

Chiffrement et déchiffrement d'une peinture numérisée

Arnaud Soulier
Université de Montpellier

2 Février 2018

1 Algorithme de chiffrement par permutation pseudo-aléatoire

L'algorithme utilisé ici (cf. algorithme 1) effectue un mélange pseudo-aléatoire sur les positions des pixels de l'image. Avant tout, un générateur de nombres pseudo-aléatoires est initialisé avec une seed passé en paramètre du programme et un tableau d'indice est créé. Chaque case de ce tableau contient un nombre dans l'ordre entre 0 et le nombre de pixels de l'image moins un. À partir de là, chaque nouvelle attribution de position se fait en quatre étapes.

- Un nombre r est tiré par le générateur pseudo-aléatoire entre 0 et la taille du tableau d'indice.
- Le nouvel indice k du pixel est pris dans le tableau à l'indice r .
- Les données du pixel de l'image sont copiées dans une nouvelle image à l'indice défini précédemment.
- La case d'indice r du tableau d'indice est supprimée et les cases suivantes sont réindexées.

```
Data: inputImage, cipherKey
Result: outputImage
srand(cipherKey);
size ← height * width;
index ← int[size];
for  $i \leftarrow 0$  to size do
    r ← random(index.size());
    newIndex ← index[r];
    outputImageR[newIndex] ← inputImageR[i];
    outputImageG[newIndex] ← inputImageG[i];
    outputImageB[newIndex] ← inputImageB[i];
    index[r].remove();
end
```

Algorithm 1: Algorithme de chiffrement par permutation pseudo-aléatoire d'une image couleur

La figure 1 montre les trois premières itérations de cet algorithme sur un tableau de cinq cases.

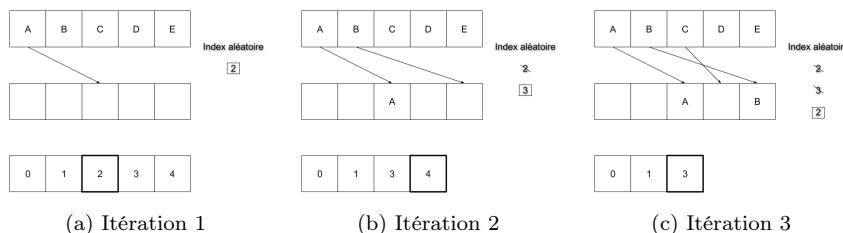


FIGURE 1 – Choix pseudo-aléatoire des nouveaux indices des pixels

Pour la première (cf. figure 1(a)), le chiffre 2 est tiré par le générateur. Le nouvel indice du premier pixel de l'image sera donc 2. On supprime ensuite la

3^{eme} case du tableau d'indice (les indices commencent à 0) et on recommence l'opération. Pour la deuxième itération (cf. figure 1(b)), le chiffre 3 est tiré. La case d'indice 3 du tableau d'indice n'ayant plus la valeur 3 mais 4. Le deuxième pixel de l'image sera donc en case 5 de la nouvelle image. Pour ce qui est de la troisième itération (cf. figure 1(c)), le chiffre 2 sort à nouveau. Cette fois ci, la troisième case du tableau d'indice contient la valeur 3. Le troisième pixel de l'image sera donc en position 4.

2 Algorithme de déchiffrement par permutation pseudo-aléatoire

L'algorithme de déchiffrement (cf. algorithme 2) se base sur le même principe que celui de chiffrement. La différence repose sur l'ordre de parcours des tableaux. L'algorithme de chiffrement parcourt dans l'ordre l'image d'entrée et dans le désordre l'image de sortie alors que l'algorithme de déchiffrement fait l'inverse. L'image chiffré est lu dans le désordre et l'image déchiffrée est écrite dans l'ordre.

```

Data: inputImage, cipherKey
Result: outputImage
srand(cipherKey);
size ← height * width;
index ← int[size];
for i ← 0 to size do
    r ← random(index.size());
    oldIndex ← index[r];
    outputImageR[i] ← inputImageR[oldIndex];
    outputImageG[i] ← inputImageG[oldIndex];
    outputImageB[i] ← inputImageB[oldIndex];
    index[r].remove();
end

```

Algorithm 2: Algorithme de chiffrement par permutation pseudo-aléatoire d'une image couleur

Selon le même procédé, la figure 2 montre les trois premières itérations pour le même tableau de cinq cases.

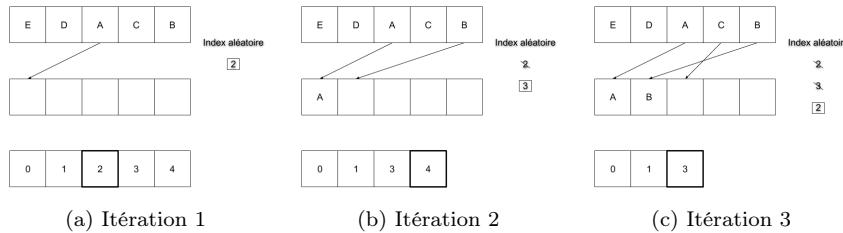


FIGURE 2 – Récupération pseudo-aléatoire des anciens indices des pixels

Les nombres tirés par le générateur pseudo-aléatoire apparaissent exactement dans le même ordre que lors du chiffrement (à condition que la clef renseignée soit la même également). Dans cette situation, on va donc chercher la valeur dans l'image d'entrée à l'indice renseigné dans le tableau d'indice en fonction du nombre pseudo-aléatoire tiré. On va ensuite stocker cette valeur dans la prochaine case libre de l'image de sortie.

3 Permutation par blocs

Afin de permettre à un appareil photo de récupérer et réorganiser correctement tous les pixels il faut les agencer et les déplacer par blocs entiers. Il faut donc ajouter aux algorithmes précédents un deuxième groupe de boucle pour les blocs.

```

Data: inputImage, cipherKey
Result: outputImage
strand(cipherKey);
size ← height * width;
index ← int[size];
for i ← 0 to size do
    r ← random(index.size());
    newIndexBlock ← index[r];
    oldIndexFirst = (i - (mod(i, nbBlockWidth))) * sizeBlock + ((mod(i,
        nbBlockWidth)) * widthBlock);
    newIndexFirst = (newBlockIndex - (mod(newBlockIndex,
        nbBlockWidth))) * sizeBlock + ((mod(newBlockIndex,
        nbBlockWidth)) * widthBlock);
    for x ← 0 to widthBlock do
        for y ← 0 to widthBlock do
            newIndex = newIndexFirst + x + (y * width);
            oldIndex = oldIndexFirst + x + (y * width);
            outputImageR[newIndex] ← inputImageR[oldIndex];
            outputImageG[newIndex] ← inputImageG[oldIndex];
            outputImageB[newIndex] ← inputImageB[oldIndex];
        end
    end
end
```

Algorithm 3: Algorithme de chiffrement par permutation pseudo-aléatoire d'une image couleur

Dans l'algorithme 3, la première boucle gère les différents blocs comme un pixel simple et calcule les coordonnées de l'angle supérieur gauche des blocs de départ et d'arrivé. La seconde section de boucle permet de calculer directement les coordonnées des pixels du bloc.

4 Moyenne des pixels

Dans le même intérêt de faciliter la récupération des images par un appareil photo, on peut remplacer chaque pixel d'un bloc par une même valeur. On peut par exemple calculer la moyenne des pixels du bloc puis l'appliquer à tous les pixels de ce bloc avant de les déplacer.

5 Résultats

5.1 Chiffrement

Les résultats des algorithmes de chiffrement sont présentés dans les figures 3 et 4.

Les figures :

- (a) montrent les images d'origines utilisé pour les tests
- (b) montrent les images chiffrées par les pixels
- (c), (d) et (e) montrent les images chiffrées avec respectivement 10, 25 et 50 blocs
- (f), (g) et (h) montrent les images chiffrées avec respectivement 10, 25 et 50 blocs avec la moyenne des blocs.

5.2 Déchiffrement

Le déchiffrement des images (b), (c), (d) et (e) permet de récupérer les images originales.

Quant au déchiffrement des images (f), (g) et (h), il renvoie les images (respectivement) 5(a), 5(b), 5(c) et 5(d), 5(e), 5(f) et



(a) Picasso



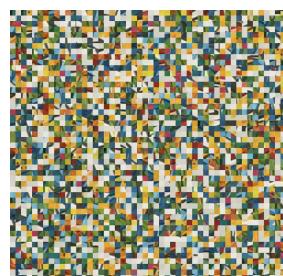
(b) Permutation par pixels



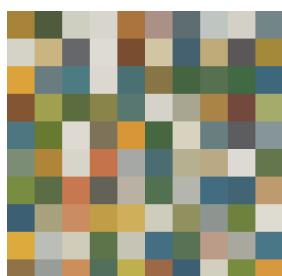
(c) Permutation 10 blocs



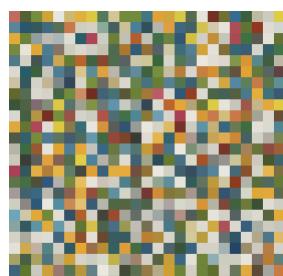
(d) Permutation 25 blocs



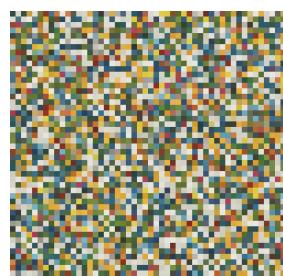
(e) Permutation 50 blocs



(f) Permutation 10 blocs
moyennés



(g) Permutation 25 blocs
moyennés



(h) Permutation 50 blocs
moyennés

FIGURE 3 – Résultat du chiffrement de la peinture "Picasso" par pixels et par blocs



(a) Van Gogh

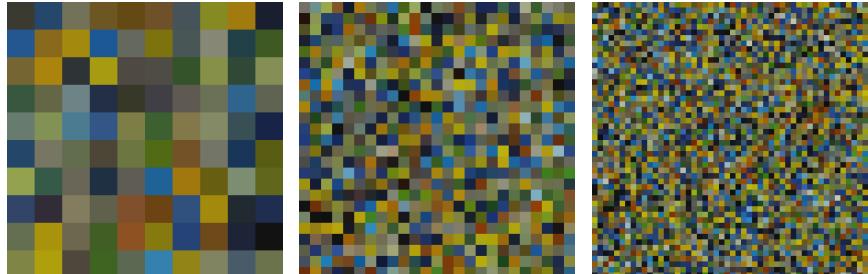
(b) Permutation par pixels



(c) Permutation 10 blocs

(d) Permutation 25 blocs

(e) Permutation 50 blocs



(f) Permutation 10 blocs moyennés (g) Permutation 25 blocs moyennés (h) Permutation 50 blocs moyennés

FIGURE 4 – Résultat du chiffrement de la peinture "Van Gogh" par pixels et par blocs



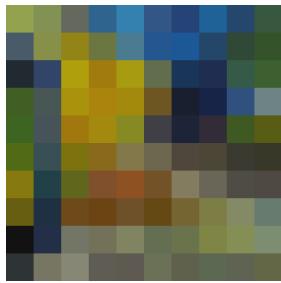
(a) Picasso 10 blocs moyennés



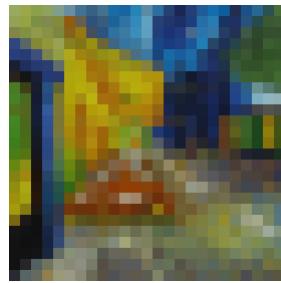
(b) Picasso 25 blocs moyennés



(c) Picasso 50 blocs moyennés



(d) Van Gogh 10 blocs moyennés



(e) Van Gogh 25 blocs moyennés



(f) Van Gogh 50 blocs moyennés

FIGURE 5 – Résultat du déchiffrement des peintures "Picasso" et "Van Gogh" par blocs avec moyenne