

SUPINFO Academic Dept.

**Arithmétique et cryptographie**

Mini - Projet

Chiffrement d’images

Version 1.0

Last update: 22/01/2015

Use: Students/Staff

Author: Laurent GODEFROY

SOMMAIRE

0 Préambule 3

1 Chiffrement d’images 4

1.1 Généralités sur cette partie 4

1.2 Implémentation en Python 4

2 DISSIMULATION D’IMAGES 5

2.1 Généralités sur cette partie 5

2.2 Implémentation en Python 6

3 DISSIMULATION DE TEXTES 7

3.1 Généralités sur cette partie 7

3.2 Implémentation en Python 8

# Préambule

Cet examen est à réaliser par groupes de deux étudiants. Dans l’unique cas où le nombre d’étudiants de la promotion est impair, un et un seul groupe de trois est autorisé.

Toute forme de plagiat ou utilisation de codes disponibles sur internet ou tout autre support, même de manière partielle, est strictement interdite et se verra sanctionnée d’un 0, d’une mention « cheater », et le cas échéant d’un conseil de discipline.

Vous devrez envoyer votre projet par mail à **votre formateur** avant le dimanche 8 février 2015 à 23h59, heure locale. Au delà de cette date et heure votre note sera de 0. Vous comprimerez vos codes sources ainsi que les images demandées dans une archive au format « .zip » que vous nommerez en respectant la convention **ID-1ARI-votreCampus-MP** (exemple : 66280-1ARI-Tours-MP). L’objet du mail sera ce même nom. Vous mettrez également **en copie** la boîte [1ARI@supinfo.com](mailto:1ARI@supinfo.com). Si vous ne mettez pas cette dernière adresse en copie aucune réclamation ultérieure ne sera recevable.

Ce mini-projet donnera lieu à des soutenances qui se dérouleront la semaine du 9 février 2015. Vos horaires de passages vous seront communiqués par votre campus.

Les soutenances sont également par groupes de deux. Elles dureront **20 minutes** pendant lesquelles vous montrerez à votre examinateur le bon fonctionnement de votre programme en en faisant la démonstration. Si vous n’avez pas implémenté le projet dans sa totalité, vous exposerez les parties fonctionnelles.

Pour appuyer votre présentation, vous devrez préparer un fichier de type Powerpoint, dans lesquels vous expliquerez les points du code que vous jugez les plus importants et significatifs. Il n’est pas nécessaire d’envoyer ce fichier à votre examinateur, ce dernier le découvrira le jour de la soutenance. Une communication précisant tout cela vous sera envoyé début février.

Un barème **indicatif** vous est donné dans la dernière partie de ce sujet.

Autre remarque, **n'hésitez pas à utiliser le forum pour échanger sur ce projet, j'ai ouvert un fil de discussion à cet usage.**

On devra nécessairement utiliser la librairie graphique Pygame. L’usage d’une autre librairie ne sera pas pris en compte. Il est fortement conseillé pour mener à bien ce projet d’utiliser le module « surfarray » de Pygame. Il utilise la librairie Numpy. Le travail de documentation sur ces outils fait partie du projet.

On pourra si on le souhaite ajouter des fonctions et procédures à celles demandées.

Les images auxquelles fait référence le sujet se trouvent sur le site courses.supinfo.com

# Chiffrement d’images

## Généralités sur cette partie

**Remarque importante**: aucun code n’est demandé dans ce paragraphe qui n’est qu’explicatif.

Le but de cette partie est d'implémenter un algorithme symétrique de chiffrement d'images.

La clé sera elle même une image que les correspondants auront échangée en amont.

Pour générer des clés nous procèderons de façon aléatoire, en attribuant une valeur au hasard à chacune des composantes R, G et B d'un pixel.

L'image à chiffrer et la clé devront avoir les mêmes dimensions. On chiffrera un pixel de l'image en utilisant le pixel situé au même emplacement dans la clé.

Pour chiffrer un pixel, on procède comme suit pour chacune des composantes R, G et B :

* on convertit en binaire (sous forme d'un octet) la valeur de la composante de l'image à chiffrer et celle de la clé.
* on effectue un XOR bit par bit.
* on reconvertit le résultat obtenu sous forme d'entier.

L'image chiffrée sera alors constituée de tous ces pixels.

Le déchiffrement se fera exactement de la même façon, en considérant l’image chiffrée et la clé.

## Implémentation en Python

**Il vous est fortement recommandé de lire l’intégralité de cette partie avant de commencer à coder. Les travaux demandés sont mis en évidence avec une couleur bleue.**

**Dans un fichier que l’on nommera « chiffrementImages.py », implémenter les sous-programmes suivants :**

* Une fonction calculant le « ou exclusif » de deux entiers valant 0, ou 1 passés en paramètre.
* Une fonction prenant en paramètre un entier supposé être entre 0 et 255 et retournant une chaîne de 8 caractères contenant la conversion de cet entier en binaire.
* Une fonction prenant en paramètre deux entiers supposés être entre 0 et 255. Elle effectuera un « ou exclusif » sur chacun des bits des deux paramètres, et retournera le résultat convertit en entier.
* Une fonction prenant en paramètre deux t-uples de 3 entiers. Elle effectuera un « ou exclusif » entre les deux premiers entiers des paramètres, puis entre les deux seconds, puis entre les deux troisièmes, et retournera ces résultats sous forme de t-uple.
* Une procédure prenant un paramètre deux entiers ‘l’ et ‘h’ supposés positifs et un nom de fichier avec une extension correspondant à une image. Elle créera une fenêtre de largeur ‘l’ et de hauteur ‘h’, et affectera une valeur aléatoire (sous forme (R,G,B)) à chacun de ses pixels. Enfin, elle enregistrera cette image dans le répertoire courant sous le nom passé en paramètre.
* Une procédure prenant en paramètre deux entiers ‘l’ et ‘h’ supposés positifs et trois noms de fichiers avec des extensions correspondant à des images de largeur ‘l’ et de hauteur ‘h’. Elle chiffrera la première image avec une clé qui sera la seconde image, et enregistrera le résultat sous le troisième nom passé en paramètre.

**Utilisation de ces procédures et fonctions :**

* Déchiffrer l’image ‘MPchiffre.bmp’ avec la clé ‘alea.bmp’.
* Chiffrer une image de votre choix avec une clé générée aléatoirement et joignez la clé et l’image chiffrée à votre projet.

**Deux petites questions :**

* Quel format d’image faut-il utiliser ?
* Est-il intelligent d’utiliser deux fois la même clé ? Pourquoi ?

# DISSIMULATION D’IMAGES

## Généralités sur cette partie

**Remarque importante**: aucun code n’est demandé dans ce paragraphe qui n’est qu’explicatif.

Le but de cette partie est d'implémenter un algorithme symétrique de masquage d'images. La clé sera elle même une image que les correspondants auront échangée en amont. Cet algorithme fonctionne avec un paramètre k réel positif.

L'image à masquer et la clé devront avoir les mêmes dimensions. On masquera un pixel de l'image en utilisant le pixel situé au même emplacement dans la clé.

Pour masquer un pixel, on procède comme suit pour chacune des composantes R, G et B :

* à la valeur de la composante de la clé, on ajoute (1/k) multiplié par la valeur de la composante de l'image à masquer.
* si ce résultat dépasse 255 on le tronque à 255.

L'image masquée sera alors constituée de tous ces pixels.

La reconstitution de l’image initiale se fera exactement de la même façon, en considérant l’image masquée et la clé. Au lieu d’effectuer un calcul du type , on calculera .

## Implémentation en Python

**Il vous est fortement recommandé de lire l’intégralité de cette partie avant de commencer à coder. Les travaux demandés sont mis en évidence avec une couleur bleue.**

**Dans un fichier que l’on nommera « masquageImages.py », implémenter les sous-programmes suivants :**

* Une fonction prenant en paramètres trois entiers ‘n’, ‘m’ et ‘op’, et un réel ‘k’. Si ‘op’ vaut 1 elle retournera , sinon elle retournera .
* Une fonction prenant en paramètre un entier ‘op’, un réel ‘k’ et deux t-uples de 3 entiers. Elle appliquera la fonction précédente sur les deux premiers entiers des t-uples, puis sur les deux seconds, puis sur les deux troisièmes et retournera le résultat sous forme de t-uple.
* Une procédure prenant en paramètre deux entiers ‘l’ et ‘h’ supposés positifs, trois noms de fichiers avec des extensions correspondant à des images de largeur ‘l’ et de hauteur ‘h’, un réel ‘k’ et un entier ‘op’ (ces deux derniers paramètres ayant le même sens que précédemment). Cette procédure masquera la première image avec une clé qui sera la seconde image et ce avec le paramètre ‘k’, et enregistrera le résultat sous le troisième nom passé en paramètre.

**Utilisation de ces procédures et fonctions :**

* Démasquer l’image ‘MPcache.bmp’ avec la clé ‘cleMasquage.bmp’ et la valeur k = 64.
* Masquer une image de votre choix avec une clé de votre choix et joignez clé et image masquée à votre projet. Vous préciserez également la valeur du paramètre ‘k’ utilisé.

# DISSIMULATION DE TEXTES

## Généralités sur cette partie

**Remarque importante**: aucun code n’est demandé dans ce paragraphe qui n’est qu’explicatif.

Le but de cette partie est d'implémenter un algorithme symétrique de masquage de texte derrière une image.

On se donne donc une image. On commence par en effectuer un pré-traitement. Chaque composante R, G, B de l’image est remplacée par 254 si elle vaut 255. Elle est conservée dans le cas contraire.

On va ensuite considérer les composantes R, G et B de chaque pixel les unes à la suite des autres (celles du premier pixel de la première ligne, puis celles du second pixel de la première ligne, etc.).

Un caractère du message d’origine sera manipulé avec sa valeur Unicode représentée sous forme d’octet, et sera dissimulé par une suite de huit de ces composantes R, G, B comme suit :

* On va considérer le premier bit avec la première composante, puis le second bit avec la seconde composante, etc.
* Si la composante est paire, on ne la modifie pas si le bit vaut 0 et on lui ajoute 1 si le bit vaut 1.
* Si la composante est impaire, on ne la modifie pas si le bit vaut 1 et on lui ajoute 1 si le bit vaut 0.

L'image dissimulant le texte sera alors constituée de tous ces pixels.

Pour retrouver un texte dissimulé dans une image selon cet algorithme, il conviendra de prendre les composantes R, G, B huit par huit et de constituer un octet selon la parité de leur valeur. Le nombre entier correspondant à cet octet sera la valeur Unicode d’un caractère. On procèdera ainsi, caractère par caractère.

## Implémentation en Python

**Il vous est fortement recommandé de lire l’intégralité de cette partie avant de commencer à coder. Les travaux demandés sont mis en évidence avec une couleur bleue.**

**Dans un fichier que l’on nommera « masquageTextes.py », implémenter les sous-programmes suivants :**

* Une procédure prenant en paramètre deux entiers ‘l’ et ‘h’ et l’ensemble des pixels d’une surface de largeur ‘l’ et de hauteur ‘h’. Elle réalisera un pré-traitement de la surface.
* Une procédure prenant en paramètre deux entiers ‘l’ et ‘h’ supposés positifs, deux noms de fichiers avec des extensions correspondant à des images de largeur ‘l’ et de hauteur ‘h’, et une chaîne de caractère. Cette procédure masquera la chaîne sous la première image, et enregistrera le résultat sous le deuxième nom passé en paramètre.
* Une fonction prenant en paramètre deux entiers ‘l’ et ‘h’ supposés positifs, et un nom de fichier avec une extension correspondant à une image de largeur ‘l’ et de hauteur ‘h’. elle retournera le texte caché sous cette image.

**Utilisation de ces procédures et fonctions :**

* Démasquer le texte caché sous l’image ‘MPtexteCache.bmp’.
* Masquer un texte de votre choix sous une image de votre choix, et joignez le résultat à votre projet.