Le calcul sécurisé, sixième cours

Preuves *zero-knowledge*

Xavier Leroy

2025-12-11

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

Preuves sans divulgation de connaissances (ZK, ZKP, *zero-knowledge proofs*)

Au début du protocole :

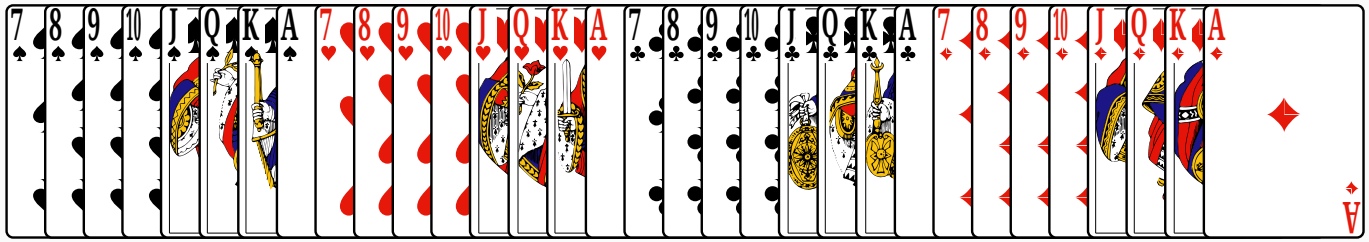
- Le prouveur connaît un secret x qui satisfait une propriété $P(x)$.

À la fin du protocole :

- La vérificatrice sait que le prouveur connaît x tel que $P(x)$.
- Elle n'a rien appris sur x qui ne soit pas une conséquence de $P(x)$.

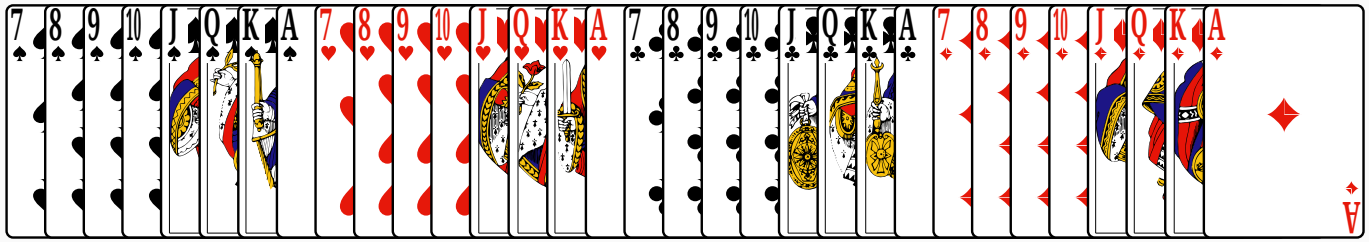
Un exemple de preuve *zero-knowledge*

Pierre (le prouveur) tire une carte à jouer et veut convaincre Véronique (la vérificatrice) que c'est une carte rouge, mais sans lui montrer la carte.



Un exemple de preuve *zero-knowledge*

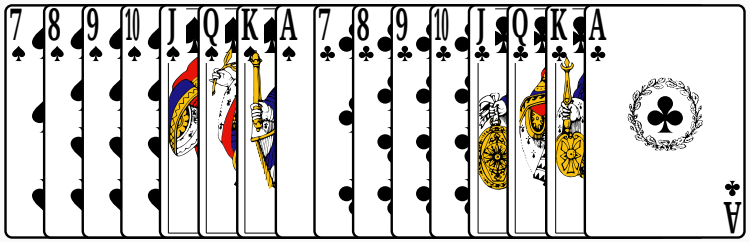
Pierre (le prouveur) tire une carte à jouer et veut convaincre Véronique (la vérificatrice) que c'est une carte rouge, mais sans lui montrer la carte.

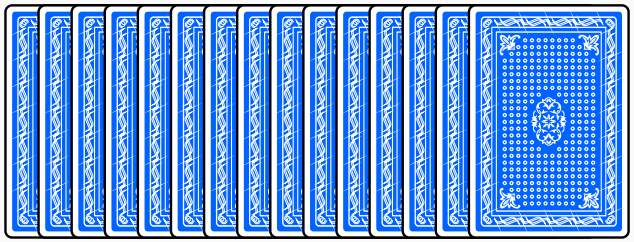


1. Pierre et Véronique vérifient le jeu de cartes :
16 rouges, 16 noires.

Un exemple de preuve *zero-knowledge*

Pierre (le prouveur) tire une carte à jouer et veut convaincre Véronique (la vérificatrice) que c'est une carte rouge, mais sans lui montrer la carte.

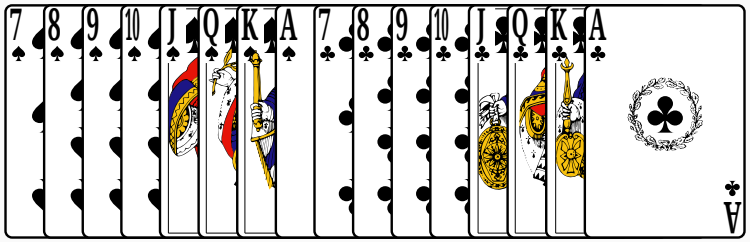


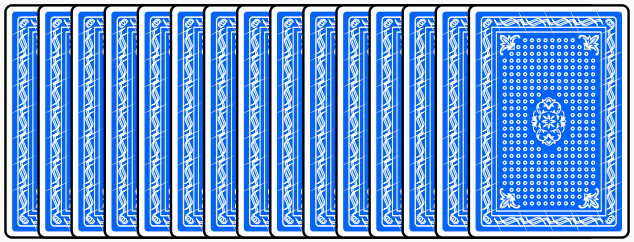


2. Pierre récupère les cartes, en garde une, pose les cartes restantes, et retourne 16 cartes noires.

Un exemple de preuve *zero-knowledge*

Pierre (le prouveur) tire une carte à jouer et veut convaincre Véronique (la vérificatrice) que c'est une carte rouge, mais sans lui montrer la carte.





3. Véronique est convaincue que la carte tirée par Pierre est rouge, mais ne sait rien de plus sur cette carte.

Quelques utilisations des preuves *zero-knowledge*

Affirmer des choses sur des données privées

(«mon vote est bien formé», «il y a assez d'argent sur mon compte», «j'ai 18 ans ou plus», etc)

Autorisation anonyme

(accès sites Web payants ; cryptomonnaies avec anonymat)

Déléguer des calculs avec garantie que le résultat est correct

(*blockchains* avec *rollups*)

Les protocoles Sigma

Forme d'un protocole Sigma

Pierre doit convaincre Véronique qu'il connaît x tel que $P(x)$.

Un protocole en 3 échanges : mise en gage (*commitment*), défi (*challenge*), réponse (*response*).

	Prouveur		Vérificatrice
Mise en gage	$k \leftarrow Commit(x)$	$\xrightarrow{k}$	
Défi		$\xleftarrow{c}$	$c \leftarrow Challenge$
Réponse	$r \leftarrow Resp(x, c)$	$\xrightarrow{r}$	
Vérification			$Verif(k, c, r)$

Exemple de protocole Sigma : le procotole de Schnorr (connaissance d'un logarithme discret)

Pierre doit convaincre Véronique qu'il connaît x tel que $g^x = a$ (g générateur du groupe G d'ordre q ; $a \in G$, connu de tous).

	Prouveur		Vérificatrice
Mise en gage	$y \in \mathbb{Z}_q$ aléatoire		
	$k = g^y$	$\xrightarrow{k}$	
Défi		$\xleftarrow{c}$	$c \in \mathbb{Z}_q$ aléatoire
Réponse	$r = y + x \cdot c \pmod{q}$	$\xrightarrow{r}$	
Vérification			$g^r = k \cdot a^c$?

(Application : Pierre s'authentifie sans révéler son mot de passe x.)

Propriétés attendues d'une preuve *zero-knowledge*

Complétude : si les deux participants suivent le protocole et si le prouveur connaît x tel que $P(x)$, la vérification réussit toujours.

Correction : si la vérification réussit, avec probabilité $1 - \varepsilon$ le prouveur connaît x tel que $P(x)$.

Non divulgation de connaissance (*zero-knowledge*) : la vérificatrice n'apprend rien sur x qu'elle ne peut déduire de $P(x)$.

Complétude du protocole

Complétude : si les deux participants suivent le protocole et si le prouveur connaît x tel que $P(x)$, la vérification réussit toujours.

$$\begin{aligned} k &\leftarrow \mathit{Commit}(x) \\ c &\leftarrow \mathit{Challenge} \\ r &\leftarrow \mathit{Resp}(x, c) \end{aligned} \quad \Longrightarrow \quad \mathit{Verif}(k, c, r) \text{ réussit}$$

Dans l'exemple du protocole de Schnorr :

$$\begin{aligned} k &= g^y \\ r &= y + x \cdot c \end{aligned} \quad \Longrightarrow \quad g^r = g^y \cdot (g^x)^c = k \cdot a^c$$

Correction du protocole

Correction : si la vérification réussit, le prouveur connaît x tel que $P(x)$ avec très forte probabilité.

Idée : si le prouveur ne répond pas au hasard, et si on avait accès au code du prouveur et des valeurs de ses variables libres, on pourrait en extraire x.

Méthode d'extraction qui ≪marche≫ pour beaucoup de protocoles : exécuter deux fois le protocole en forçant le prouveur à mettre en gage le même k.

Correction spéciale du protocole

Correction spéciale : il existe une fonction d'extraction *Extr* en temps polynomial probabiliste qui retrouve x à partir de deux traces valides (k, c_1, r_1) et (k, c_2, r_2) avec le même gage k et des défis différents $c_1 \neq c_2$.

$$x = Extr((k, c_1, r_1), (k, c_2, r_2))$$
satisfait $P(x)$ avec probabilité $1 - \varepsilon$

Dans l'exemple du protocole de Schnorr :

$$\begin{aligned} r_1 &= y + x \cdot c_1 \\ r_2 &= y + x \cdot c_2 \end{aligned} \quad \Longrightarrow \quad x = \frac{r_1 - r_2}{c_1 - c_2} \pmod{q}$$

Non divulgation de connaissance (*zero-knowledge*)

***Zero-knowledge* :** la vérificatrice n'apprend rien sur x qu'elle ne peut déduire de $P(x)$.

Idée : si le prouveur ne connaissait pas x mais pouvait voyager dans le temps et changer son gage k après avoir reçu le défi c, serait-il capable de produire une réponse r correcte ?

Si c'est le cas, on voit que l'existence d'une trace (k, c, r) qui passe la vérification n'implique rien sur x, pas même qu'il existe.

Non divulgation de connaissance (*zero-knowledge*)

Zero-knowledge : il existe une fonction $Simu(c) = (k, r)$ qui produit une trace (k, c, r) valide pour tout défi c.

$$\begin{array}{l} c \leftarrow Challenge \\ (k, r) \leftarrow Simu(c) \end{array} \implies Verif(k, c, r) \text{ réussit}$$

Dans l'exemple du protocole de Schnorr, on prend

$$Simu(c) = (g^r \cdot a^{-c}, r) \quad \text{pour } r \in \mathbb{Z}_q \text{ aléatoire}$$

La vérification $g^r = k \cdot a^c$ passe, puisque $k \cdot a^c = g^r \cdot a^{-c} \cdot a^c = g^r$.

Généralisation : le protocole de Maurer

(U. Maurer, *Unifying Zero-Knowledge Proofs of Knowledge*, AFRICACRYPT 2009.)

Deux groupes G, H d'ordre q et un morphisme $\varphi : G \to H$ difficilement inversible (*one-way*).

Étant donné $a \in H$ public, prouver $\exists x \in G,\ \varphi(x) = a$.

	Prouveur		Vérificatrice
Mise en gage	$y \in G$ aléatoire		
	$k = \varphi(y)$	$\xrightarrow{k}$	
Défi		$\xleftarrow{c}$	$c \in \mathbb{Z}_q$ aléatoire
Réponse	$r = y \cdot x^c$	$\xrightarrow{r}$	
Vérification			$\varphi(r) = k \cdot a^c$

Sécurité du protocole de Maurer

Complétude : si $a = \varphi(x)$ et $k = \varphi(y)$ et $r = y \cdot x^c$,

$$\varphi(r) = \varphi(y) \cdot \varphi(x)^c = k \cdot a^c \quad \text{(car } \varphi \text{ est un morphisme)}$$

Correction : si on a deux exécutions $r_1 = y \cdot x^{c_1}$ et $r_2 = y \cdot x^{c_2}$, on a $r_1/r_2 = x^{c_1}/x^{c_2} = x^{c_1 - c_2}$ d'où $x = (r_1/r_2)^{c_2 - c_1}$.

***Zero-knowledge* :** pour c donné, on simule une exécution (k, c, r) valide en prenant $r \in G$ aléatoire et $k = \varphi(r) \cdot a^{-c}$.

On a bien $k \cdot a^r = \varphi(r) \cdot a^{-c} \cdot a^r = \varphi(r)$.

Cas particuliers de l'approche de Maurer

Schnorr : connaissance d'un logarithme discret $\exists x,\ g^x = a$

$\varphi : \mathbb{Z}_q \to G \qquad \varphi(x) = g^x$

Guillou-Quisquater : connaissance d'une racine e-ième $\exists x,\ x^e = a$

$\varphi : \mathbb{Z}_m^* \to \mathbb{Z}_m^* \qquad \varphi(x) = x^e \bmod m$

Chaum-Pedersen : $\exists x,\ a = g^x \wedge b = h^x$ c.à.d. $\log_g(a) = \log_h(b)$

$\varphi : \mathbb{Z}_q \to G \times G \qquad \varphi(x) = (g^x, h^x)$

(a, b) **est un chiffré ElGamal de** m **:** $\exists x,\ a = g^x \wedge b = h^x \cdot m$

Exemple donné dans le 1er cours.

Utiliser Chaum-Pedersen avec a et b/m.

Prouver une disjonction

Étant donnés des protocoles Sigma pour les propriétés P_1 et P_2,
peut-on construire une preuve Sigma pour la disjonction $P_1 \vee P_2$?

Exemple d'utilisation : prouver qu'un bulletin de vote chiffré avec ElGamal est soit un chiffré de 0 soit un chiffré de 1,
sans révéler dans quel cas on se trouve.

Idée : une réponse à un défi est une paire de réponses :
une ≪vraie≫ réponse construite avec $Resp_i$ (si P_i est vraie);
une réponse simulée construite avec $Simu_j$ pour $j \neq i$.

Vérifier la preuve d'une disjonction

Gage : une paire (k_1, k_2) de gages pour les protocoles P_1 et P_2.

Réponse : un quadruplet (c_1, c_2, r_1, r_2)
où r_1 est une réponse au défi c_1 et r_2 au défi c_2.

Vérification d'une réponse au défi c :

$$\begin{aligned} \mathit{Verif}(\ (k_1, k_2),\ c,\ (c_1, c_2, r_1, r_2)\ ) = \quad & c = c_1 \oplus c_2 \\ & \wedge\ \mathit{Verif}_1(k_1, c_1, r_1) \\ & \wedge\ \mathit{Verif}_2(k_2, c_2, r_2) \end{aligned}$$

La vérification est symétrique en P_1 et P_2 et n'indique pas laquelle des deux propriétés est vraie.

Produire la preuve d'une disjonction

Supposons que $P_1(x)$ est vraie. On n'a pas de preuve de $P_2(x)$, alors on va en simuler une pour un défi c_2 choisi par le prouveur.

	Prouveur		Vérificatrice
Mise en gage	$k_1 \leftarrow Commit_1(x)$		
	$c_2 \leftarrow Challenge$		
	$(k_2, r_2) \leftarrow Simu_2(c_2)$	$\xrightarrow{k_1, k_2}$	
Défi		$\xleftarrow{c}$	$c \leftarrow Challenge$
Réponse	$c_1 = c \oplus c_2$		
	$r_1 \leftarrow Resp_1(x, c_1)$	$\xrightarrow{c_1, c_2, r_1, r_2}$	
Vérification			$c = c_1 \oplus c_2$?
			$Verif_1(k_1, c_1, r_1)$?
			$Verif_2(k_2, c_2, r_2)$?

Preuves non-interactives

Vers un protocole non-interactif

	Prouveur		Vérificatrice
Mise en gage	$k \leftarrow Commit(x)$	$\xrightarrow{k}$	
Défi		$\xleftarrow{c}$	$c \leftarrow Challenge$
Réponse	$r \leftarrow Resp(x, c)$	$\xrightarrow{r}$	
Vérification			$Verif(k, c, r)$

Le défi c n'a pas besoin d'être choisi aléatoirement par la vérificatrice. N'importe qui pourrait choisir c pourvu que

- c est choisi après la mise en gage ;
- c n'est pas contrôlé par le prouveur ;
- c est pseudo-aléatoire.

L'heuristique de Fiat-Shamir

(A. Fiat, A. Shamir, *How To Prove Yourself : Practical Solutions to Identification and Signature Problems*, CRYPTO 1986.)

Utiliser une fonction de hachage $\mathcal{H}$ pour produire le défi à partir du gage k et des paramètres publics pp du problème :

$$c = \mathcal{H}(pp \parallel k)$$

Si $\mathcal{H}$ est vue comme un oracle aléatoire, c est aléatoire et on ne peut pas manipuler k pour obtenir le c qu'on veut.

Si $\mathcal{H}$ est une fonction de hachage cryptographique usuelle (p.ex. SHA-256), on admet que ces propriétés sont vraies aussi.

Transformation d'un protocole Sigma en protocole non-interactif

	Prouveur		Vérificatrice
Mise en gage	$k \leftarrow Commit(x)$		
Défi	$c = \mathcal{H}(pp \parallel k)$		
Réponse	$r \leftarrow Resp(x, c)$	$\xrightarrow{k,r}$	
Vérification			$c = \mathcal{H}(pp \parallel k)$
			$Verif(k, c, r)$

Fiat-Shamir : le défi est choisi par le prouveur en utilisant $\mathcal{H}$.

La preuve est la paire (k, r). Elle peut être vérifiée à tout moment sans interaction avec le prouveur.

Exemple de preuve non interactive

Je connais le logarithme discret de a en base g dans $\mathbb{Z}_p^*$.

$$p = 25644269200652980450766820164246153935 3$$

$$g = 781944113$$

$$a = 660237491474363027736483369857459075 35$$

J'en veux pour preuve

$$k = 200299568312215464498549432374020738 31$$

$$r = 221824598860809771154727139215467720 68$$

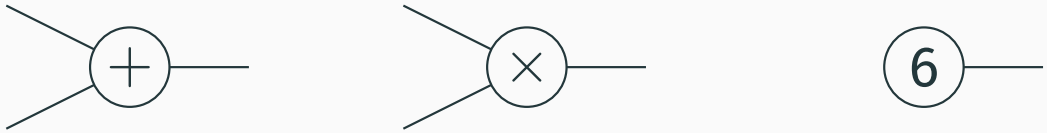
(Protocole de Schnorr + fonction de hachage SHA-256.)

Circuits arithmétiques et programmes quadratiques

Circuits arithmétiques

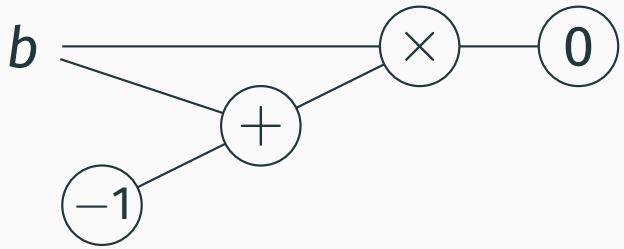
Des circuits combinatoires qui vont au delà des circuits booléens :

- Les fils prennent des valeurs dans un corps $\mathbb{F}_q$ d'ordre q.
- Portes de base : addition, multiplication, constante.



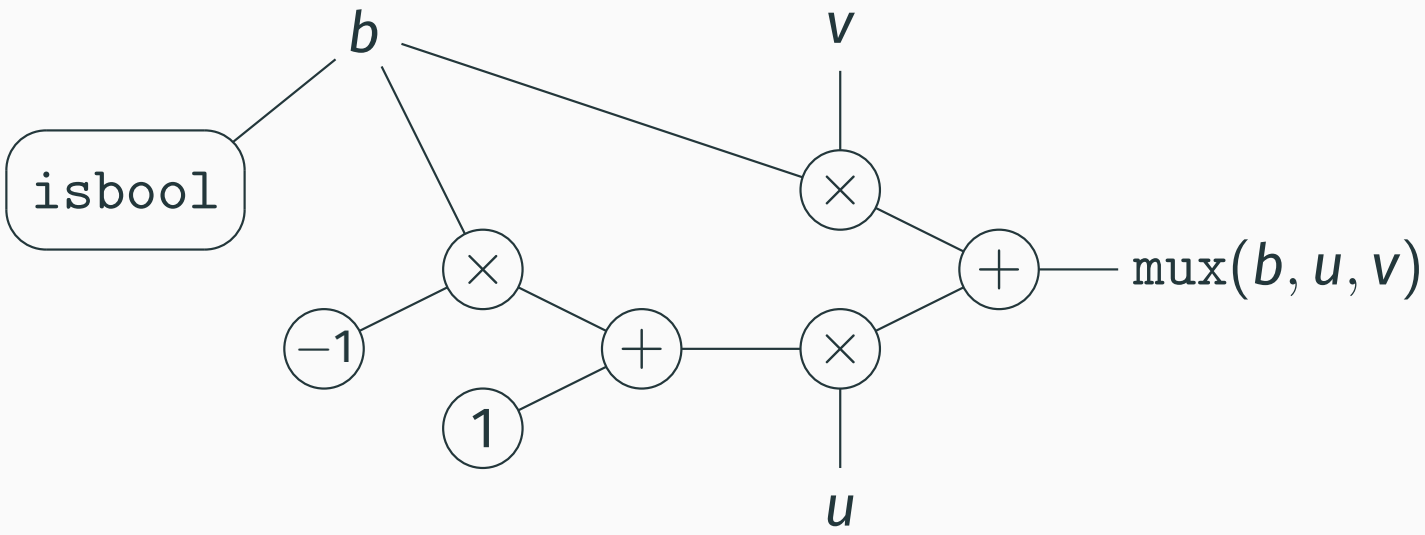
- On peut contraindre la sortie d'une porte à être égale à une constante ou à la sortie d'une autre porte.

  Exemple : le circuit $\texttt{isbool}(b)$ qui prouve que $b \in \{0, 1\}$.



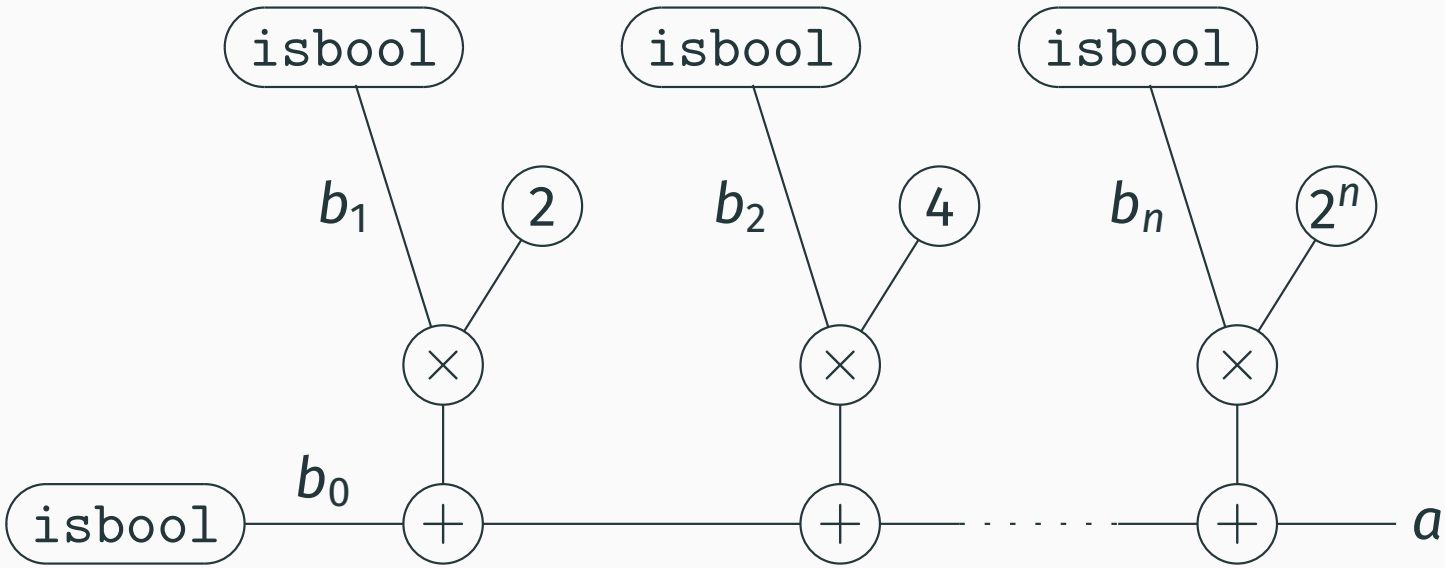
Exemples de circuits arithmétiques

Multiplexeur : $\texttt{mux}(b, u, v) = \texttt{if } b \texttt{ then } v \texttt{ else } u$



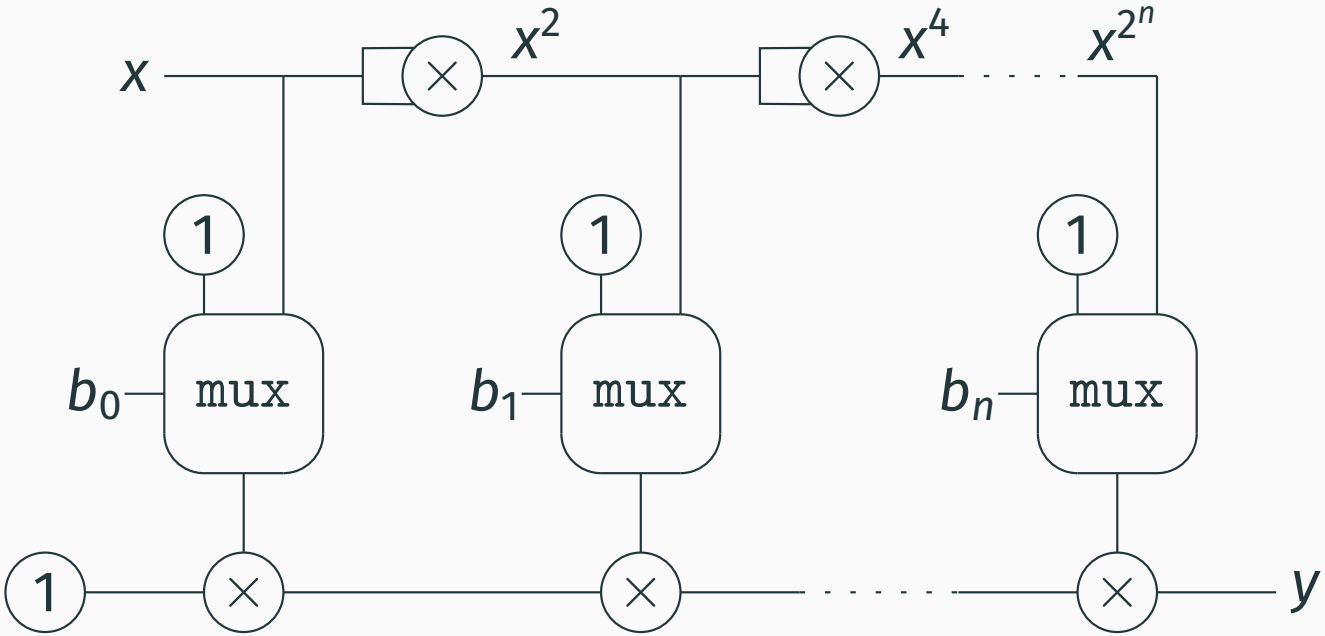
Combiner opérations arithmétiques et opérations booléennes

Décomposition en bits : $a = b_0 + 2b_1 + \ldots + 2^n b_n$



Combiner opérations arithmétiques et opérations booléennes

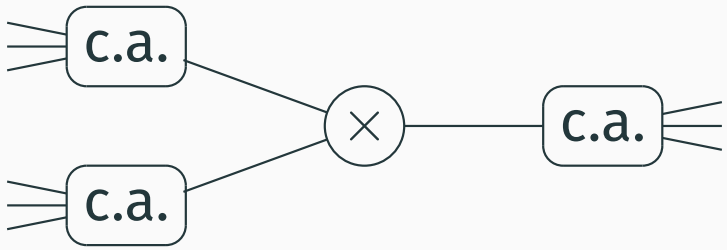
Exponentiation rapide : $y = x^a$ avec $a = b_0 + 2b_1 + \cdots 2^n b_n$.



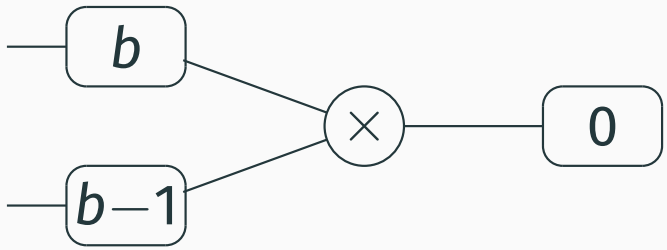
Si on fixe x et y, ce circuit prouve la connaissance du logarithme discret de y en base x .

Mise en équations des portes

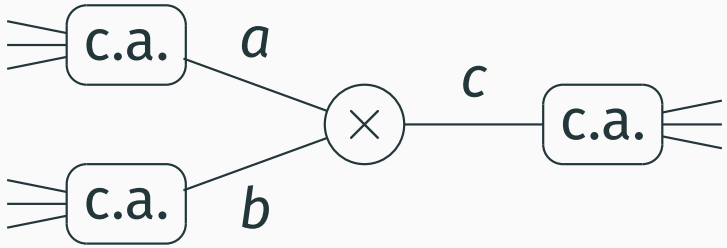
On décompose le circuit en macro-portes comprenant chacune exactement une porte «produit» (non linéaire), avec deux entrées et une sortie qui sont des combinaisons affines de fils.



Exemple du circuit `isbool` :



Mise en équations des portes



Soient $w_1, \ldots, w_n$ les valeurs des n fils du circuit. On prend $w_0 = 1$.

Les deux entrées a, b et la sortie c de la porte «produit» sont des combinaisons linéaires des w_i :

$$a = a_0 w_0 + \cdots + a_n w_n \quad b = b_0 w_0 + \cdots + b_n w_n \quad c = c_0 w_0 + \cdots + c_n w_n$$

Puisque $c = a \times b$, on obtient la contrainte

$$(a_0 w_0 + \cdots a_n w_n)\,(b_0 w_0 + \cdots b_n w_n) = c_0 w_0 + \cdots + c_n w_n$$

Système de contraintes de rang 1 (R1CS, *Rank-1 Constraint System*)

En exprimant ainsi chacune des m portes «produit» du circuit d'origine, on obtient un ensemble de m équations du second degré en les w_i :

$$(a_{1,0}w_0 + \cdots a_{1,n}w_n)\,(b_{1,0}w_0 + \cdots b_{1,n}w_n) = c_{1,0}w_0 + \cdots c_{1,n}w_n$$

$$\vdots$$

$$(a_{m,0}w_0 + \cdots a_{m,n}w_n)\,(b_{m,0}w_0 + \cdots b_{m,n}w_n) = c_{m,0}w_0 + \cdots c_{m,n}w_n$$

Les coefficients $a_{j,i}$, $b_{j,i}$, $c_{j,i}$ décrivent la structure du circuit.

Programme arithmétique quadratique (QAP, *Quadratic Arithmetic Program*)

Un codage du système de contraintes sous forme de polynômes.

On se donne m points $x_1, \ldots, x_m$ de $\mathbb{F}_q$.
Le point x_j identifie la j-ème macro-porte.

Par interpolation de Lagrange, on construit des polynômes $A_0, \ldots, A_n, B_0, \ldots, B_n, C_0, \ldots, C_n$ tels que

$$A_i(x_j) = a_{j,i} \qquad B_i(x_j) = b_{j,i} \qquad C_i(x_j) = c_{j,i}$$

et on forme le polynôme

$$P = (w_0 A_0 + \cdots + w_n A_n)\,(w_0 B_0 + \cdots + w_n B_n) - (w_0 C_0 + \cdots + w_n C_n)$$

Programme arithmétique quadratique (QAP, *Quadratic Arithmetic Program*)

$$P = (w_0 A_0 + \cdots + w_n A_n)\,(w_0 B_0 + \cdots + w_n B_n) - (w_0 C_0 + \cdots + w_n C_n)$$

Le circuit initial s'exécute correctement si et seulement si

$$P(x_j) = 0 \text{ pour tout } j \in \{1, \ldots, m\}$$

ou, de manière équivalente, si et seulement si P s'écrit comme

$$P = H\,T \quad \text{avec} \quad T = (X - x_1)\,(X - x_2) \cdots (X - x_m)$$

Lien avec les preuves *zero-knowledge*

On a des preuves qui sont des ≪arguments de connaissance≫ :

je connais des valeurs **x** telles que $Prop(\mathbf{x})$ est vraie.

Si la propriété *Prop* peut s'exprimer comme un circuit C, l'argument devient

je connais des valeurs des fils **w** qui satisfont le circuit C.

Après transformation en QAP, l'argument devient

je connais des valeurs **w** et un polynôme H tels que

$$(w_0 A_0 + \cdots + w_n A_n)\,(w_0 B_0 + \cdots + w_n B_n) - (w_0 C_0 + \cdots + w_n C_n) = H\,T$$

où les polynômes A_i, B_i, C_i et le polynôme cible T sont connus de tous et ne dépendent que du circuit.

Évaluation masquée de polynômes

Prouver l'égalité de deux polynômes

Alice connaît un polynôme $A = a_0 + a_1 X + \cdots + a_n X^n$ de degré n.

Bob connaît un polynôme $B = b_0 + b_1 X + \cdots + b_n X^n$ de degré n.

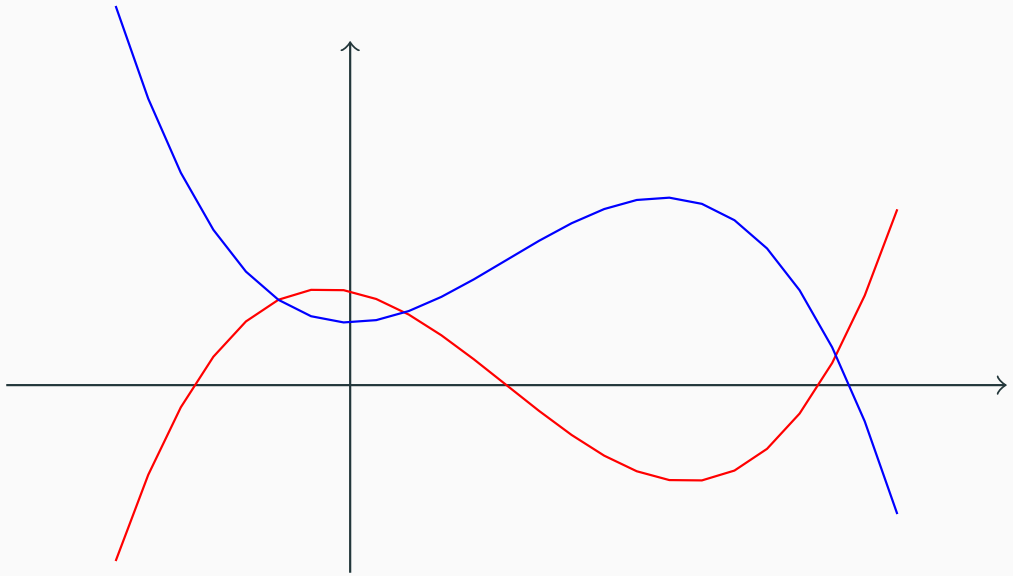
Ils veulent s'assurer que $A = B$.

Preuve naïve : vérifier que $a_0 = b_0, \ldots, a_n = b_n$
$\Rightarrow$ preuve de taille $\mathcal{O}(n)$.

Preuve succinte : évaluer A et B en un point z aléatoire.
Si $A(z) = B(z)$, alors $A = B$ avec forte probabilité.

Prouver l'égalité de deux polynômes

Si deux polynômes A et B de degré n diffèrent, ils sont égaux en au plus n points.



On se place dans un corps fini $\mathbb{F}_q$ d'ordre $q \gg n$.

La probabilité que $z \in \mathbb{F}_q$ soit un des points d'égalité est $n/q \ll 1$.

Donc, si $A(z) = B(z)$, on a $A = B$ avec forte probabilité.

Prouver qu'un polynôme a certaines racines (premier essai)

Pierre a un polynôme P et veut convaincre Véronique qu'il s'annule en $x_1, \ldots x_n$.

Prouveur		Vérificatrice
		$T = (X - x_1) \cdots (X - x_n)$
	$\xleftarrow{T,z}$	$z \in \mathbb{F}_q$ aléatoire
$H = P/T$		
$p = P(z) \quad h = H(z)$	$\xrightarrow{p,h}$	
		$p = h \cdot T(z)$?

Si $p = h \cdot T(z)$, avec forte probabilité on a $P = H \cdot T$ et donc $x_1, \ldots, x_n$ sont racines de P.

Sécuriser l'évaluation de polynômes

Dans le protocole précédent, le prouveur peut tricher de nombreuses manières. Par exemple, il prend h aléatoire et $p = h \cdot T(z)$.

Idée : pour limiter ce que le prouveur peut faire, on va transmettre z chiffré et utiliser du calcul homomorphe.

Soit E une fonction «à sens unique» (*one-way*) homomorphe pour l'addition et la multiplication par une constante :

$$E(x + y) = E(x) \oplus E(y)$$
$$E(c \cdot x) = c \odot E(x)$$

mais surtout pas homomorphe pour la multiplication générale :

$E(x \cdot y)$ ne peut pas être déterminé à partir de $E(x)$ et $E(y)$

Exemple de fonction à sens unique

Soit $(G, \cdot)$ un groupe d'ordre q où le logarithme discret est difficile à calculer. Soit g un générateur de G.

On définit $E : \mathbb{F}_q \to G$ comme $E(x) = g^x$.

On a bien

$$E(x + y) = g^{x+y} = g^x \cdot g^y = E(x) \cdot E(y) \qquad (\oplus \text{ est le produit dans } G)$$
$$E(c \cdot x) = g^{cx} = (g^x)^c = E(x)^c \qquad (\odot \text{ est l'exponentiation dans } G)$$

Mais à partir de g^x et g^y on ne sait pas calculer facilement g^{xy} (hypothèse Diffie-Hellman).

Note : l'égalité des chiffrés $E(x) = E(y)$ implique l'égalité des clairs $x = y \pmod{q}$.

Évaluation homomorphe d'un polynôme

On peut évaluer homomorphiquement des combinaisons linéaires :

$$E(c_1 x_1 + \cdots + c_n x_n) = c_1 \odot E(x_1) \oplus \cdots \oplus c_n \odot E(x_n)$$

Si en plus on nous donne les chiffrés des puissances de z

$$E(1), E(z), \ldots, E(z^n)$$

on peut aussi évaluer $P(z)$ pour tout polynôme P de degré n :

$$E(c_0 + c_1 z + \cdots + c_n z^n) = c_0 \odot E(1) \oplus c_1 \odot E(z) \oplus \cdots \oplus c_n \odot E(z^n)$$

Prouver qu'un polynôme a certaines racines (deuxième essai)

Prouveur		Vérificatrice
		$T = (X - x_1) \cdots (X - x_n)$
		$z \in \mathbb{F}_q$ aléatoire
	$\xleftarrow{T, z_0, \ldots, z_n}$	$z_i = E(z^i)$ pour $i = 0, \ldots, n$
$H = P/T$		
$p = E(P(z))$ $\quad h = E(H(z))$	$\xrightarrow{p, h}$	
		$p = T(z) \odot h$?

Prouver qu'un polynôme a certaines racines (deuxième essai)

Prouveur		Vérificatrice
		$T = (X - x_1) \cdots (X - x_n)$
		$z \in \mathbb{F}_q$ aléatoire
	$\xleftarrow{T, z_0, \ldots, z_n}$	$z_i = E(z^i)$ pour $i = 0, \ldots, n$
$H = P/T$		
$p = E(P(z))$ $\quad h = E(H(z))$	$\xrightarrow{p, h}$	
		$p = T(z) \odot h$?

Note : le prouveur peut encore tricher !

P.ex. $h = E(r)$ et $p = r \odot E(T(z))$ pour r aléatoire.

Authentifier les chiffrés

Soit $k \in \mathbb{F}_q$ un secret connu uniquement de la vérificatrice.

Un chiffré authentifié de x est une paire de chiffrés de x et kx :

$$\bar{E}(x) = (E(x), E(kx))$$

Connaissant k, la vérificatrice peut facilement vérifier que le chiffré (a, b) est authentique en testant $b = k \odot a$.

Authentifier les chiffrés

$$\bar{E}(x) = (E(x), E(kx))$$

Sans connaître k, le prouveur peut calculer homomorphiquement sur les chiffrés authentifiés :

$$\bar{E}(x + y) = \bar{E}(x) \oplus \bar{E}(y) \qquad \text{(point à point)}$$
$$\bar{E}(cx) = c \odot \bar{E}(x) \qquad \text{(point à point)}$$

Sans connaître k, les seuls chiffrés authentifiés valides que le prouveur peut construire sont des combinaisons linéaires de chiffrés authentifiés fournis par la vérificatrice.

(Hypothèse KEA, *Knowledge of Exponent Assumption*.)

Prouver qu'un polynôme a certaines racines (version sécurisée)

Prouveur		Vérificatrice
		$T = (X - x_1) \cdots (X - x_n)$
		$z \in \mathbb{F}_q$ aléatoire
	$\xleftarrow{T, z_0, \ldots, z_n}$	$z_i = \bar{E}(z^i)$ pour $i = 0, \ldots, n$
$H = P/T$		
$p = \bar{E}(P(z))$ $\quad h = \bar{E}(H(z))$	$\xrightarrow{p, h}$	
		valider p et h
		$p = T(z) \odot h$?

Version *zero-knowledge*

Pour rendre le protocole *zero-knowledge*, il suffit de masquer les réponses du prouveur par un facteur r aléatoire :

$$p = r \odot \bar{E}(P(z)) \qquad h = r \odot \bar{E}(H(z))$$

La vérification $p = T(z) \odot h$ est inchangée.

Version non-interactive

Pour rendre le protocole non-interactif, il faut choisir les secrets z et k à l'avance, et partager des infos chiffrées avec les prouveurs et les vérificatrices :

Clé d'évaluation : $\bar{E}(1), \bar{E}(z), \ldots, \bar{E}(z^n)$

Clé de vérification : $E(1), E(T(z)), E(k)$

Problème : sans connaître k en clair, comment vérifier la validité d'un chiffré authentifié (a, b) ? (plus possible de tester $b = k \odot a$).

$\Rightarrow$ Utilisation d'un couplage.

Couplage cryptographique (*pairing, bilinear map*)

Étant donnés deux groupes multiplicatifs G (engendré par g) et G', un couplage $e : G \times G \to G'$ est une forme bilinéaire non dégénérée :

$$e(g^a, g^b) = e(g, g^b)^a = e(g^a, g)^b = e(g, g)^{ab}$$

$$e(g, g) \neq 1 \text{ est un générateur de } G'$$

Des couplages existent pour de nombreuses courbes elliptiques.

Un couplage fournit un moyen de tester homomorphiquement l'égalité de deux produits :

$$e(g^a, g^b) = e(g^c, g^d) \quad \Longleftrightarrow \quad ab = cd$$

Vérifications à l'aide d'un couplage

Validité d'un chiffré authentifié (a, b) :

Connaissant k : $b = k \odot a$

Connaissant uniquement $E(k)$: $e(b, E(1)) = e(E(k), a)$

Vérification de $P(z) = H(z) \cdot T(z)$:

Connaissant $T(z)$: $p = T(z) \odot h$

Connaissant uniquement $E(T(z))$: $e(p, E(1)) = e(E(T(z)), h)$

Le protocole Pinocchio : des ZK-SNARK pour QAP

Le protocole Pinocchio

(B. Parno, J. Howell, C. Gentry, M. Raykova, *Pinocchio : Nearly Practical Verifiable Computation*, S&P 2013, CACM 2016.)

Un protocole ZK-SNARK :

Zero-Knowledge	
Succint	(preuves de taille constante)
Non-interactive	
ARgument of Knowledge	(il existe $\mathbf{s}$ tel que $Prop(\mathbf{s})$)

La propriété *Prop* est exprimée par un circuit arithmétique, représenté sous forme QAP : des polynômes de degré m

$$T, A_0, \ldots, A_n, B_0, \ldots, B_n, C_0, \ldots, C_n$$

où n est le nombre de secrets et m le nombre de portes ≪produit≫ dans le circuit.

Preuve de connaissance

Le prouveur doit montrer qu'il connaît des secrets $s_0, \ldots, s_n$ et un polynôme H tels que

$$(s_0 A_0 + \cdots + s_n A_n)\,(s_0 B_0 + \cdots + s_n B_n) - (s_0 C_0 + \cdots + s_n C_n) = H\,T$$

On note $\quad A = \sum s_i A_i \quad B = \sum s_i B_i \quad C = \sum s_i C_i\,.$

On va utiliser une variante du protocole qui prouve qu'un polynôme a certaines racines :

- Montrer que $A(z)\,B(z) - C(z) = H(z)\,T(z)$ pour un z aléatoire.
- De plus : montrer que A, B, C sont des combinaisons linéaires des A_i, B_i, C_i avec les mêmes coefficients $s_0, \ldots, s_n$.

Initialisation du protocole

Une autorité tire au hasard $z, k, \alpha, \beta, \gamma, \delta$ dans $\mathbb{F}_q$ et publie les connaissances partagées (*Common Reference String*, CRS) :

Clé d'évaluation :

$\bar{E}(z^i)$ pour $i = 0, \ldots, m$

$\bar{E}(A_i(z))$ pour $i = 0, \ldots, n$

$\bar{E}(B_i(z))$ pour $i = 0, \ldots, n$

$\bar{E}(C_i(z))$ pour $i = 0, \ldots, n$

$E(\alpha A_i(z) + \beta B_i(z) + \gamma C_i(z))$ pour $i = 0, \ldots, n$

Clé de vérification :

$E(1)$ $E(k)$ $E(T(z))$

$E(\delta)$ $E(\alpha\delta)$ $E(\beta\delta)$ $E(\gamma\delta)$

Génération de la preuve

Connaissant les secrets $s_0, \ldots, s_n$, le prouveur

- forme les polynômes $A = \sum s_i A_i \quad B = \sum s_i B_i \quad C = \sum s_i C_i$
- calcule H tel que $A\,B - C = H\,T$ par division de polynômes.

La preuve se compose des chiffrés authentifiés des polynômes A, B, C, H évalués homomorphiquement au point z

$$\bar{E}(A(z)) \qquad \bar{E}(B(z)) \qquad \bar{E}(C(z)) \qquad \bar{E}(H(z))$$

ainsi que de la «somme de contrôle»

$$E(\alpha A(z) + \beta B(z) + \gamma C(z)) = \bigoplus s_i \odot E(\alpha A_i(z) + \beta B_i(z) + \gamma C_i(z))$$

(Taille de la preuve : 9 éléments du groupe G, indépendamment de la taille m du circuit et du nombre n de secrets.)

(*Zero-knowledge* : ajouter des multiples aléatoires de T à A, B, C.)

Vérification de la preuve

La vérificatrice a quatre chiffrés authentifiés a, b, c et h, ainsi qu'une ≪somme de contrôle≫ chiffrée ck.

Elle vérifie que a, b, c et h sont valides. Ceci garantit qu'ils ont été obtenus par combinaisons linéaires de valeurs de la CRS.

Elle vérifie la condition de divisibilité :

$$e(a, b) = e(c, E(1)) \cdot e(E(T(z)), h)$$

Ceci montre que $A(z)\, B(z) = C(z) + T(z)\, H(z)$.

Elle vérifie que :

$$e(ck, E(\delta)) = e(a, E(\alpha\delta)) \cdot e(b, E(\beta\delta)) \cdot e(c, E(\gamma\delta))$$

Ceci montre que A, B, C sont des combinaisons linéaires des A_i, B_i, C_i avec les mêmes coefficients s_i.

(Conséquence non évidente de l'hypothèse KEA.)

Point d’étape

Point d'étape

Les protocoles Sigma :

- Le principal ≪patron≫ pour concevoir des protocoles ZK interactifs et en analyser la sécurité.
- Passage à des protocoles non-interactifs via l'heuristique de Fiat-Shamir.

Les preuves zk-SNARK :

- Capables de prouver la connaissance d'une solution pour une très large classe de problèmes.
- L'approche QAP : circuits arithmétiques encodés par des polynômes.
- D'autres approches sont possibles, voir le survey de Nitulescu (2020).

Bibliographie

Bibliographie

Protocoles Sigma :

- *Cryptography made simple*, Nigel P. Smart, Springer, 2016. Chapitre 21.

Un survey sur les ZK-SNARKs :

- Anca Nitulescu, *zk-SNARKs : A Gentle Introduction*, 2020. https://www.di.ens.fr/~nitulesc/files/Survey-SNARKs.pdf

Un tutoriel sur l'approche QAP :

- Maksym Petkus, *Why and How zk-SNARK Works : Definitive Explanation*, 2019, http://arxiv.org/abs/1906.07221

Une synthèse sur QAP et ses applications aux blockchains :

- Thomas Chen, Hui Lu, Teeramet Kunpittaya, Alan Luo, *A Review of zk-SNARKs*, 2023, https://arxiv.org/abs/2202.06877