\gamma = 0$  et il n'existe aucune instance  $\Pi_Q^j$  qui match  $\Pi_P^i$
- Modification pour l'authentification unilatérale :
  - ❖ S'il s'agit de l'authentification du serveur auprès du client, alors  $P \in \text{CLIENTS}$  et  $Q \in \text{SERVERS}$
  - ❖ Si c'est l'inverse, alors  $P \in \text{SERVERS}$  et  $Q \in \text{CLIENTS}$



# SÉCURITÉ DE L'ÉCHANGE DE CLÉS

## ➤ Nouvel attribut, nouvel oracle :

- ❖ Un bit d'indistinguabilité par instance  $\Pi_p^i. b \in \{0,1\}$
- ❖ Oracle de test :  $\text{oTest}_b(\Pi_p^i)$  : si  $\Pi_p^i.sk \neq \perp$ , alors si  $b = 1$ , l'oracle retourne  $\Pi_p^i.sk$ ; si  $b = 0$ , retourner  $sk$  aléatoire

## ➤ Sécurité pour une authentification mutuelle :

- ❖ L'adversaire aura accès à  $\text{oNewSession}$ ,  $\text{oSend}$ ,  $\text{oReveal}$ ,  $\text{oCorrupt}$  et une seule fois  $\text{oTest}$
- ❖ Finalement, l'adversaire retourne une instance valide  $\Pi_p^i$  et un bit  $d$

❖ Il gagnera si  $d = \Pi_p^i.b$  et : A peut distinguer la vraie clé d'une clé aléatoire

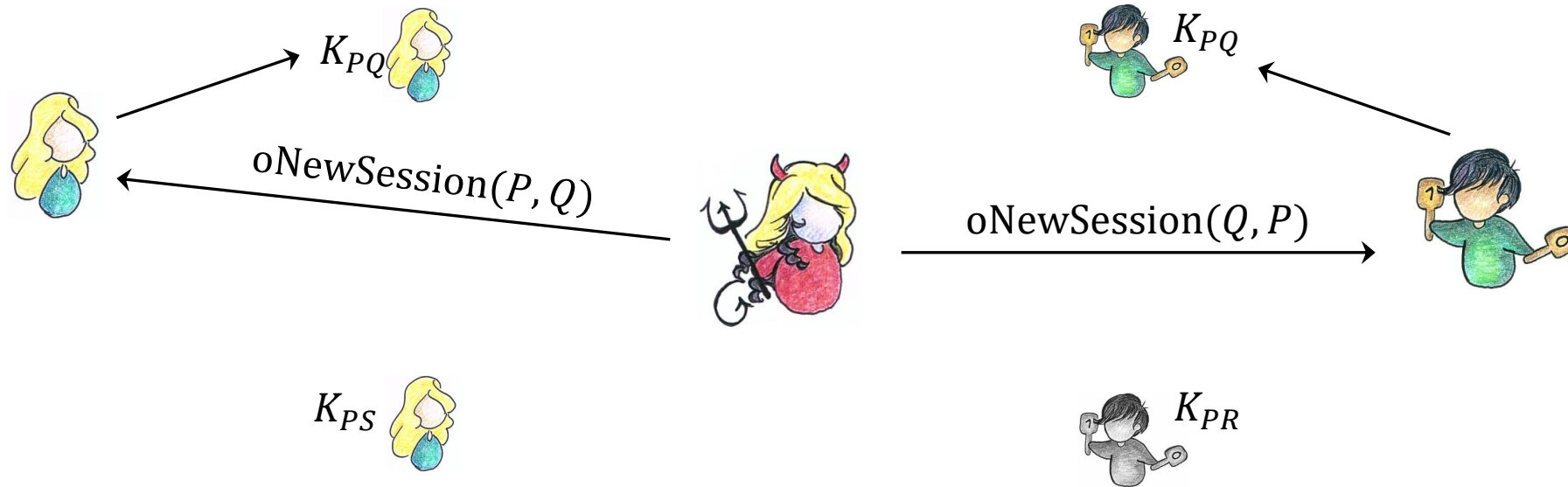
❖  $\Pi_p^i.\alpha = 1$  L'instance accepte

❖  $\Pi_p^i.\gamma = 0$  et si  $\Pi_p^i.pid = Q$  alors  $\Pi_Q^*. \gamma = 0$  Ni la partie ciblée ni son partenaire sont corrompus

❖  $\Pi_p^i.\rho = 0$  et pour toute  $\Pi_Q^j$  qui match  $\Pi_p^i$  on a  $\Pi_Q^j.\rho = 0$

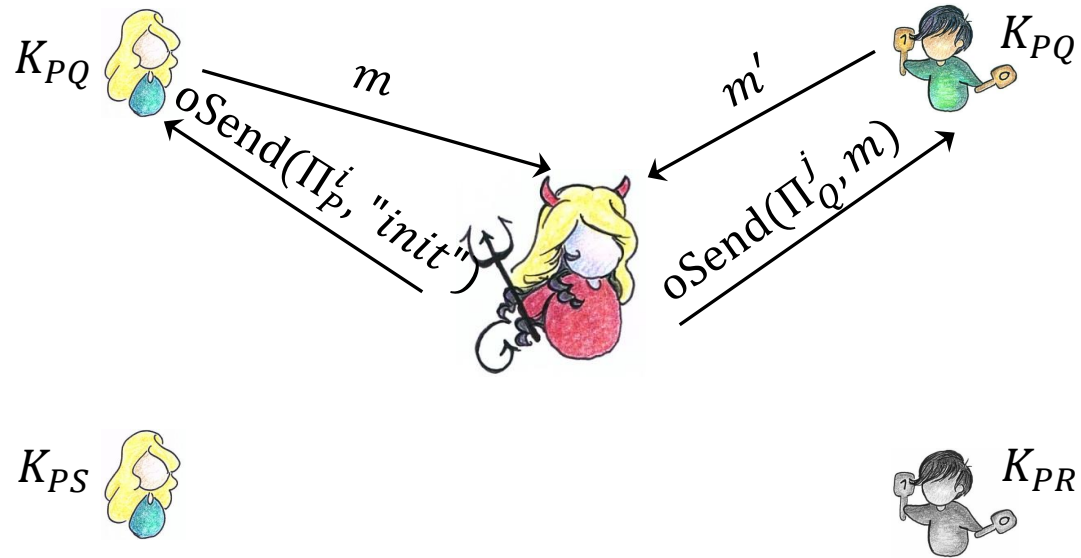
La clé de session n'est révélée ni d'un bout ni de l'autre

# INTUITION SÉC. AUTHENTIFICATION MUTUELLE



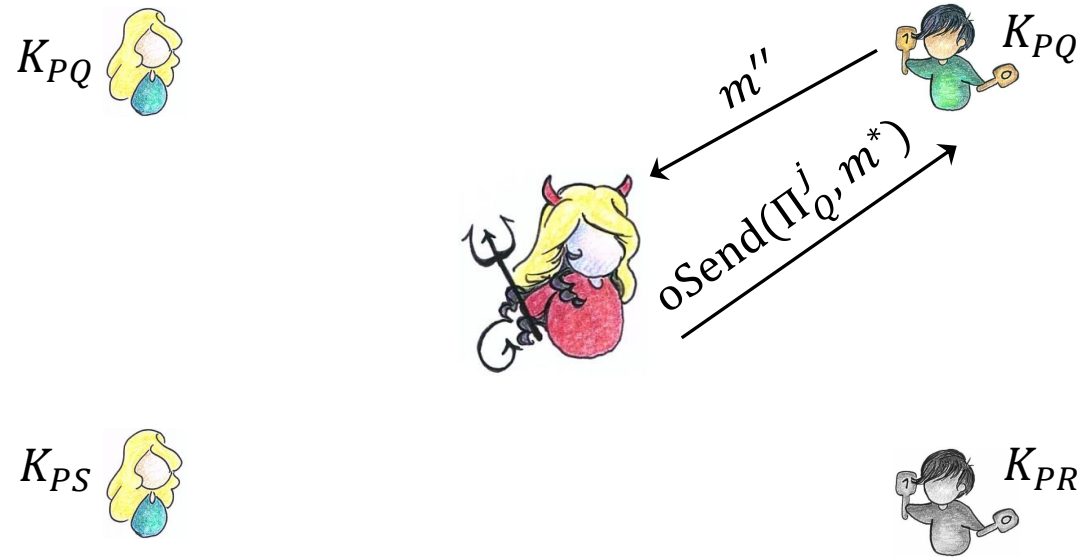
➤ L'attaquant peut créer des instances

# INTUITION SÉC. AUTHENTIFICATION MUTUELLE



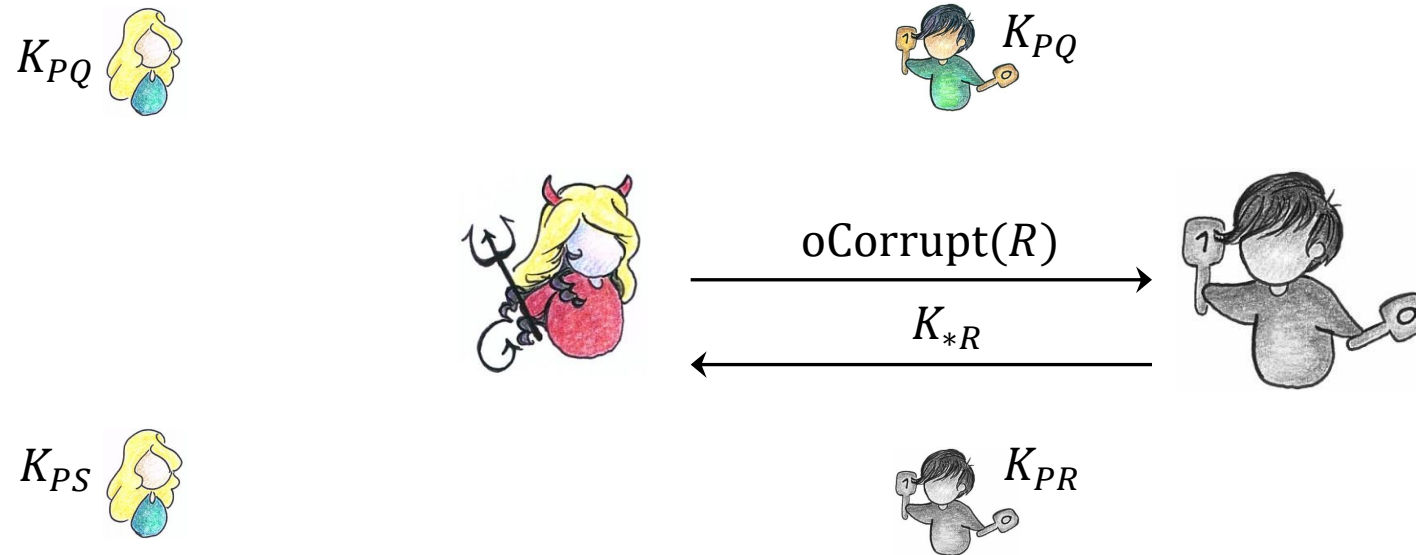
- L'attaquant peut observer un échange honnête entre deux parties honnêtes

# INTUITION SÉC. AUTHENTIFICATION MUTUELLE



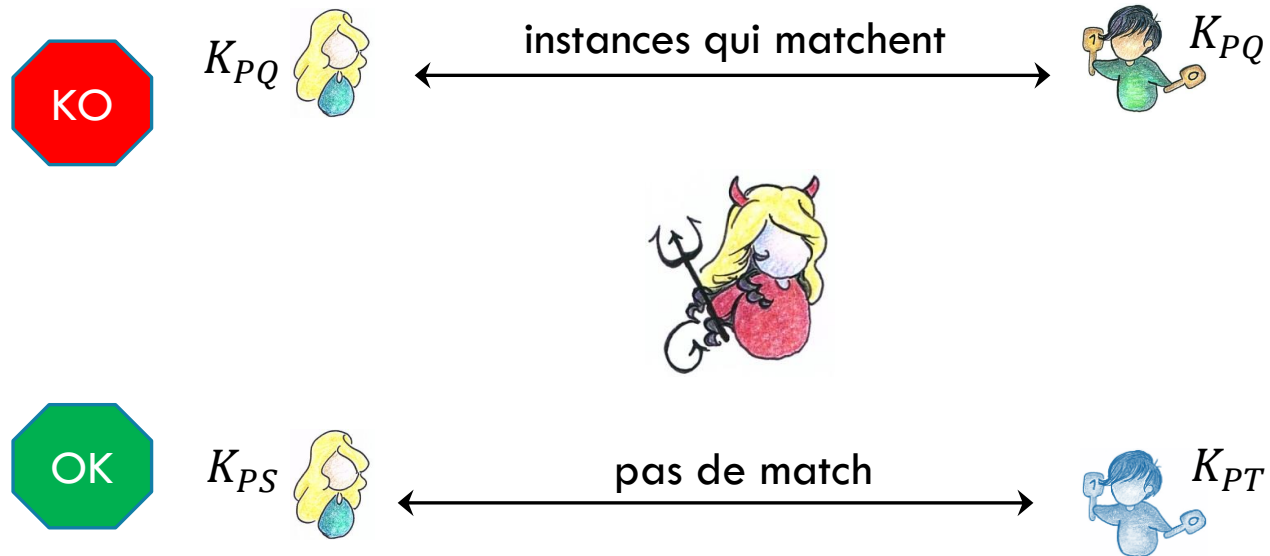
➤ L'attaquant peut insérer des messages arbitraires

# INTUITION SÉC. AUTHENTIFICATION MUTUELLE



- L'attaquant peut corrompre des parties honnêtes...
- ❖ ... mais pas la partie ciblée, ni son partenaire

# INTUITION SÉC. AUTHENTIFICATION MUTUELLE



- L'attaquant gagne si une instance accepte sans partenaire

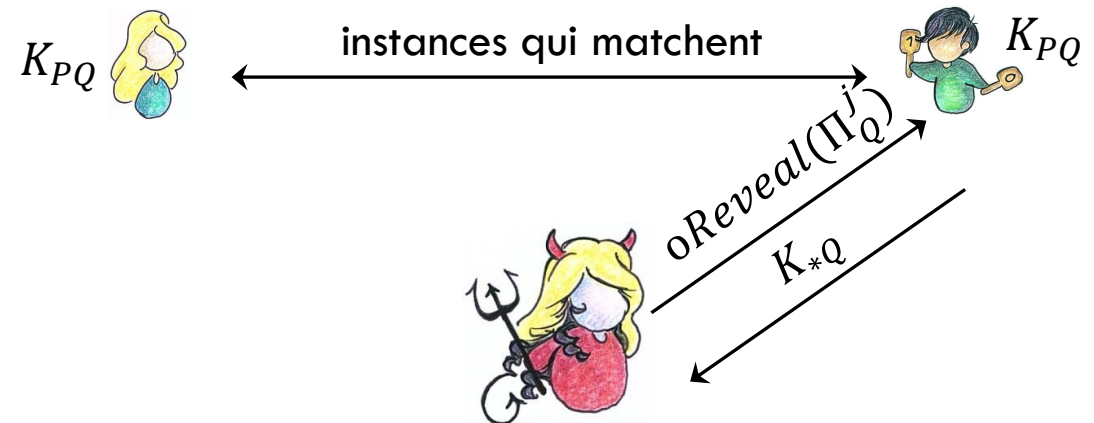
# INTUITION SÉCURITÉ AKE

➤ De même que pour l'authentification, l'adversaire peut :

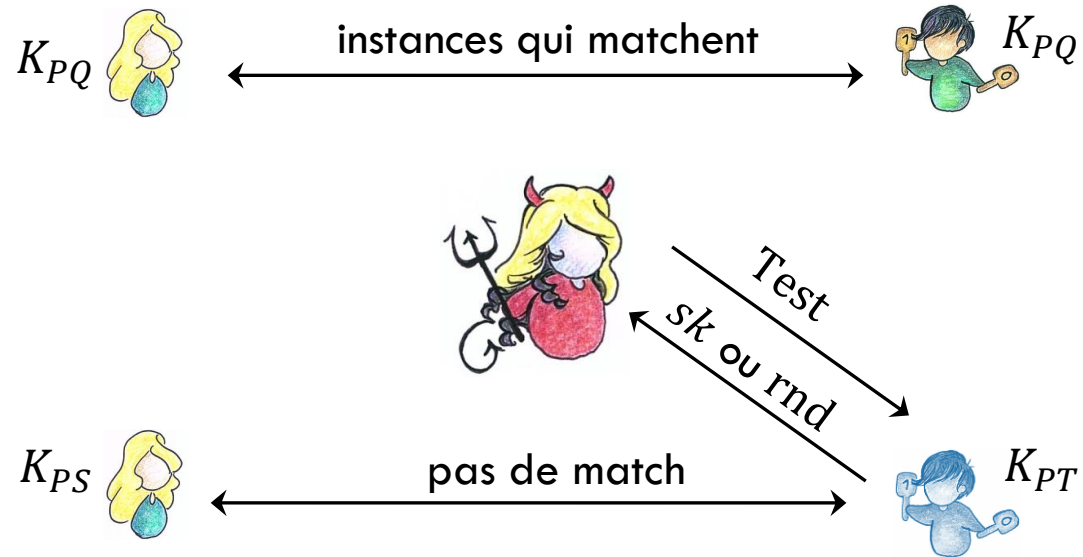
- ❖ créer de nouvelles instances
- ❖ observer des exécutions honnêtes
- ❖ envoyer des messages de son choix
- ❖ corrompre toute partie sauf l'instance ciblée et son partenaire

➤ ... et de plus :

- ❖ Révélation de clé de session ...
- ❖ ... seulement pour des instances non-ciblées
- ❖ ... et des instances sans match pour l'instance ciblée



# INTUITION SÉCURITÉ AKE -- ORACLE TEST



- L'attaquant peut faire une seule requête à une instance "fraîche"



# ET SI L'AUTHENTIFICATION EST UNILATÉRALE ?

- Le jeu AKE change pour une authentification unilatérale

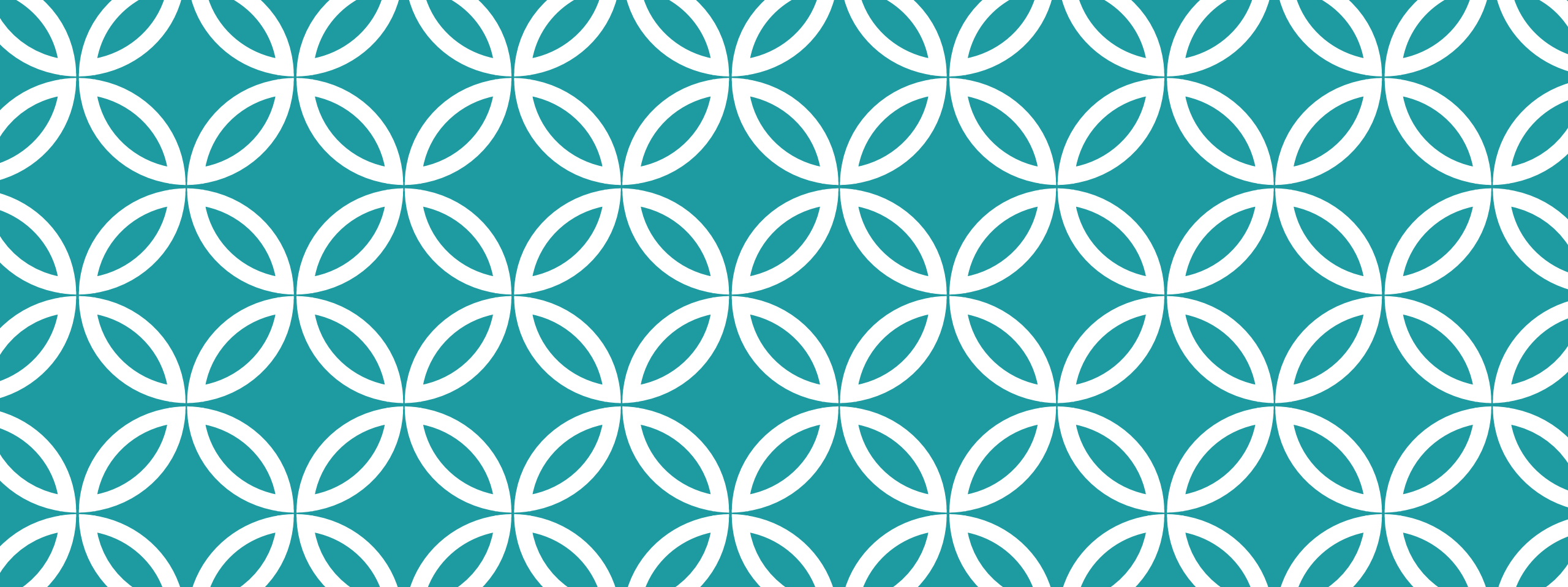
L'authentification du serveur assure qu'il y aura deux bouts honnêtes

- Supposons une authentification unilatérale du côté du serveur

- L'attaquant devra cibler une instance client acceptante, avec un partenaire non-corrompu

- ❖ ... ou une instance de serveur avec une instance client honnête qui match

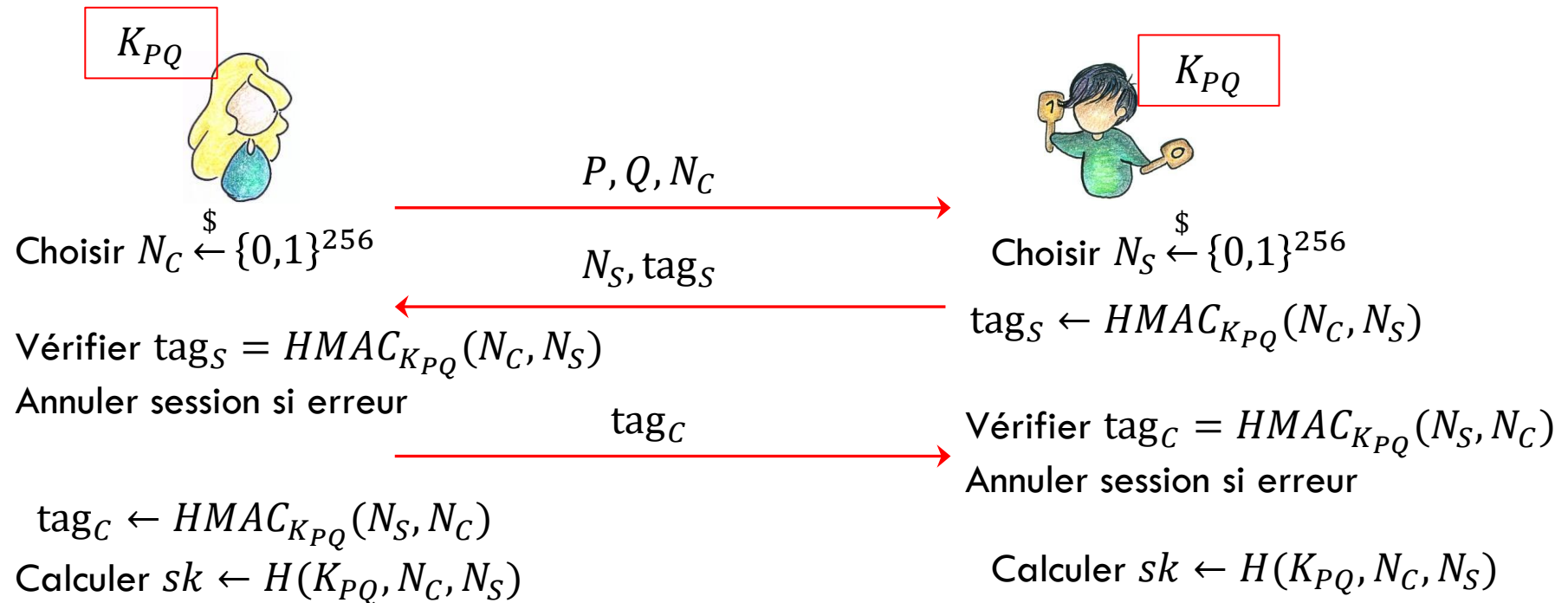
On s'assure qu'il y a deux bouts honnêtes



# LA SÉCURITÉ D'UN PROTOCOLE

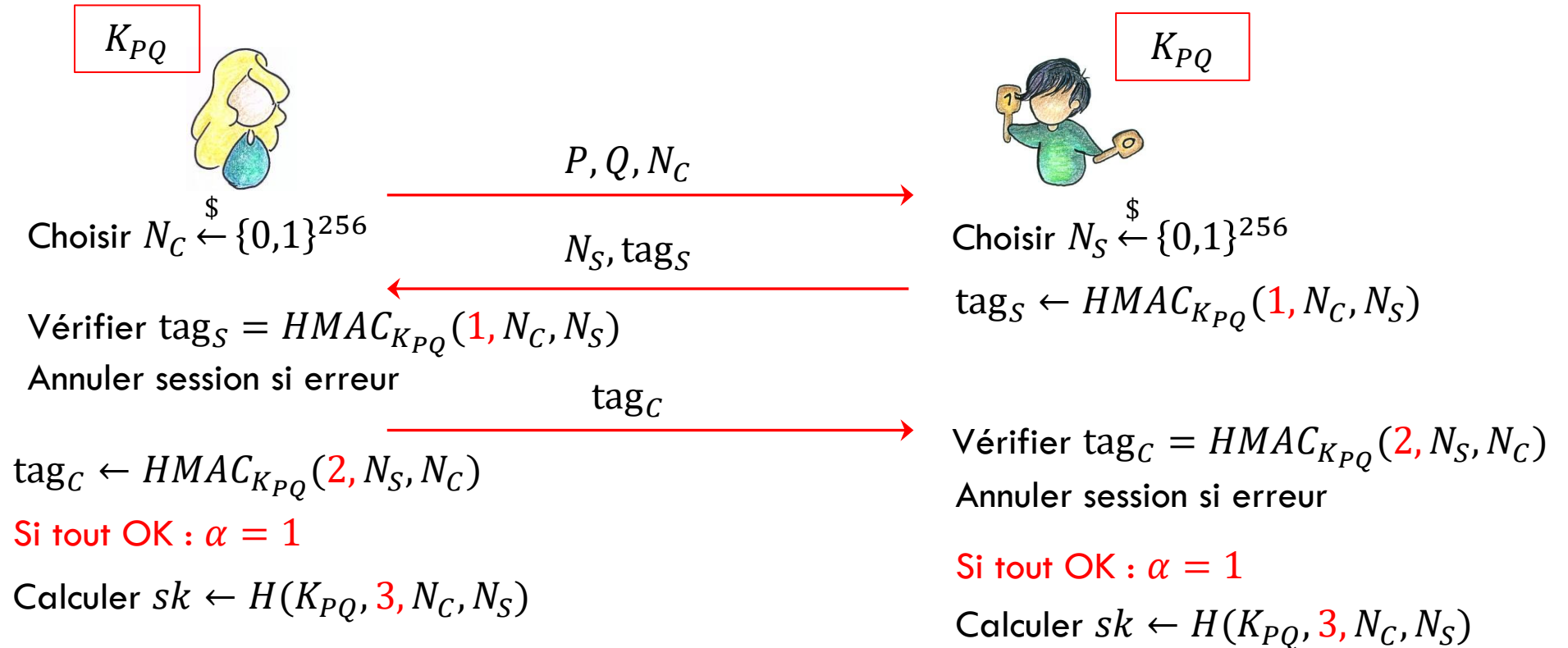


# EXEMPLE DE PROTOCOLE



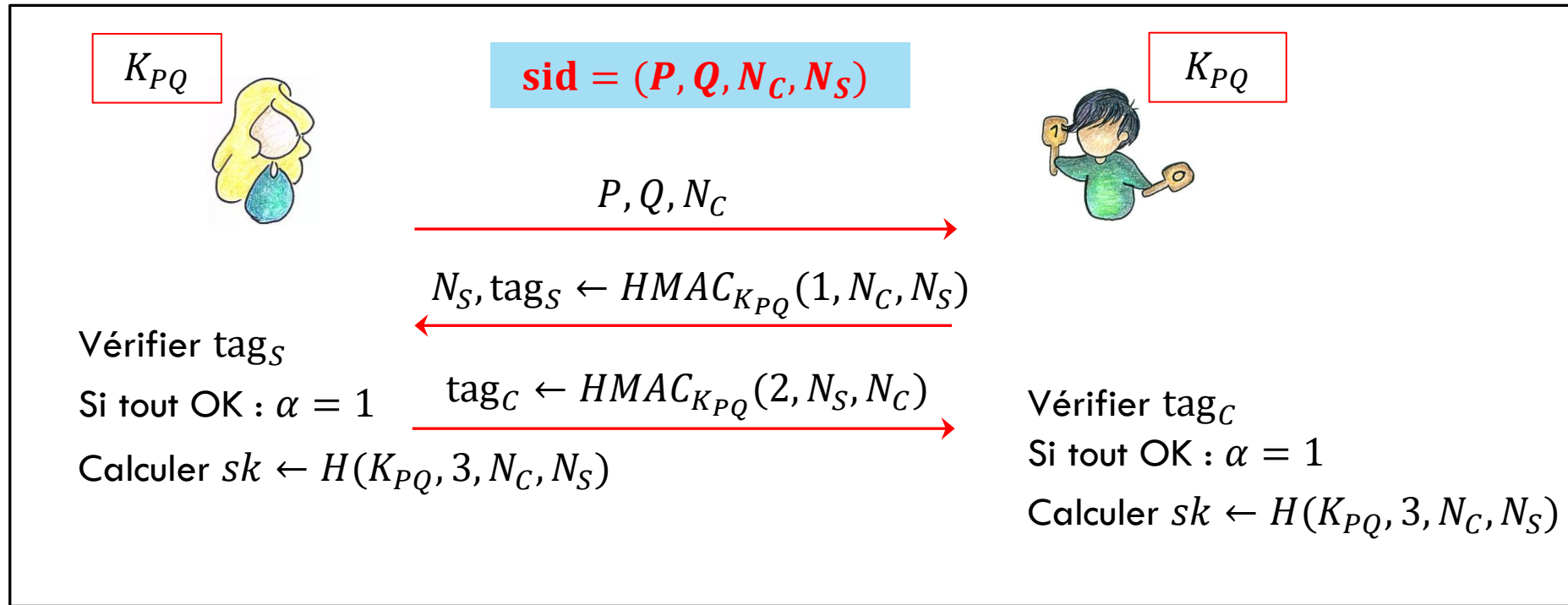
- Un cas un peu bizarre : choisir  $N_S = N_C$
- Même si on pouvait faire un raisonnement de preuve, plus difficile !

# UN PROTOCOLE MODIFIÉ



**Identifiant de session :  $\text{sid} = (P, Q, N_C, N_S)$**

# SÉCURITÉ



- Le protocole est sécurisé si on remplace HMAC et H par  $RO$
- Pourquoi ?

# INTUITION DE PREUVE

➤ Le protocole est sécurisé si on remplace HMAC et H par  $RO$

➤ Authentification Q vers P :

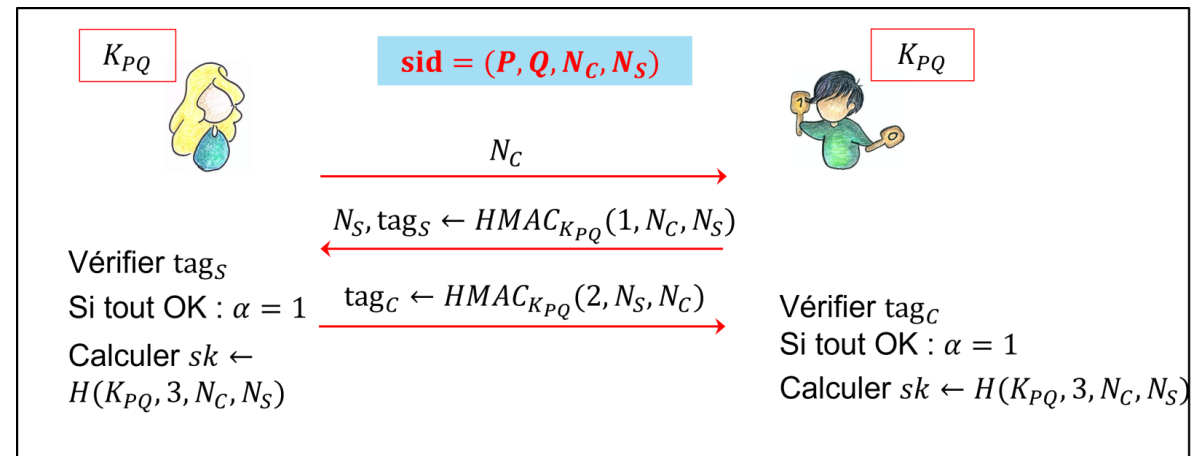
❖ Pour un tuple unique  $(N_C, N_S)$  et une clé  $K_{PQ}$  privée, la valeur  $tag_S$  est imprévisible/unforgeable

➤ Authentification P vers Q :

❖ Pour un tuple unique  $(N_C, N_S)$  et une clé  $K_{PQ}$  privée, la valeur  $tag_C$  est imprévisible/unforgeable

➤ Sécurité de la clé de session :

- ❖ L'instance ciblée est d'une partie non-corrompue, (avec un partenaire non-corrompu) : pas d'impersonification
- ❖ Les nonces sont quasi-unique : chaque clé honnêtement calculée est connue par  $\leq$  deux instances honnêtes
- ❖ Comme H est remplacée par un RO, la clé est ensuite pseudo-aléatoire (sa sécurité dépendra de  $K_{PQ}$ )



# AUTH. SERVEUR VERS CLIENT : STRATÉGIE

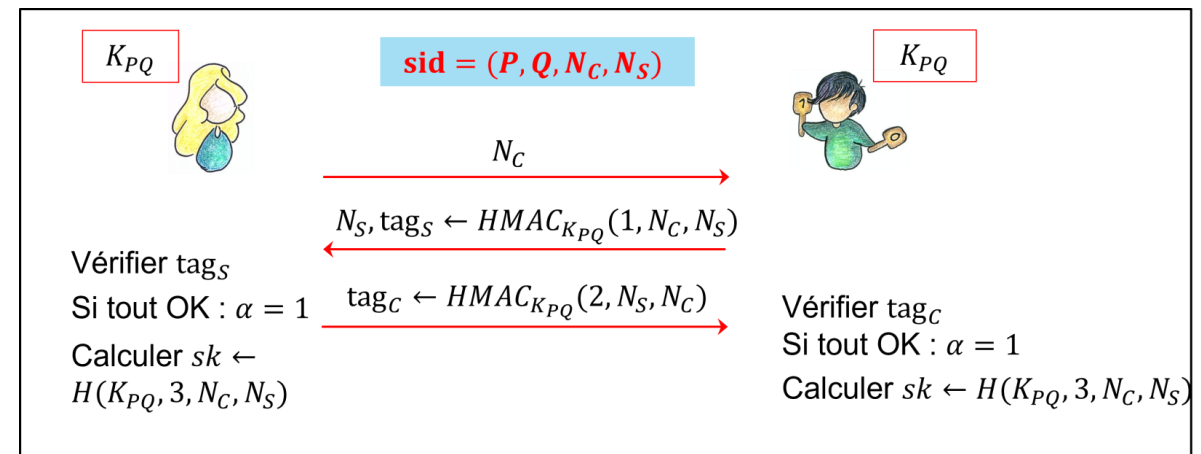
- L'attaquant aura le droit de créer des instances de protocole, envoyer des messages, et aussi de corrompre certaines entités
- L'attaquant devra obliger une instance de P à l'accepter en tant qu'un serveur honnête Q
  - ❖ Ni P ni Q ne peuvent pas être corrompus
  - ❖ L'instance ciblée ne doit pas avoir une instance qui match

➤ Sans compromission de P,Q :  $K_{PQ}$  reste privée

➤ Instance de P honnête : aléa fraîche

- ❖ On peut argumenter l'unicité de  $N_C$  en instance honnête
- ❖ Argumenter : aléa imprévisible par l'attaquant
- ❖ Donc :  $\text{tag}_S$  quasi-unique

➤ Finalement : sans connaître  $K_{PQ}$ , sans avoir  $\text{tag}_S$ , la clé est indistinguible d'aléatoire



# AUTH. SERVEUR VERS CLIENT : PREUVE

➤ On suppose :  $n_C$  clients,  $n_S$  serveurs,  $q_{inst}$  instances créées,  $q_{RO}$  requêtes au RO

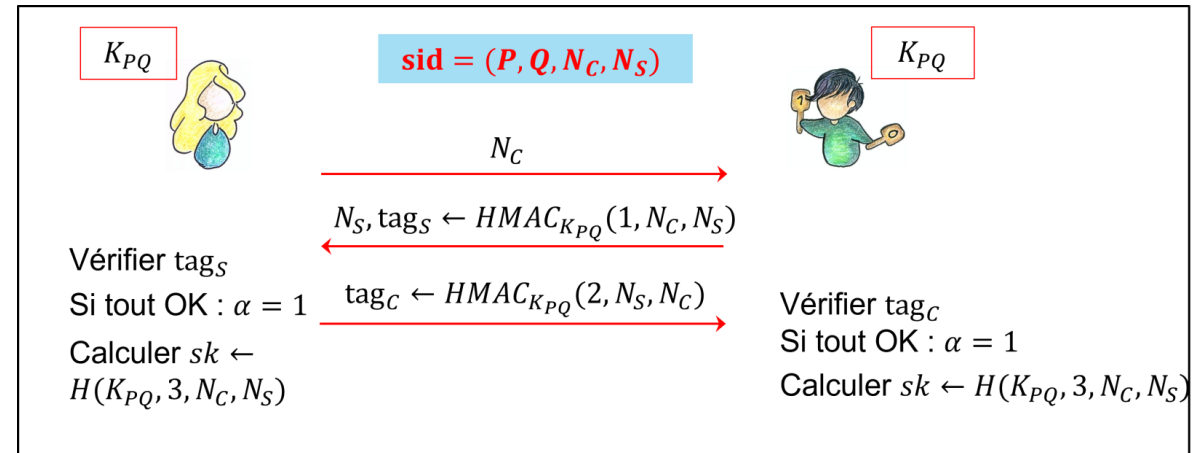
➤  $G_0$  : le jeu initial

➤  $G_1$  : Pas deux fois même  $N_C$  pour inst. honnête

❖ La probabilité d'une collision est  $\binom{q_{inst}}{2} \cdot 2^{-|N_C|}$

❖ Donc :

$$\Pr[A \text{ wins } G_0] \leq \Pr[A \text{ wins } G_1] + \binom{q_{inst}}{2} \cdot 2^{-|N_C|}$$





# AUTH. SERVEUR VERS CLIENT : PREUVE

➤ On suppose :  $n_C$  clients,  $n_S$  serveurs,  $q_{inst}$  instances créées,  $q_{RO}$  requêtes au RO

➤  $G_0$  : le jeu initial

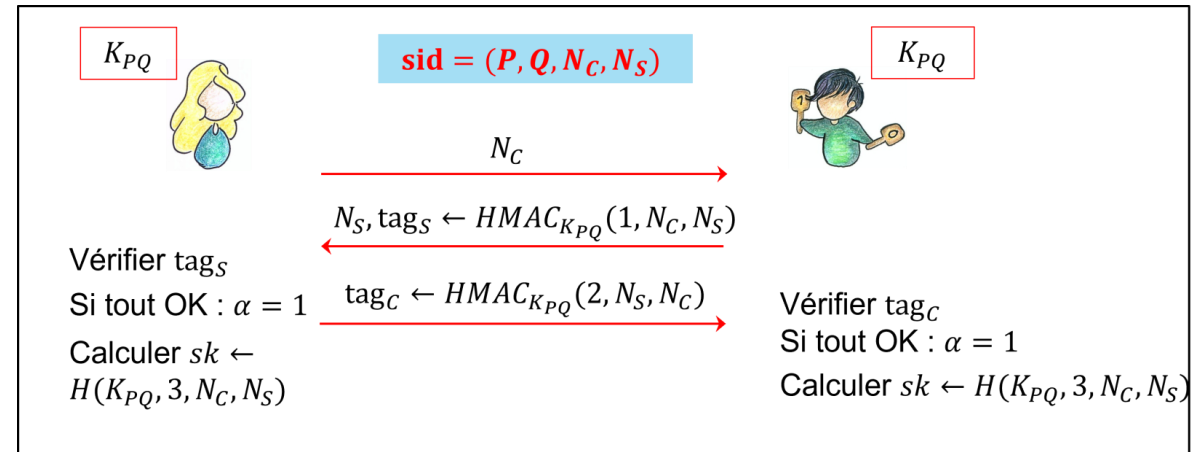
➤  $G_1$  : Pas deux fois même  $N_C$  pour inst. honnête

➤  $G_2$  : A ne peut pas anticiper  $N_C$  honnête :

❖ La probabilité d'une collision est  $\binom{q_{inst}}{2} \cdot 2^{-|N_C|}$  aussi

❖ Donc :

$$\Pr[A \text{ wins } G_1] \leq \Pr[A \text{ wins } G_2] + \binom{q_{inst}}{2} \cdot 2^{-|N_C|}$$



# AUTH. SERVEUR VERS CLIENT : PREUVE

➤ On suppose :  $n_C$  clients,  $n_S$  serveurs,  $q_{inst}$  instances créées,  $q_{RO}$  requêtes au RO

➤  $G_0$  : le jeu initial

➤  $G_1$  : Pas deux fois même  $N_C$  pour inst. honnête

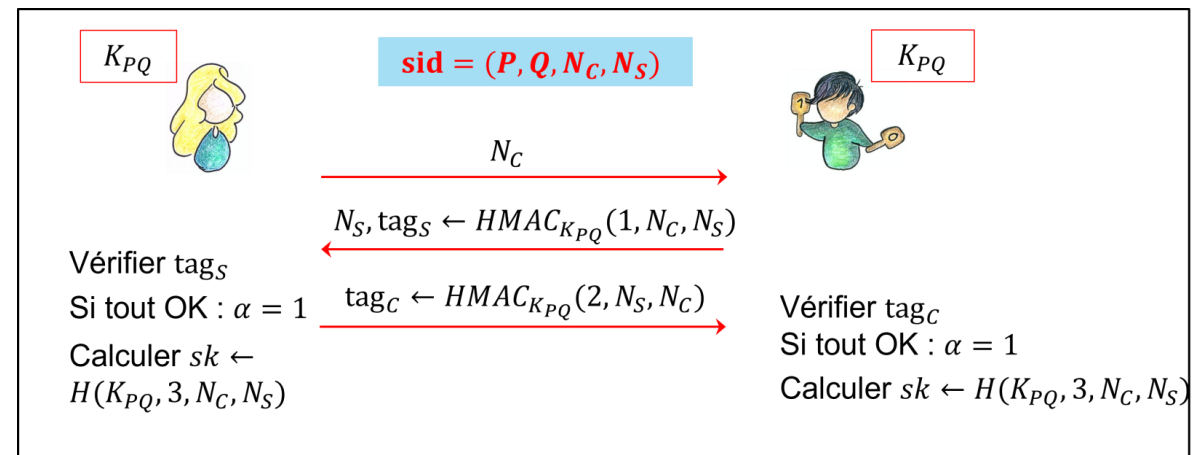
➤  $G_2$  : A ne peut pas anticiper  $N_C$  honnête

➤  $G_3$  : Challenger devine  $P, Q$  ciblés par A

❖ Il y a  $n_C$  clients en total dans le système

❖ Donc :

$$\Pr[A \text{ wins } G_2] \leq n_C \cdot n_S \Pr[A \text{ wins } G_3]$$



# AUTH. SERVEUR VERS CLIENT : PREUVE

➤ On suppose :  $n_C$  clients,  $n_S$  serveurs,  $q_{inst}$  instances créées,  $q_{RO}$  requêtes au RO

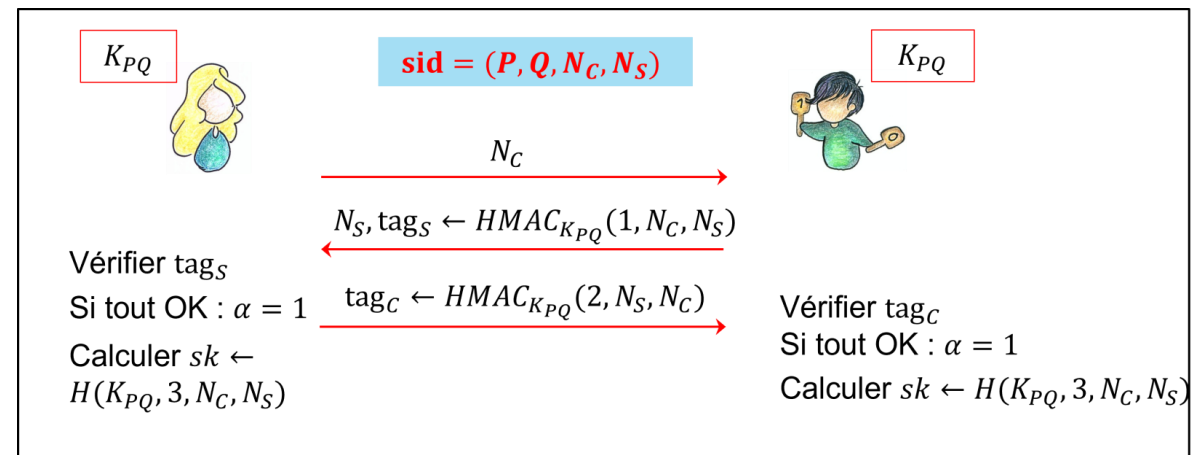
➤  $G_0$  : le jeu initial

➤  $G_1$  : Pas deux fois même  $N_C$  pour inst. honnête

➤  $G_2$  : A ne peut pas anticiper  $N_C$  honnête

➤  $G_3$  : Challenger devine  $P, Q$  ciblés par A

➤  $G_4$  : A n'envoie pas  $(K_{PQ}, 1, N_C, N_S)$  à RO



❖ Notons pas KEYS l'ensemble de clés duquel on choisit les clés long-terme  $K_{PQ}$

❖ Les valeurs  $N_C, N_S$  et 1 sont publiques, mais pas le  $K_{PQ}$  ciblé, utilisé dans les sessions ciblées

❖ Le fait d'avoir deviné  $P, Q$  réduit le nombre de requêtes pertinentes à  $q_{RO}$

❖ Donc :

$$\Pr[A \text{ wins } G_3] \leq \Pr[A \text{ wins } G_4] + q_{RO} \cdot \frac{1}{|\text{KEYS}|}$$

# AUTH. SERVEUR VERS CLIENT : PREUVE

➤ On suppose :  $n_C$  clients,  $n_S$  serveurs,  $q_{inst}$  instances créées,  $q_{RO}$  requêtes au RO

➤  $G_0$  : le jeu initial

➤  $G_1$  : Pas deux fois même  $N_C$  pour inst. honnête

➤  $G_2$  : A ne peut pas anticiper  $N_C$  honnête

➤  $G_3$  : Challenger devine  $P, Q$  ciblés par A

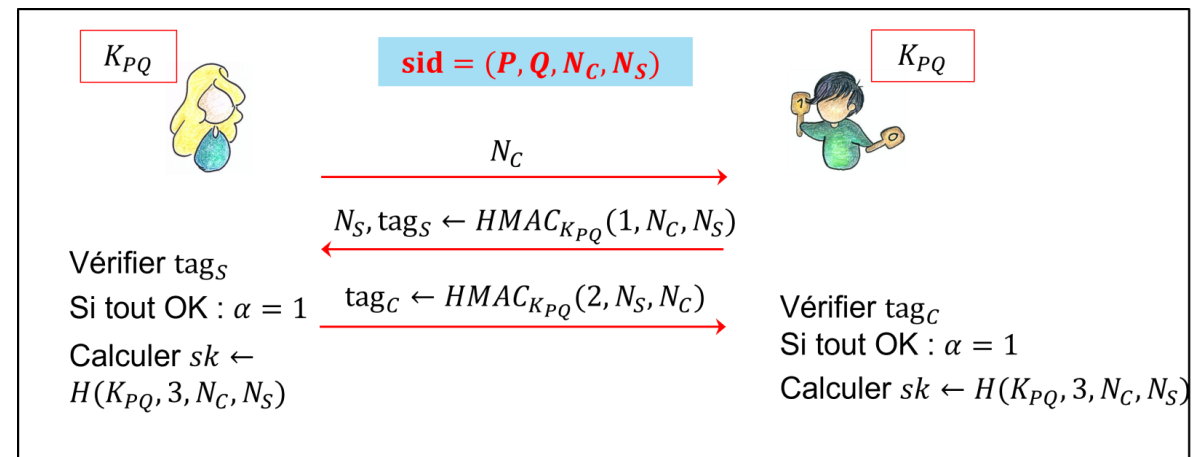
➤  $G_4$  : A n'envoie pas  $(K_{PQ}, 1, N_C, N_S)$  à RO

➤ Gagner  $G_4$  :

❖ La probabilité de gagner maintenant se réduit à soit distinguer HMAC de RO, soit deviner la valeur  $tag_S$

❖ Donc :

$$\Pr[A \text{ wins } G_4] \leq Adv_{RO} + q_{inst} \cdot 2^{-|tag_S|}$$



# RÉSUMER LA PREUVE

➤ **Théorème** : Dans le modèle de l'oracle aléatoire, le protocole est sécurisé en termes de l'authentification serveur vers client. Plus particulièrement :

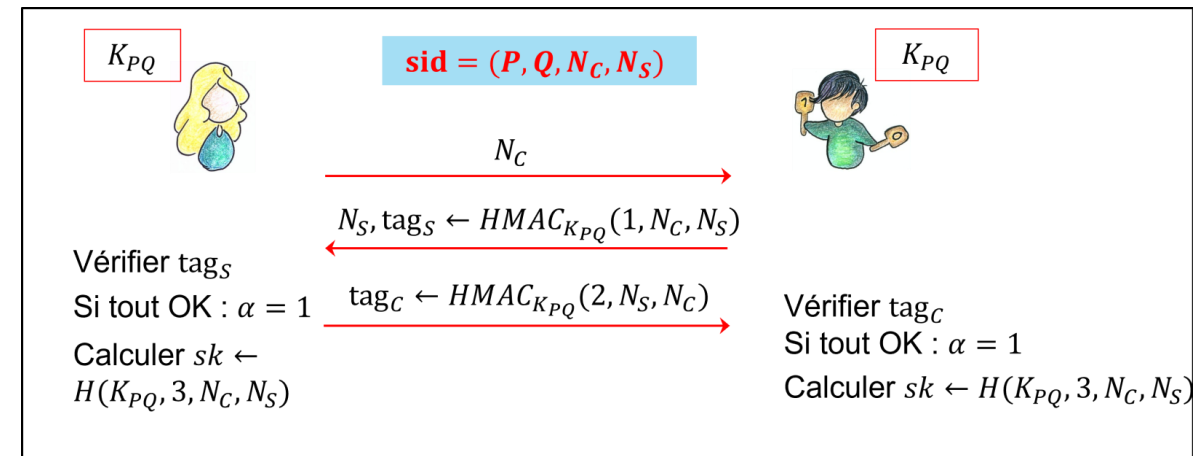
$$\Pr[A \text{ gagne}] \leq 2 \cdot \binom{q_{\text{inst}}}{2} \cdot 2^{-|N_C|} + n_C n_S (\text{Adv}_{RO} + q_{\text{inst}} \cdot 2^{-|\text{tag}_S|} + q_{RO} \cdot \frac{1}{|\text{KEYS}|})$$

➤ **Étapes** :

- ❖ Jeu  $G_1$  : chaque instance honnête de client génère une valeur aléatoire unique :
- ❖ Jeu  $G_2$  : A ne peut pas anticiper une valeur aléatoire utilisée dans une session unique
- ❖ Jeu  $G_3$  : Le challenger devine au moins un client ciblé  $P$  et son partenaire  $Q$
- ❖ Jeu  $G_4$  : A ne peut pas apprendre  $\text{tag}_S$  en envoyant une requête au RO
- ❖ Gagner  $G_4$  : Soit distinguer le HMAC du RO, soit deviner  $\text{tag}_S$

# SÉCURITÉ D'ÉCHANGE DE CLÉS : INTUITION

- L'authentification client vers serveur identique. Et l'échange de clés ?
- L'attaquant pourra créer des instances, envoyer des messages, corrompre et révéler des clés
- A la fin, il devra distinguer d'aléatoire la clé (non-révoquée) d'une instance acceptante, honnête
  - ❖ ... dont le partenaire est non-corrompu
- La clé  $K_{PQ}$  utilisée dans la session reste privée
  - ❖ Car pas de corruption de P, Q
- Argumenter l'unicité de  $N_C, N_S$ , session ciblée
  - ❖ Unicité du tuple  $(N_C, N_S)$  dans une session honnête
  - ❖ Ceci se traduit par une unicité de clés/session
- Argumenter l'impossibilité de calculer la clé d'une session honnête en la demandant au RO
- Maintenant le RO garantit l'indistinguabilité d'aléatoire



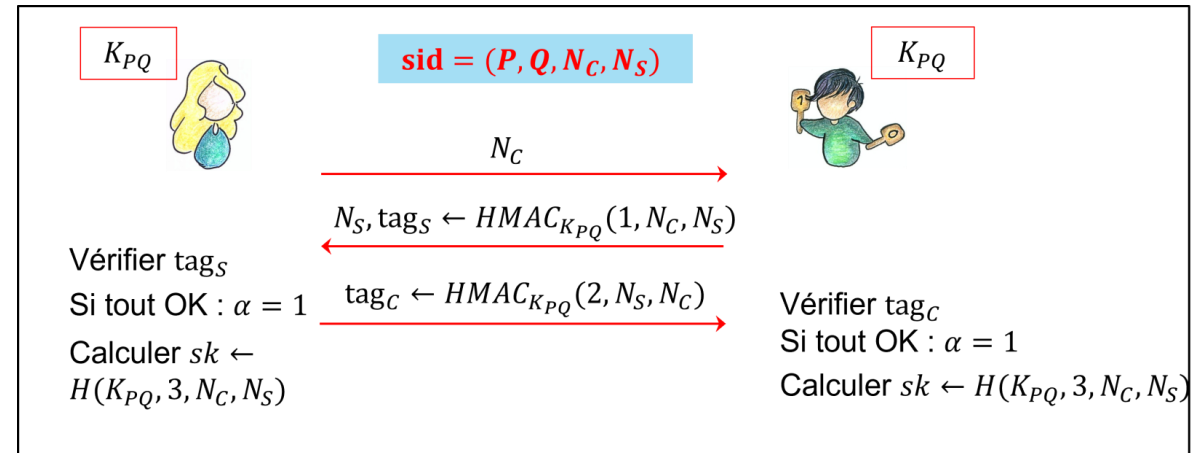
# SÉCURITÉ ÉCHANGE DE CLÉS : PREUVE

- On suppose :  $n_C$  clients,  $n_S$  serveurs,  $q_{inst}$  instances créées,  $q_{RO}$  requêtes au RO
- $G_0$  : le jeu initial
- $G_1$  : Pas de collision sur  $N_C$  en 2 inst. honnêtes

❖ La probabilité d'une collision est  $\binom{q_{inst}}{2} \cdot 2^{-|N_C|}$

❖ Donc :

$$\text{Adv}[A \text{ wins } G_0] \leq \text{Adv}[A \text{ wins } G_1] + \binom{q_{inst}}{2} \cdot 2^{-|N_C|}$$



# SÉCURITÉ ÉCHANGE DE CLÉS : PREUVE

➤ On suppose :  $n_C$  clients,  $n_S$  serveurs,  $q_{\text{inst}}$  instances créées,  $q_{\text{RO}}$  requêtes au RO

➤  $G_0$  : le jeu initial

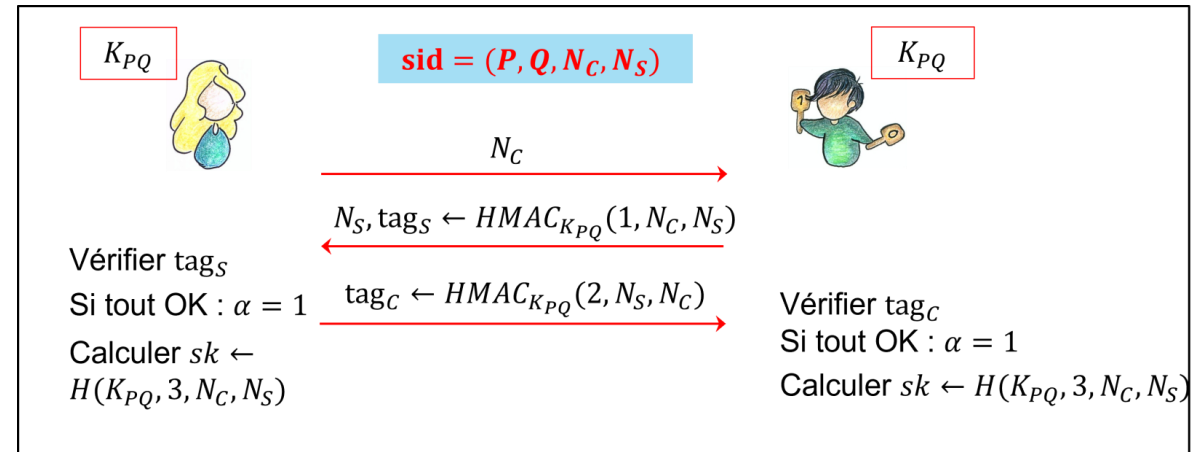
➤  $G_1$  : Pas de collision sur  $N_C$  en 2 inst. honnêtes

➤  $G_2$  : Pas de collision sur  $N_S$  en 2 inst. honnêtes

❖ La probabilité d'une collision est  $\binom{q_{\text{inst}}}{2} \cdot 2^{-|N_S|}$

❖ Donc :

$$\text{Adv}[A \text{ wins } G_1] \leq \text{Adv}[A \text{ wins } G_2] + \binom{q_{\text{inst}}}{2} \cdot 2^{-|N_S|}$$





# SÉCURITÉ ÉCHANGE DE CLÉS : PREUVE

➤ On suppose :  $n_C$  clients,  $n_S$  serveurs,  $q_{inst}$  instances créées,  $q_{RO}$  requêtes au RO

➤  $G_0$  : le jeu initial

➤  $G_1$  : Pas de collision sur  $N_C$  en 2 inst. honnêtes

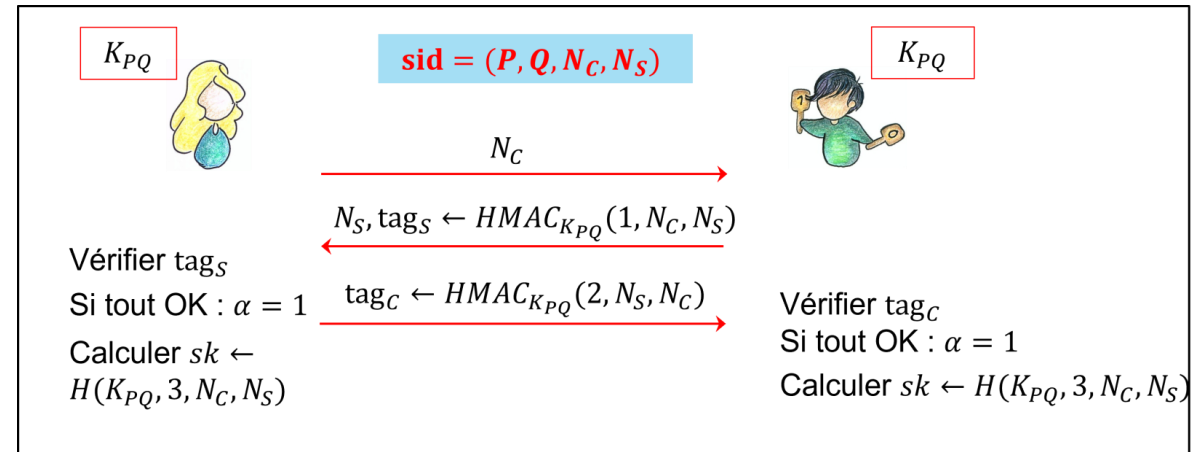
➤  $G_2$  : Pas de collision sur  $N_S$  en 2 inst. honnêtes

➤  $G_3$  : Deviner  $P$  ciblé et son partenaire  $Q$

❖ Soit  $P$  est un client et  $Q$ , un serveur, soit l'inverse

❖ Donc :

$$\text{Adv}[A \text{ wins } G_2] \leq n_C \cdot n_S \cdot \text{Adv}[A \text{ wins } G_3]$$



# SÉCURITÉ ÉCHANGE DE CLÉS : PREUVE

➤ On suppose :  $n_C$  clients,  $n_S$  serveurs,  $q_{inst}$  instances créées,  $q_{RO}$  requêtes au RO

➤  $G_0$  : le jeu initial

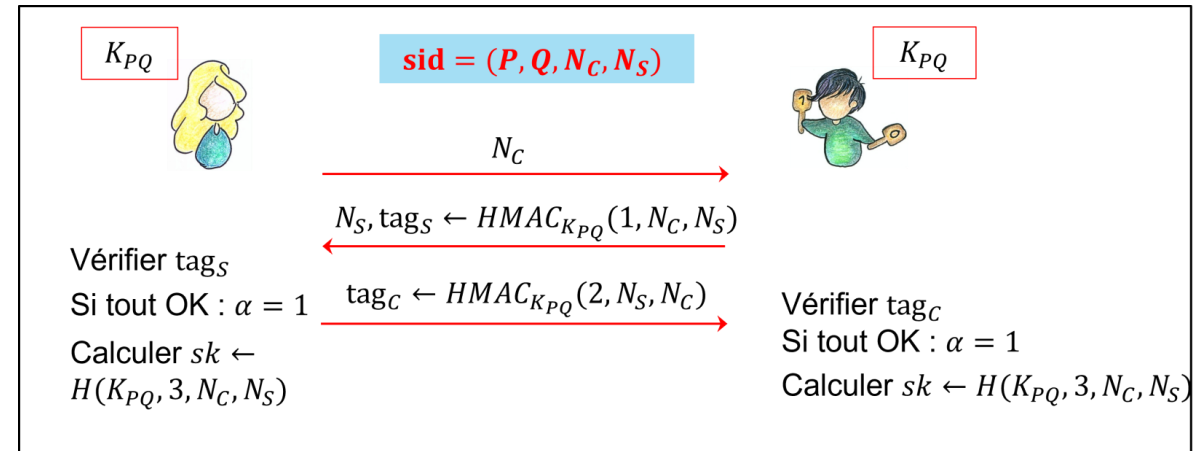
➤  $G_1$  : Pas de collision sur  $N_C$  en 2 inst. honnêtes

➤  $G_2$  : Pas de collision sur  $N_S$  en 2 inst. honnêtes

➤  $G_3$  : Deviner  $P$  ciblé et son partenaire  $Q$

➤ A ce point, la session ciblée a des nonces uniques

$G_4$  : A ne peut plus envoyer  $(K_{PQ}, 3, N_C, N_S)$  au RO



❖ Notant par KEYS l'espace de clés d'où on génère les clés long-terme des utilisateurs :

$$\text{Adv}[A \text{ wins } G_3] \leq \text{Adv}[A \text{ wins } G_4] + q_{RO} \cdot \frac{1}{|\text{KEYS}|}$$

# SÉCURITÉ ÉCHANGE DE CLÉS : PREUVE

➤ On suppose :  $n_C$  clients,  $n_S$  serveurs,  $q_{inst}$  instances créées,  $q_{RO}$  requêtes au RO

➤  $G_0$  : le jeu initial

➤  $G_1$  : Pas de collision sur  $N_C$  en 2 inst. honnêtes

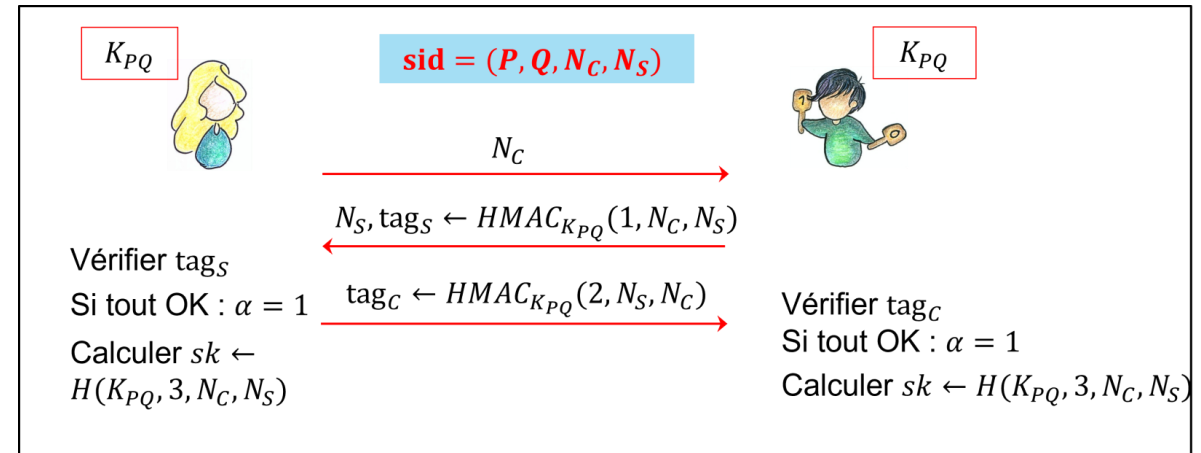
➤  $G_2$  : Pas de collision sur  $N_S$  en 2 inst. honnêtes

➤  $G_3$  : Deviner  $P$  ciblé et son partenaire  $Q$

➤ A ce point, la session ciblée a des nonces uniques

$G_4$  : A ne peut plus envoyer  $(K_{PQ}, 3, N_C, N_S)$  au RO

➤ Gagner  $G_4$  : l'attaquant ne peut plus distinguer entre les deux valeurs, sauf en distinguant la fonction de hachage d'un oracle aléatoire



$$Adv[A \text{ wins } G_4] = Adv_{RO}$$

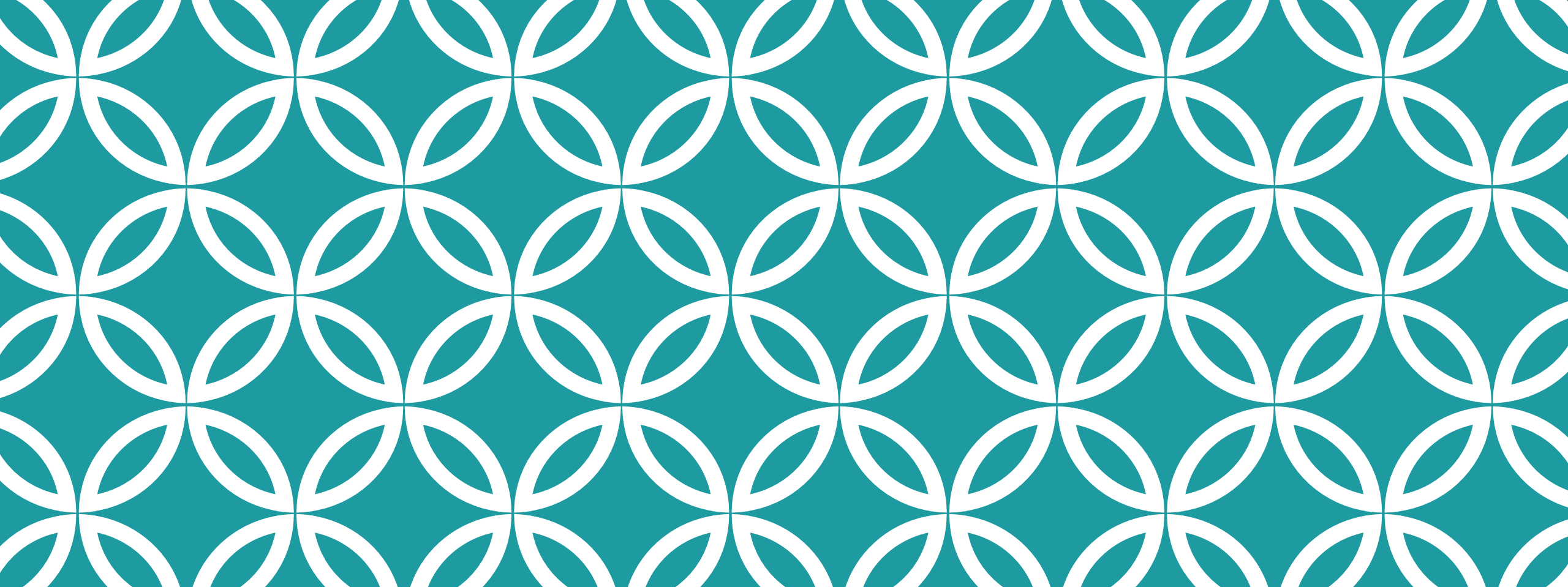
# RÉSUMER LA PREUVE

➤ **Théorème** : Dans le modèle de l'oracle aléatoire, le protocole est AKE-sécurisé. Plus particulièrement :

$$\Pr[A \text{ gagne}] \leq \binom{q_{\text{inst}}}{2} \cdot (2^{-|N_C|} + 2^{-|N_S|}) + n_C \cdot n_S (\text{Adv}_{RO} + q_{RO} \cdot \frac{1}{|\text{KEYS}|})$$

➤ **Étapes** :

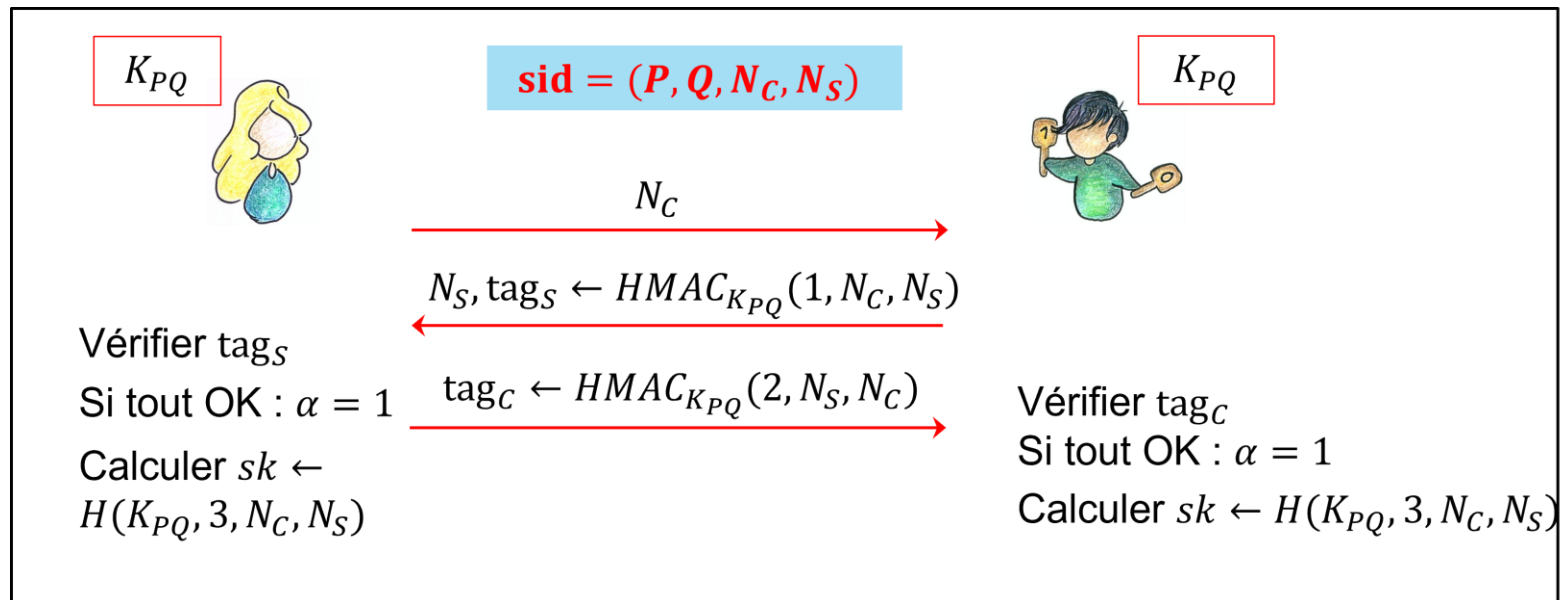
- ❖ Jeu  $G_1$  : chaque  $N_C$  honnêtement généré est unique
- ❖ Jeu  $G_2$  : chaque  $N_S$  honnêtement généré est unique
- ❖ Jeu  $G_3$  : Le challenger devine au la partie ciblée et son partenaire
- ❖ Jeu  $G_4$  : A ne peut pas trouver la clé en envoyant une requête au RO
- ❖ Gagner  $G_4$  : distinguer le HMAC du RO



# QUELQUES RÉFLEXIONS



# HMAC VS. HASH



**Si on idéalise les deux par un RO, pourquoi utiliser deux primitives différentes ?**

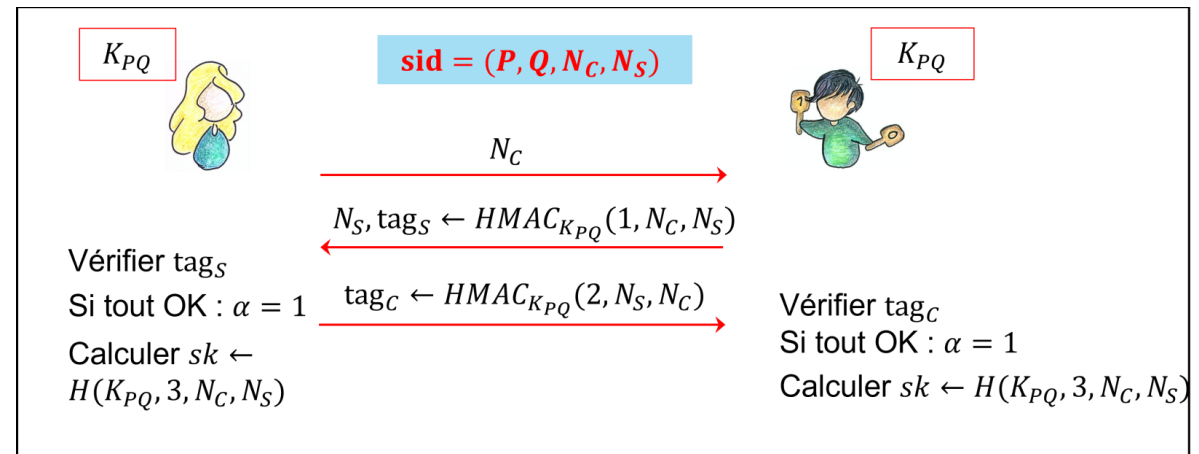
# DIFFÉRENCES KDF, PRF, HMAC, H

➤ Une raison pour différencier entre deux primitives est leur utilité (en termes de sécurité) :

- ❖ Une HMAC permet de s'authentifier
- ❖ Une dérivation de clé peut se faire avec une H (en RO)
- ❖ ... ou avec une KDF

➤ Une autre raison : les hypothèses de sécurité :

- ❖ L'hypothèse du RO est forte
- ❖ Réduction possible au jeu de sécurité d'une fonction pseudo-aléatoire (PRF) pour HMAC
- ❖ Réduction au jeu de sécurité d'une KDF pour remplacer H
- ❖ Ces preuves sont, par contre, plus compliquées



# PERFECT FORWARD SECRECY (PFS)

- Dans la sécurité AKE : instance ciblée honnête, partenaire honnête
- Notion PFS : disons qu'à un moment, une clé long-terme est corrompue par un attaquant
  - ❖ La sécurité des sessions futures est compromise
  - ❖ Mais qu'est-ce qui se passe pour la sécurité des sessions passées ?
- Notre protocole n'est pas PFS-sécurisé
- Pour avoir la PFS, les clés long-termes doivent évoluer (mises à jour par RO)

