Le calcul sécurisé, cinquième cours

Calcul multipartite sécurisé : circuits brouillés et transfert inconscient

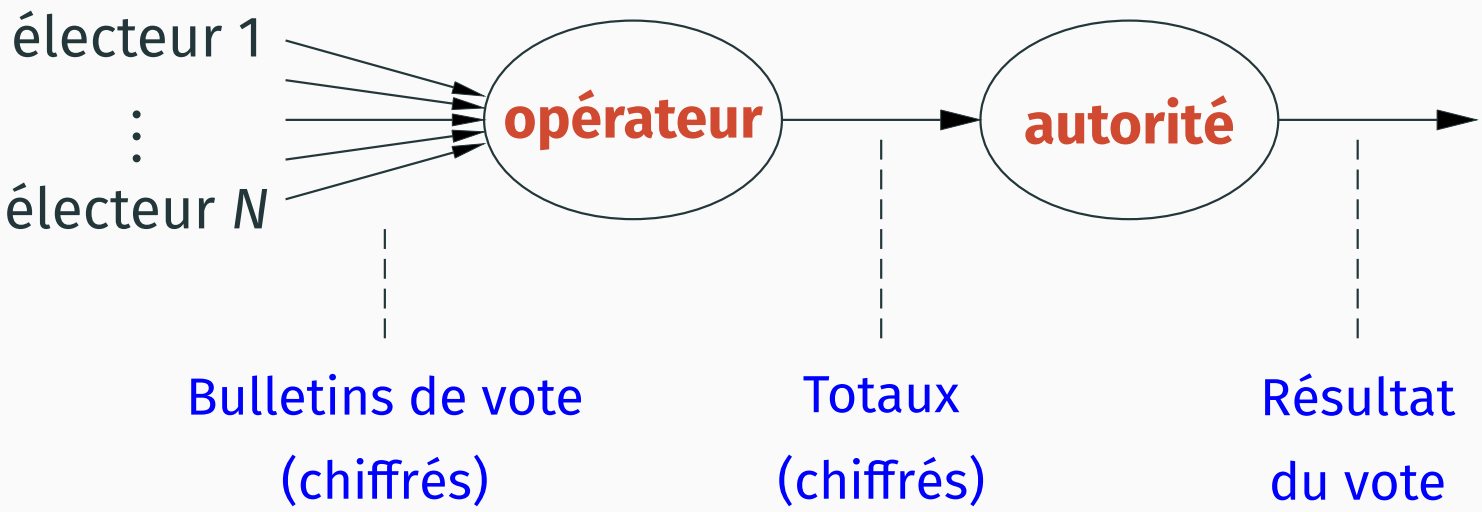
Xavier Leroy

2025-12-04

Collège de France, chaire de sciences du logiciel
xavier.leroy@college-de-france.fr

Partage de clés et déchiffrement à seuil

Dépouillement simplifié d'une élection électronique (rappel)



Une paire de clés pk, sk pour un chiffrement semi-homomorphe.

Les bulletins sont chiffrés avec pk.

Les bulletins sont comptabilisés par addition homomorphe.

Le total doit être déchiffré avec la clé sk.

On voudrait que la clé sk soit partagée entre n trustees, de sorte que k trustees puissent déchiffrer le total.

Un algorithme multipartite naïf

On partage la clé secrète sk entre les n trustees avec un partage de Shamir (polynômes de degré $t = k - 1$).

Lorsque les bulletins sont comptabilisés, les trustees révèlent leurs parts, reconstruisent sk, et déchiffrent le total.

Problèmes :

- Tout trustee peut déchiffrer tous les bulletins.
- La clé secrète ne peut pas être réutilisée pour un autre vote.

Le chiffrement d'ElGamal (rappel)

Un groupe fini $(G, \cdot)$ d'ordre q engendré par g.

Clé secrète : $s \in \{1, \ldots, q-1\}$.

Clé publique : $h \stackrel{def}{=} g^s$.

Chiffrement :

$$\mathcal{E}_h(m) = (g^r, h^r \cdot m) \quad \text{avec } r \in \{1, \ldots, q-1\} \text{ aléatoire}$$

Déchiffrement :

$$\mathcal{D}_s((a, b)) = b/a^s$$

Déchiffrement multipartite

Supposons un partage additif complet de la clé secrète s entre les n participants :

$$s = s_1 + \cdots + s_n \pmod{q}$$

Pour déchiffrer conjointement (a, b)

- Chaque participant i calcule $y_i = a^{s_i}$ et l'envoie aux autres.
- Un ou plusieurs participants calculent $y = y_1 \cdots y_n$, puis b/y.

On a bien déchiffré le message, puisque

$$y = a^{s_1} \cdots a^{s_n} = a^{s_1 + \cdots + s_n} = a^s$$

La clé secrète s n'est pas révélée, uniquement un partage multiplicatif de $y = a^s$.

Déchiffrement multipartite à seuil

Si on a un partage de Shamir, ou un partage linéaire (LSSS) général, la clé secrète est une combinaison linéaire des parts :

$$s = \lambda_1 s_1 + \cdots + \lambda_n s_n \pmod{q}$$

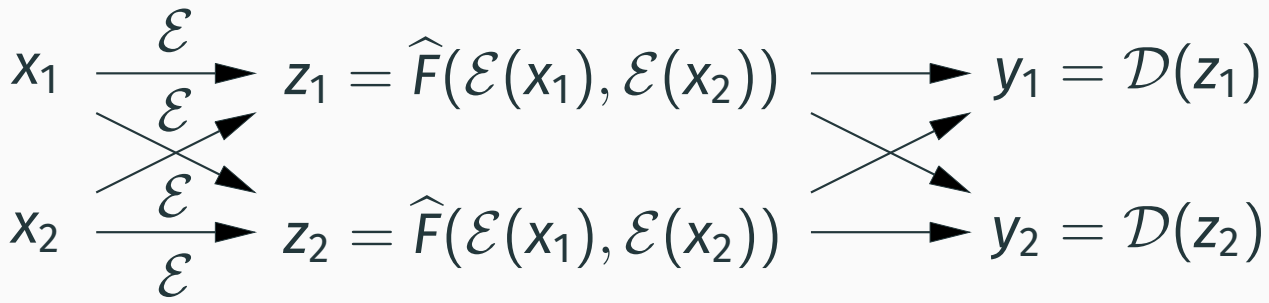
où certains λ_i peuvent être égaux à 0 si on n'a pas besoin des parts correspondantes. ($\Rightarrow 4^{\text{e}}$ cours)

Dans ce cas, chaque participant i calcule $y_i = a^{\lambda_i s_i}$, et on a bien

$$y = y_1 \cdots y_n = a^{\lambda_1 s_1} \cdots a^{\lambda_n s_n} = a^{\lambda_1 s_1 + \cdots + \lambda_n s_n} = a^s$$

Calcul multipartite et chiffrement homomorphe

On peut réaliser un calcul multipartite $y = F(x_1, \ldots, x_n)$ par évaluation homomorphe du circuit F.

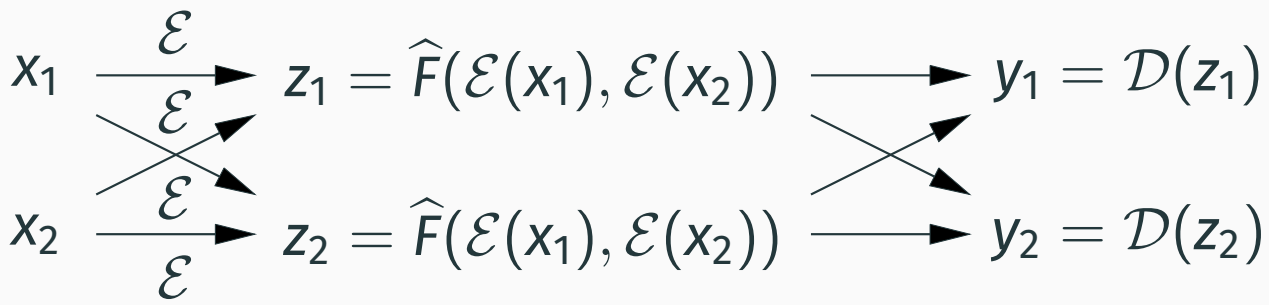


On suppose la clé secrète sk partagée entre les participants, et la clé publique pk connue de tous.

Chaque participant i envoie son secret chiffré $\mathcal{E}_{pk}(x_i)$ aux autres participants.

Calcul multipartite et chiffrement homomorphe

On peut réaliser un calcul multipartite $y = F(x_1, \ldots, x_n)$ par évaluation homomorphe du circuit F.



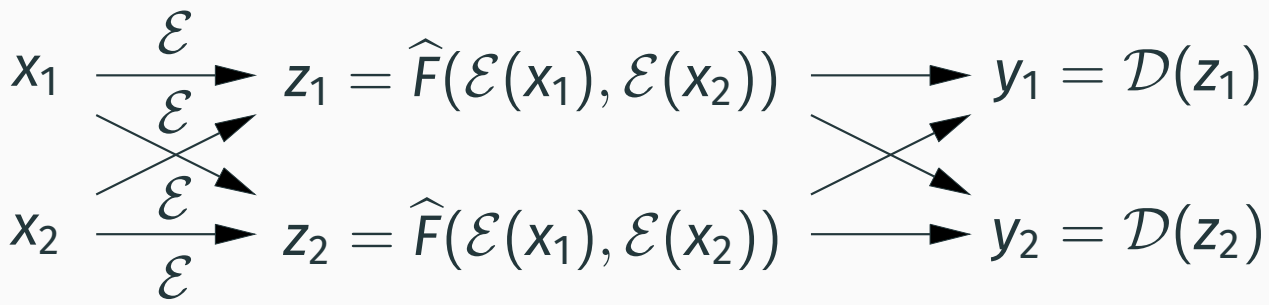
Chaque participant i calcule

$$z_i = \widehat{F}(\mathcal{E}_{pk}(x_1), \ldots, \mathcal{E}_{pk}(x_n))$$

où $\widehat{F}$ est l'évaluation homomorphe de F.

Calcul multipartite et chiffrement homomorphe

On peut réaliser un calcul multipartite $y = F(x_1, \ldots, x_n)$ par évaluation homomorphe du circuit F.



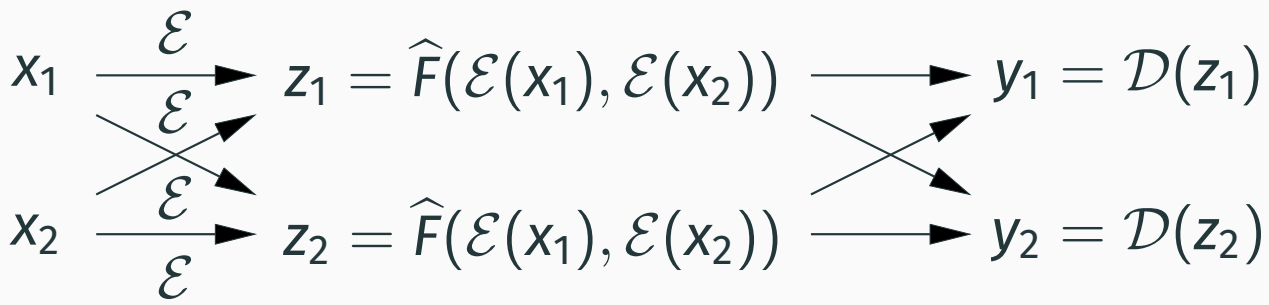
Tous les participants coopèrent pour déchiffrer les z_i :

$$y_i = \mathcal{D}_{sk}(z_i) \qquad \text{(sans révéler } sk\text{)}$$

et vérifient que $y_1 = \ldots = y_n$.

Calcul multipartite et chiffrement homomorphe

On peut réaliser un calcul multipartite $y = F(x_1, \ldots, x_n)$ par évaluation homomorphe du circuit F.



Point fort : le nombre de rounds de communication est indépendant de la profondeur multiplicative de F.

Point faible : l'évaluation homomorphe est coûteuse en temps CPU.

Circuits brouillés
(*Yao's garbled circuits*)

Le problème des millionaires de Yao

(Andrew C. Yao, *Protocols for Secure Computation*, SFCS 1982.)

Alice et Bob souhaitent savoir lequel des deux est le plus riche, sans révéler le montant de sa fortune à l'autre.

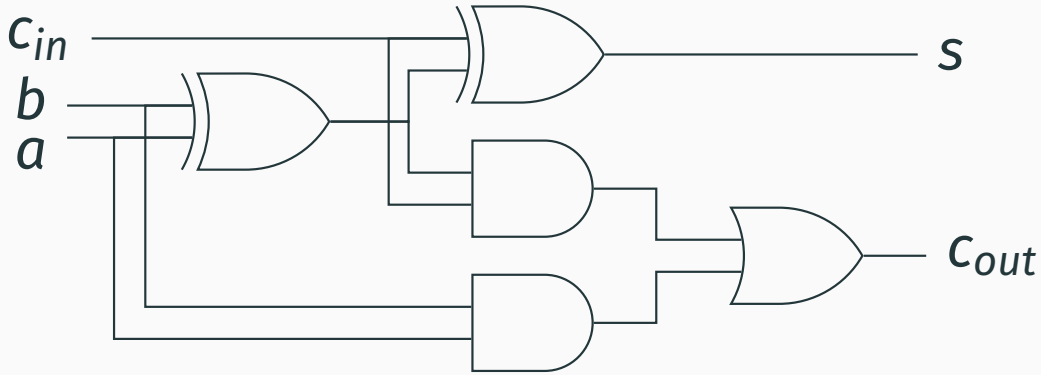
Une version bipartite du problème de l'appel d'offre.

Formellement : calculer le booléen $a \geq b$ en gardant a et b secrets.

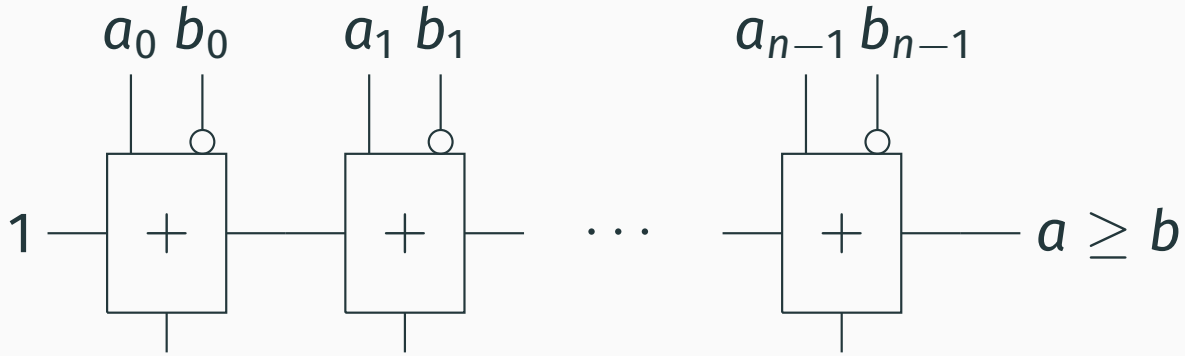
(Variante : le problème des millionaires socialistes, où il faut déterminer si $a = b$.)

Un circuit booléen pour la comparaison

L'additionneur complet :



Le comparateur n bits :



Évaluation bipartite sécurisée du circuit

En utilisant un des protocoles de partage de secrets du 4e cours, p.ex. le protocole GMW.

- Alice écrit sa fortune en binaire $A = \sum a_i 2^i$ et partage les bits $a_0, \ldots, a_{39}$ avec Bob.
- Bob écrit sa fortune en binaire $B = \sum a_b 2^i$ et partage les bits $b_0, \ldots, b_{39}$ avec Alice.
- Alice et Bob évaluent ensemble le circuit de comparaison.
- Une fois calculé le partage $[c]$ du résultat, Alice et Bob le révèlent, obtenant c.

Problème potentiel : le nombre de communications pendant le calcul (3 multiplications par bit $\rightarrow$ au moins 120 communications).

Les circuits brouillés de Yao

$$c = F(a, b)$$

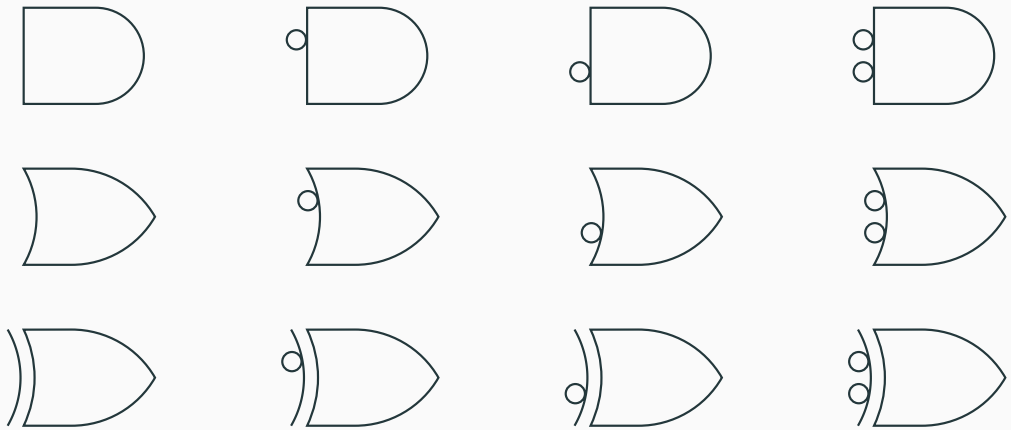
a : données privées d'Alice
b : données privées de Bob
c : résultats partagés

Une alternative dissymétrique au partage de secrets :

1. Alice prépare une version ≪brouillée≫ du circuit F et l'envoie à Bob, ainsi que ses secrets a brouillés.
2. Bob brouille ses secrets b à l'aide de transferts inconscients avec Alice.
3. Bob évalue le circuit brouillé, obtenant $c = F(a, b)$ brouillé. (Évaluation purement locale ; aucune communication.)
4. Bob envoie ce résultat à Alice, qui le dé-brouille et annonce c publiquement.

Les portes logiques utilisées

On considère les portes AND, OR, XOR, avec une négation possible sur chaque entrée :



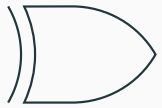
Plus besoin de porte NOT : la négation se fait sur l'entrée de la porte suivante.

Représentation par des tables de vérité

Chaque porte F peut être représentée par sa table de vérité :

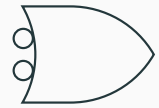
0	0	valeur de $F(0,0)$
0	1	valeur de $F(0,1)$
1	0	valeur de $F(1,0)$
1	1	valeur de $F(1,1)$

Exemples :



0	0	0
0	1	1
1	0	1
1	1	0

0	0	0
0	1	1
1	0	0
1	1	0



0	0	1
0	1	1
1	0	1
1	1	0

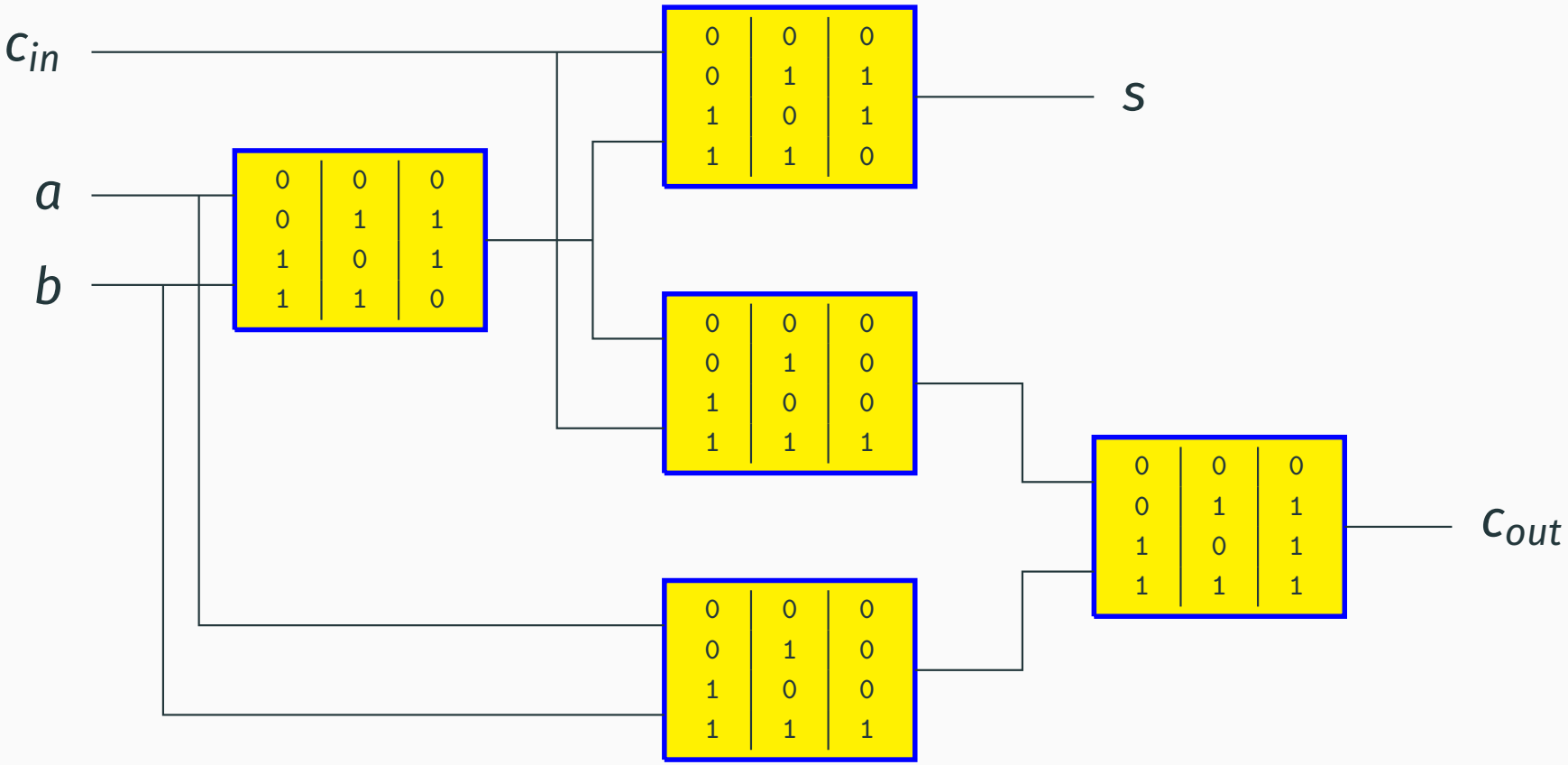
Brouillage des fils

Pour chaque fil w du circuit, Alice choisit deux vecteurs de bits w_0 représentant le bit 0 et w_1 représentant le bit 1.

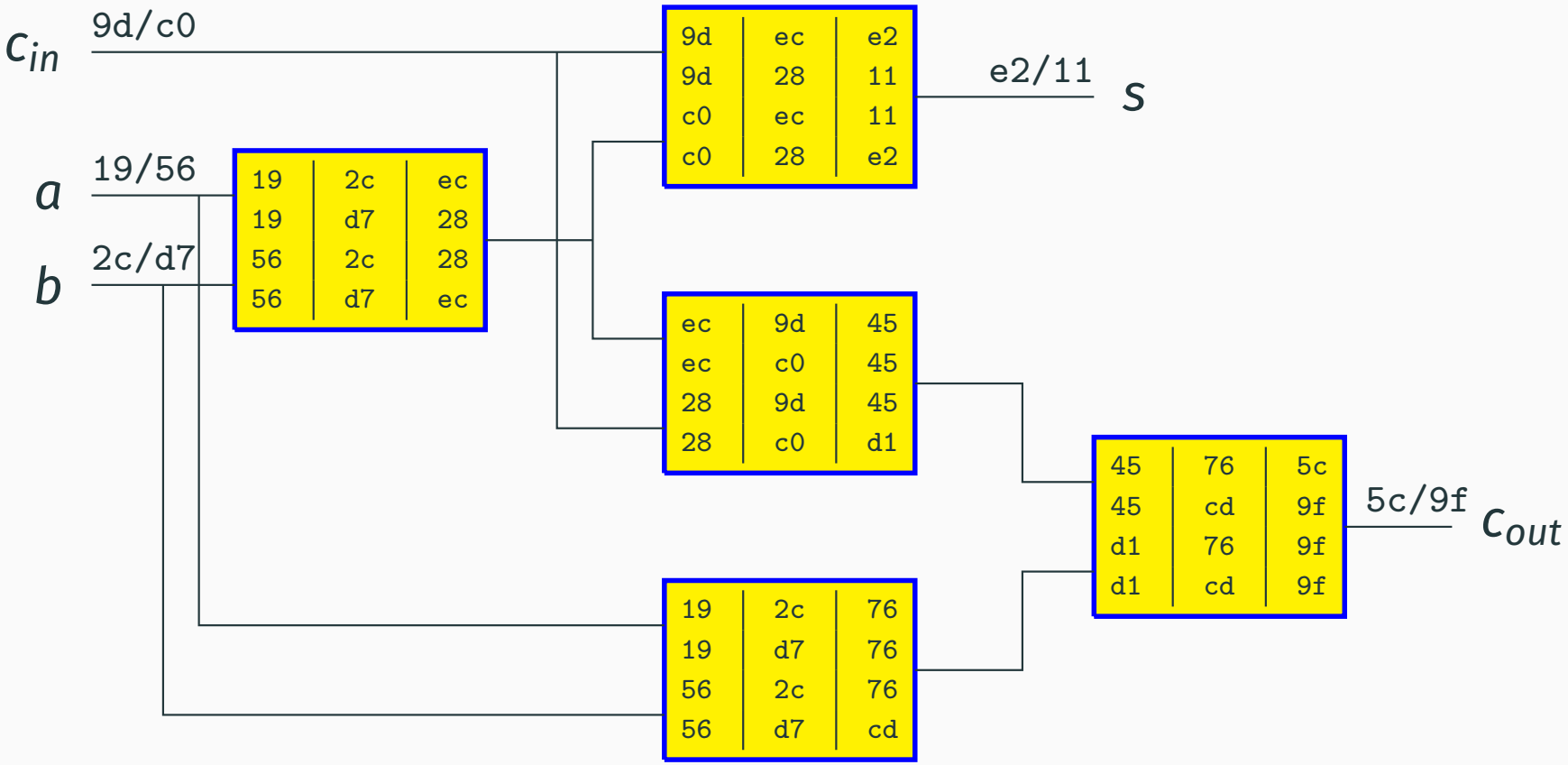
(Chaque w_b est la moitié d'une clé de chiffrement symétrique, i.e. un entier 128 bits dans le cas d'AES-256.)

Elle réécrit les tables de vérité en conséquence.

Exemple : brouillage des fils de l'additionneur complet



Exemple : brouillage des fils de l'additionneur complet



Chiffrement des portes logiques

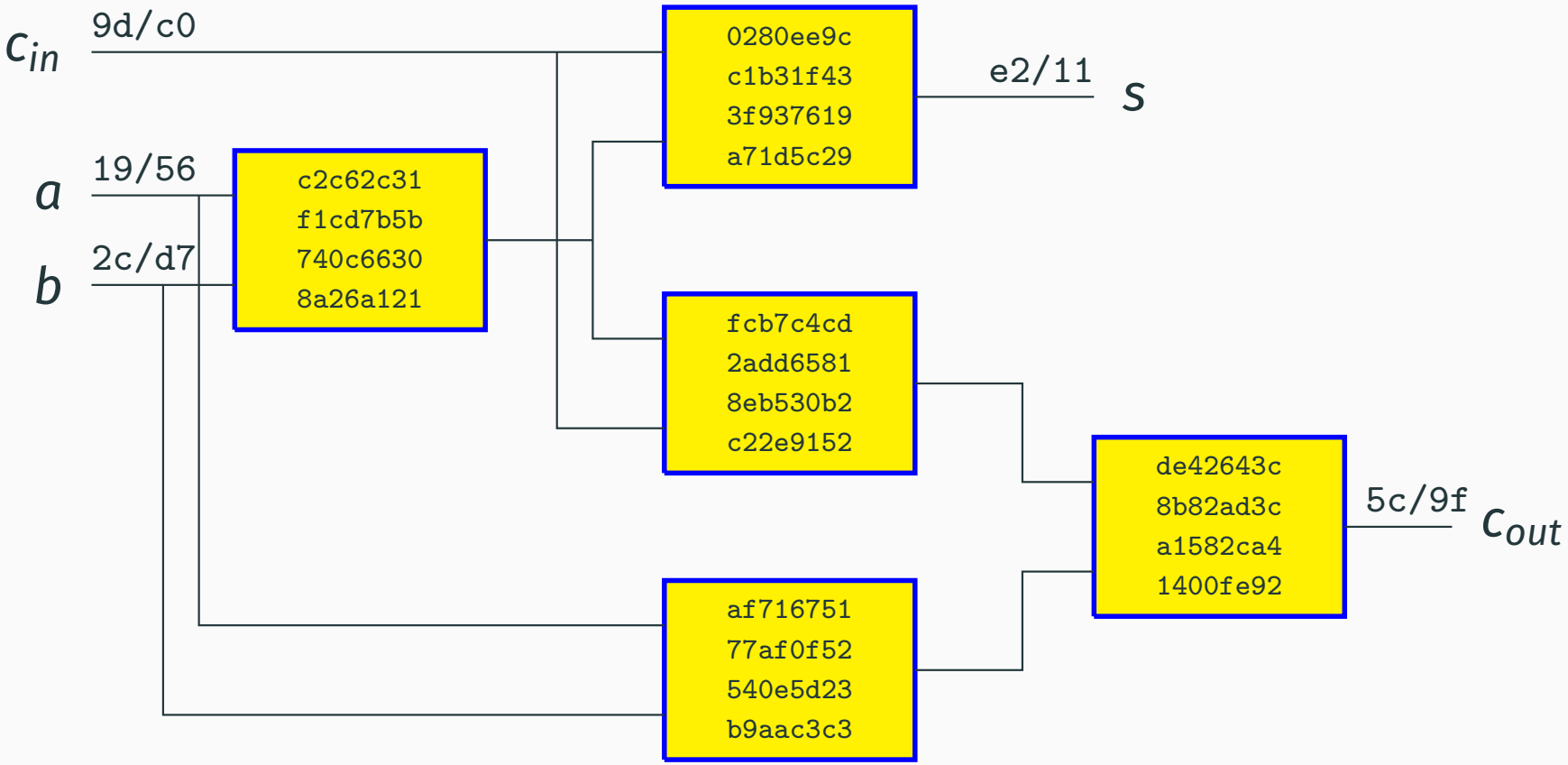
Pour la porte logique numéro g d'entrées a, b et de sortie c :

$$\begin{array}{c|c|c} a_0 & b_0 & c_{F(0,0)} \\ a_0 & b_1 & c_{F(0,1)} \\ a_1 & b_0 & c_{F(1,0)} \\ a_1 & b_1 & c_{F(1,1)} \end{array} \quad \Longrightarrow \quad \begin{array}{ll} \{ & \mathcal{E}_{a_0 \| b_0}(g \parallel c_{F(0,0)}), \\ & \mathcal{E}_{a_0 \| b_1}(g \parallel c_{F(0,1)}), \\ & \mathcal{E}_{a_1 \| b_0}(g \parallel c_{F(1,0)}), \\ & \mathcal{E}_{a_1 \| b_1}(g \parallel c_{F(1,0)}) \quad \} \end{array}$$

Chaque valeur de sortie (c_0 ou c_1 suivant la valeur de $F(i, j)$) est chiffrée avec comme clé la concaténation $a_i \parallel b_j$ des entrées correspondantes.

Les 4 chiffrés résultants sont permutés aléatoirement.

Exemple : chiffrement des portes de l'additionneur complet



Évaluation d'une porte chiffrée

$$
\begin{array}{c|c|c}
a_0 & b_0 & c_{F(0,0)} \\
a_0 & b_1 & c_{F(0,1)} \\
a_1 & b_0 & c_{F(1,0)} \\
a_1 & b_1 & c_{F(1,1)}
\end{array}
\quad \Longrightarrow \quad
\left\{ \begin{array}{l}
\mathcal{E}_{a_0 \| b_0}(g \parallel c_{F(0,0)}), \\
\mathcal{E}_{a_0 \| b_1}(g \parallel c_{F(0,1)}), \\
\mathcal{E}_{a_1 \| b_0}(g \parallel c_{F(1,0)}), \\
\mathcal{E}_{a_1 \| b_1}(g \parallel c_{F(1,0)})
\end{array} \right\}
$$

Bob connaît uniquement l'identifiant g de la porte, ses 4 lignes chiffrées, et les entrées brouillées a, b.

Il déchiffre les 4 lignes avec la clé $a \parallel b$.

Avec une très forte probabilité, un seul clair est de la forme $g \parallel c$ pour un certain code c. Ce c est la sortie brouillée du circuit.

Évaluation d'une porte chiffrée

$$
\begin{array}{c|c|c}
a_0 & b_0 & c_{F(0,0)} \\
a_0 & b_1 & c_{F(0,1)} \\
a_1 & b_0 & c_{F(1,0)} \\
a_1 & b_1 & c_{F(1,1)}
\end{array}
\quad \Longrightarrow \quad
\left\{ \begin{array}{l}
\mathcal{E}_{a_0 \| b_0}(g \parallel c_{F(0,0)}), \\
\mathcal{E}_{a_0 \| b_1}(g \parallel c_{F(0,1)}), \\
\mathcal{E}_{a_1 \| b_0}(g \parallel c_{F(1,0)}), \\
\mathcal{E}_{a_1 \| b_1}(g \parallel c_{F(1,0)})
\end{array} \right\}
$$

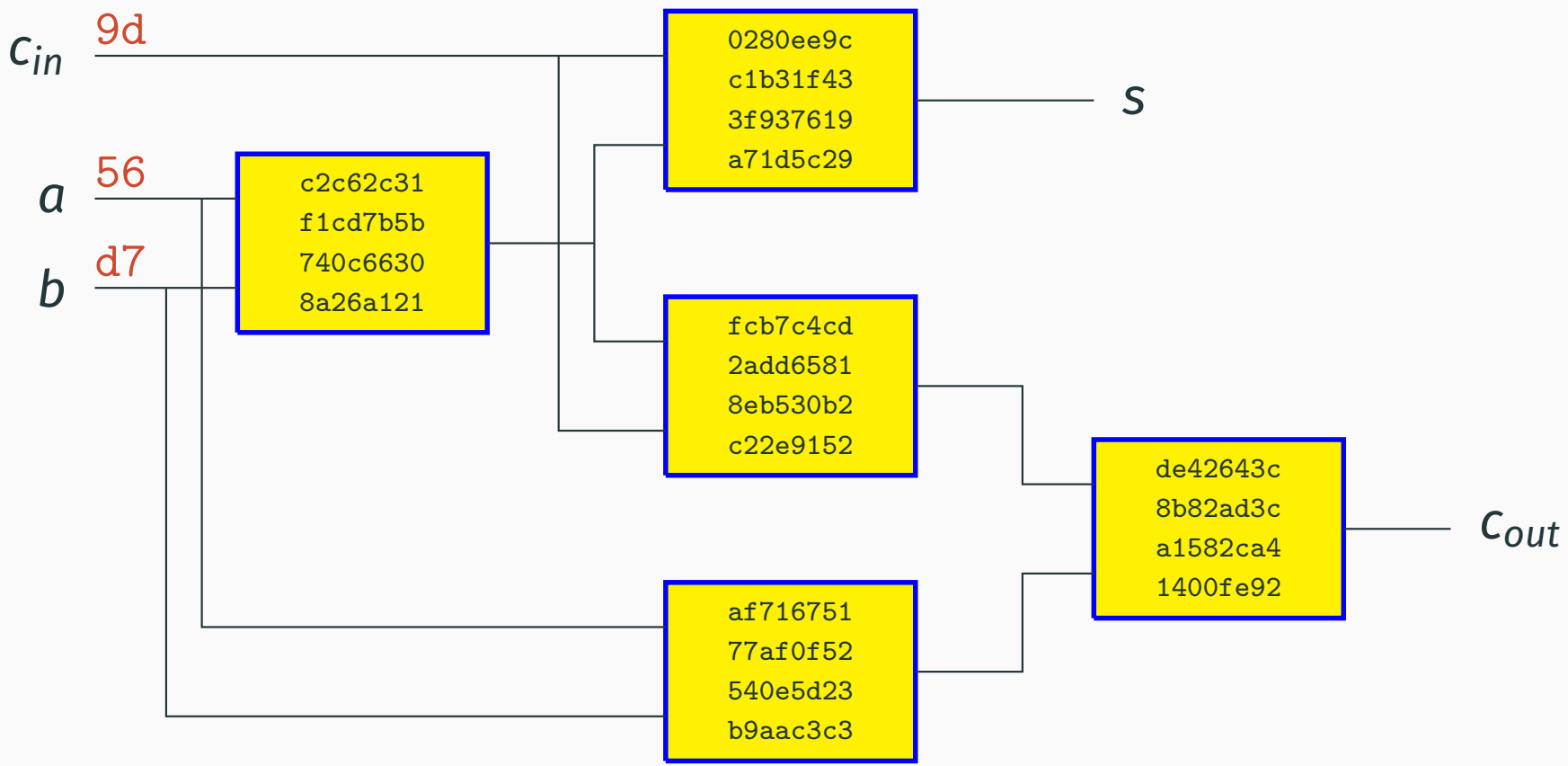
Bob connaît uniquement l'identifiant g de la porte, ses 4 lignes chiffrées, et les entrées brouillées a, b.

Il déchiffre les 4 lignes avec la clé $a \parallel b$.

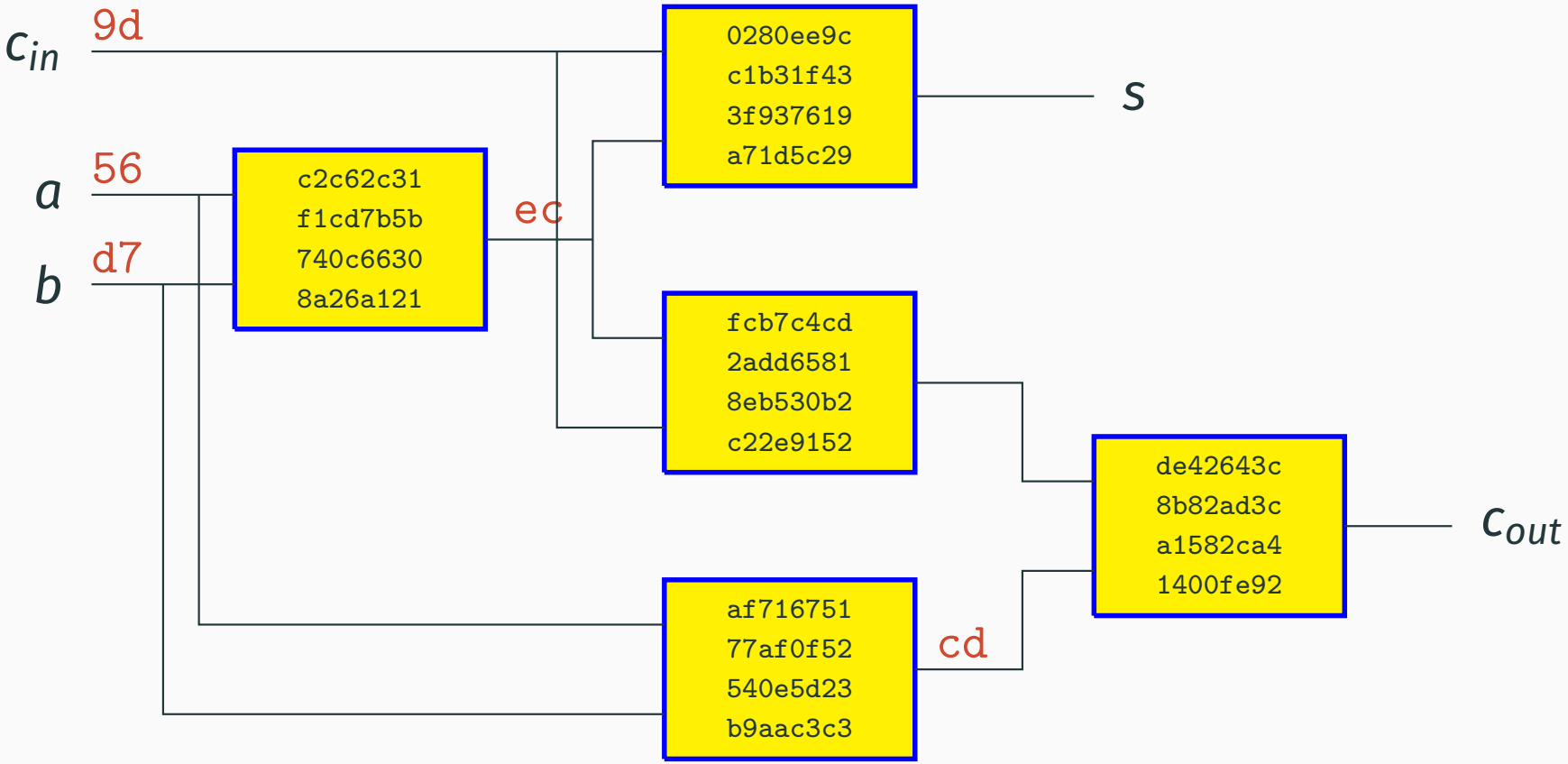
Avec une très forte probabilité, un seul clair est de la forme $g \parallel c$ pour un certain code c. Ce c est la sortie brouillée du circuit.

Bob peut évaluer la porte logique, mais ne sait pas quels bits représentent a, b, c, et ne connaît pas les 3 autres lignes.

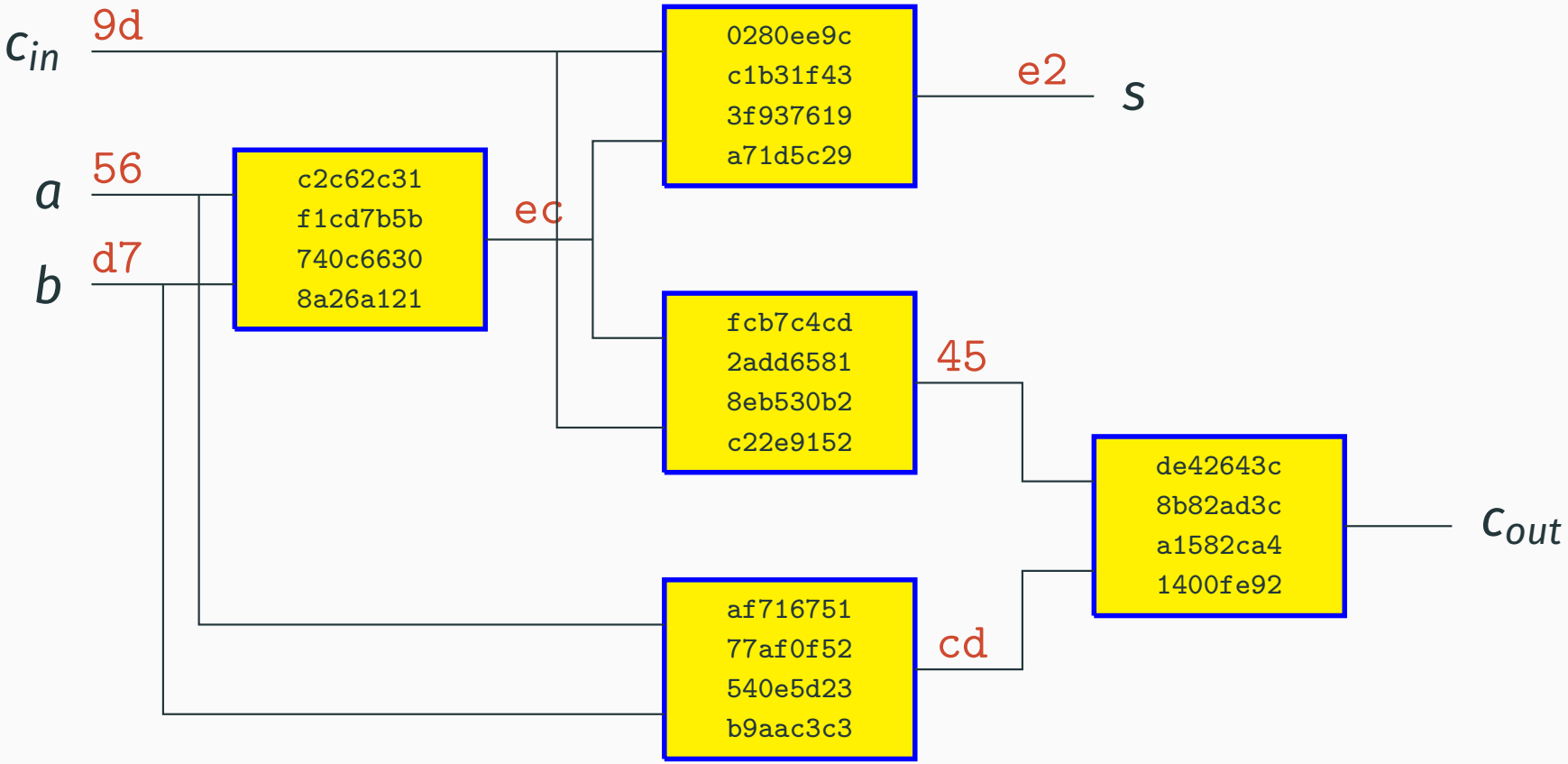
Exemple : une exécution de l'additionneur complet



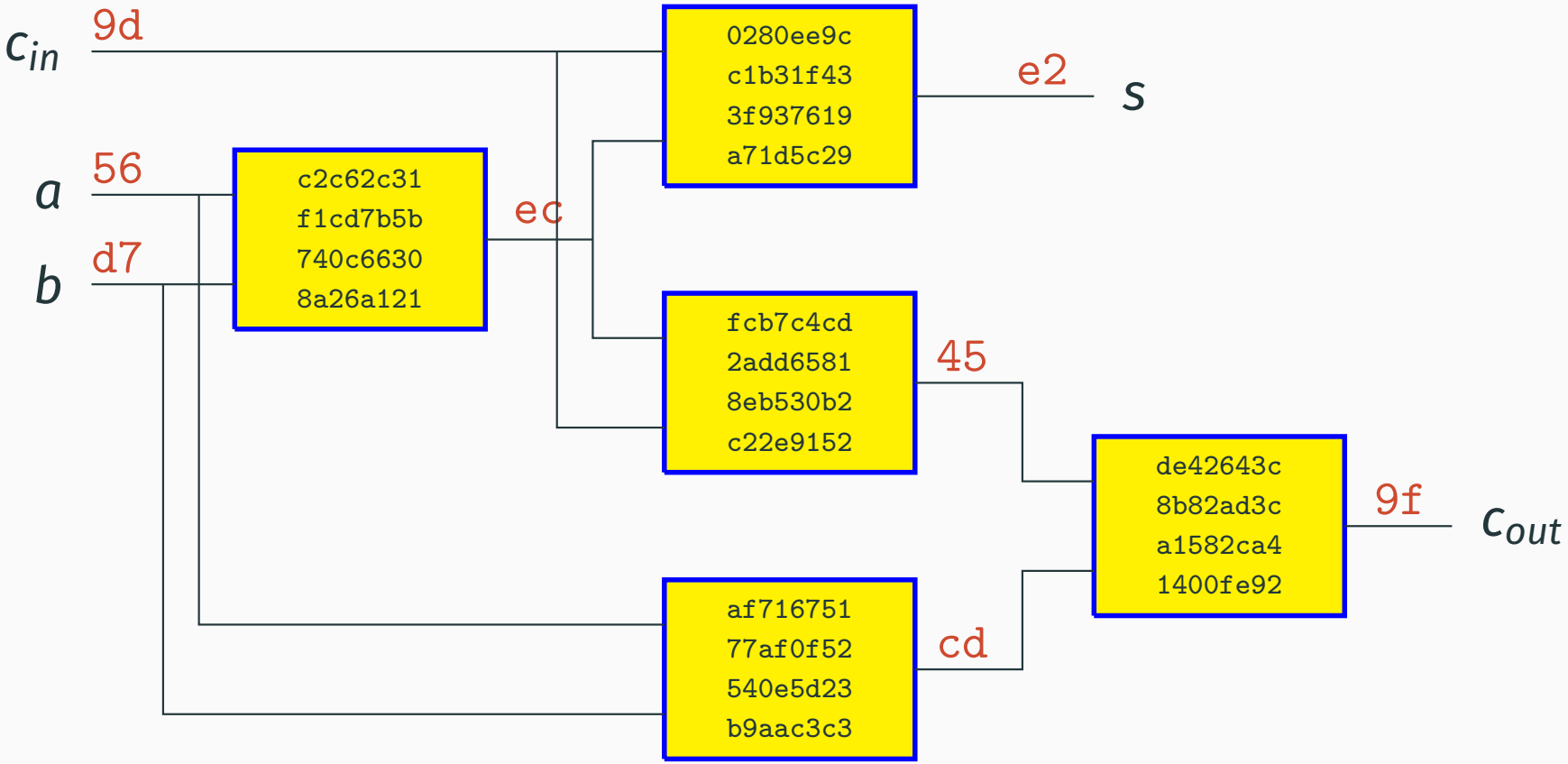
Exemple : une exécution de l'additionneur complet



Exemple : une exécution de l'additionneur complet



Exemple : une exécution de l'additionneur complet



Le protocole complet

1. Alice brouille les fils : $k \mapsto k_0, k_1$ aléatoires.
   Alice brouille le circuit et l'envoie à Bob.
   Alice, pour chacune de ses entrées a de valeur x,
   envoie l'entrée brouillée a_x à Bob.

2. Transfert inconscient : pour chaque entrée b de Bob de valeur y, Alice propose b_0 et b_1, Bob choisit y, et reçoit b_y.

3. Bob évalue (localement) le circuit brouillé.

4. Pour chaque sortie c du circuit, Bob envoie sa valeur brouillée c_0 ou c_1 à Alice, qui retrouve le bit 0 ou 1 correspondant et l'annonce.

Le protocole complet

1. Alice brouille les fils : $k \mapsto k_0, k_1$ aléatoires.
   Alice brouille le circuit et l'envoie à Bob.
   Alice, pour chacune de ses entrées a de valeur x,
   envoie l'entrée brouillée a_x à Bob.

2. Transfert inconscient : pour chaque entrée b de Bob de valeur y, Alice propose b_0 et b_1, Bob choisit y, et reçoit b_y.

3. Bob évalue (localement) le circuit brouillé.

4. Pour chaque sortie c du circuit, Bob envoie sa valeur brouillée c_0 ou c_1 à Alice, qui retrouve le bit 0 ou 1 correspondant et l'annonce.

(Variante : utiliser un brouillage trivial $k \mapsto 0, 1$ pour les sorties k.
Alors, Bob connaît la sortie en clair et l'annonce directement.)

Sécurité de ce protocole

Sécurité passive :

- Bob n'apprend rien des secrets a d'Alice
  (ils sont masqués par le brouillage des bits $0, 1 \mapsto k_0, k_1$).
- Alice n'apprend rien des secrets b de Bob
  (en supposant que le transfert inconscient est sécurisé).

Sécurité active :

- Si Bob ne suit pas le protocole, il va obtenir des valeurs impossibles (ni k_0 ni k_1) pour les fils de sortie k. Alice le détectera.
- Alice peut tricher de nombreuses manières, p.ex. en envoyant un circuit brouillé qui renvoie le secret de Bob en résultat : $F(a, b) = b$.

Accélérer l'évaluation d'une porte brouillée

Pour évaluer une porte brouillée $\{z_1, \ldots, z_4\}$ sur les entrées a, b, il faut déchiffrer deux z_i en moyenne et quatre dans le cas le pire.

On peut utiliser les bits de poids faibles de a et de b pour savoir exactement quel z_i déchiffrer.

Accélérer l'évaluation d'une porte brouillée

On choisit les brouillages de fils $k \mapsto k_0, k_1$ de sorte que $LSB(k_0) \neq LSB(k_1)$.

On ordonne les 4 chiffrés $\mathcal{E}_{a_x \| b_y}(g \parallel c_{F(x,y)})$ par $2 \times LSB(a_x) + LSB(b_y)$.

L'évaluateur sait quelle ligne déchiffrer : la ligne numéro $2 \times LSB(a) + LSB(b)$.

Exemple : table initiale / table réordonnée par LSB.

19	2c	af716751		56	2c	540e5d23		540e5d23
19	d7	77af0f52	$\Longrightarrow$	56	d7	b9aac3c3	$\Longrightarrow$	b9aac3c3
56	2c	540e5d23		19	2c	af716751		af716751
56	d7	b9aac3c3		19	d7	77af0f52		77af0f52

Accélérer le déchiffrement

On peut simplifier le chiffrement des lignes en utilisant une fonction de hachage $\mathcal{H}$.

Les 4 lignes z sont obtenues par

$$z_{2\times LSB(a_x)+LSB(b_y)} = \mathcal{H}(g \parallel a_x \parallel b_y) \oplus c_{F(x,y)}$$

avec comme déchiffrement pendant l'exécution de la porte

$$c = z_{2\times LSB(a)+LSB(b)} \oplus \mathcal{H}(g \parallel a \parallel b)$$

Accélérer le déchiffrement

On peut simplifier le chiffrement des lignes en utilisant une fonction de hachage $\mathcal{H}$.

Les 4 lignes z sont obtenues par

$$z_{2\times LSB(a_x)+LSB(b_y)} = \mathcal{H}(g \parallel a_x \parallel b_y) \oplus c_{F(x,y)}$$

avec comme déchiffrement pendant l'exécution de la porte

$$c = z_{2\times LSB(a)+LSB(b)} \oplus \mathcal{H}(g \parallel a \parallel b)$$

(La fonction de hachage peut être implémentée efficacement avec un chiffrement par bloc type AES et une clé fixée à l'avance et connue de tous.)

Des portes XOR qui ne coûtent presque rien

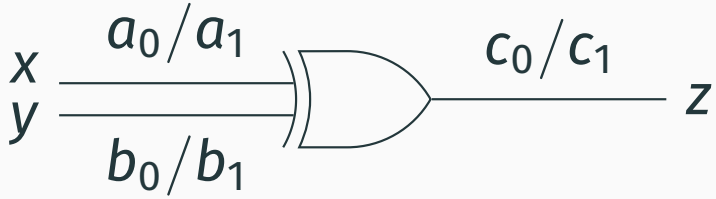
Pour un fil k, au lieu de prendre k_0 et k_1 aléatoires,
on peut prendre k_0 aléatoire et $k_1 = k_0 \oplus \Delta$
où Δ est un secret choisi par Alice au début du protocole.

Le brouillage du bit x sur le fil k est alors $k_0 \oplus x \cdot \Delta$.

Des portes XOR qui ne coûtent presque rien

Pour un fil k, au lieu de prendre k_0 et k_1 aléatoires,
on peut prendre k_0 aléatoire et $k_1 = k_0 \oplus \Delta$
où Δ est un secret choisi par Alice au début du protocole.

Le brouillage du bit x sur le fil k est alors $k_0 \oplus x \cdot \Delta$.



Si on choisit $c_0 = a_0 \oplus b_0$, cette porte XOR s'évalue sans déchiffrement, par un simple XOR de ses entrées :

$$\begin{aligned} a_x \oplus b_y &= (a_0 \oplus x \cdot \Delta) \oplus (b_0 \oplus y \cdot \Delta) \\ &= (a_0 \oplus b_0) \oplus (x \oplus y) \cdot \Delta = c_0 \oplus z \cdot \Delta = c_z \end{aligned}$$

Sécurité active : la technique du *cut-and-choose*

Comment s'assurer que le circuit brouillé envoyé par Alice calcule bien la fonction F et non pas $F'(a, b) = b$ par exemple ?

La technique du *cut-and-choose* :

- Alice produit n circuits brouillés $C_1, \ldots, C_n$ avec des aléas différents et les transmet à Bob.
- Bob choisit $i \in \{1, \ldots, n\}$ et demande à Alice de révéler les aléas utilisés pour construire $C_j, j \neq i$.
- Bob vérifie que les circuits $C_j, j \neq i$ sont bien des brouillages de F.
- Bob et Alice poursuivent le protocole avec le circuit C_i.

Sécurité active : expanser les entrées secrètes

Une autre attaque possible par Alice :

2. Transfert inconscient : pour chaque entrée b de Bob de valeur y, Alice propose b_0 et b_1, et Bob choisit y, et reçoit b_y.

Au lieu de proposer b_0 et b_1, Alice pourrait proposer b_0 et 0.

Si Bob produit un résultat correct, c'est qu'il n'a pas utilisé la valeur 0. Alice apprend donc que $y = 0$.

Contre-mesure : expanser l'entrée b en n entrées $b_1, \ldots, b_n$, avec un bout de circuit qui calcule $b = b_1 \oplus \cdots \oplus b_n$. Combiner cela avec la technique *cut-and-choose*.

(Y. Lindell, B. Pinkas : *An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries*. J. Cryptol, 2015).

Transfert inconscient

Le transfert inconscient (*Oblivious Transfer*, OT)

Un protocole entre deux participants :

- Alice (l'expéditrice) connaît n valeurs $m_1, \ldots, m_n$.
- Bob (le récepteur) choisit $i \in \{1, \ldots, n\}$.

À la fin du protocole,

- Bob connaît la valeur m_i.
- Alice ne connaît pas le choix i de Bob.
- Bob n'a rien appris sur les autres valeurs m_j pour $j \neq i$.

Un protocole de transfert inconscient 1 parmi 2 (EGL)

(S. Even, O. Goldreich, A. Lempel, *A Randomized Protocol for Signing Contracts*, CRYPTO 1982.)

Utilise un chiffrement à clé publique (*Keygen*) pour lequel on sait tirer au hasard des «fausses» clés publiques (*PubKeySamp*) indistinguables des «vraies».

1. Le récepteur Bob tire une paire de clés $(pk, sk) \leftarrow Keygen$ et une fausse clé publique $pk' \leftarrow PubKeySamp$.

S'il choisit $i = 0$ il envoie (pk, pk') à Alice.
S'il choisit $i = 1$ il envoie (pk', pk) à Alice.

2. L'expéditrice Alice reçoit deux clés publiques pk_0, pk_1 et chiffre ses deux messages avec :

$$c_0 = \mathcal{E}_{pk_0}(m_0) \qquad c_1 = \mathcal{E}_{pk_1}(m_1)$$

Elle envoie les chiffrés c_0 et c_1 à Bob.

3. Bob reçoit c_0, c_1 et déchiffre c_i avec sa clé privée :

$$m_i = \mathcal{D}_{sk}(c_i)$$

Correction : pk_i est la clé publique associée à sk, donc on a bien $\mathcal{D}_{sk}(\mathcal{E}_{pk_i}(m_i)) = m_i$.

Sécurité du protocole EGL

Sécurité passive :

- Alice reçoit deux clés publiques mais ne peut pas distinguer la vraie de la fausse.
  $\rightarrow$ Alice n'apprend rien sur le choix i de Bob.
- Bob reçoit deux chiffrés c_0, c_1 et peut déchiffrer c_i mais pas c_{1-i} (il n'a pas la clé secrète correspondant à pk').
  $\rightarrow$ Bob n'apprend rien sur m_{1-i} .

Sécurité active : Bob peut facilement tricher.

Au lieu de $pk' \leftarrow PubKeySamp$ il fait $(pk', sk') \leftarrow Keygen$.

Il peut alors déchiffrer les deux messages envoyés par Alice et apprendre m_0 et m_1.

Variante : transfert inconscient 1 parmi 4

(Se généralise facilement à 1 parmi 2^n.)

1. Bob le récepteur tire deux paires de clés et deux fausses clés

$$(pk_0, sk_0) \leftarrow \mathit{Keygen} \qquad pk'_0 \leftarrow \mathit{PubKeySamp}$$
$$(pk_1, sk_1) \leftarrow \mathit{Keygen} \qquad pk'_1 \leftarrow \mathit{PubKeySamp}$$

Il décompose son choix en binaire : $i = i_0 + 2i_1$.

Il envoie (pk_0, pk'_0) si $i_0 = 0$ ou (pk'_0, pk_0) si $i_0 = 1$.

Il envoie (pk_1, pk'_1) si $i_1 = 0$ ou (pk'_1, pk_1) si $i_1 = 1$.

Variante : transfert inconscient 1 parmi 4

2. Alice l'expéditrice reçoit deux paires de clés publiques (u_0, u_1) et (v_0, v_1), et chiffre doublement ses quatre messages avec :

$$c_0 = \mathcal{E}_{u_0}(\mathcal{E}_{v_0}(m_0))$$
$$c_1 = \mathcal{E}_{u_1}(\mathcal{E}_{v_0}(m_1))$$
$$c_2 = \mathcal{E}_{u_0}(\mathcal{E}_{v_1}(m_2))$$
$$c_3 = \mathcal{E}_{u_1}(\mathcal{E}_{v_1}(m_3))$$

3. Bob reçoit $c_0, \ldots, c_3$ et déchiffre c_i avec ses clés privées :

$$m_i = \mathcal{D}_{sk_0}(\mathcal{D}_{sk_1}(c_i))$$

Un protocole de transfert inconscient avec sécurité active (NP)

(M. Naor, B. Pinkas, *Efficient oblivious transfer protocols*, SODA 2001.)

Idée : donner à Alice un moyen de vérifier qu'une seule des clés pk_1, pk_2 est «vraie», au sens où Bob détient la clé privée correspondante.

On va utiliser une propriété du chiffrement d'ElGamal :

> si $pk = g^s$ est une «vraie» clé publique,
> et C est un élément quelconque fixé à l'avance,
> alors C/pk est une «fausse» clé publique
> (il est difficile de l'écrire comme $C/pk = g^t$).

Un protocole de transfert inconscient avec sécurité active (NP)

0. Au préalable : Alice tire C au hasard et l'envoie à Bob.

1. Bob tire une paire de clés $(pk, sk) = (g^s, s)$ avec $s \in \{1, \ldots, q-1\}$ aléatoire.

S'il choisit $i = 0$, il envoie $(pk, C/pk)$ à Alice.
S'il choisit $i = 1$, il envoie $(C/pk, pk)$ à Alice.

Un protocole de transfert inconscient avec sécurité active (NP)

2. Alice reçoit deux clés publiques pk_0, pk_1.

Elle vérifie que $pk_0 \cdot pk_1 = C$ et échoue sinon.

Elle chiffre ses deux messages avec pk_0, pk_1 :

$$c_0 = \mathcal{E}_{pk_0}(m_0) \qquad c_1 = \mathcal{E}_{pk_1}(m_1)$$

Elle envoie les chiffrés c_0 et c_1 à Bob.

3. Bob reçoit c_0, c_1 et déchiffre c_i avec sa clé privée :

$$m_i = \mathcal{D}_{sk}(c_i)$$

Sécurité du protocole NP

Sécurité passive :

- Alice ne peut pas distinguer la fausse clé C/pk d'une vraie clé g^y pour y aléatoire, car C/pk est aléatoire.
- Bob ne peut pas facilement retrouver une clé secrète y correspondant à la fausse clé C/pk :
  s'il retrouvait y, il connaîtrait $z = s + y$ tel que $C = g^z$, et donc il aurait trouvé le logarithme discret de C.

Sécurité active : Bob n'a aucun degré de liberté pour choisir la fausse clé, qui doit être C/pk pour qu'Alice l'accepte.

Variante : transfert inconscient aléatoire (ROT, *Random* OT)

Une variante de OT où les messages d'Alice et le choix de Bob sont tirés au hasard par le protocole.

Au début du protocole : pas d'information.

À la fin du protocole :

- Alice connaît deux messages r_0 et r_1 aléatoires.
- Bob connaît un bit $b \in \{0, 1\}$ aléatoire, ainsi que le message r_b .
- Alice ne connaît pas le bit b .
- Bob ne connaît rien sur r_{1-b} .

Construire OT à partir de ROT

Initialement :
Alice a deux messages m_0 et m_1 ; Bob a un choix $b \in \{0, 1\}$.

Exécution du protocole ROT :
Alice reçoit r_0, r_1 aléatoires ; Bob reçoit $s \in \{0, 1\}$ aléatoire et r_s.

Bob calcule $t = b \oplus s$ et l'envoie à Alice (Masquage.)

Si $t = 0$, Alice envoie $c_0 = m_0 \oplus r_0$ et $c_1 = m_1 \oplus r_1$ à Bob.
Si $t = 1$, Alice envoie $c_0 = m_0 \oplus r_1$ et $c_1 = m_1 \oplus r_0$ à Bob.
(Masquage.)

Bob retrouve $m_b = c_b \oplus r_s$.

(Correction : si $t = 0$, on a $s = b$ et $c_b \oplus r_s = m_b \oplus r_b \oplus r_b = m_b$.
Si $t = 1$, on a $s = 1 - b$ et $c_b \oplus r_s = m_b \oplus r_{1-b} \oplus r_{1-b} = m_b$.)

Étendre un protocole de transfert inconscient

Les protocoles OT utilisent tous du chiffrement à clé publique, qui est coûteux.

Le problème de l'extension OT : après avoir effectué n transferts inconscients utilisant du chiffrement à clé publique, peut-on effectuer $N \gg n$ transferts inconscients qui n'en utilisent pas ?

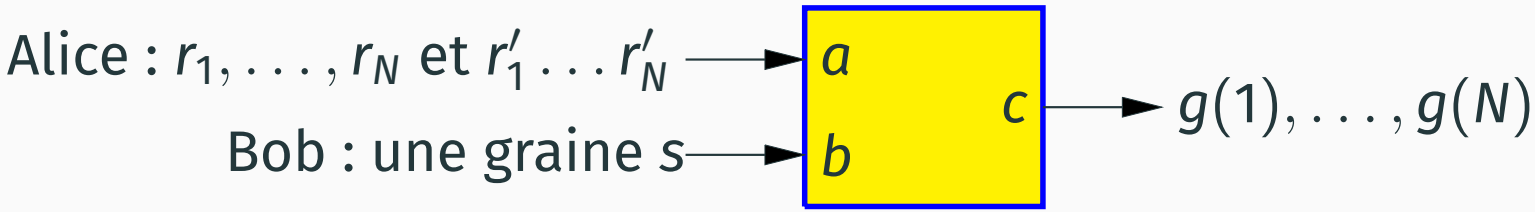
Transfert inconscient aléatoire avec un circuit brouillé

(D. Beaver, *Correlated pseudorandomness and the complexity of private computations*, STOC 1996.)

On se donne un générateur pseudo-aléatoire *PRNG* :

$$PRNG : \text{graine} \times \mathbb{N} \to \text{bit}$$

Alice prépare un brouillage du circuit suivant :



$$\text{Sorties} : g(i) = \begin{cases} (0, r_i) & \text{si } PRNG(s, i) = 0 \\ (1, r'_i) & \text{si } PRNG(s, i) = 1 \end{cases}$$

Transfert inconscient aléatoire avec un circuit brouillé

Alice brouille le circuit et le transmet à Bob.

Alice tire (pseudo-)aléatoirement $2N$ nombres
$r_1, \ldots, r_N$ et $r'_1, \ldots, r'_N$ et les transmet brouillés à Bob.

Bob choisit une graine s au hasard et la fait brouiller par Alice par OT classique. (n OT si n est la taille d'une graine en bits.)

Bob exécute le circuit, obtenant $g(1), \ldots, g(N)$.

Résultat : les paires $((r_i, r'_i), g(i))$ pour $i = 1, \ldots, N$
sont N transferts inconscients aléatoires, utilisables pour N transferts inconscients ensuite.

Extension d'un transfert aléatoire

La construction de Beaver montre qu'on peut obtenir N OT sans opérations à clé publique à partir de $n \ll N$ OT classiques.

Elle est limitée principalement par la taille du circuit brouillé.

Pour de meilleures techniques d'extension OT, voir
la section 7.3 du livre *A pragmatic introduction to MPC*
et le séminaire de Geoffroy Couteau.

Point d’étape

Point sur les circuits brouillés de Yao

Une des premières formes de calcul multipartite sécurisé.

Une des plus efficaces encore aujourd'hui !
(Peu de rounds de communication + crypto symétrique.)

Extension à $n > 2$ participants possible mais délicate.

Sécurité passive sans grandes difficultés
(mais : ne jamais évaluer deux fois le même circuit brouillé !)

Sécurité active atteignable mais coûteuse
(techniques *cut and choose* à base de sacrifices).

Point sur le transfert inconscient

Une primitive utilisée dans de nombreux protocoles.

Nécessite un minimum de crypto à clé publique et de rounds de communication.

Techniques d'extension pour amortir les coûts sur un grand nombre de transferts.

Bibliographie

Bibliographie

Pour aller un peu plus loin :

- *A pragmatic introduction to secure multi-party computation*, David Evans, Vladimir Kolsnikov, Mike Rosulek, NOW Publishers, 2018.
  Section 3.1 : circuits brouillés de Yao.
  Section 3.7 : transfert inconscient.

Pour aller beaucoup plus loin :

- *Foundations of garbled circuits*, Mihir Bellare, Viet Tung Hoang, Phillip Rogaway, CCS 2012, https://doi.org/10.1145/2382196.2382279
- *Oblivious Transfer Is in MiniQCrypt*, Alex B. Grilo, Huijia Lin, Fang Song, Vinod Vaikuntanathan, Eurocrypt 2021, https://doi.org/10.1007/978-3-030-77886-6_18