La sécurité prouvable

MISES-EN-GAGE, PROTOCOLES SIGMA ET LA DIVULGATION NULLE DE CONNAISSANCES

Cristina Onete
cristina.onete@gmail.com

PROUVER ET AUTHENTIFIER

- Authentification :
  - Un prouveur montre sa **legitimité** envers un vérificateur
  - Différents outils : différentes propriétés

    Signatures : la clé identifie son propriétaire

    MAC : sans non-repudiation, inéfficace à l'échelle

    Une clé : la légitimité de faire une chose

- Aujourd'hui :
  - Beaucoup de cas dans lesquels il faut s'authentifier …
  - Chaque cas : une nouvelle clé
    - Age > 17 ans : peut voter
    - Carte d'étudiant ou de sénior : réduction de tarif
    - …

PROUVER SANS DIVULGUER

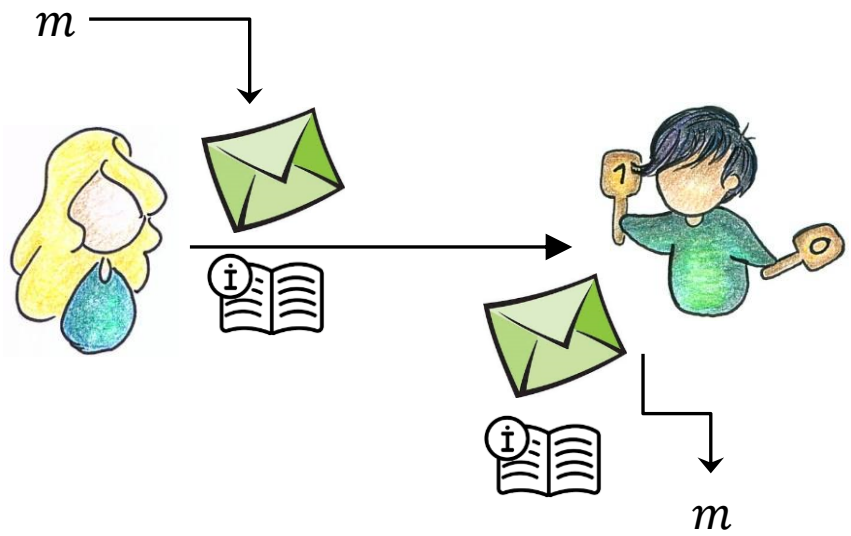
- Se légitimer pour un séjour d'hôtel :
  - Passeport
  - Carte d'identité
  - Carte vitale
  - Permis de conduire
  - …
- Le vérificateur apprend beaucoup d'informations non-nécessaires
- Prouver sans divulguer – possible ?

LES SCHÉMAS DE MISE-EN-GAGE (COMMITMENT SCHEMES)

MISE-EN-GAGE – L'IDÉE

- **Une primitive** (souvent utilisée dans des protocoles) qui permet :
  1. De choisir une valeur secrète m appartennant à un ensemble M
  2. De bâtir une enveloppe pour la contenir (le commitment)
  3. D'ouvrir l'enveloppe sans pouvoir tricher

- Deux propriétés :
  - ✓ **Hiding (Cacher)** : la mise-en-gage cache la valeur engagée
  - ✓ **Binding (Engager)** : la mise-en-gage ne peut s'ouvrir que sur la valeur initialement mise en gage



MISE-EN-GAGE – SYNTAXE

- CScheme $= (\text{Gen}, \text{Commit}, \text{Open})$ pour M tel que :

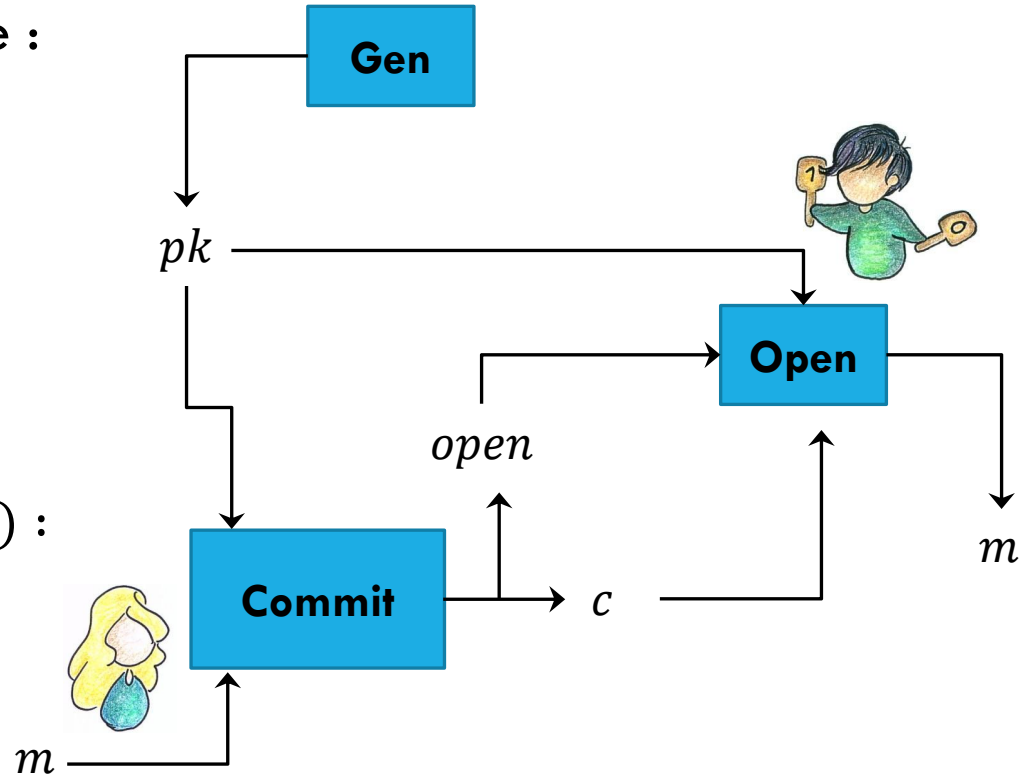
  $\text{Gen}(1^\lambda) \rightarrow pk$

  $\text{Commit}(pk; m) \rightarrow (c, open)$ avec $m \in M$

  $\text{Open}(pk; c, open) \rightarrow m \cup \perp$

- Fonctionnalité :

$$\forall pk \leftarrow \text{Gen}(1^\lambda);\ \ \forall m \in M, (c, open) \leftarrow \text{Commit}(pk; m) :$$
$$\text{Open}(pk; c, open) = m$$

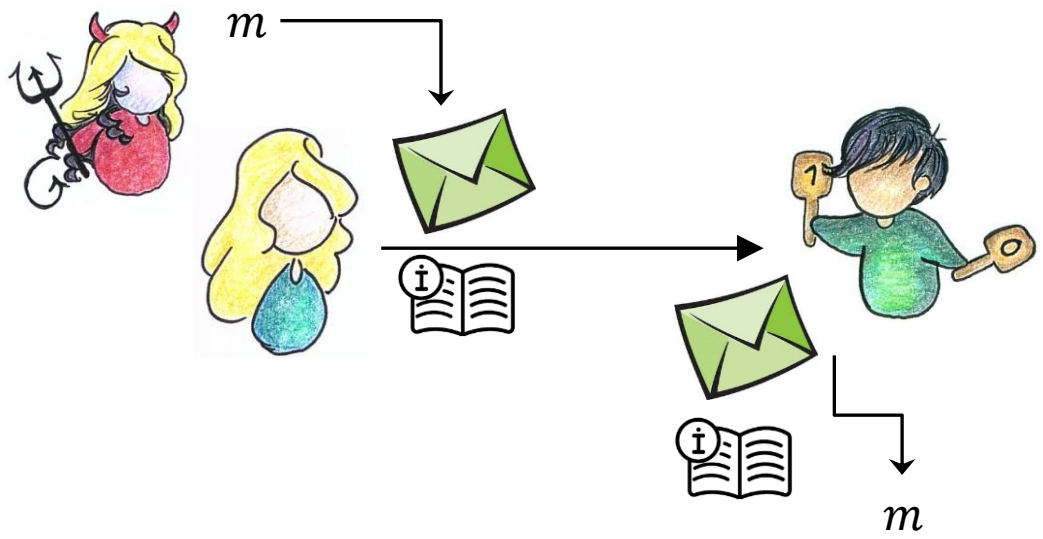


SÉCURITÉ : NOTION DE BINDING

- La mise-en-gage ne peut s'ouvrir que sur la valeur initialement mise en gage

$$pk \leftarrow \mathrm{Gen}(1^{\lambda})$$
$$(c, open_1, open_2) \leftarrow A(\lambda, pk)$$
$$m_1 \leftarrow Open(pk; c, open_1)$$
$$m_2 \leftarrow Open(pk; c, open_2)$$

A gagne ssi. : $\perp \neq m_1 \neq m_2$



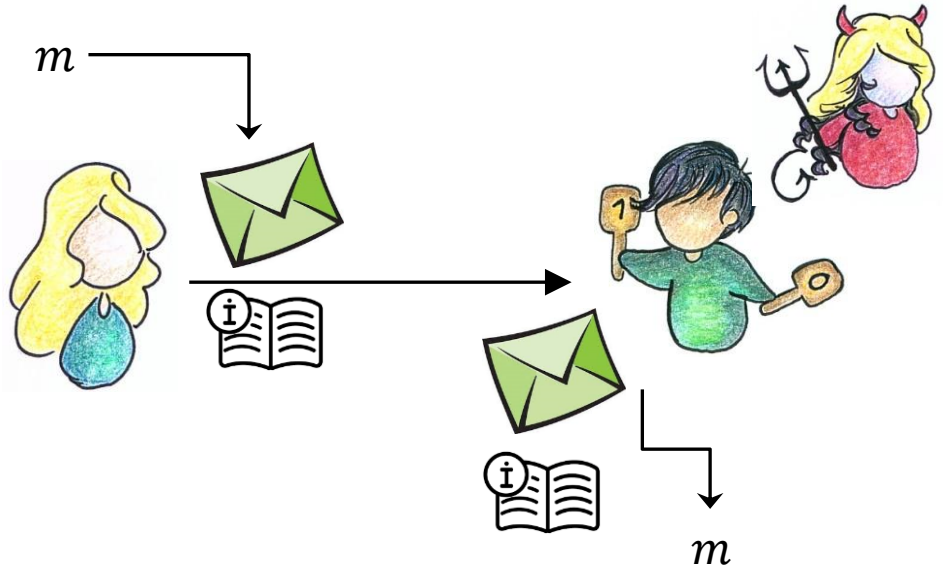
- Quelle est la définition de sécurité ?

SÉCURITÉ : NOTION DE HIDING

- La mise-en-gage cache la valeur mise en gage
  - ✓ Fonctionnalité : avec c et open, on retrouve m
  - ✓ Mais avec c, sans open, impossible de retrouver m

$$pk \leftarrow \mathrm{Gen}(1^{\lambda})$$
$$b \xleftarrow{\$} \{0,1\}$$
$$(m_0, m_1) \leftarrow A(\lambda, pk)$$
$$(c_b, open_b) \leftarrow \mathrm{Commit}(pk; m_b, r)$$
$$d \leftarrow A(\lambda, pk, m_0, m_1, c_b)$$

A gagne ssi. : $m_0, m_1 \in M$ et $b = d$



Peut-on définir la sécurité de la même façon que pour la notion de binding ?

FONCTIONS DE HACHAGE & LES MISES-EN-GAGE

- Prenons une famille de fonctions de hachage $H: K \times \{0,1\}^* \rightarrow \{0,1\}^{\ell}$
- Instancions le schéma d'engagement pour M avec $|M| \leq 2^{\ell}$:

  $\mathrm{Gen}(1^{\lambda})$: tirer aléatoirement : $k \xleftarrow{\$} K$

  $\mathrm{Commit}(k; m)$: calculer $c \leftarrow H_k(m)$ et $open \leftarrow m$

  $\mathrm{Open}(k; c, open)$: vérifier $c = H_k(open)$

- Supposons que la fonction de hachage résiste aux collisions. Quelles propriétés pour le schéma d'engagement ?

PKE & LES MISES-EN-GAGE

- Soit un schéma PKE = (KGen, Enc, Dec) qui est ϵ-IND-CPA secure
  - Avec $Enc(\mathrm{pk}; \mathrm{m})$ probabiliste et $Enc(\mathrm{pk}; \mathrm{m}, \mathrm{r})$ déterministe (l'aléa décide la valeur de c)

- Instanciation schéma de mise-en-gage :

  $\mathrm{Gen}(1^\lambda)$: exécuter : $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathrm{KGen}(1^\lambda)$, garder en output pk, jeter sk

  $\mathrm{Commit}(pk; m)$: choisir r aléatoirement, calculer $c \leftarrow \mathrm{Enc}\ (pk;\ m, r)$ et $open \leftarrow (m, r)$

  $\mathrm{Open}(k; c, open)$: calculer $\hat{c} = \mathrm{Enc}(pk; open)$, vérifier $\hat{c} = c$.

PKE & LES MISES-EN-GAGE

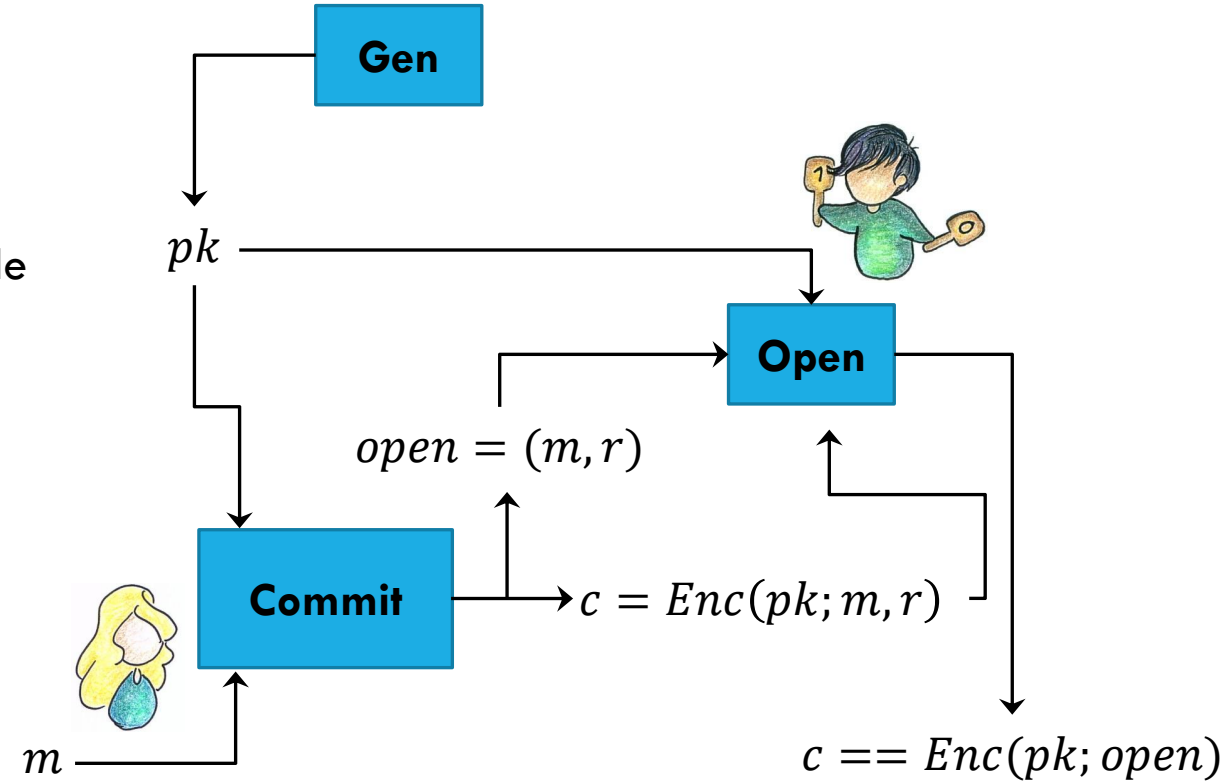
- Binding :
  - L'adversaire doit trouver $(m_1, r_1) \neq (m_2, r_2)$ t.q. :

$$c = Enc(pk; m_1, r_1) = Enc(pk; m_2, r_2)$$

- 2 cas :
  1. $m_1 \neq m_2$: Si on peut chiffrer deux messages dans le même texte chiffré (la différence est l'aléa), alors le déchiffrement est ambigue

     (il manque une étape du raisonnement – laquelle ?)
  2. $m_1 = m_2, r_1 \neq r_2$: Si on peut arrive au même texte chiffré avec un seul m et deux aléas possibles. Mais, on avait supposé que l'aléa rend le chiffrement déterministe… Donc non.

- Et donc le schéma est binding.

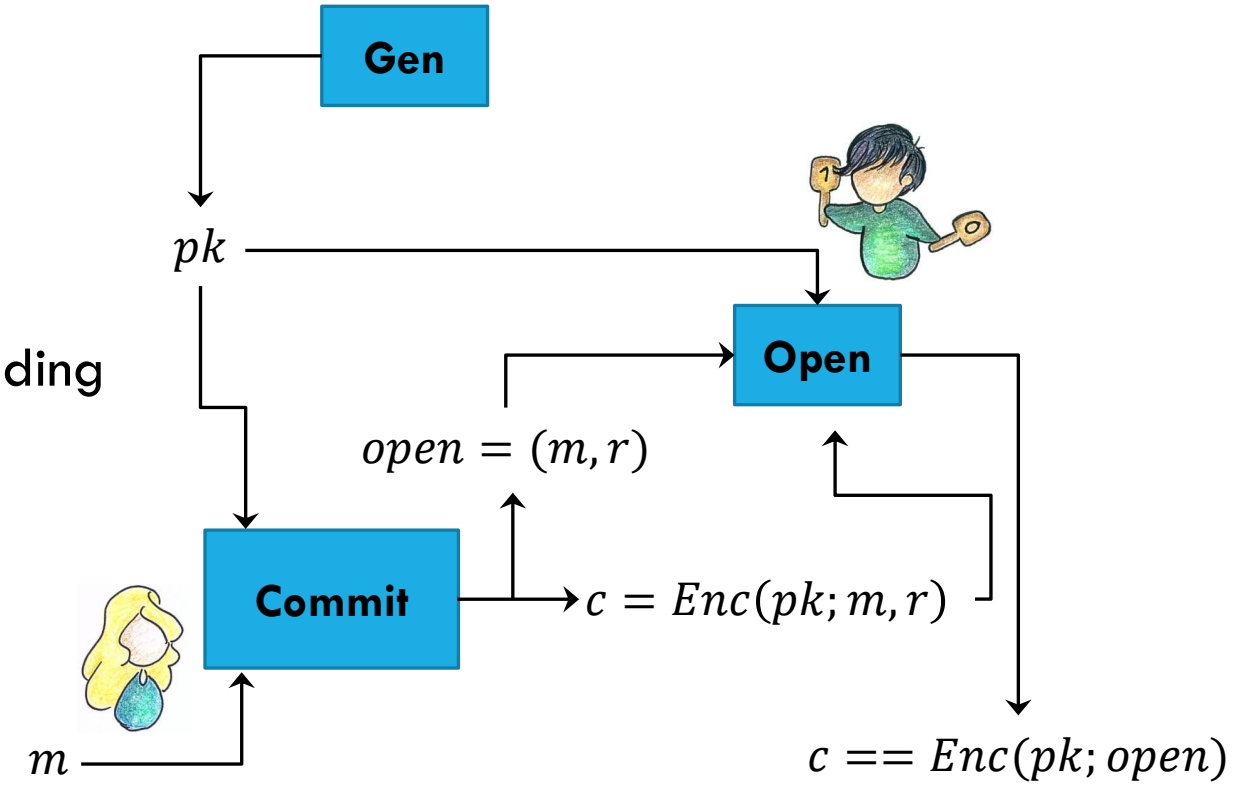


PKE & LES MISES-EN-GAGE

- Hiding :
  - L'adversaire doit distinguer entre $\mathrm{Enc}(pk; m_0, r) = \mathrm{Enc}(pk; m_1, r)$

- Si PKE est ϵ-IND-CPA, alors CScheme est ϵ-hiding

Preuve ????



LES PROTOCOLES SIGMA

UNE AUTRE FORME D'AUTHENTIFICATION

- Départ : une relation NP R pour des entrées $(x, y) \in X \times Y$:
  - **Vérification efficace** : étant données $(x, y) \in X \times Y$, on peut vérifier en temps polynomial si $(x, y) \in R$
  - **Succinct** : il existe un polynomial tel que $|y| < p(|x|)$

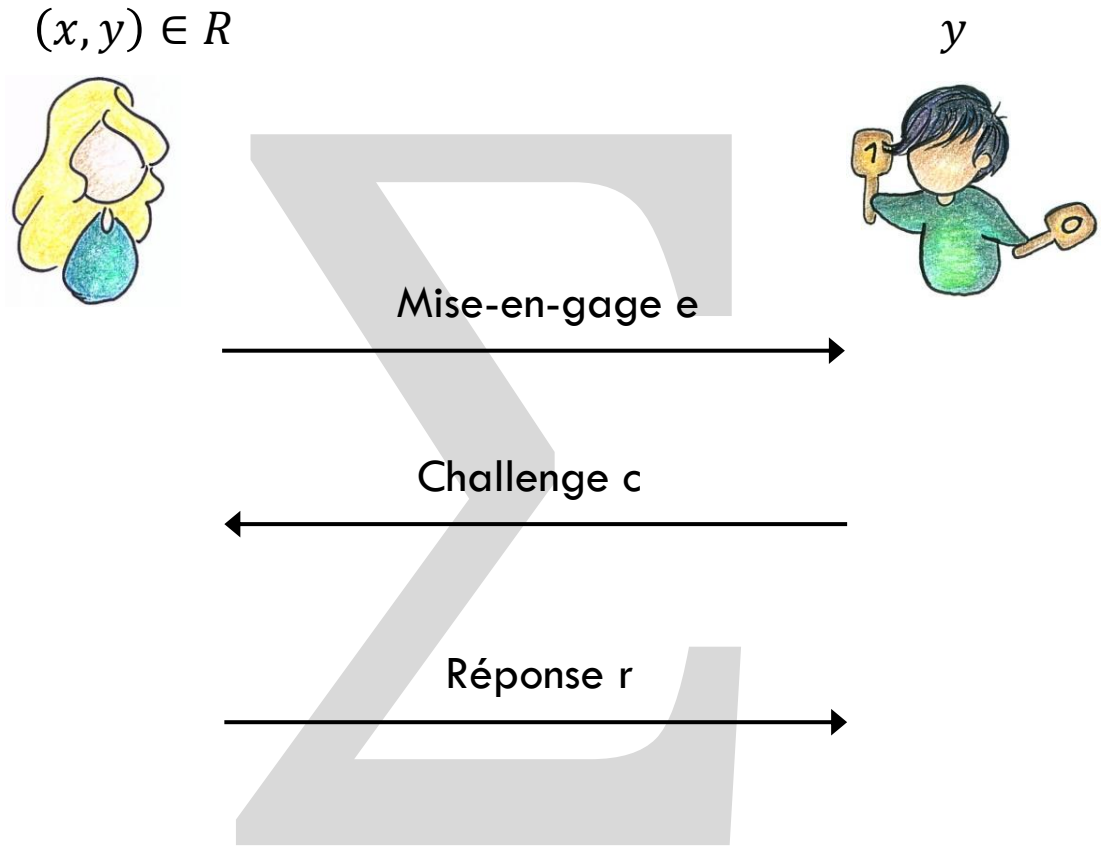
- Protocole Sigma : un prouveur et un vérificateur partagent y et R
  - ✓ Le prouveur veut prouver sa légitimité : notamment qu'il possède un x t.q. $(x, y) \in R$
  - ✓ Structure particulière : 3 messages – mise-en-gage challenge, response
  - ✓ Un protocole Sigma doit protéger la privacy de l'entrée x, tout en garantissant l'authentification
  - ✓ S'il existent plusieurs x valides pour un certain y, nous aurons une notion de privacy

PROTOCOLE SIGMA

- ✓ Le prouveur commence avec une mise-en-gage e
- ✓ Le vérificateur choisit une challenge aléatoire
- ✓ Le prouveur envoie une réponse, et le vérificateur devra ensuite décider la légitimité du prouveur

➢ Protocole : P(x,y) ↔ V(y) output :

1 ssi. P légitime, ou 0 sinon

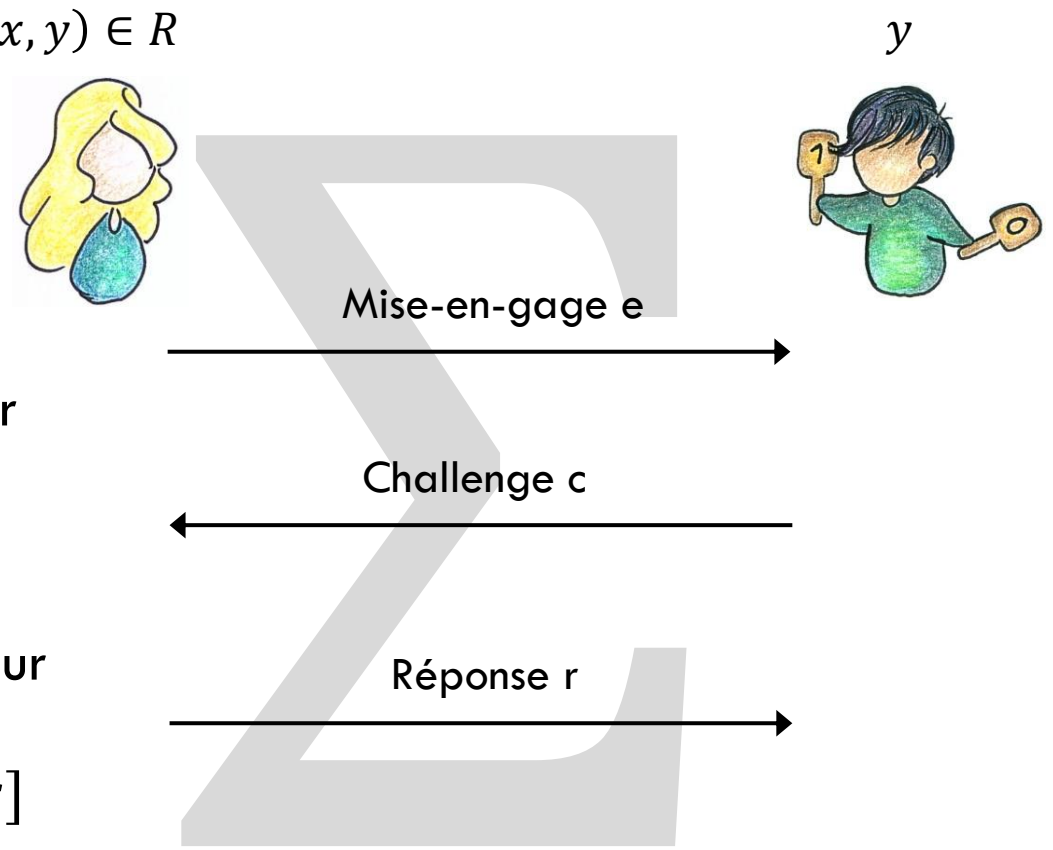


SÉCURITÉ PROTOCOLE SIGMA

- **Fonctionnalité** : $\forall\ (x, y) \in X \times Y$ t.q. $(x, y) \in R$:
$$\mathrm{Out}\big(P(x, y) \leftarrow V(y)\big) = 1$$

- **Special soundness** : Pour R et y, il existe un extracteur polynomial Ext, qui, à partir de (e, c_0, r_0) et (e, c_1, r_1) valides avec $c_0 \neq c_1$, trouve $x \in X$ t.q. $(x, y) \in R$

- **Honest-verifier Zero-Knowledge** : il existe un simulateur polynomial Sim, qui aura en entrée $(y, \hat{c} \leftarrow C)$ t.q. :
$$Sim(y, \hat{c}) \approx_c \mathrm{Dist}[(e, c, r) : P(x, y) \rightarrow (e, r); V \rightarrow c]$$



LE PROTOCOLE SCHNORR (LOG DISCRET)

- **Contexte** : groupe $G = \langle g \rangle$ de taille q, gén. G
- $(x, y) \in R$ ssi. pour $y = (q, g, p, G = \langle g \rangle . h)$,
  on a $h = g^x \bmod p$

$x \in \{1, q-1\}$
$y = (q, g, p, G = \langle g \rangle . h = g^x)$

y

Mise-en-gage e

- Prouveur : choisit z , calcule $e = g^z \bmod p$
- Vérificateur : choisit $c \in \{0,1\}^t$

Challenge c

- Prouveur : calcule $r = (z + cx) \bmod q$
- Vérificateur : vérifie $g^r == eh^c \bmod p$

Réponse r

SÉCURITÉ PROTOCOLE SCHNORR

- **Fonctionnalité** : Supposons : G, p, q, g t.q. indiqués, et $h = g^x \bmod p$

  exécution honnête : $g^r = g^{z+cx} = g^z g^{cx} = eh^c \bmod p$

- **Sp. Sound** : Supposons : (e, c_0, r_0) et $(e, c_1 \neq c_0, r_1)$ valides :

$r_0 = z + c_0 x$ $\qquad$ $r_1 = z + c_1 x$

$$x = \frac{r_0 - r_1}{c_0 - c_1}$$

$x \in \{1, q-1\}$

$y = (q, g, p, G = \langle g \rangle . h = g^x)$

y

$e = g^z \bmod p$ →

$c \leftarrow \{0,1\}^t$ ←

$r = z + cx \bmod q$ →

$g^r == eh^c \bmod p$

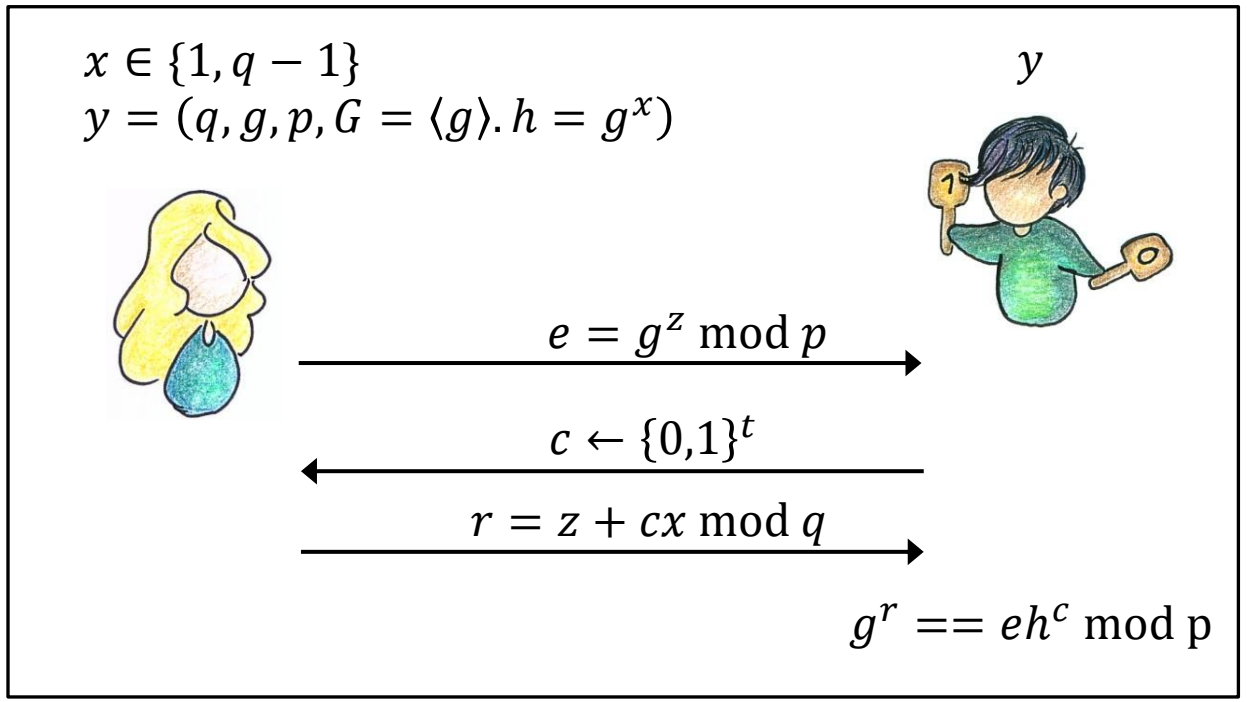
SÉCURITÉ PROTOCOLE SCHNORR

- HVZK : Le simulateur a y et c aléatoire , mais pas x . Comment générer un transcript valide ?

  Typiquement reverse engineering :

  * choisir r aléatoirement,

  * calculer : $e = g^r h^{-c} \bmod p$



LE PROTOCOLE GUILLOU-QUISQUATER (RSA)

- **Contexte** : module RSA $n = pq$, pk, $y = (n, a, sk^a \ mod \ n)$
- $(x, y) \in R$ ssi. pour $x = sk$ ssi. $sk \cdot pk = 1 \ mod \ \varphi(n)$

- Prouveur : choisit z , calcule $e = z^a \bmod p$
- Vérificateur : choisit $c \in \{1, \dots n-1\}$
- Prouveur : calcule $r = z \cdot sk^c \ mod \ n$
- Vérificateur : vérifie $r^a = e \cdot sk^{a\,c}$

Preuve ????