Attaque de Compatibilité contre la Stéganographie JPEG

Etienne LEVECQUE Patrick BAS Jan BURTORA
Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL Lille, France

Résumé – Cet article présente une nouvelle attaque de compatibilité pour détecter un message stego inséré dans le domaine DCT d'une image JPEG pour un facteur de qualité de 100 (QF100). Chaque bloc DCT d'une image cover aura toujours un antécédent dans le domaine pixel, mais ce n'est pas forcément le cas pour une image stego. Bien que théoriquement cette propriété de non-surjectivité permette de construire un détecteur sans fausse alarme, la complexité du problème est trop grande pour y arriver en pratique. Cependant, il est possible de chercher un antécédent avec un problème d'optimisation. Une attaque temporelle peut s'appuyer sur le budget temps alloué à la résolution de ce problème permettant d'avoir une précision dépendant du temps.

Abstract – This paper presents a new compatibility attack to detect a steganographic message embedded in the DCT domain of a JPEG image at QF100. Every DCT block in a cover image will always have an antecedent in the pixel domain, but this is not necessarily the case for a stego image. Although theoretically this non-surjectivity property would allow the design of a detector without false alarms, the complexity of the problem is too big to achieve this in practice. However, we can look for an antecedent with an optimisation problem. Playing with the time budget given to the solver to find a solution, we can design a timing attack that discriminates stego from cover with time-dependent accuracy.

1 Introduction

Les attaques de compatibilité sont un type d'algorithme dans la boîte à outils des experts en stéganalyse. Elles sont considérées comme universelles dans le sens où elles ne ciblent pas un algorithme d'insertion particulier, mais un type de cover. L'idée principale est de trouver une propriété présente dans chaque cover du même type. Si cette propriété n'est pas présente dans une image, nous pouvons en déduire, avec un très faible degré d'incertitude, qu'elle cache un message.

Le format JPEG utilise la transformée en cosinus discrète (DCT) et les fonctions d'arrondi pour compresser une image. Ces opérations créent des propriétés utiles pour les attaques de compatibilité. La littérature relative à ces attaques sur les images JPEG peut être divisée en deux catégories principales : l'une basée sur les propriétés mathématiques et l'autre basée sur les modèles statistiques.

Dans le domaine des images JPEG, Fridrich *et al.* [3] est un exemple d'une propriété mathématique. La compatibilité étudiée est celle des stego après insertion dans le domaine pixels, cette modification crée des blocs pixels n'ayant aucun antécédent dans le domaine DCT. Notre approche est très similaire mais le domaine d'insertion n'est pas le même. Un exemple de modèle statistique est présenté par Butora *et. al.* [1] et concerne l'erreur d'arrondi sur les pixels après décompression. Une caractéristique dont le modèle est connu et stable à QF 100 et 99, ce qui permet de discriminer les stegos dont la distribution est différente.

Ce papier présente une nouvelle attaque par compatibilité utilisant une propriété mathématique des images JPEG à QF100. Elle vise à détecter les messages insérés dans le domaine DCT peu importe l'algorithme utilisé. La section 2 introduira la compression JPEG utilisée, la section 3 définira la théorie de cette attaque et la section 4 présentera l'attaque temporelle qui en découle.

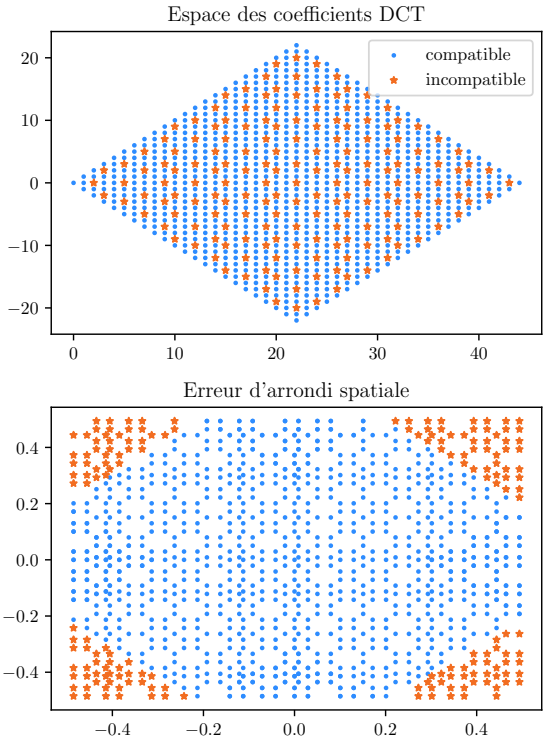

FIGURE 1 : Exemple jouet en utilisant des blocs JPEG de taille $(1, 2)$. Les points bleus sont des blocs DCT compatibles obtenus par compression de toutes les paires de pixels dont la valeur est entre 0 et 32. Les étoiles oranges sont des blocs DCT sans aucun antécédent dans le domaine spatial, ils sont donc incompatibles. La distribution de l'erreur d'arrondi spatial (en bas) peut être utilisée pour détecter les blocs incompatibles mais la forme de l'espace devient plus compliquée pour des blocs de taille $(8, 8)$.

2 Compression JPEG

Les notations introduites dans cette section seront utilisées dans tout le papier. La compression JPEG agit de manière indépendante sur tous les blocs $(8, 8)$ d'une image pixel. Notons $\mathbf{x} \in [\![0; 255]\!]^{64}$ un tel bloc sous forme de vecteur. La transformée en cosinus discrète (DCT) sera représentée par la matrice orthonormale $\mathbf{M}$. À QF100, l'étape de compression correspond à arrondir un flottant vers un entier. Pour ce faire les compresseurs utilisent des fonctions différentes parmi les fonctions d'arrondi ou de troncature. Dans notre cas nous supposerons qu'il s'agit de la fonction d'arrondi à l'entier vers l'infini (donc $[-0.5] = -1$ et $[0.5] = 1$ avec $[\cdot]$ l'opérateur d'arrondi). Les étapes de la compression puis de la décompression peuvent se résumer ainsi :

$$\begin{aligned}
\mathbf{d} &= \mathbf{M}\mathbf{x} && \in \mathbb{R}^{64} && \text{Coefficients DCT flottants,}\\
\mathbf{c} &= [\mathbf{d}] && \in \mathbb{Z}^{64} && \text{Coefficients DCT entiers,}\\
\mathbf{y} &= \mathbf{M}^T\mathbf{c} && \in \mathbb{R}^{64} && \text{Bloc décompressé flottant.}
\end{aligned}$$

Et on peut extraire des erreurs d'arrondi utiles à notre modèle comme suit :

$$\begin{aligned}
\mathbf{e} &= [\mathbf{y}] - \mathbf{y} && \text{Erreur d'arrondi domaine spatial,}\\
\mathbf{u} &= [\mathbf{d}] - \mathbf{d} && \text{Erreur d'arrondi domaine DCT,}
\end{aligned}$$

On fera aussi l'hyptohèse que le compresseur ne modifie pas les coefficients DCT après arrondi. En particulier, la méthode n'est pas adaptée au compresseur Mozjpeg.

3 Attaque par compatibilité : Théorie

3.1 Non-surjectivité de la compression JPEG

À QF100 l'opération de quantification est une division de chaque coefficient DCT par 1. Il est donc possible de construire des compressions JPEG jouets avec des tailles de blocs variables (n, m) avec $1 \leq n, m \leq 8$. En particulier, la figure 1 représente les coefficients DCT pour la taille $(1, 2)$. Il y a les points bleus qui proviennent de blocs pixels compressés et il y a les étoiles oranges qui sont des positions dans l'espace des coefficients DCT qui ne sont pas possibles à atteindre. Ce sont des blocs incompatibles avec la compression JPEG.

Il est tout à fait possible de trouver de tels blocs dans toutes les dimensions jusqu'à $n, m \leq 7$. Au delà, la complexité devient trop grande. Il est donc raisonnable d'assumer cette propriété pour des blocs $(8, 8)$. Nous ferons donc l'hypothèse que la compression JPEG à QF100 n'est pas surjective. C'est-à-dire qu'il existe des blocs DCT incompatibles dans l'espace $\mathbb{Z}^{64}$ qui n'ont pas d'antécédent dans le domaine pixel $[\![0; 255]\!]^{64}$ par la compression JPEG.

Ainsi, si on trouve un tel bloc incompatible qui n'a aucun antécédent, on peut être sûr qu'il a été modifié dans le domaine DCT. Sinon on ne peut rien dire donc par défaut nous supposerons que c'est un block cover.

3.2 Ensemble des antécédents

Comment prouver qu'un bloc DCT n'a pas d'antécédent ? Commençons par décrire l'ensemble des antécédents avec les

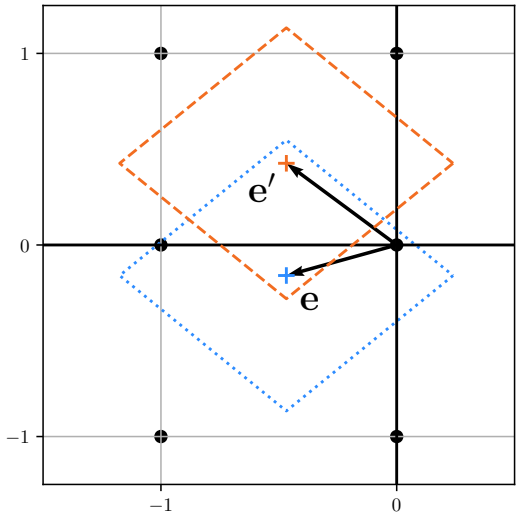

FIGURE 2 : Exemple jouet avec des blocs de taille $(1, 2)$. Le vecteur $\mathbf{e}$ est l'erreur d'arrondi spatial d'un bloc cover et $\mathbf{e}'$ est l'erreur d'arrondi spatial du même bloc après avoir inséré un message stego dedans. Le carré en pointillés bleus et le carré en tirets oranges représentent les contraintes de l'ensemble des erreurs de compression possible $\mathcal{S}$ dans chacun des cas. Notez qu'il n'y a aucun point de coordonnées entières dans le carré en tirets oranges prouvant que ce bloc est incompatible.

informations connues dans l'image JPEG compressée. Les seuls éléments à notre disposition sont les coefficients DCT compressés $\mathbf{c}$. Il est donc possible d'en extraire l'image décompressée $\mathbf{y}$ et l'erreur d'arrondi spatial $\mathbf{e}$. On peut ensuite faire un premier calcul préliminaire pour avoir plus d'information sur les antécédents :

$$\begin{aligned}
[\mathbf{y}] - \mathbf{x} &= [\mathbf{y}] - \mathbf{M}^T\mathbf{d},\\
&= \mathbf{e} + \mathbf{M}^T\mathbf{u}.
\end{aligned} \tag{1}$$

Notons $\mathbf{k} = [\mathbf{y}] - \mathbf{x} = \mathbf{e} + \mathbf{M}^T\mathbf{u}$ l'erreur de compression JPEG. D'après la première égalité, ce terme est un vecteur de valeurs entières donc $\mathbf{k} \in \mathbb{Z}^{64}$. Dans l'autre partie de cette égalité, nous connaissons $\mathbf{e}$ et $\mathbf{M}^T$. On sait de plus que $\mathbf{u} = \mathbf{M}(\mathbf{k} - \mathbf{e}) \in [-0.5; 0.5]^{64}$ car c'est une erreur d'arrondi. On peut donc écrire l'ensemble des erreurs de compression avec des contraintes linéaires :

$$\mathcal{S} = \left\{ \mathbf{k} \in \mathbb{Z}^{64} \;\middle|\; \begin{array}{l} -0.5 < (\mathbf{M}(\mathbf{k} - \mathbf{e}))_i \leq 0.5, \text{ si } d_i \geq 0,\\ -0.5 \leq (\mathbf{M}(\mathbf{k} - \mathbf{e}))_i < 0.5, \text{ si } d_i < 0. \end{array} \right\}$$

Un bloc n'est pas compatible si son ensemble des erreurs de compression est vide. Pour décider, il faut soit trouver un élément de $\mathcal{S}$, soit prouver que $\mathcal{S} = \{\varnothing\}$. Cependant, il existe beaucoup de solutions possibles. En effet, si on regarde les erreurs de compression sur des images JPEG, on peut constater empiriquement que $\mathbf{k} \in \{-1, 0, 1\}^{64}$ ce qui donne 3^{64} candidats potentiellement dans $\mathcal{S}$. D'après nos observations, il est très rare d'avoir des solutions triviales, la plupart des solutions ont au moins 4 valeurs non nulles.

3.3 Exemple en 2D

Afin d'illustrer cet ensemble des antécédents, nous allons utiliser un exemple jouet en 2D pour pouvoir le visualiser. Sup-

posons qu'on travaille sur un bloc de 2 pixels dont la valeur est $\mathbf{x} = (195, 84)$. Les coefficients DCT associés après avoir soustrait 128 sont $\mathbf{c} = (16, 78)$ et le bloc décompressé a pour valeur $[\mathbf{y}] = (194, 84)$. L'erreur d'arrondi spatial associée à ce bloc est environ $\mathbf{e} = (-0.46, -0.16)$. Ce point est représenté par le vecteur e dans la figure 2. Le cube en pointillés bleus autour de e représente l'espace des erreurs de compression possible $\mathcal{S}$. On constate qu'il y a deux points entiers dedans : $\mathcal{S} = \{(0, 0), (-1, 0)\}$. Si on utilise ces erreurs pour construire des antécédents, on obtient deux blocs pixels différents : $(194, 84)$ et $(195, 84)$. Après compression de ces deux blocs pixels on vérifie qu'il donne à nouveau $\mathbf{c}$ prouvant que ce sont deux antécédents.

Si maintenant on insère un message stégo dans le domaine DCT de notre bloc : $\mathbf{m} = (1, -1)$, la décompression de ce nouveau bloc DCT donne $[\mathbf{y}'] = (194, 86)$ et une erreur d'arrondi différente : $\mathbf{e}' = (-0.47, 0.43)$. On voit sur la même figure que l'espace des erreurs de compression illustré par le carré en tirets oranges ne contient aucun point de coordonnées entières. Ce bloc n'a donc aucun antécédent dans le domaine pixel, il est incompatible.

4 Attaque temporelle

4.1 Trouver des antécédents avec un problème d'optimisation

Prouver que l'ensemble des antécédents est vide est complexe mais si l'ensemble n'est pas vide, trouver un antécédent est beaucoup plus rapide. Pour cela nous utilisons un problème d'optimisation avec les contraintes définissant $\mathcal{S}$. Il est possible d'utiliser n'importe quelle fonction objective puisque seule la faisabilité du problème nous intéresse. Si une solution est trouvée, on ne peut rien dire sur le bloc en question. Mais si l'algorihme prend trop de temps à trouver une solution, il est possible que ce bloc n'ait pas de solution et qu'il soit donc incompatible.

Les contraintes linéaires qui définissent l'ensemble $\mathcal{S}$ ont des inégalités strictes et dépendent du signe des éléments de $\mathbf{d}$ qui n'est pas connu du stéganalyste. Pour résoudre cela nous allons introduire ε très petit (empiriquement fixé à 10^{-5}) pour éviter les inégalités strictes. De plus, nous utiliserons le signe de $\mathbf{c}$ pour approximer le signe de $\mathbf{d}$. Cette approximation peut être fausse si $c_i = 0$ car dans ce cas, d_i peut aussi bien être positif que négatif. Si cela arrive, les contraintes sont possiblement mal définies et la solution n'est alors pas correcte. Il est cependant possible de vérifier à posteriori que la solution du problème d'optimisation permet de reconstruire un antécédent. Si ce n'est pas le cas, on ignore le bloc en question. Dans le cas où ces deux hypothèses (ε suffisamment petit et $\mathbf{d}$ de même signe) sont vraies, on peut alors écrire :

$$\begin{aligned}\mathbf{k} \in \mathcal{S} \Leftrightarrow & \begin{cases} -0.5 < \left(\mathbf{M}(\mathbf{k} - \mathbf{e})\right)_i \leq 0.5, \text{ si } c_i \geq 0, \\ -0.5 \leq \left(\mathbf{M}(\mathbf{k} - \mathbf{e})\right)_i < 0.5, \text{ si } c_i < 0. \end{cases} \\ & \Leftrightarrow \mathbf{A}\mathbf{k} \leq \mathbf{b}_\varepsilon\end{aligned}$$

Avec $\mathbf{A} = \begin{pmatrix} \mathbf{M} \\ -\mathbf{M} \end{pmatrix}$ et $\mathbf{b}_\varepsilon = 0.5 + \mathbf{A}\mathbf{e} - \begin{pmatrix} \boldsymbol{\delta} \\ 1 - \boldsymbol{\delta} \end{pmatrix} \varepsilon$. Le symbole $\boldsymbol{\delta}$ est un masque binaire tel que $\delta_i = 1$ si $c_i \geq 0$, 0 sinon.

4.2 Expérience

Le but de l'expérience est d'évaluer l'attaque temporelle de la compatibilité JPEG en trouvant des antécédents à chaque bloc le plus rapidement possible. Elle a été menée sur le dataset ALASKA2 [2] composé d'images RAW en nuances de gris et de dimension 256×256. On utilise ces images RAW pour contrôler le compresseur JPEG et en particulier la fonction d'arrondi. Un filtre est utilisé pour ignorer les blocs uniformes ou ayant des pixels dont la valeur est soit 0 soit 255. Ces blocs "clippés" peuvent présenter des erreurs d'arrondi supérieur à 0.5 en valeur absolue et les blocs uniformes ont des erreurs d'arrondi spatial nulles.

J-UNIWARD [5] a été utilisé pour créer 100 images stego avec un payload de 0.1 bpnzac (*bit per non-zero AC coefficients*) et 100 autres images cover sont également utilisées.

Afin d'accélérer la recherche de solution, nous apportons deux modifications au modèle. La première modifications est de restreindre l'espace de recherche à l'aide de l'observation empirique discutée plus tôt : $\mathbf{k} \in \{-1, 0, 1\}$. Ensuite, nous ajoutons une fonction objectif qui donne plus d'informations au solveur. L'espace des contraintes est un hypercube de côté 1 auquel une rotation (la DCT) a été appliquée, puis cet hypercube est translaté par le vecteur $\mathbf{e}$. Nous allons donc utiliser la distance euclidienne par rapport au centre de l'espace des contraintes : $\min_{\mathbf{k} \in \{-1,0,1\}} (\mathbf{k} - \mathbf{e})^2$.

Le problème utilisé dans ce modèle d'attaque temporelle est donc le suivant :

$$\begin{aligned} \min_{\mathbf{k} \in \{-1,0,1\}} \quad & (\mathbf{k} - \mathbf{e})^2, \\ \text{s.c.} \quad & \mathbf{A}\mathbf{k} \leq \mathbf{b}_\varepsilon, \end{aligned} \tag{2}$$

Afin de résoudre le problème 2, nous utiliserons le logiciel `Gurobi` [4] à travers le module Python `Gurobipy` avec le paramètre `MIPFocus = 3` qui permet de converger plus rapidement vers une solution.

Chaque image est découpée en bloc dont sont extraits les erreurs d'arrondi spatial pour construire les problèmes 2 correspondant à chaque bloc. `Gurobi` est utilisé pour essayer de résoudre chacun de ces problèmes avec une limite de temps de 100s maximum. Une fois cette limite atteinte le problème peut avoir 3 status possibles :

- *Résolu*, si une solution a été trouvée et qu'elle correspond bien à un antécédent,
- *Faux*, si une solution a été trouvée mais qu'elle ne correspond pas à un antécédent (à cause de l'approximation du signe de $\mathbf{d}$ par le signe de $\mathbf{c}$ comme expliqué à la fin de la section 4.1),
- *Non-résolu*, si aucune solution n'a été trouvée.

En théorie il existe un quatrième status dans le cas d'un bloc stego sans antécédent : *infaisable*. Cependant, le temps nécéssaire pour atteindre un tel status est bien plus grand que le temps disponible dans cette expérience.

4.3 Résultats

La figure 3 représente la distribution du ratio de bloc dont le status est *non-résolu* à la fin de l'expérience. Ce ratio est calculé comme la division entre le nombre de blocs non-résolus

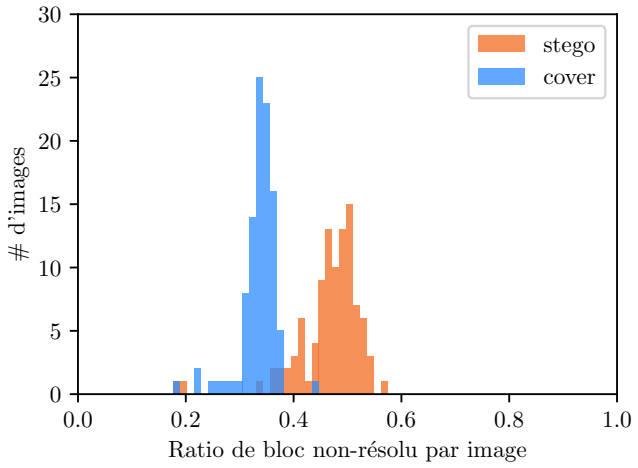

FIGURE 3 : Distribution du ratio de bloc non-résolu après 100s par rapport au nombre de blocs non-ignorés sur 100 images cover et 100 images stego.

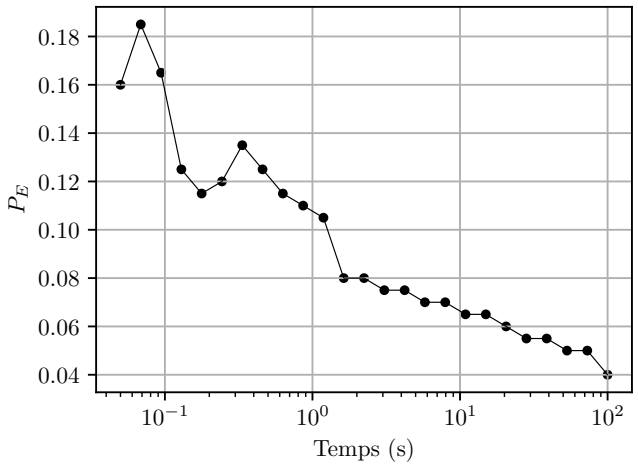

FIGURE 4 : Evolution de la probabilité d'erreur minimale P_E en fonction du budget temps sur 100 images cover et 100 images stego.

et le nombre total de blocs retenus dans l'image après filtrage. On constate que les images stego sont plus longues à être résolues que les images cover. En observant l'évolution de cette distribution dans le temps, on constate aussi que les deux distributions se déplacent vers la gauche. Comportement prévisible puisque le budget temps alloué à la résolution du problème est de plus en plus grand. Dans la limite d'un temps infini, on peut penser que la distribution des cover va devenir un Dirac en 0 et que le ratio pour chaque image stego convergera vers le ratio de bloc sans antécédent dans l'image (donc potentiellement non nul).

La figure 4 représente la probabilité d'erreur minimale à priors égaux (EER) P_E en fonction du temps et permet de constater que plus le temps alloué à la résolution est grand, plus le ratio de blocs non-résolus est un attribut discriminant entre stego et cover. Il faut tout de même noter la progression logarithmique de cette courbe : en effet il faut multiplier le temps par 100 pour diminuer notre erreur par 2 seulement.

5 Conclusion

Ce papier propose une attaque de stéganalyse permettant de détecter un message inséré dans le domaine DCT d'une image JPEG à QF100. Si un bloc DCT n'a pas d'antécédent dans le domaine pixel, cela prouve qu'il est incompatible car il a été modifié. Prouver une telle incompatibilité n'est pas évident, nous avons donc proposé une attaque temporelle à l'aide d'un problème d'optimisation dans lequel on cherche un antécédent à chaque bloc. Les résultats sont prometteurs car la précision dépend du temps d'execution, mais celui-ci devient vite très grand.

Un travail futur pourrait être mené sur l'ingénierie de cette méthode afin de la rendre plus rapide et donc précise. Il serait aussi intéressant de tester cette attaque sur d'autres compresseurs avec des fonctions d'arrondi et de troncatures différentes ainsi que sur des facteurs de qualité plus faibles. En effet, la formulation du problème permet d'utiliser n'importe quel QF, mais la propriété de non-surjectivité va devenir fausse à partir d'un certain QF et rendra donc cette méthode inutile.

Pour finir, du côté de la stéganographie, il est naturel de se demander si c'est possible de prendre en compte (et à quel coût) cette notion de compatibilité pour être immunisé à cette attaque. Cette question n'a pas encore été abordée dans notre travail.

6 Remerciements

Les travaux présentés dans ce papier ont également reçu un financement du programme H2020 de l'Union Européenne, accord de financement No 101021687, projet "UNCOVER".

Références

[1] Jan BUTORA et Jessica FRIDRICH : Reverse jpeg compatibility attack. *IEEE Transactions on Information Forensics and Security*, 15:1444–1454, 2019.

[2] Rémi COGRANNE, Quentin GIBOULOT et Patrick BAS : ALASKA-2 : Challenging Academic Research on Steganalysis with Realistic Images. *In IEEE International Workshop on Information Forensics and Security*, New York City (Virtual Conference), United States, décembre 2020.

[3] Jessica FRIDRICH, Miroslav GOLJAN et Rui DU : Steganalysis based on jpeg compatibility. *In Multimedia Systems and Applications IV*, volume 4518, pages 275–280. SPIE, 2001.

[4] GUROBI OPTIMIZATION, LLC : Gurobi Optimizer Reference Manual, 2023.

[5] Vojtěch HOLUB, Jessica FRIDRICH et Tomáš DENEMARK : Universal distortion function for steganography in an arbitrary domain. *EURASIP Journal on Information Security*, 2014(1):1–13, 2014.