# Multi-Cipher Cracker v2.1 – Revue de code & Documentation technique

## 1. Objectif du projet

Multi-Cipher Cracker est une application **HTML/JavaScript autonome** visant l'analyse et le cassage assisté de chiffrements classiques (pré-informatiques). L'outil combine :

- **Analyse statistique** (Indice de Coïncidence, fréquences)
- **Heuristiques linguistiques** (segmentation par dictionnaire FR/EN)
- **Bruteforce raisonné** (réduction d'espace de clés)
- **Parallélisation navigateur** via *Web Workers*

L'objectif n'est pas le cassage exhaustif cryptanalytique académique, mais une **exploration rapide et pragmatique** des pistes plausibles.

---

## 2. Architecture générale

### 2.1 Séparation des responsabilités

| Composant | Rôle |
|---|---|
| `index.html` (main thread) | UI, interaction utilisateur, analyse rapide, orchestration |
| `workerCode` (Web Worker) | Calcul intensif, bruteforce, scoring |
| CSS intégré | Rendu UI cyber/terminal |

**État de l'art respecté** : - Pas de calcul lourd sur le thread UI - Communication asynchrone par `postMessage` - Code *single-file* déployable offline

---

## 3. Pipeline de traitement

### 3.1 Vue d'ensemble

```
Texte chiffré
   ↓
Analyse rapide (IC, stats, lisibilité)
   ↓
Sélection dynamique des chiffrements
   ↓
Worker : génération clés / déchiffrement
   ↓
Scoring linguistique + fréquentiel
   ↓
```

```
Déduplication + tri
   ↓
Top résultats
```

---

# 4. Analyse cryptostatistique

## 4.1 Indice de Coïncidence (IC)

Principe :

$$IC = \frac{\sum f_i(f_i - 1)}{N(N-1)}$$

Interprétation empirique :

| IC | Interprétation |
|---|---|
| ~0.038 | Aléatoire |
| ~0.055 | Vigenère / polyalphabétique |
| ~0.065 | Langue naturelle |

Utilisation : - Orientation du choix de chiffrements - Estimation des longueurs de clé

## 4.2 Estimation des longueurs de clé

Méthode : - Découpage du texte modulo *k* - Calcul IC par colonne - Pondération et seuil >1%

➡️Conforme aux pratiques classiques (Kasiski/IC heuristique).

---

# 5. Score linguistique (cœur du système)

## 5.1 Score fréquentiel

Distance L1 entre fréquences observées et cibles (FR/EN) :

```
score = max(0, 100 − Σ |f_obs − f_ref|)
```

## 5.2 Segmentation lexicale dynamique

Algorithme DP (programmation dynamique) : - Recherche de mots connus (longueur $\leq$10) - Gain pondéré par $l^2$ (mots longs favorisés) - Reconstruction arrière

Résultat : - Liste de mots reconnus - Taux de couverture (%)

### 5.3 Fusion des scores

```
score_final = 0.7 × couverture_dictionnaire
            + 0.3 × score_frequence
```

**État de l'art** : - Très proche des heuristiques utilisées dans les CTF crypto - Compromis performant sans NLP lourd

---

## 6. Implémentation des chiffrements

### 6.1 Substitution simple

| Chiffrement | Méthode |
|---|---|
| César | Bruteforce 25 décalages |
| Atbash | Involution directe |
| Affine | Couples (a,b) avec a ∈ $Z_{(26)}$* |

➡️Gestion correcte des inverses modulaires.

### 6.2 Polyalphabétique

#### Vigenère / Beaufort

Optimisations : - Découpage par colonnes - Sélection des *N* meilleurs décalages par colonne - Génération combinatoire limitée

➡️Réduction exponentielle de l'espace de clés (état de l'art pratique).

#### Autokey

- Amorçage par mots clés plausibles
- Extension automatique de clé

⚠️Limite volontaire (dictionnaire) assumée.

### 6.3 Digraphique

#### Playfair

- Grille 5×5 (I/J fusionnés)
- Gestion correcte des doublons et padding

### 6.4 Transposition

| Type | Méthode |
|---|---|
| Rail Fence | Reconstruction inverse |

| Type | Méthode |
| --- | --- |
| Columnar | Calcul offsets colonnes |

---

## 7. Mode CASCADE (méta-heuristique)

Principe : 1. Sélection des meilleurs résultats intermédiaires 2. Application de transformations simples (Atbash, ROT13, Reverse) 3. Acceptation si score ↑

➡️Simule un raisonnement humain multi-étapes.

**Approche moderne** : exploration locale guidée par score.

---

## 8. Parallélisation & performance

### 8.1 Web Worker

- Calcul isolé
- UI toujours fluide
- Progression incrémentale

### 8.2 Optimisations notables

- Cache de segmentation (`Map` LRU simplifié)
- Limitation combinatoire (≤500 clés)
- Déduplication par préfixe

➡️Excellent ratio puissance / simplicité.

---

## 9. Revue vs état de l'art

### Points forts

✔️Architecture propre, moderne ✔️Heuristiques pertinentes ✔️Lisibilité du code crypto ✔️Aucun framework, zéro dépendance ✔️Très bon outil pédagogique / CTF

### Limites assumées

⚠️Dictionnaires statiques ⚠️Pas de n-grammes (quadgrams) ⚠️Pas de hill-climbing avancé ⚠️Pas de crypto moderne (AES, RSA…)

---

## 10. Améliorations possibles (optionnelles)

- Ajout quadgrams log-likelihood
- Apprentissage auto des mots

• Détection automatique du type de chiffrement
• Export PDF / JSON
• Mode "forensic" déterministe

---

## 11. Conclusion

Ce code est **solide, cohérent et aligné avec les meilleures pratiques** pour la cryptanalyse classique côté navigateur.

Il se situe clairement **au-dessus d'un simple bruteforce**, sans tomber dans la complexité excessive des frameworks NLP.

♓ Base parfaitement saine pour : - un projet GitHub public - un outil pédagogique - un moteur crypto-CTF

---

*Documentation rédigée dans une optique professionnelle et audit technique.*