

La sécurité prouvable



LES PROTOCOLES D'AUTHENTIFICATION

Cristina Onete
cristina.onete@gmail.com

PROTOCOLES ET PRIMITIVES

- Une séparation souvent "flexible"
- En general :
 - ❖ **une primitive** ne contient pas d'algorithmes interactifs
 - ❖ **un protocole** suppose l'interaction
- Exemples :
 - ❖ Une fonction de hachage est une primitive
 - ❖ Un schéma de chiffrement/MAC est une primitive
 - ❖ On parle d'un protocole d'échange de clé, d'un protocole d'authentification, etc.

PRIMITIVES ET PROTOCOLES D'AUTHENTIFICATION

➤ L'authentification **de messages** est **une primitive** :

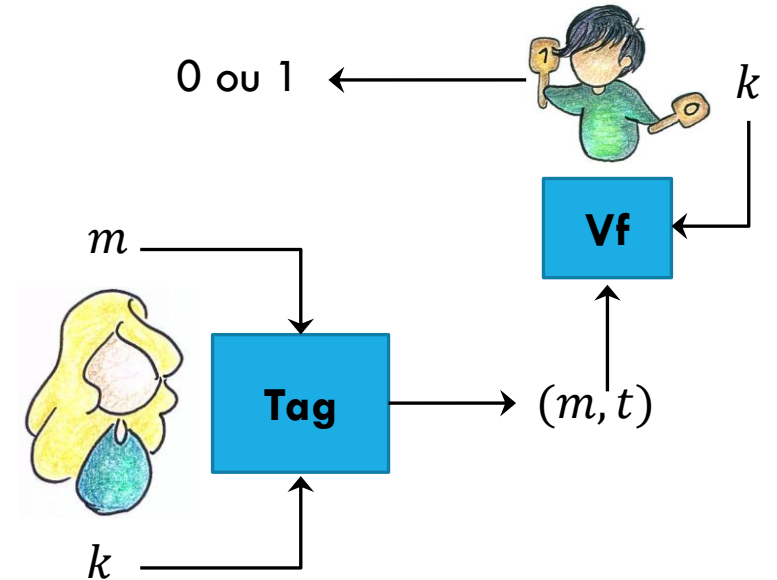
But : authentifier la provenance d'un message

- ✓ Le possesseur de la clé génère le message également
- ✓ La vérification peut se faire immédiatement ou après un délai
- ✓ Les **réjeux** de tuples (m, t) ne consistent pas une attaque !

➤ L'authentification **de personnes** est souvent **un protocole** :

But : établir qu'un certain acteur est le possesseur d'une clé légitime

- ✓ Cela suppose de l'interaction : le possesseur de la clé ne doit pas tout choisir
- ✓ De plus on veut justement **éviter les réjeux** : les réjeux sont des attaques !



LES PROTOCOLES D'AUTHENTIFICATION

➤ Les acteurs :

- ❖ Un prouveur : l'entité qui doit s'authentifier
- ❖ Un vérificateur/vérifieur : l'entité qui vérifie l'authentification
- ❖ Parfois une entité qui gère/génère des clés, indépendante

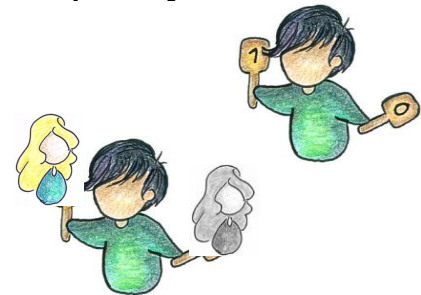


➤ Clé publique, clé secrète :

- ❖ L'authentification à clé publique : le prouveur P possède (sk_P, pk_P) et le vérificateur V aura pk_P
Parfois le vérificateur a accès à une base de données
- ❖ L'authentification à clé secrète : le prouveur et le vérificateur ont accès à une clé symétrique k_P

➤ Authentification : le vérificateur output un bit 1 (valide) ou 0 (invalide)

➤ Identification : le vérificateur output l'identité d'un prouveur P



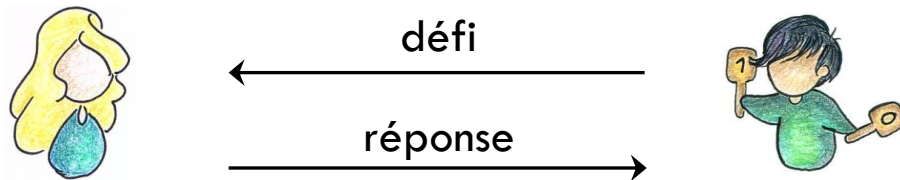
L'AUTHENTIFICATION À CLÉ SYMÉTRIQUE

➤ **Syntaxe** : $\text{Auth} = (\text{KGen}, \text{Prove}, \text{Verify})$

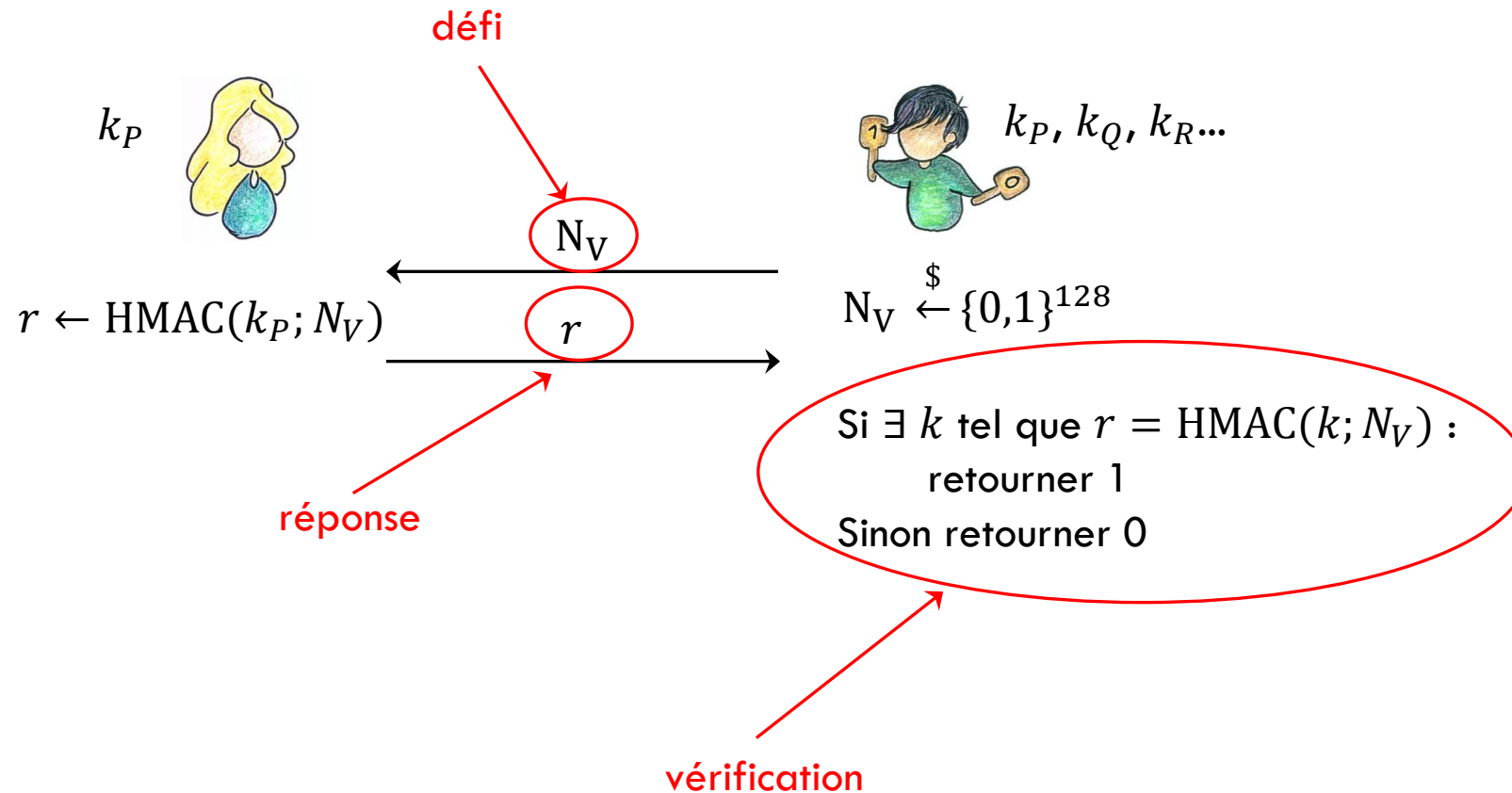
- ❖ Génération de clés : $\text{KGen}(1^\lambda, P)$ donnera en output k_P
- ❖ Prouver-vérifier : $\text{Prove}(k_P)$ et $\text{Verify}(k_P)$ interagissent pour que, à la fin, le vérificateur sort 0 ou 1

➤ **Structure de base** : défi – réponse (challenge-response)

- ❖ Le vérificateur génère un défi frais, pour que l'authentification ne soit pas un réjeu
- ❖ Le prouveur génère une réponse basée sur le défi



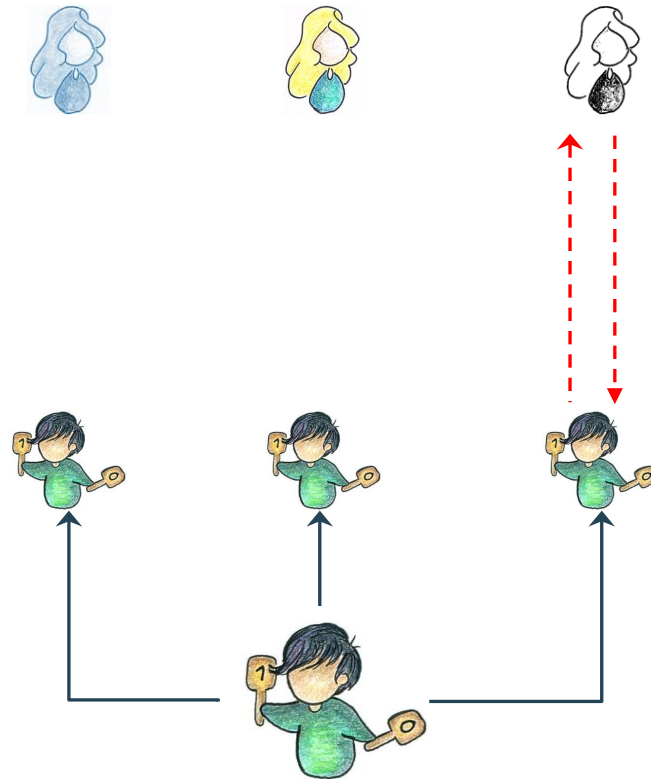
EXEMPLE DE PROTOCOLE



SESSIONS, INSTANCES

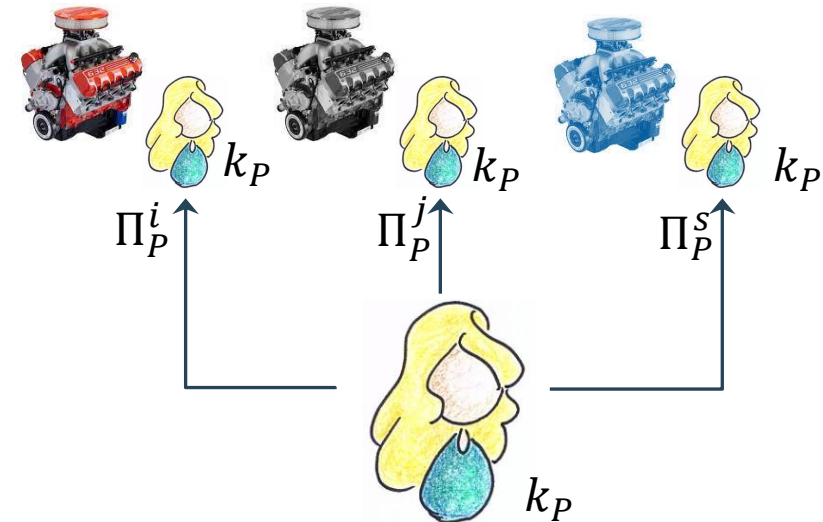
- Les protocoles d'authentification fonctionnent par **session**
 - ❖ Une session : une exécution du protocole, entre deux partenaires

- Chaque partenaire contribue $\frac{1}{2}$ de session : **une instance**
 - ❖ Plusieurs instances peuvent être **exécutées en même temps**
 - ❖ Chaque instance produit des **valeurs aléatoires indépendantes**
 - ❖ Une instance peut être aperçue comme un **algorithme**
 - ❖ Entrées provenant du partenaire
 - ❖ Sorties arrivent au partenaire
 - ❖ Dans l'authentification, le **vérificateur exécute des instances parallèles**
 - ❖ Le prouveur, plutôt des **instances séquentielles**



DES ACCRÉDITATIONS D'AUTHENTIFICATION

- Chaque **participant P** possède :
 - ❖ Une clé symétrique k_P
 - ❖ La capacité de générer des valeurs aléatoires
 - ❖ La capacité de calculer les messages du protocole prévu
- Chaque **instance de P** :
 - ❖ Utilise la clé symétrique
 - ❖ Génère des aléas et fait des calculs indépendamment
 - ❖ Une mise à jour de la clé affecte toutes les instances, une valeur aléatoire générée n'affecte qu'une



!! Exécutions concurrentes seulement modélisables pour protocole sans état !!



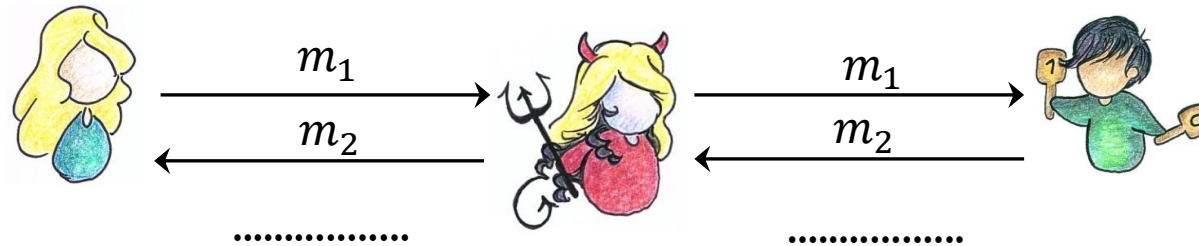
LA SÉCURITÉ DE L'AUTHENTIFICATION



LA SÉCURITÉ DES PROT. D'AUTHENTIFICATION

- **Intuition** : un attaquant (c-à-d un participant illégitime) ne doit jamais pouvoir s'authentifier
 - ❖ L'attaquant aura accès à certaines exécutions honnêtes du protocole
 - ❖ Il pourra même contacter un prouveur/vérificateur de son choix
 - ❖ Par contre Il ne peut pas contacter un prouveur légitime pendant une session d'authentification "cible"

L'authentification est susceptible aux attaques par relais !!!



VERS UNE DÉFINITION FORMELLE

➤ Intuition 1 :

- ❖ L'attaquant aura accès à certaines exécutions honnêtes du protocole
- ❖ Il pourra même contacter un prouveur/vérificateur de son choix

➤ Formalisation :

- ❖ Un **oracle de création d'instance** d'une partie choisie : $\text{NewInstance}(P)$ qui crée une nouvelle instance Π_P^i
- ❖ Un **oracle d'envoi de message** : $\text{Send}(\Pi_P^i, m)$ qui envoie un message à l'instance Π_P^i et output la réponse

VERS UNE DÉFINITION FORMELLE

➤ Intuition 2 :

- ❖ L'attaquant doit s'authentifier auprès du vérificateur dans une session dans laquelle aucun prouveur n'intervient

Comment différencier formellement entre les sessions ?

➤ Instances, sessions, matching :

- ❖ Associer chaque session à un identifiant (**session identifier**, noté sid) :
 - En pratique, une concaténation de valeurs propres à la session : quelques messages, tout le transcript, etc.
 - L'identifiant de la session évolue donc au cours du protocole
- ❖ Deux instances qui ont le même sid exécutent la même session : **les deux instances matchent**
- ❖ Corollaire : si un prouveur légitime intervient dans une session : il y a une instance du prouveur dont sid match le sid de l'instance du vérificateur

VERS UNE DÉFINITION FORMELLE

➤ Intuition 2 :

- ❖ L'attaquant doit s'authentifier auprès du vérificateur dans une session dans laquelle aucun prouveur n'intervient

➤ Formalisation :

- ❖ L'attaquant doit causer le vérificateur de produire une sortie de 1 (accepter)
- ❖ ... Dans une session Π_V^i avec identifiant sid tel que :
 - il n'existe aucune session d'un prouveur P
 - et aucune instance Π_P^j avec
 - $\Pi_V^i.\text{sid} = \Pi_P^j.\text{sid}$

FORMALISATION

➤ Environnement :

- ❖ Un nombre n de prouveurs $(P_1, P_2 \dots P_n)$, un vérificateur V
- ❖ Des clés k_{P_i} pour chaque P_i , données au vérificateur aussi
- ❖ Un adversaire au milieu

$$\text{fini} \leftarrow A^{\text{oNewSession}(\cdot), \text{oSend}(\cdot, \cdot)}(\lambda)$$

A gagne ssi. $\exists \Pi_V^x$ tel que :

la sortie de Π_V^x est 1

et $\nexists P, y$ tel que $\Pi_V^x.\text{sid} = \Pi_P^y.\text{sid}$

➤ Oracles :

- ❖ **$\text{oNewSession}(P_i) \rightarrow (\Pi_{P_i}^s, \Pi_V^t)$** : étant donné l'identifiant d'un prouveur P_i , cet oracle retourne deux nouvelles instances, une du prouveur et une du vérificateur
- ❖ **$\text{oSend}(\Pi, m) \rightarrow m^*$** : étant donné une instance (d'un prouveur ou d'un vérificateur) et un message, l'oracle retourne un message m^* . Si $m = \text{"Start"}$ et Π est une instance de la partie qui commence le protocole (prouveur ou vérificateur), alors m^* est le premier message du protocole

QUE PEUT FAIRE L'ATTAQUANT ?

➤ **Voir une exécution honnête** du protocole entre un prouveur P_i et un vérificateur V :

1. Générer une instance du prouveur et une du vérificateur : $\text{oNewSession}(P_i) \rightarrow (\Pi_{P_i}^s, \Pi_V^t)$
2. Supposant que le vérificateur commence : $\text{oSend}(\Pi_V^t, \text{Start}) \rightarrow m_1$, générer le premier message du protocole
3. L'attaquant fait suivre le message à P_i : $\text{oSend}(\Pi_{P_i}^s, m_1) \rightarrow m_2$, générer la réponse m_2
4. L'attaquant fait suivre m_2 à V : $\text{oSend}(\Pi_V^t, m_2) \rightarrow m_3$, etc.
5. ... continuer jusqu'à la fin du protocole

➤ À la fin : l'attaquant aura tout le transcript

$\text{fini} \leftarrow A^{\text{oNewSession}(\cdot), \text{oSend}(\cdot, \cdot)}(\lambda)$

A gagne ssi. $\exists \Pi_V^x$ tel que :

la sortie de Π_V^x est 1

et $\nexists P, y$ tel que $\Pi_V^x.\text{sid} = \Pi_P^y.\text{sid}$

QUE PEUT FAIRE L'ATTAQUANT ?

➤ Exécuter le protocole avec un prouveur P_i (sans vérificateur) :

1. Générer une instance du prouveur : $\text{oNewSession}(P_i) \rightarrow (\Pi_{P_i}^s, \Pi_V^t)$. Ignorer Π_V^t
2. Supposant que le vérificateur commence : l'attaquant génère un message \widehat{m}_1
3. L'attaquant fait suivre le message à P_i : $\text{oSend}(\Pi_{P_i}^s, \widehat{m}_1) \rightarrow m_2$, générer la réponse m_2
4. L'attaquant génère un \widehat{m}_3 et utilise : $\text{oSend}(\Pi_{P_i}^s, \widehat{m}_3) \rightarrow m_4$, pour apprendre la réponse d'un prouveur honnête
5. ... continuer jusqu'à la fin du protocole

➤ À la fin : une session attaquant-prouveur

$$\text{fini} \leftarrow A^{\text{oNewSession}(\cdot), \text{oSend}(\cdot, \cdot)}(\lambda)$$

A gagne ssi. $\exists \Pi_V^x$ tel que :

la sortie de Π_V^x est 1

et $\nexists P, y$ tel que $\Pi_V^x.\text{sid} = \Pi_P^y.\text{sid}$

QUE PEUT FAIRE L'ATTAQUANT ?

- **Exécuter le protocole avec le vérificateur** (sans prouveur) :
 - ❖ Identique à la stratégie avec un prouveur (sans vérificateur)
- À la fin : l'attaquant aura exécuté une session avec le vérificateur

$$\text{fini} \leftarrow A^{\text{NewSession}(\cdot), \text{Send}(\cdot, \cdot)}(\lambda)$$

A gagne ssi. $\exists \Pi_V^x$ tel que :

la sortie de Π_V^x est 1

et $\nexists P, y$ tel que $\Pi_V^x.\text{sid} = \Pi_P^y.\text{sid}$

QUE PEUT FAIRE L'ATTAQUANT ?

- **Exécuter le protocole en tant que Personne-au-milieu** entre un prouveur et un vérificateur :
 - ❖ Identique à la stratégie de session entre un prouveur et un vérificateur, mais l'adversaire peut modifier des messages
- Attention : on ne peut pas gagner avec une telle session !

$$\text{fini} \leftarrow A^{\text{NewSession}(\cdot), \text{Send}(\cdot, \cdot)}(\lambda)$$

A gagne ssi. $\exists \Pi_V^x$ tel que :

la sortie de Π_V^x est 1

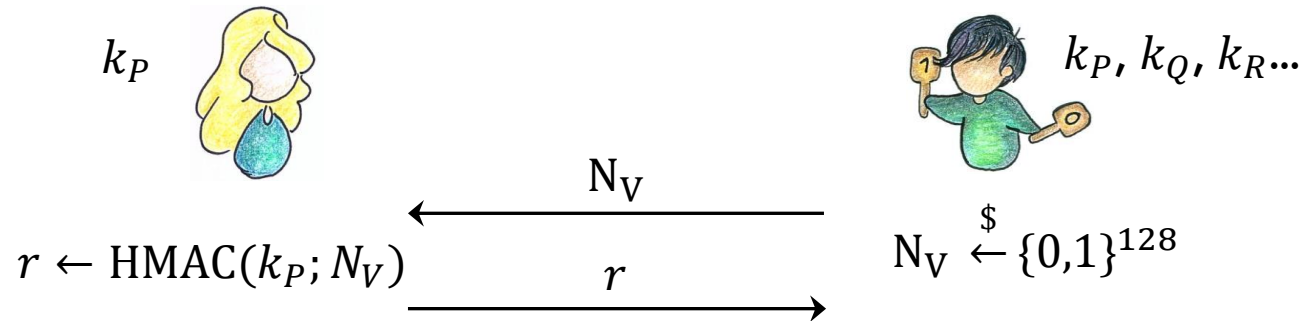
et $\nexists P, y$ tel que $\Pi_V^x.\text{sid} = \Pi_P^y.\text{sid}$



LA SÉCURITÉ DE NOTRE EXEMPLE



EXEMPLE DE PROTOCOLE



Si $\exists k$ tel que $r = \text{HMAC}(k; N_V)$:
retourner 1
Sinon retourner 0

CE QUE L'ATTAQUANT APPREND DES ORACLES

➤ Exécution honnête :

- ❖ des tuples (N_V, r) pour certaines valeurs de N_V

➤ Exécution avec le prouveur :

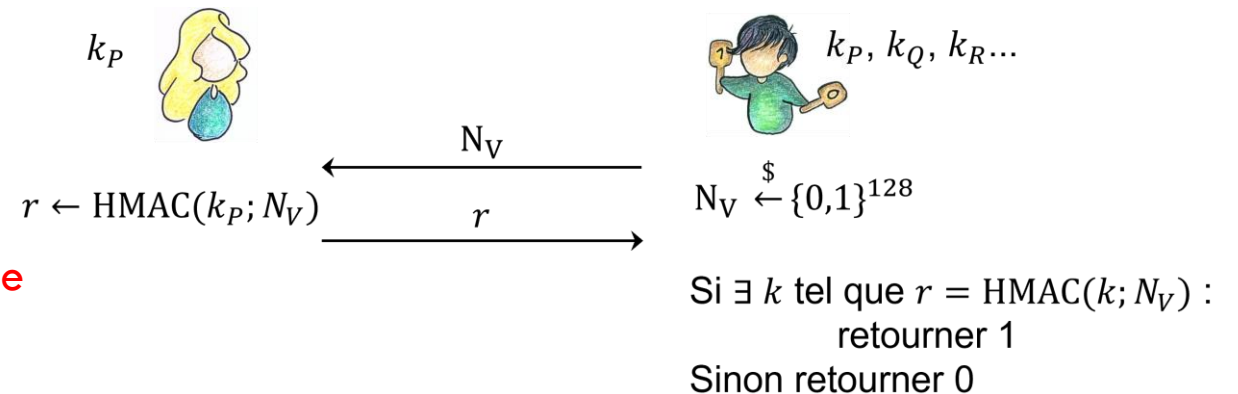
- ❖ des réponses r à des défis N_V pour des valeurs choisies par l'adversaire

➤ Exécution avec le vérificateur :

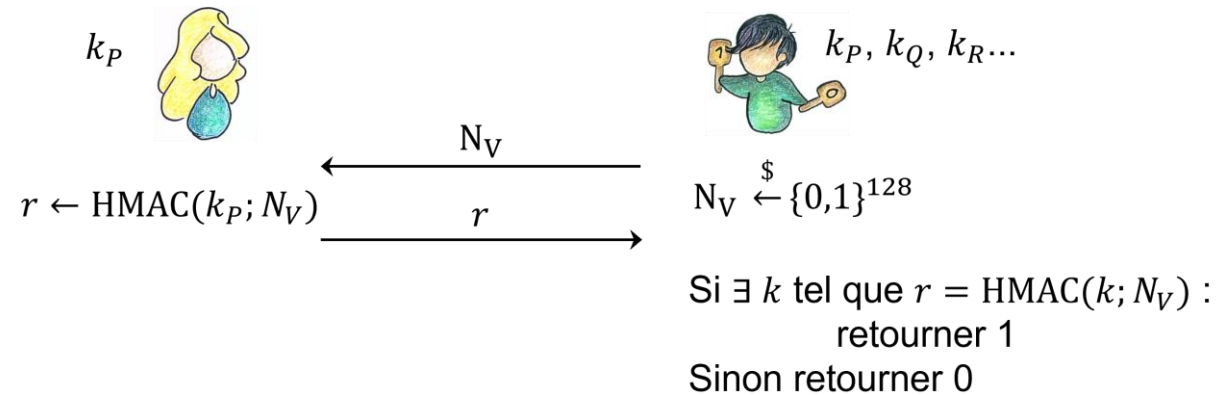
- ❖ L'attaquant doit répondre avec une valeur r correcte pour un frais défi N_V

➤ Exécution prouveur/vérificateur modifiée :

- ❖ modifier le défi \Rightarrow nouvelle valeur r
- ❖ modifier $r \Rightarrow$ la réponse de V sera 0



INTUITION PREUVE



➤ **Pour gagner** l'attaquant aurait besoin que :

- ❖ Le défi N_V de la session ciblée **se répète** dans une autre session qu'il a vu **avec le prouveur**
- ❖ OU : Le défi N_V de la session ciblée se répète dans une **session préalable avec le prouveur légitime**
- ❖ OU : Pour un **défi N_V frais**, l'attaquant peut **trouver la réponse** souhaitée (sans la demander à un prouveur)
 - ❖ SOIT : **en exploitant** la/les faiblesses de l'utilisation de HMAC
 - ❖ SOIT : **en devinant** sur un petit ensemble

STRATÉGIE DE PREUVE

➤ **Théorème** : Ce protocole est sécurisé dans le modèle de l'oracle aléatoire (HMAC => RO)

➤ **Preuve** : Stratégie de preuve :

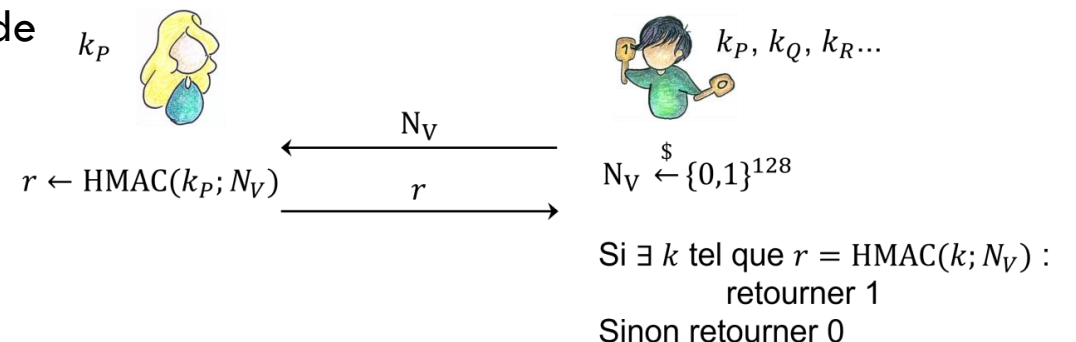
❖ Pré-preuve : transition au RO coûte ϵ_{RO}

❖ Premier pas : Nous allons supposer que le vérificateur ne génère jamais la même valeur deux fois
Ceci n'est pas vrai (collisions possibles !)

❖ Deuxième pas : Pour gagner, l'attaquant devra gagner dans une session avec V, sans aide
On suppose que A n'a pas deviné la valeur du défi utilisé dans cette session avant l'heure !

❖ Troisième pas : A ne peut pas envoyer $(k; N_V)$ pour un k valide

❖ A ce point : l'attaquant ne peut que deviner la réponse



PREMIER PAS

- **Théorème** : Ce protocole est sécurisé dans le modèle de l'oracle aléatoire (HMAC \Rightarrow RO)
- **Preuve** : transition vers RO : ϵ_{RO}
 - ❖ Premier pas : Nous allons supposer que le vérificateur ne génère jamais la même valeur deux fois
 - ❖ G_0 : jeu initial
 - ❖ G_1 : V ne répète jamais un défi généré

 - ❖ Disons que A fait exécuter :
 - q_{PV} sessions avec le prouveur et vérificateur (honnêtes et malhonnêtes)
 - q_P sessions avec un prouveur
 - q_V sessions avec le vérificateur
 - ❖ $\Pr[A \text{ gagne } G_0] \leq \Pr[A \text{ gagne } G_1] + \binom{q_{PV} + q_V}{2} \cdot 2^{-|N_V|}$

DEUXIÈME PAS

- **Théorème** : Ce protocole est sécurisé dans le modèle de l'oracle aléatoire (HMAC \Rightarrow RO)
- **Preuve** : transition vers RO : ϵ_{RO}
 - ❖ Deuxième pas : Nous allons supposer que l'adversaire n'est pas capable, dans ses q_P sessions avec le prouveur, deviner "avant l'heure" aucun défi utilisé dans une session avec le vérificateur
 - ❖ G_0 : jeu initial
 - ❖ G_1 : V ne répète jamais un défi généré
 - ❖ G_2 : A ne devine aucun défi utilisé dans une session avec le vérificateur
 - ❖ Pour un défi fixe (dans une session avec V), proba de deviner dans un des q_P sessions : $q_P \cdot 2^{-|N_V|}$
 - ❖ $\Pr[A \text{ gagne } G_1] \leq \Pr[A \text{ gagne } G_2] + q_P q_V \cdot 2^{-|N_V|}$

TROISIÈME PAS

- **Théorème** : Ce protocole est sécurisé dans le modèle de l'oracle aléatoire (HMAC \Rightarrow RO)
- **Preuve** : transition vers RO : ϵ_{RO}
 - ❖ Troisième pas : Nous allons interdire à l'adversaire d'envoyer une requête $RO(k; N_V)$ pour aucun défi vu dans une session avec le vérificateur (total de q_V valeurs distinctes de N_V)
 - ❖ G_0 : jeu initial
 - ❖ G_1 : V ne répète jamais un défi généré
 - ❖ G_2 : A ne devine aucun défi utilisé dans une session avec le vérificateur
 - ❖ G_3 : ayant reçu un défi frais et imprévisible, A n'a plus le droit de demander $RO(k; N_V)$

 - ❖ La valeur k est la seule partie inconnue de la requête
 - ❖ $\Pr[A \text{ gagne } G_2] \leq \Pr[A \text{ gagne } G_3] + q_V \cdot 2^{-|k|}$

ETAPE FINALE

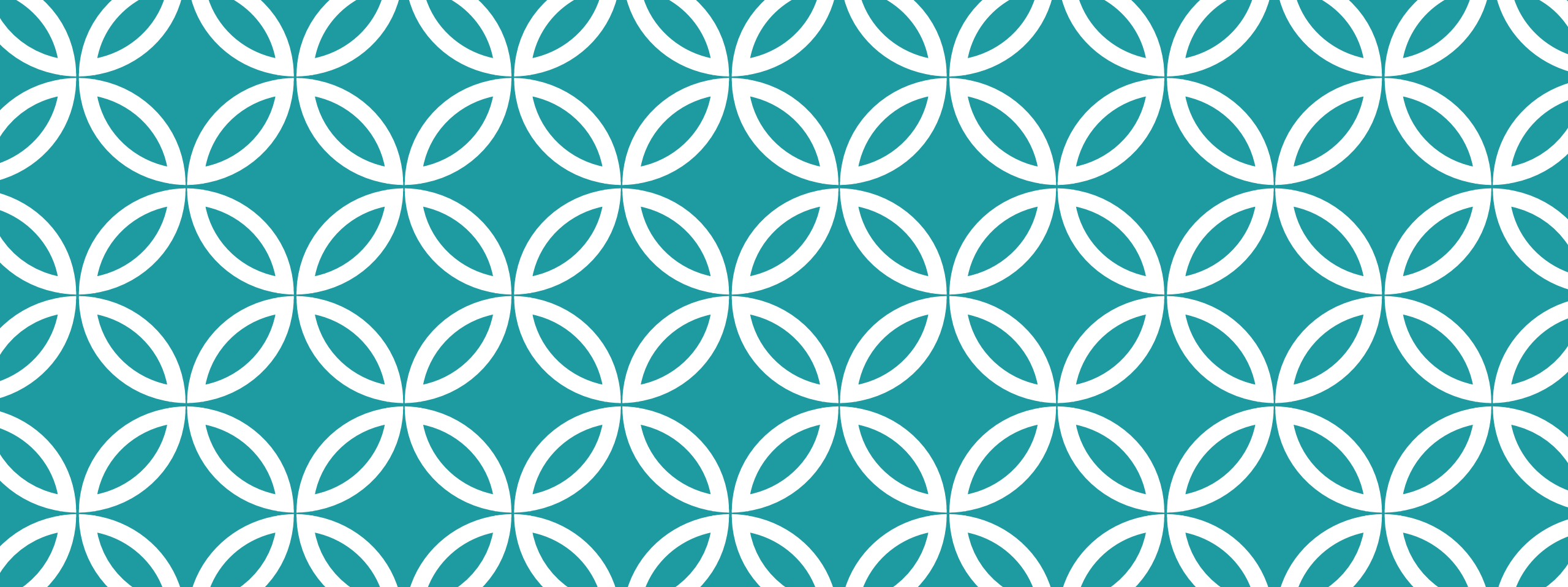
- **Théorème** : Ce protocole est sécurisé dans le modèle de l'oracle aléatoire (HMAC \Rightarrow RO)
- **Preuve** : transition vers RO : ϵ_{RO}
 - ❖ Étape finale : Produire une bonne réponse à une session avec le vérificateur parmi les q_V sessions
 - ❖ G_0 : jeu initial
 - ❖ G_1 : V ne répète jamais un défi généré
 - ❖ G_2 : A ne devine aucun défi utilisé dans une session avec le vérificateur
 - ❖ G_3 : ayant reçu un défi frais et imprévisible, A n'a plus le droit de demander $RO(k; N_V)$
 - ❖ Gagner G_3 : Maintenant A n'a plus le droit d'apprendre une réponse r valide à un défi N_V frais via RO.
Il n'a pas le droit d'utiliser le prouveur en tant qu'oracle pour trouver la bonne réponse (par modèle)
- ❖ Son seul recours serait de deviner la valeur de r : $\Pr[A \text{ gagne } G_3] \leq q_V \cdot 2^{-|r|}$

QUANTIFICATION TOTALE

- **Théorème** : Ce protocole est sécurisé dans le modèle de l'oracle aléatoire (HMAC \Rightarrow RO).
Pour chaque adversaire contre la sécurité du protocole :

$$\Pr[A \text{ gagne}] \leq \epsilon_{RO} + \left[q_P q_V + \binom{q_{PV} + q_V}{2} \right] \cdot 2^{-|N_V|} + q_V \cdot (2^{-|k|} + 2^{-|r|})$$

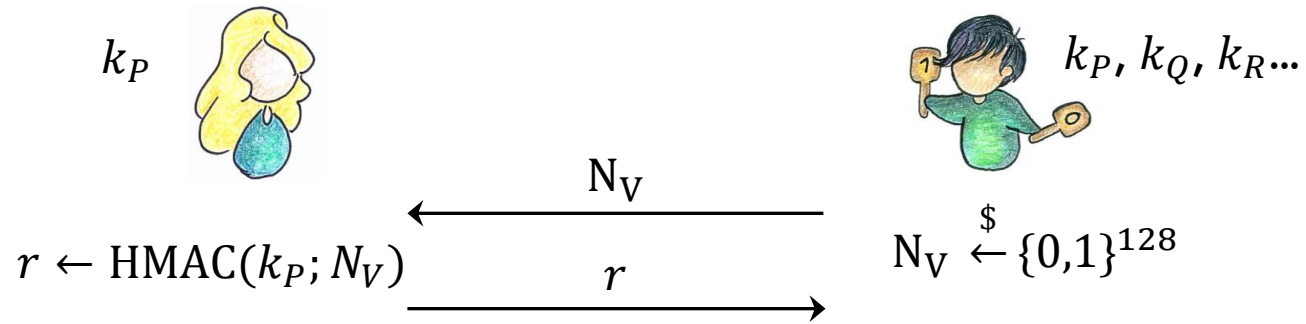
- ❖ G_0 : jeu initial
- ❖ G_1 : Enlever les collisions sur N_V prouveur honnête : $\Pr[A \text{ gagne } G_0] \leq \Pr[A \text{ gagne } G_1] + \binom{q_{PV} + q_V}{2} \cdot 2^{-|N_V|}$
- ❖ G_2 : A ne devine pas un N_V "important" : $\Pr[A \text{ gagne } G_1] \leq \Pr[A \text{ gagne } G_2] + q_P q_V \cdot 2^{-|N_V|}$
- ❖ G_3 : A n'a plus le droit de demander $RO(k; N_V)$: $\Pr[A \text{ gagne } G_2] \leq \Pr[A \text{ gagne } G_3] + q_V \cdot 2^{-|k|}$
- ❖ Gagner G_3 : $\Pr[A \text{ gagne } G_3] \leq q_V \cdot 2^{-|r|}$



PROCOLES D'AUTHENTIFICATION



NOTRE PREMIER PROTOCOLE

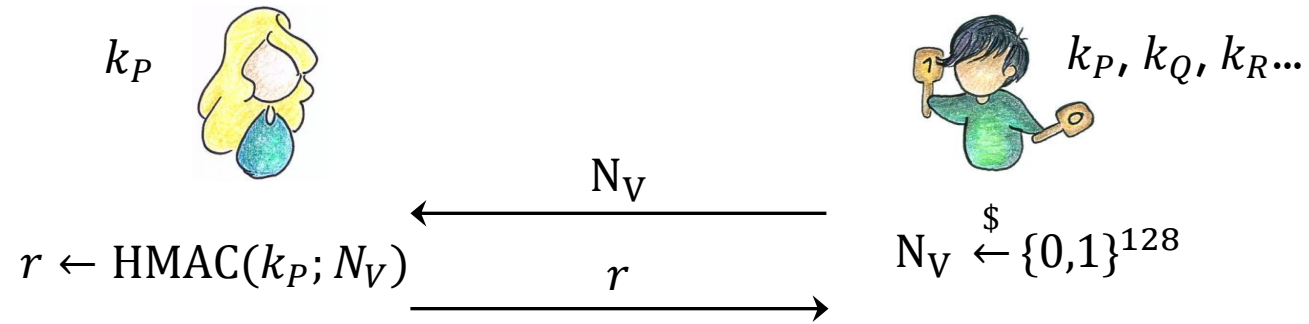


Si $\exists k$ tel que $r = \text{HMAC}(k; N_V)$:
retourner 1
Sinon retourner 0

➤ Le protocole est sécurisé

Peut-on vouloir plus ?

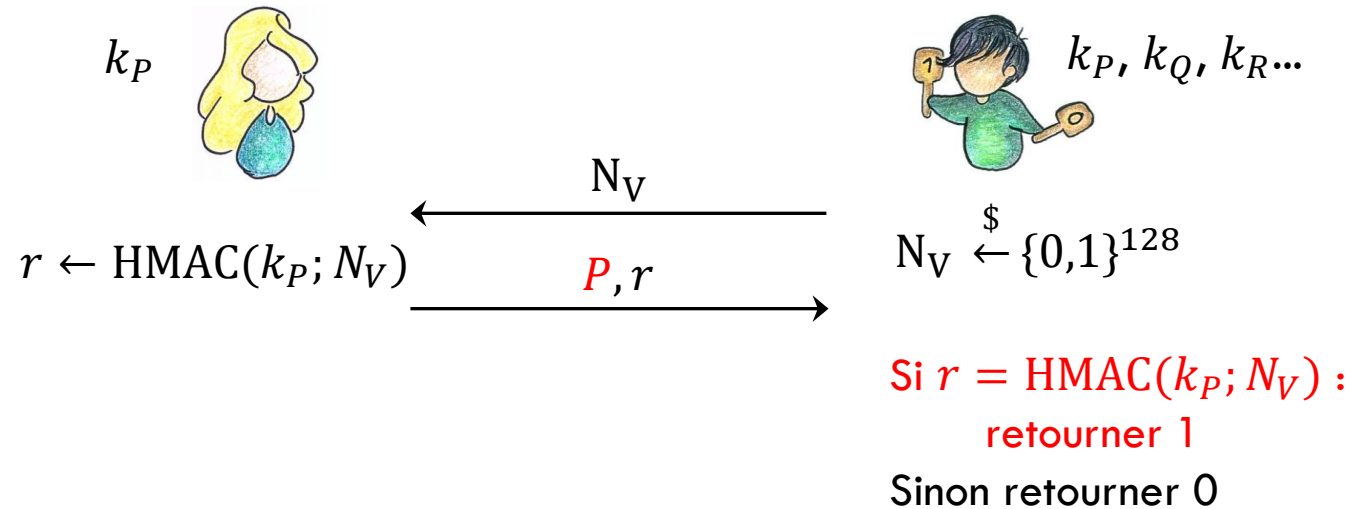
NOTRE PREMIER PROTOCOLE



Si $\exists k$ tel que $r = \text{HMAC}(k; N_V)$:
retourner 1
Sinon retourner 0

- Une attaque de déni de service pour des vérificateurs qui gèrent bcp de prouveurs :
- Juste envoyer des faux r -- car le vérificateur devra chercher parmi toutes les clés pour trouver la bonne !

DEUXIÈME PROTOCOLE



- Maintenant la vérification est plus rapide
- Inconveniant : la protection de la vie privée souffrira



LA VIE PRIVÉE DANS L'AUTHENTIFICATION



PRIVACY, ANONYMAT, INDISTINGUABILITÉ

- Le respect de la vie privée connaît des **dizaines de degrés** [PfitzmannHansen, 2010] :
 - ❖ **L'anonymat** : rester anonyme parmi un ensemble d'entités
 - ❖ **Unlinkability** : ne pas pouvoir lier deux objets d'intérêt appartenant à une même entité
 - ❖ **Non-déteçtabilité** : ne pas savoir si un certain objet d'intérêt appartenant à une certaine entité existe ou non
 - ❖ **Inobservabilité** : non-déteçtabilité par rapport aux externes de l'objet d'intérêt, anonymat pour les internes
 - ❖ **Pseudonymat** : anonymat via l'utilisation d'un pseudonym
 - ❖ ...
- De plus des besoins contradictoires : accountability
 - ❖ Parfois on veut savoir après le coup qui a accédé à un certain service

Quelle serait la notion la plus utile dans l'authentification ?

QUELQUES TYPES D'ADVERSAIRES

➤ Typiquement les adversaires sont des personnes au milieu :

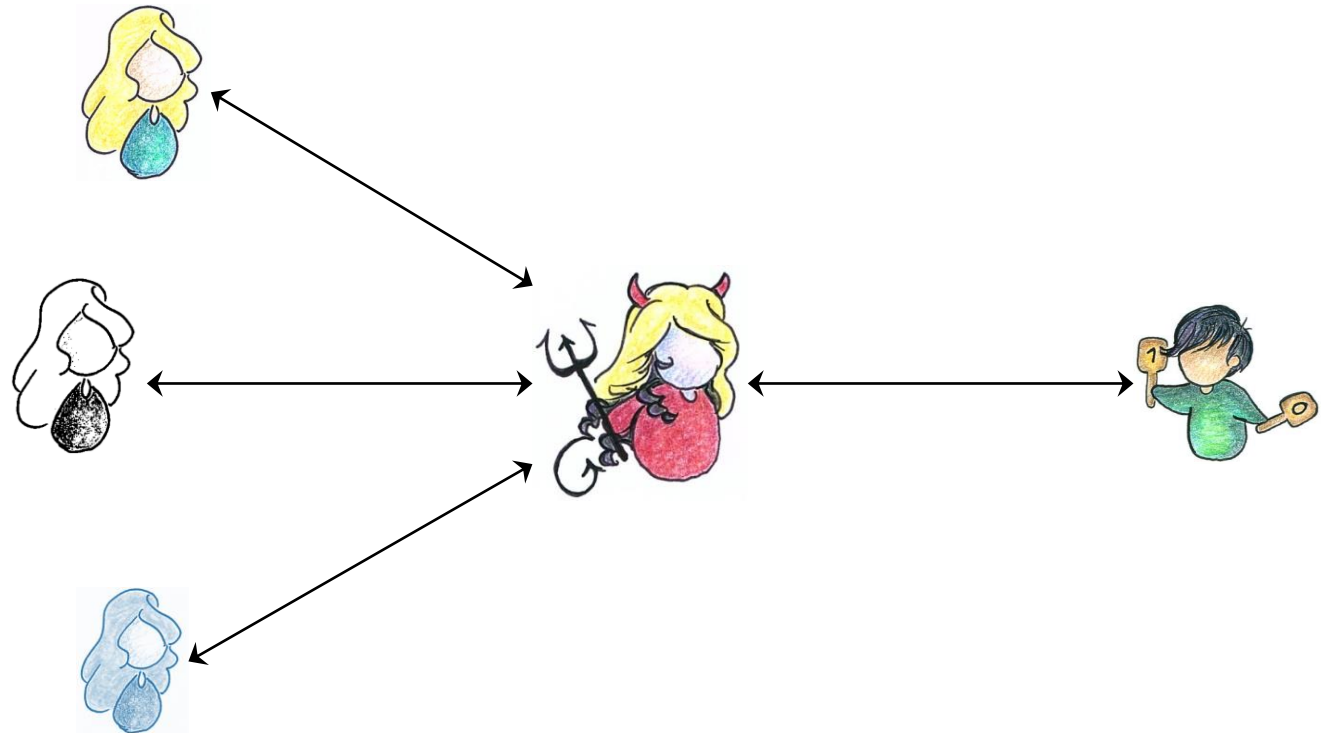
❖ **Passifs** : l'adversaire peut voir les communications mais pas insérer ses propres messages

❖ **Actifs** : l'adversaire peut insérer des messages soit séparément vers une des parties, soit dans les deux directions

➤ Adversaires internes et externes :

❖ **Externes** : Personnes au milieu

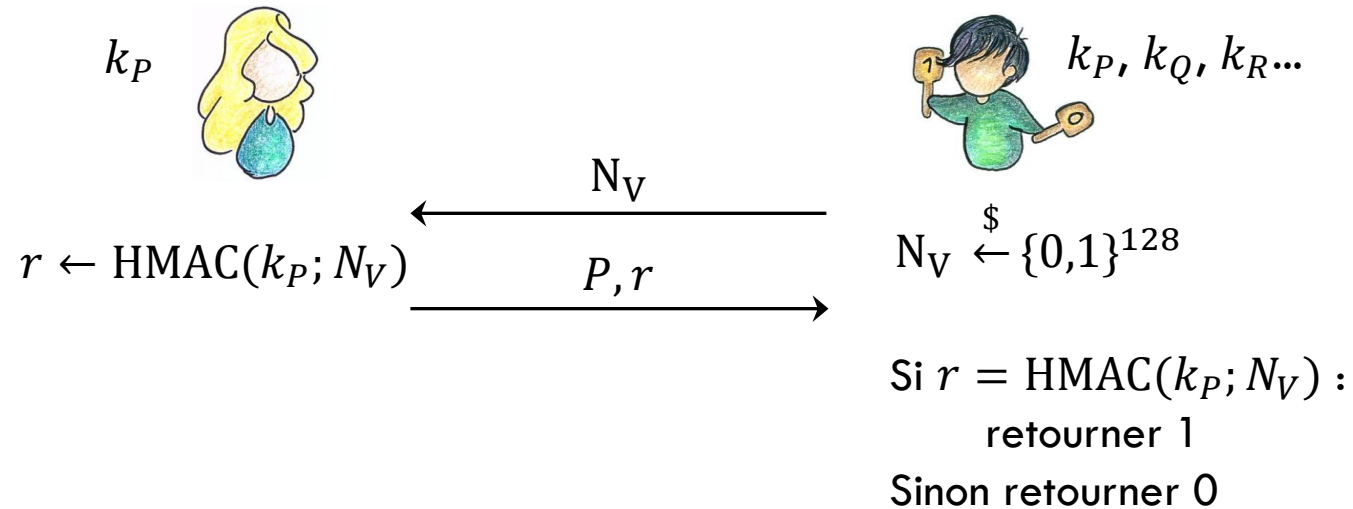
❖ **Internes** : Contrôle le vérificateur aussi



L'ANONYMAT DANS L'AUTHENTIFICATION

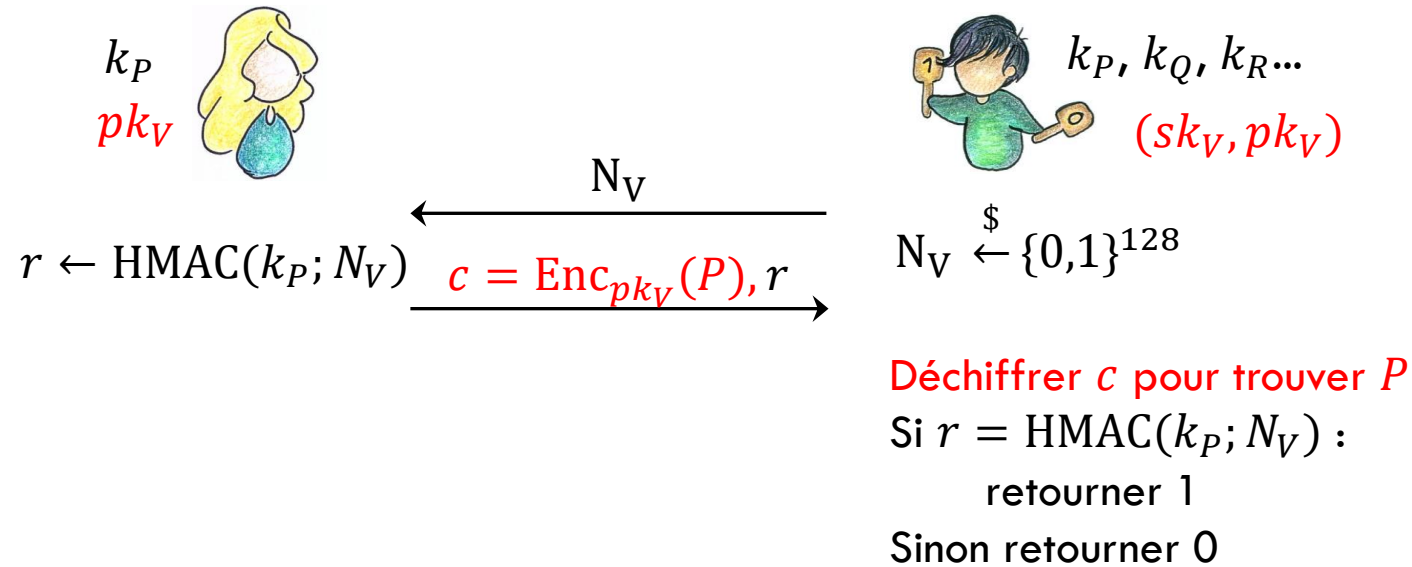
- Prenons un attaquant qui est une Personne au Milieu
- Plusieurs degrés d'anonymat :
 1. Un adversaire externe et passif ne peut pas retrouver l'identités des prouveurs
 2. Un adversaire externe et actif ne peut pas retrouver/distinguer les identités des prouveurs

DEUXIÈME PROTOCOLE



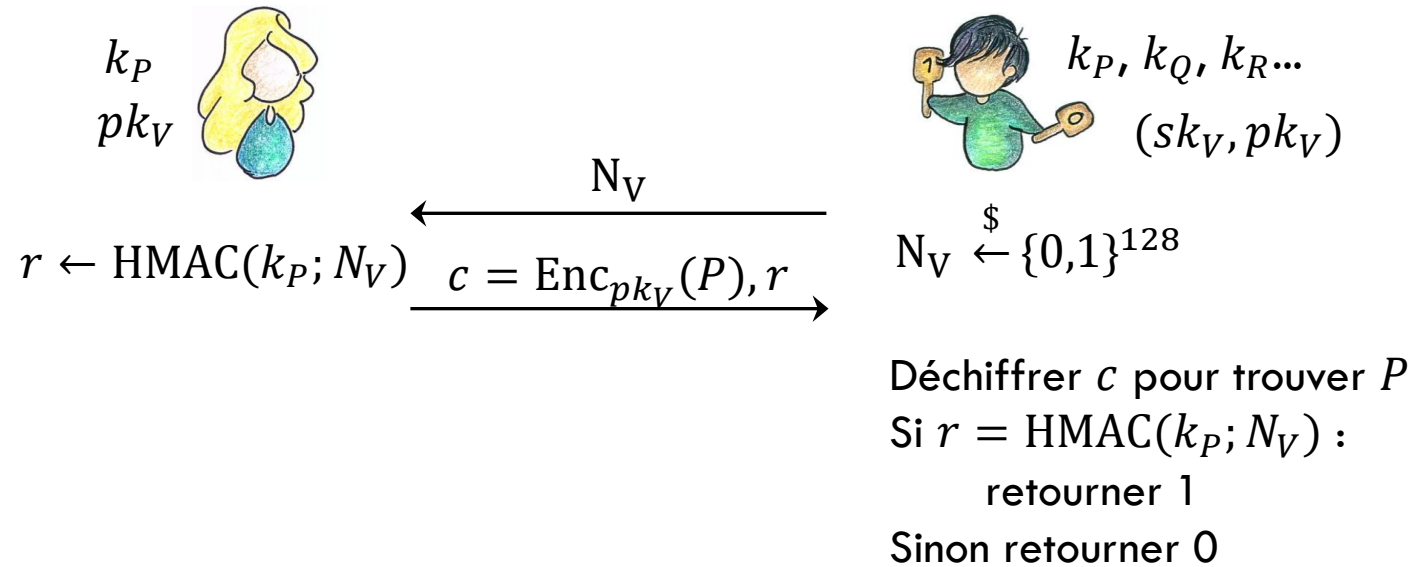
- Ne protège pas du tout la vie privée des prouveurs

TROISIÈME PROTOCOLE



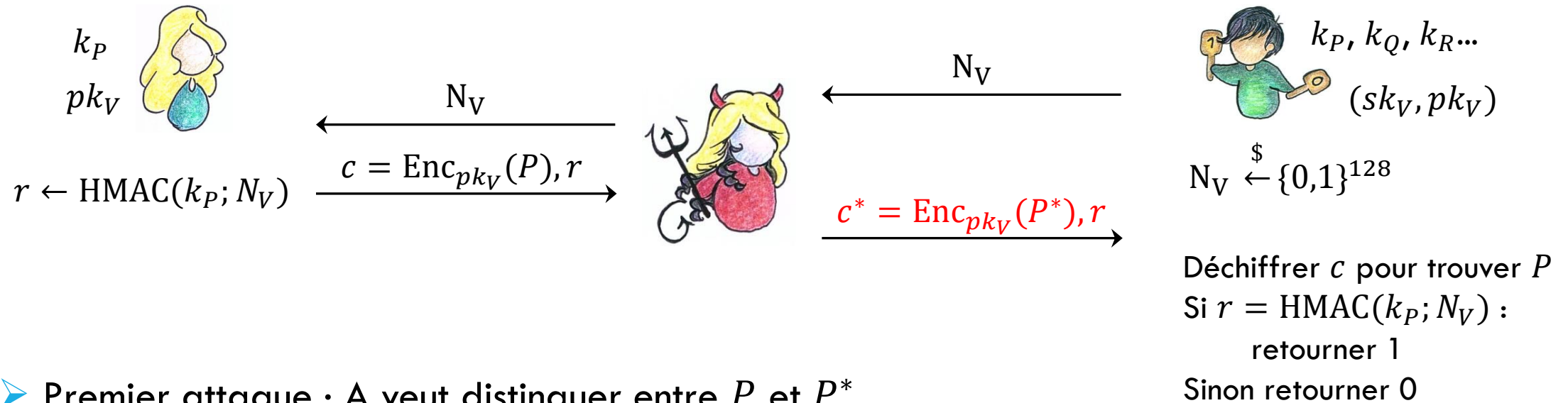
- Protège l'identité des prouveurs contre les attaquants passifs
 - ❖ Car le chiffrement protège la confidentialité des informations envoyées
- Mais ne protège pas ces identités dans la présence d'un attaquant actif

ATTAQUE ACTIF VS LE TROISIÈME PROTOCOLE



- Astuce : tout personne peut chiffrer à clé publique
l'authentification n'est pas incluse dans le chiffrement

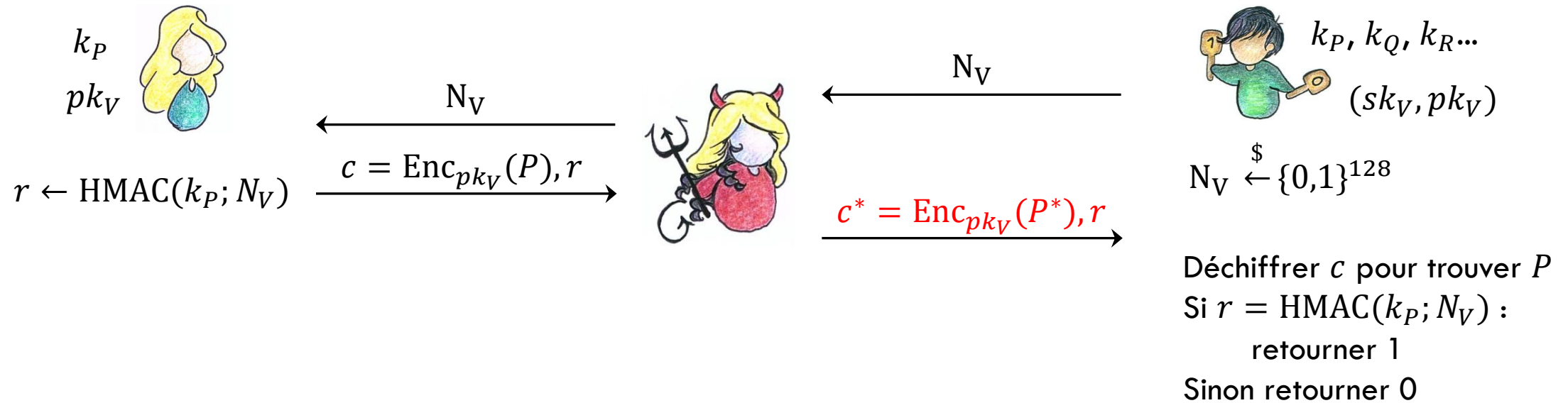
ATTAQUE ACTIF VS LE TROISIÈME PROTOCOLE



➤ Premier attaque : A veut distinguer entre P et P^*

- ❖ Un prouveur inconnu (soit P soit P^*) démarre une session
- ❖ l'attaquant fait suivre le défi vers le prouveur et le prouveur inconnu répond
- ❖ A conserve r et calcule $c^* = \text{Enc}_{pk_V}(P^*)$
- ❖ Il attend la réponse du vérificateur : si la réponse est 1, alors A dit qu'il s'agit de P^* ; si c'est 0, alors A dit P

ATTAQUE ACTIF VS LE TROISIÈME PROTOCOLE



- Extension de l'attaque : A veut savoir quel prouveur parle
 - ❖ Répéter l'attaque de base, en redemandant à chaque fois de l'information au prouveur présent

UNLINKABILITY DES SESSIONS

- Un protocole permettant à l'identité du prouveur de fuir présente un danger clair
- Mais même sans révéler leurs identités, on peut mettre en danger les prouveurs :
 - ❖ Si on peut connecter lier deux exécutions à un même prouveur...
 - ❖ ... et on continue à faire cela dans la durée dans des endroits différents ...
 - ❖ ... on peut reconstituer le chemin d'un utilisateur
- Particulièrement important dans les réseaux véhiculaires, l'authentification mobile...
- Un sujet très riche qui mérite la découverte