Cracker - CRyptographic pACKER

Benjamin CAILLAT

bcaillat(at)security-labs.org

Mastère spécialisé SIS - ESIEA

5 juin 2008

Présentation de la problématique (1)

- Après compromission d'une machine lors d'un pentest, consultant veut uploader des outils pour conforter accès et poursuivre attaque
- Action critique car :
  - si upload outils personnels, ils risquent d'être capturés/volés
  - si upload outils publics, ils risquent d'être reconnus par logiciel protection et faire détecter attaque

Présentation de la problématique (2)

- Une solution : utiliser un programme externe chiffrant l'exécutable et le déchiffrant à la volée : cracker (CRyptographic pACKER)

Présentation de la problématique (2)

- Une solution : utiliser un programme externe chiffrant l'exécutable et le déchiffrant à la volée : cracker (CRyptographic pACKER)
- Principe général :
  - Sur poste du consultant : programme à protéger et ses ressources sont chiffrés avec une clé dérivée d'une passphrase
  - Fichiers chiffrés + cracker.exe sont uploadés sur le serveur
  - cracker est lancé avec la passphrase en paramètre ; il déchiffre le programme en mémoire uniquement et l'exécute

Présentation de la problématique (2)

- Une solution : utiliser un programme externe chiffrant l'exécutable et le déchiffrant à la volée : cracker (CRyptographic pACKER)
- Principe général :
  - Sur poste du consultant : programme à protéger et ses ressources sont chiffrés avec une clé dérivée d'une passphrase
  - Fichiers chiffrés + cracker.exe sont uploadés sur le serveur
  - cracker est lancé avec la passphrase en paramètre ; il déchiffre le programme en mémoire uniquement et l'exécute
- Objectif de la protection :
  Rendre impossible la récupération de l'outil ou même son identification, même en disposant :
  - des fichiers chiffrés
  - de la ligne de commande utilisée lors de l'exécution

Principe de fonctionnement détaillé (1)

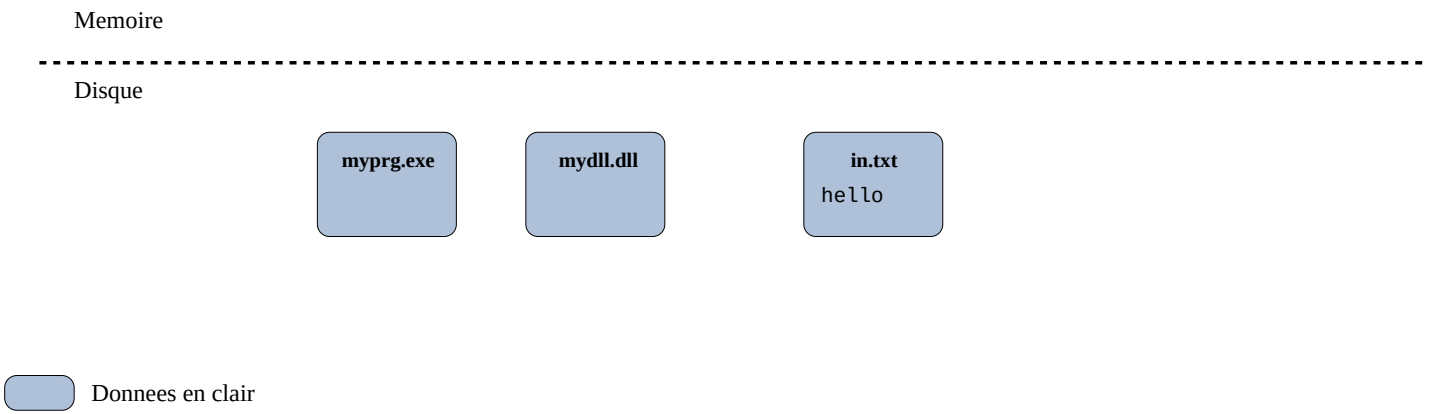


Fig.: Exécution du programme de test sans protection

Principe de fonctionnement détaillé (1)

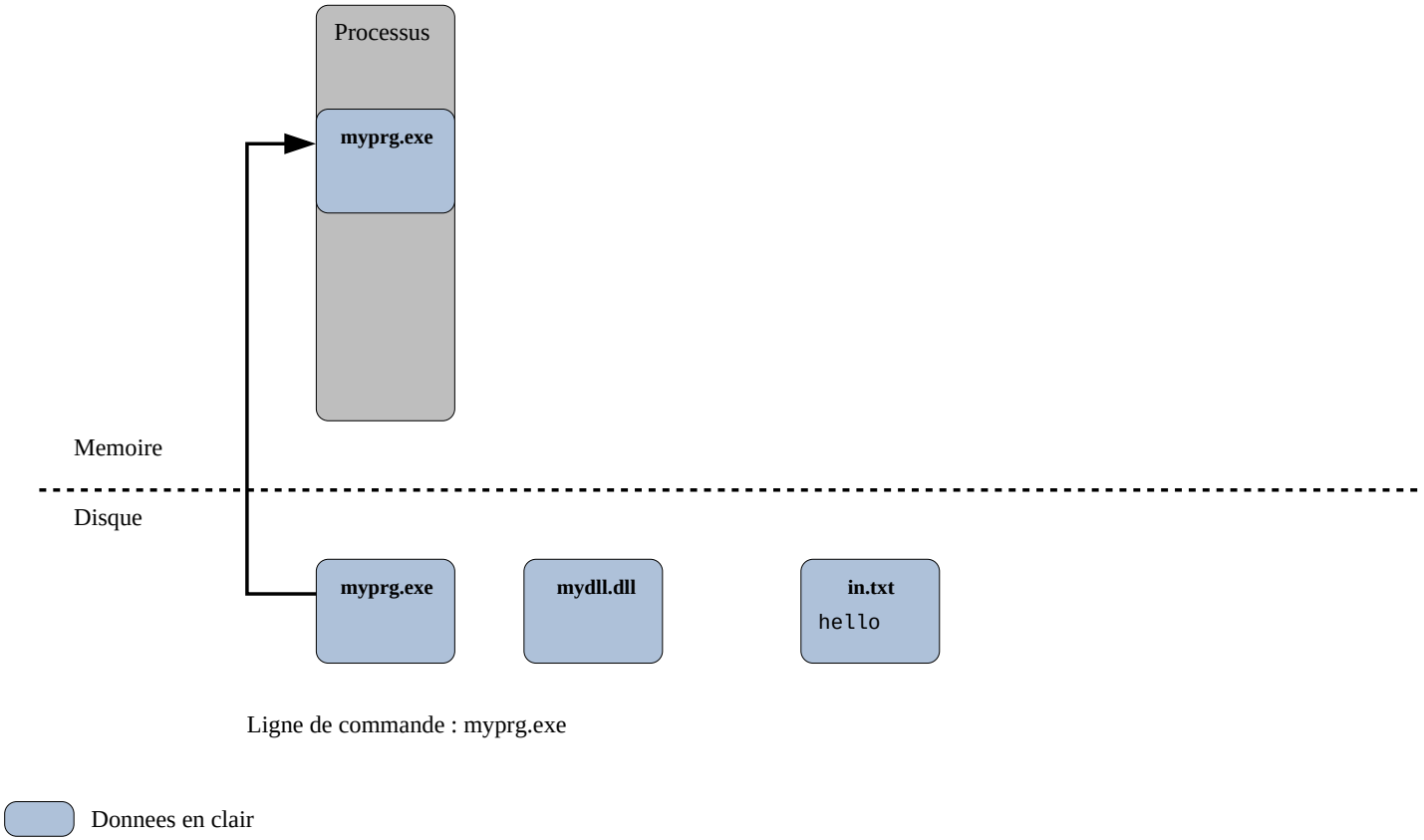


Fig.: Exécution du programme de test sans protection

Principe de fonctionnement détaillé (1)

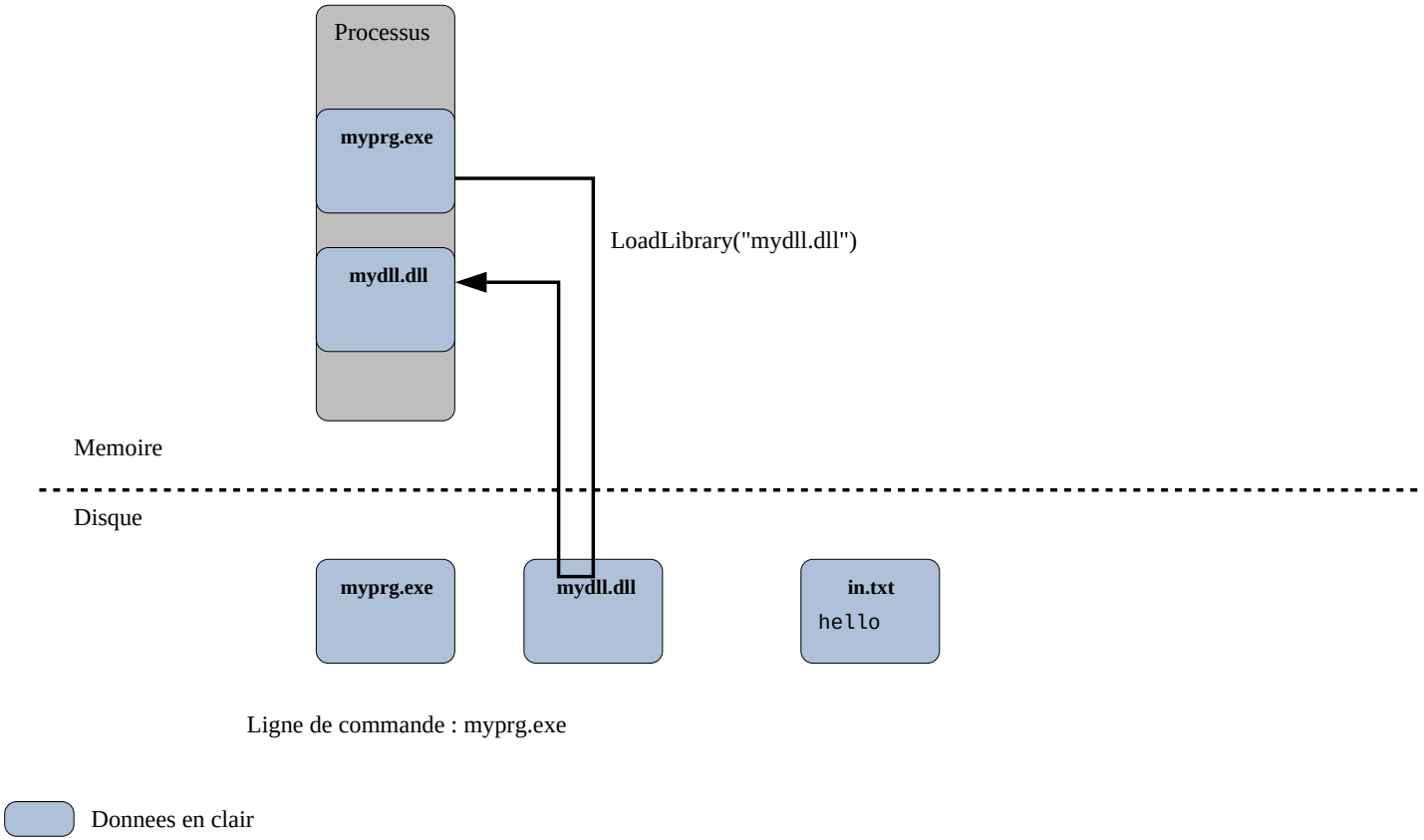


Fig.: Exécution du programme de test sans protection

Principe de fonctionnement détaillé (1)

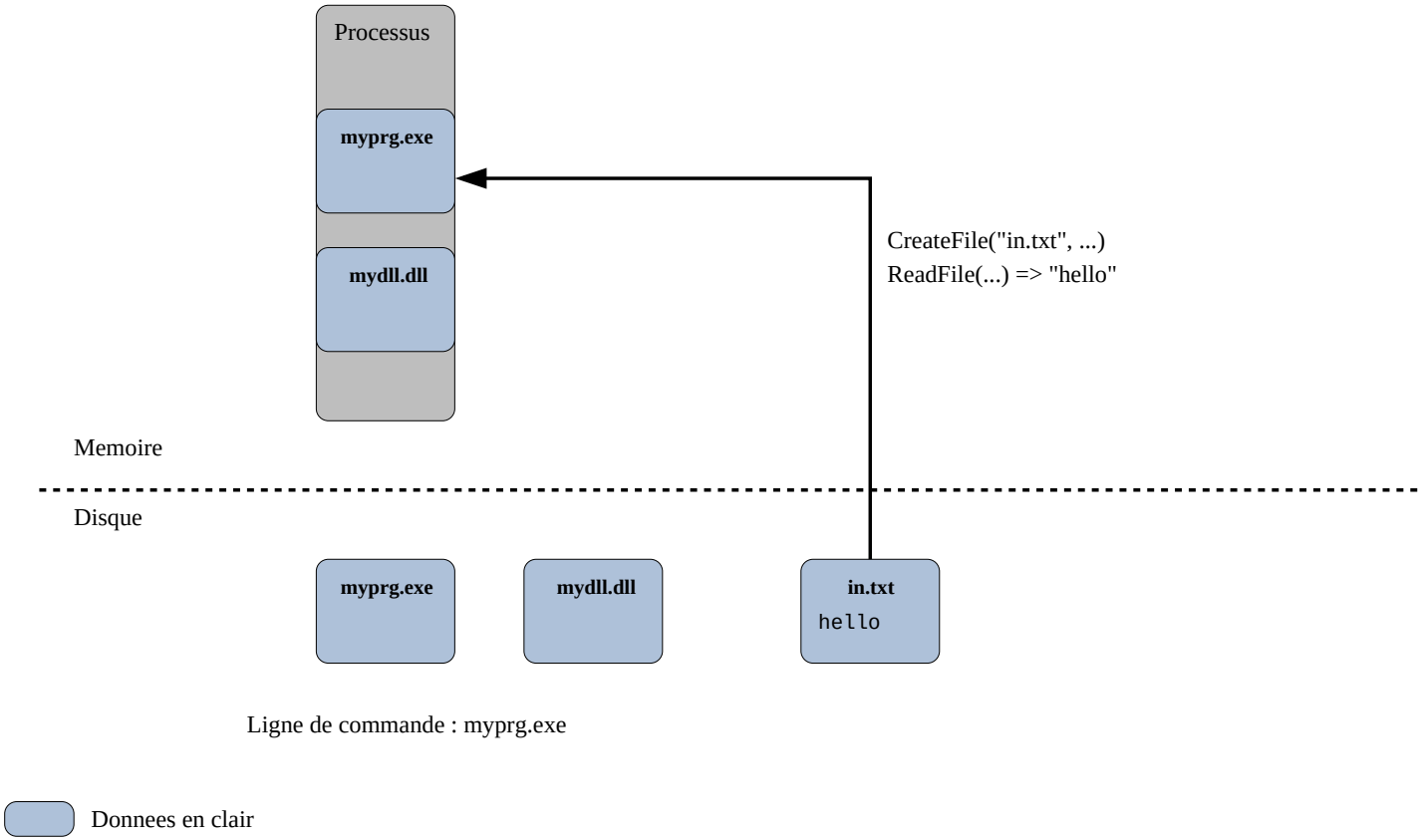


Fig.: Exécution du programme de test sans protection

Principe de fonctionnement détaillé (1)

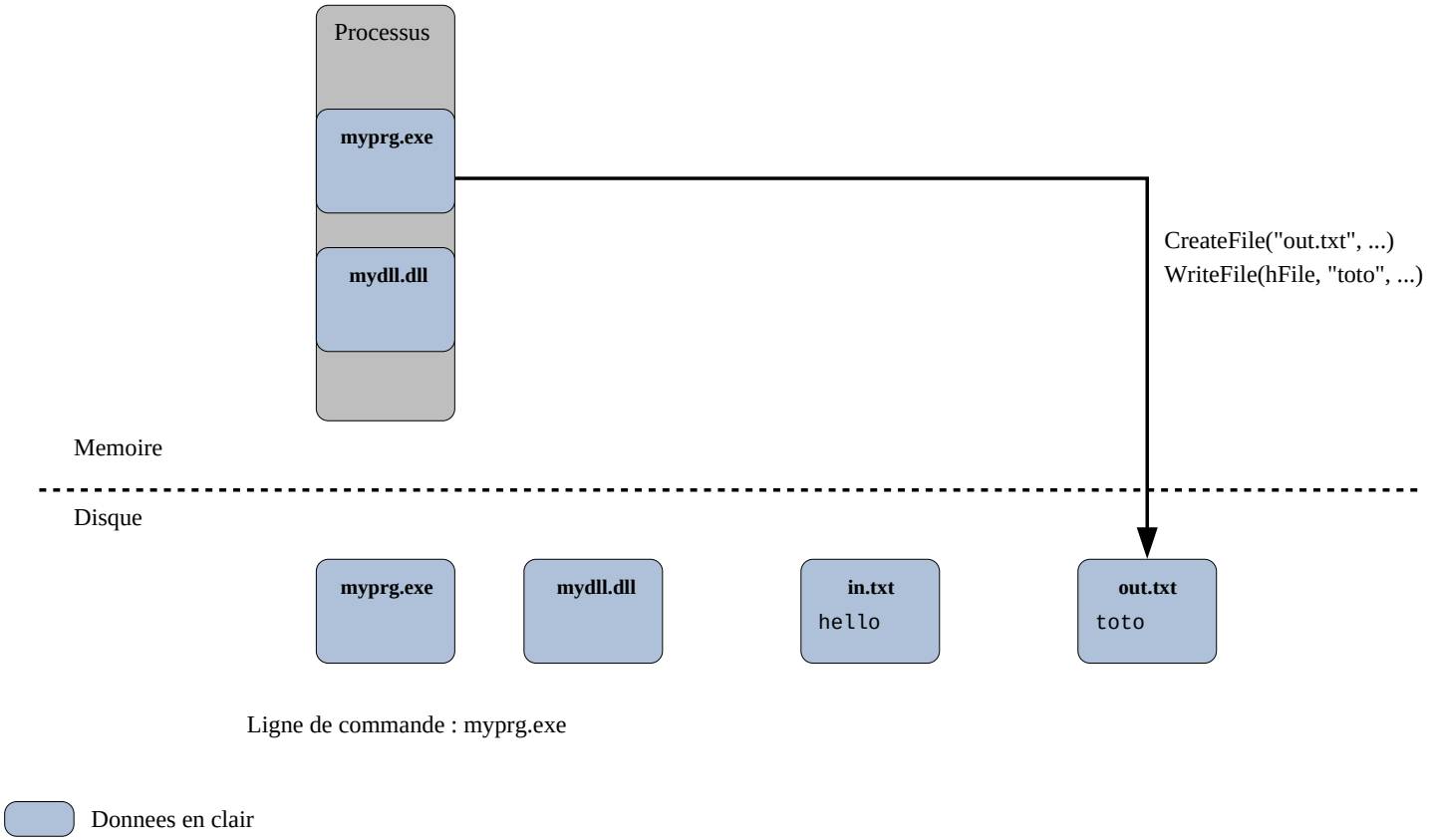


Fig.: Exécution du programme de test sans protection

Principe de fonctionnement détaillé (1)

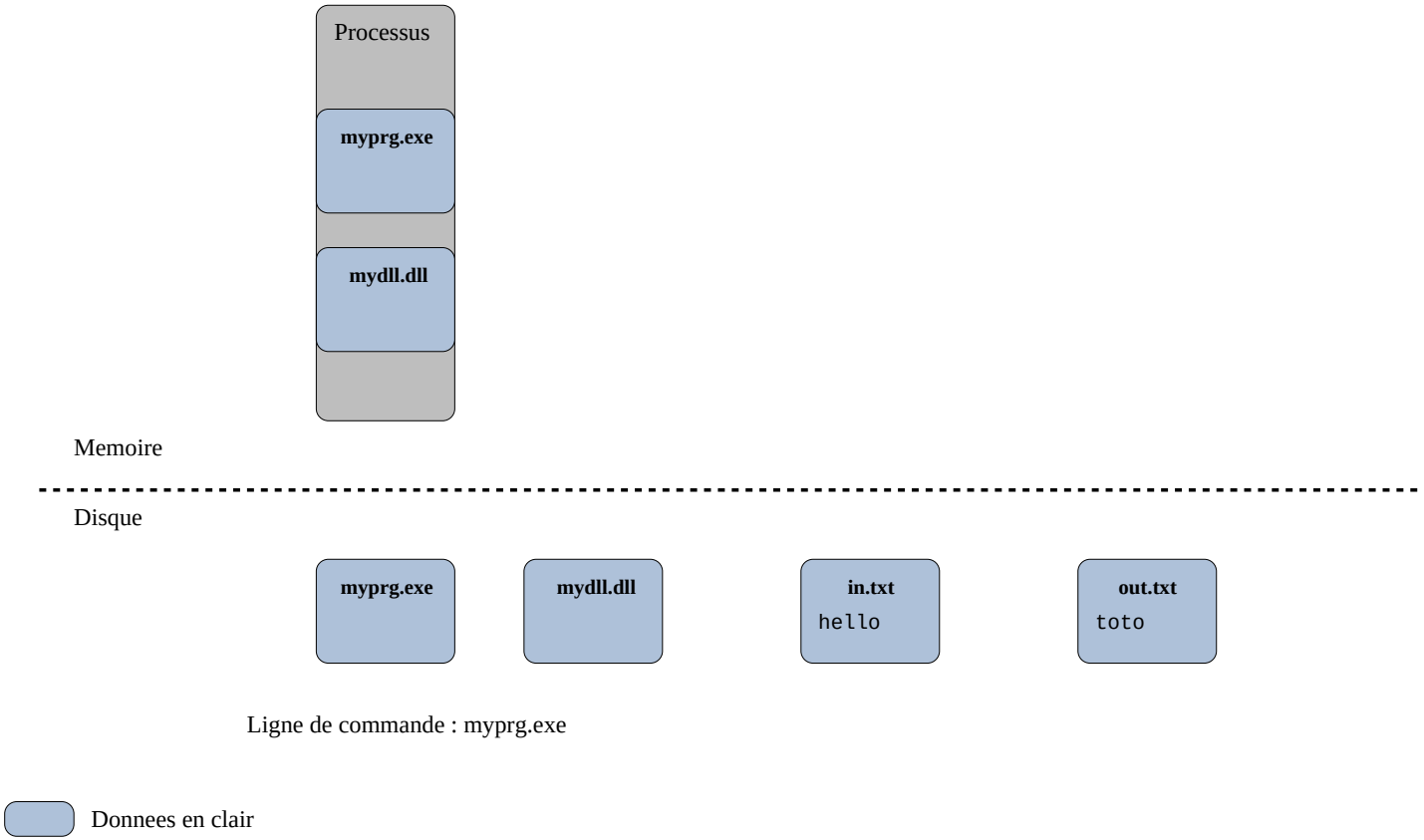


Fig.: Exécution du programme de test sans protection

Principe de fonctionnement détaillé (2)

Sur le poste du consultant : génération de l'archive chiffrée

- Création d'une archive chiffrée avec clé symétrique S contenant l'exécutable et les dlls dont il dépend
- Chiffrement des fichiers ressources spécifiés avec clé symétrique S
- Création d'un fichier de configuration chiffré avec clé symétrique S, utilisé pour la gestion interne
- Création d'un fichier contenant la clé symétrique S et un secret, chiffré par la clé jetable n°1

Principe de fonctionnement détaillé (3)

Sur le serveur : lancement du programme

- Lancement de cracker.exe $\Rightarrow$ création d'un processus
- Déchiffrement de l'archive et remappage dans la mémoire de l'exécutable protégé + des dlls dépendantes
- Installation d'une couche d'interception des appels de fonctions (API hooking par patch du header en user-land)

Principe de fonctionnement détaillé (3)

Sur le serveur : lancement du programme

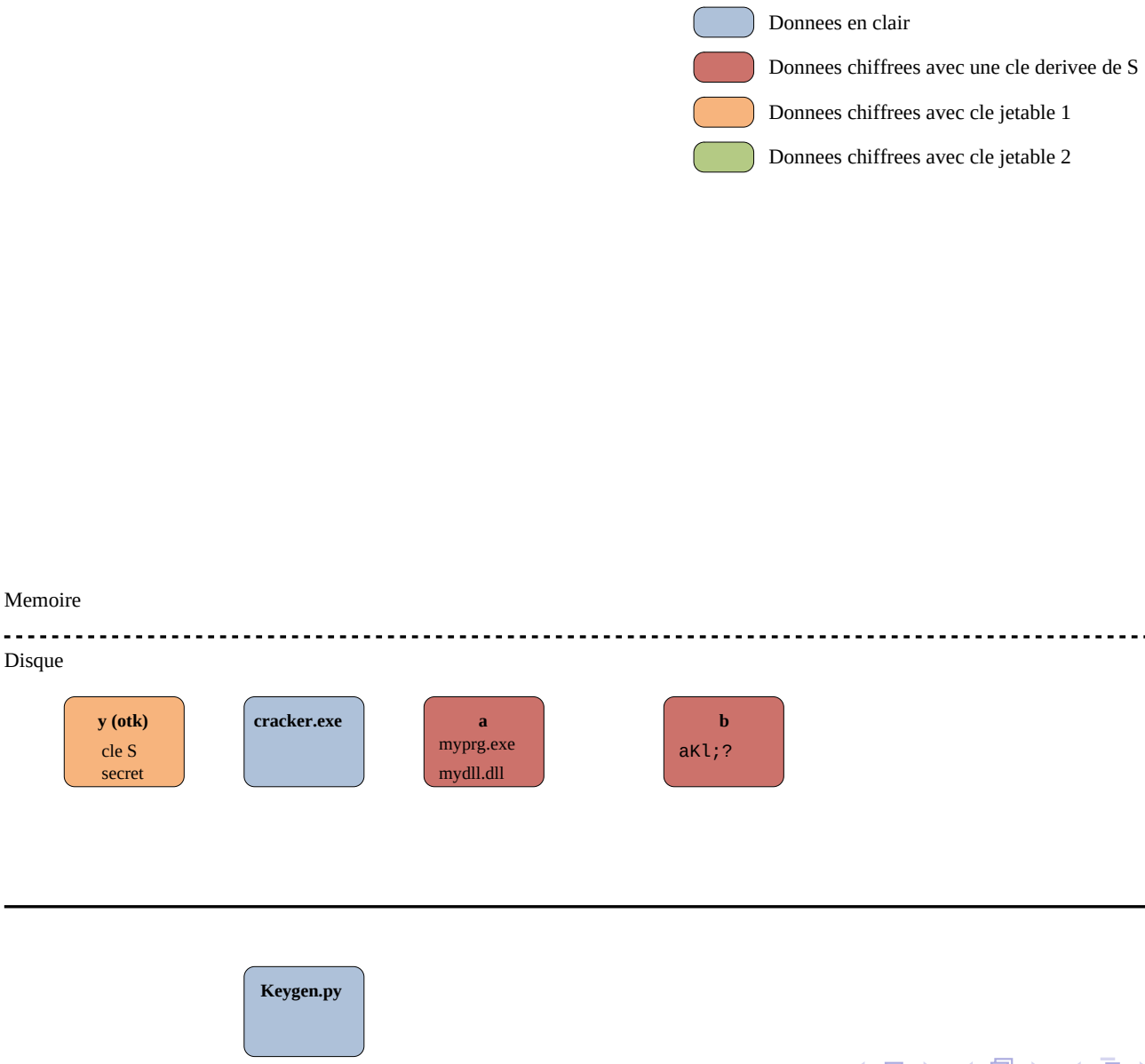
- Lancement de cracker.exe ⇒ création d'un processus
- Déchiffrement de l'archive et remappage dans la mémoire de l'exécutable protégé + des dlls dépendantes
- Installation d'une couche d'interception des appels de fonctions (API hooking par patch du header en user-land)

Sur le serveur : exécution du programme

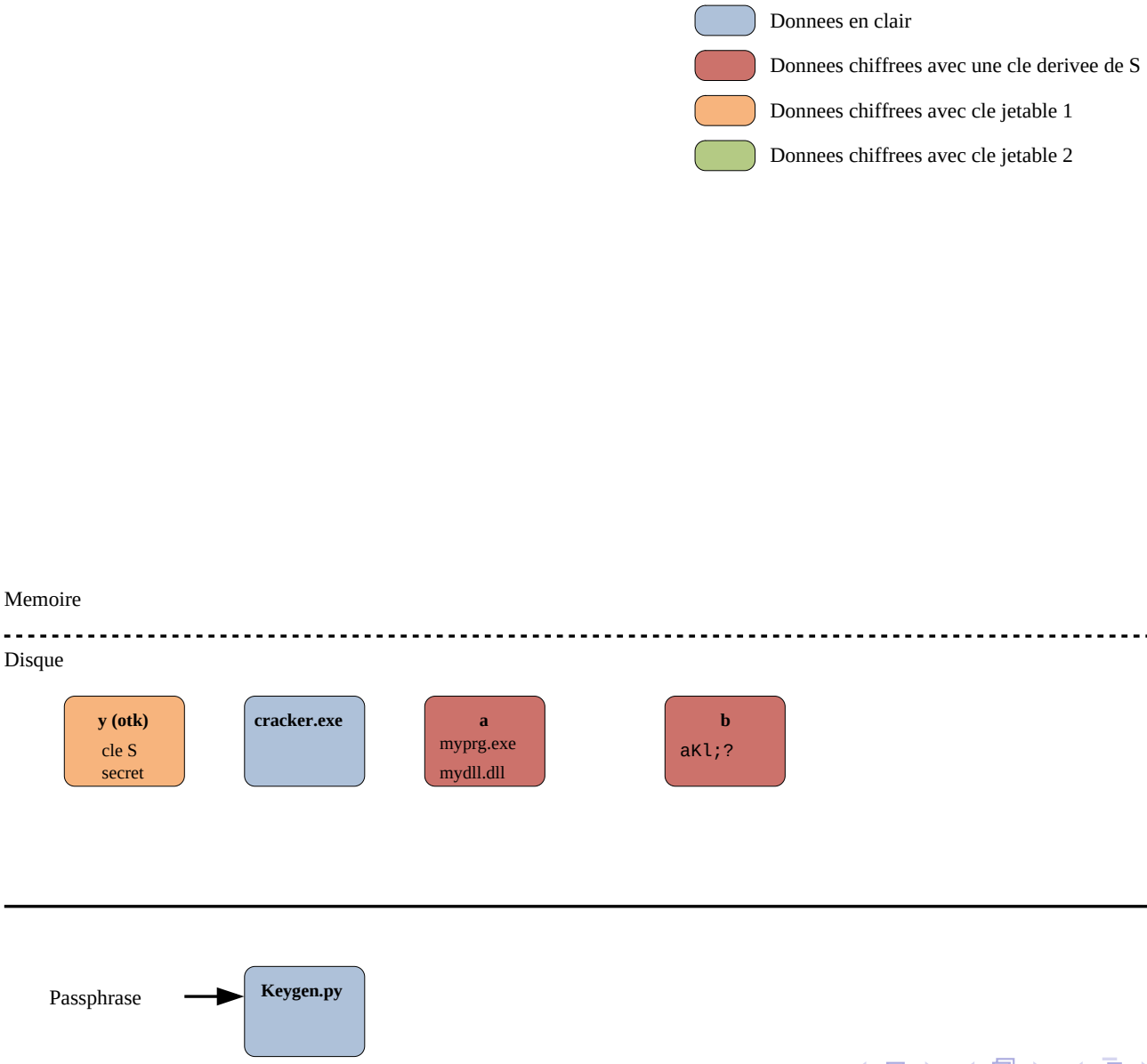
Interception des appels et modification à la volée des paramètres / des résultats des fonctions.
Exemple :

- Chargement d'une dll : redirection pour chargement ait lieu à partir de l'archive chiffrée
- Ouverture d'un fichier : patch du nom du fichier
- Ecriture/lecture du contenu d'un fichier : chiffrement/déchiffrement des données écrites/lues

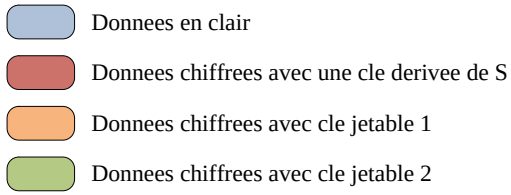
Principe de fonctionnement détaillé (3)

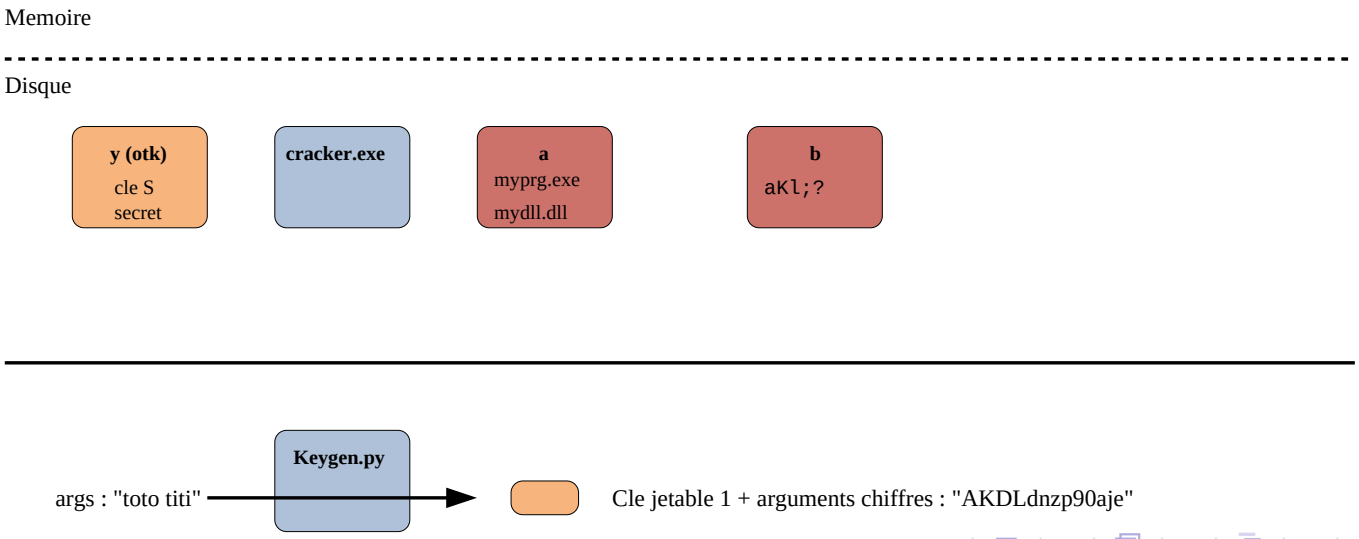


Principe de fonctionnement détaillé (3)

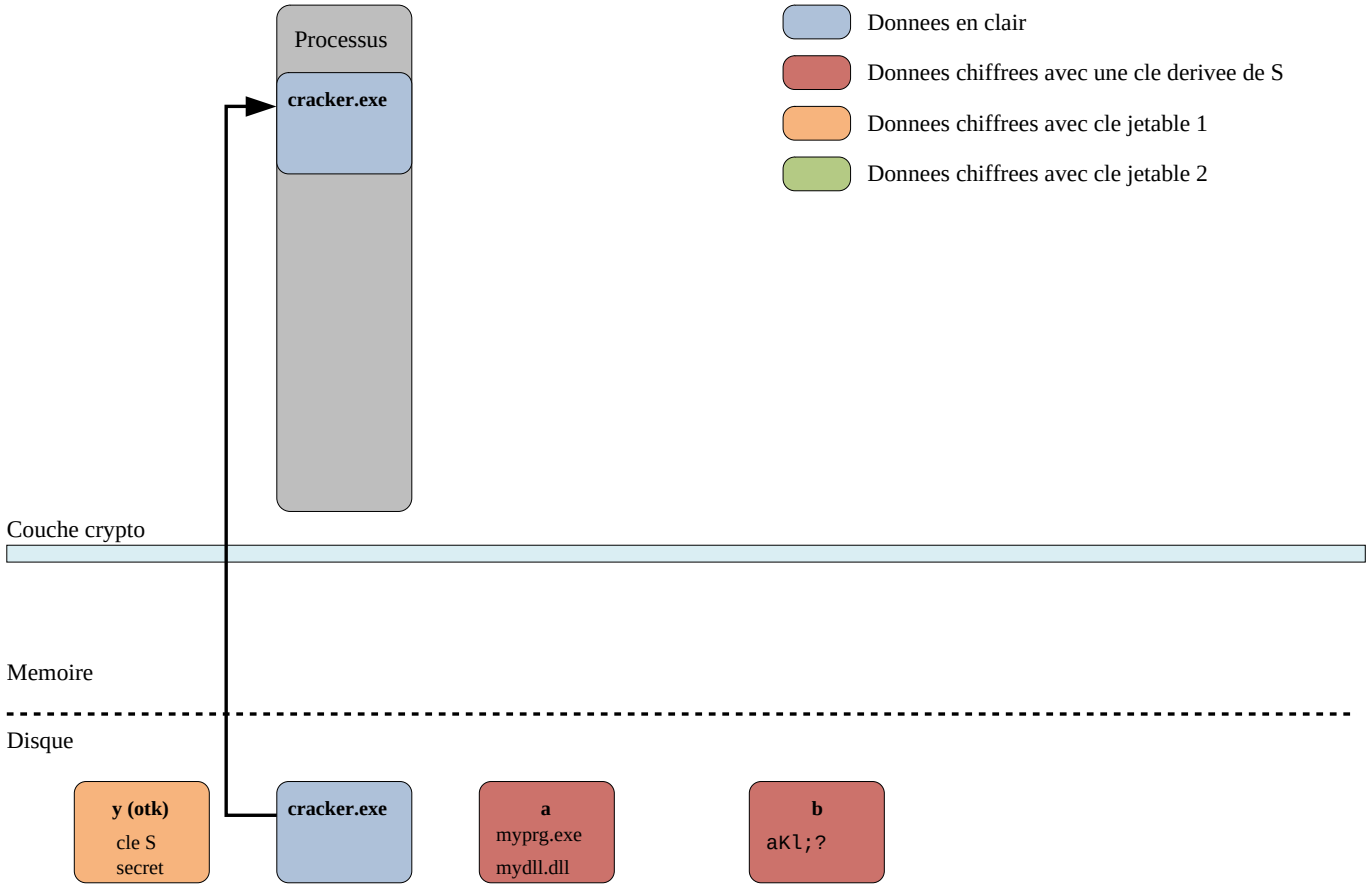


Principe de fonctionnement détaillé (3)

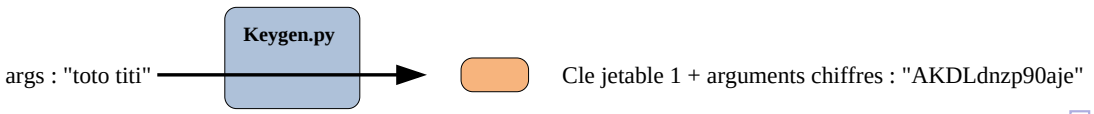




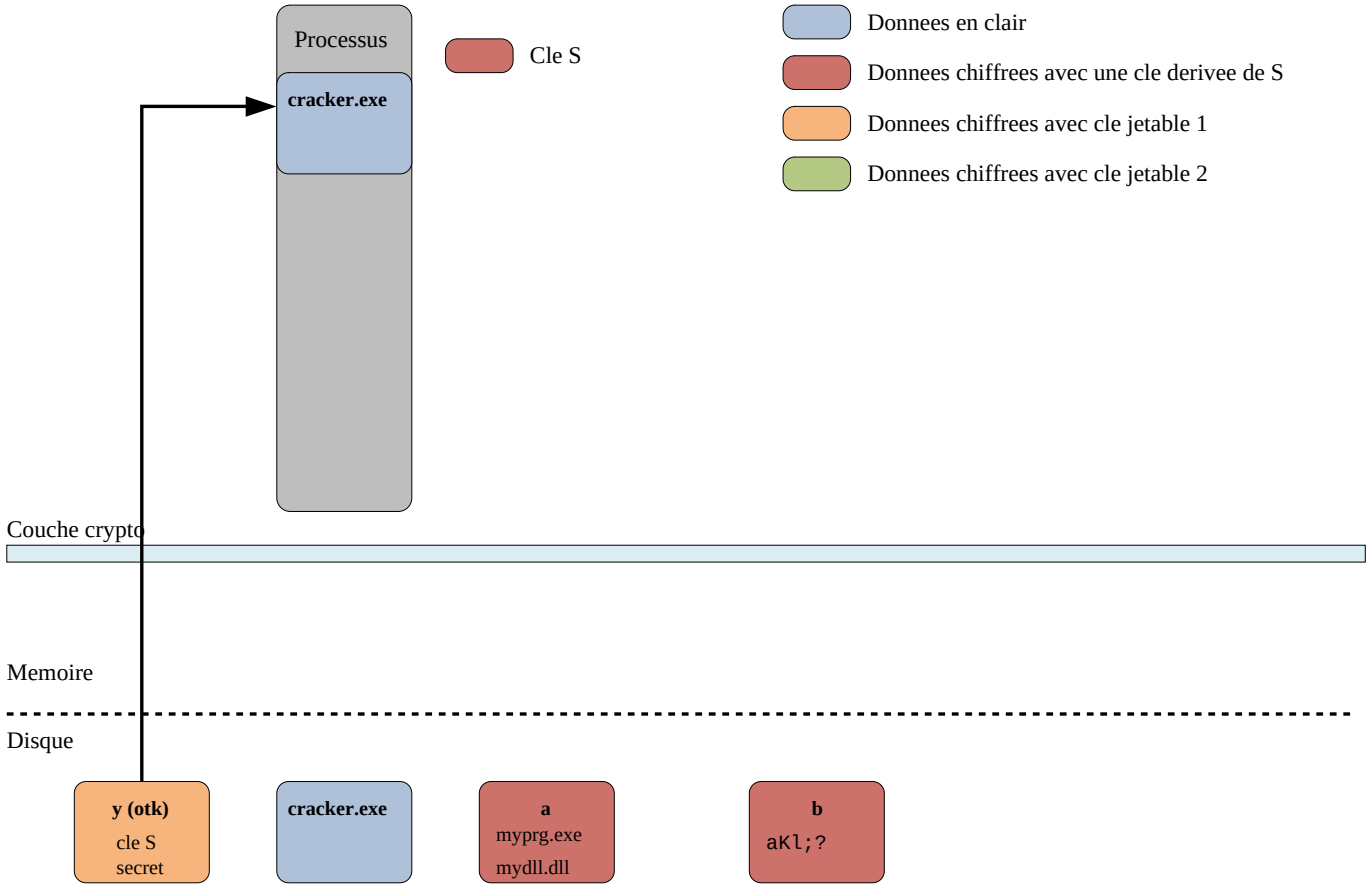
Principe de fonctionnement détaillé (3)



Ligne de commande : cracker.exe −t a −−otk y −k [cle jetable 1] −− AKDLdnzp90aje

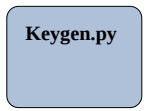


Principe de fonctionnement détaillé (3)

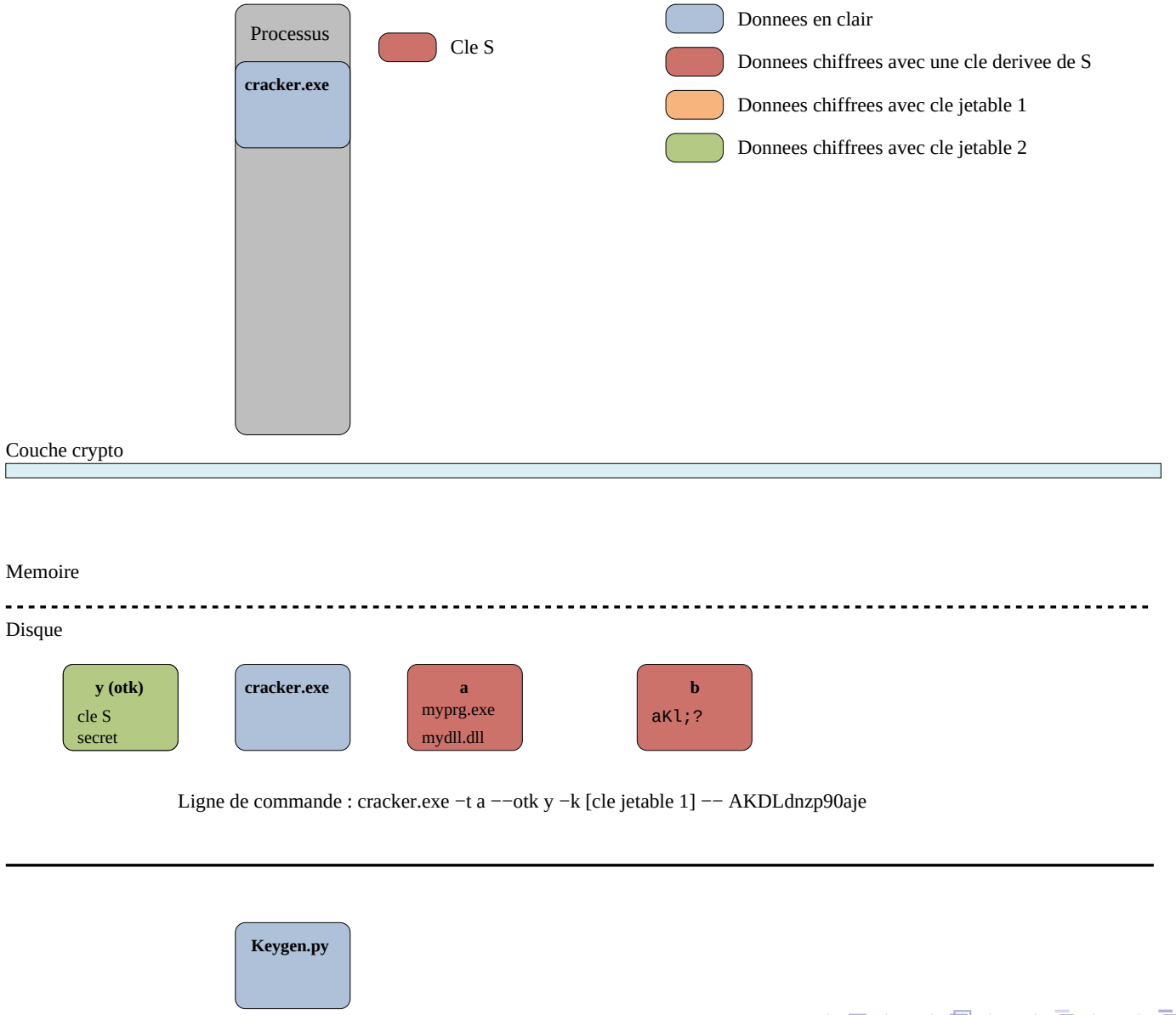


Ligne de commande : cracker.exe −t a −−otk y −k [cle jetable 1] −− AKDLdnzp90aje

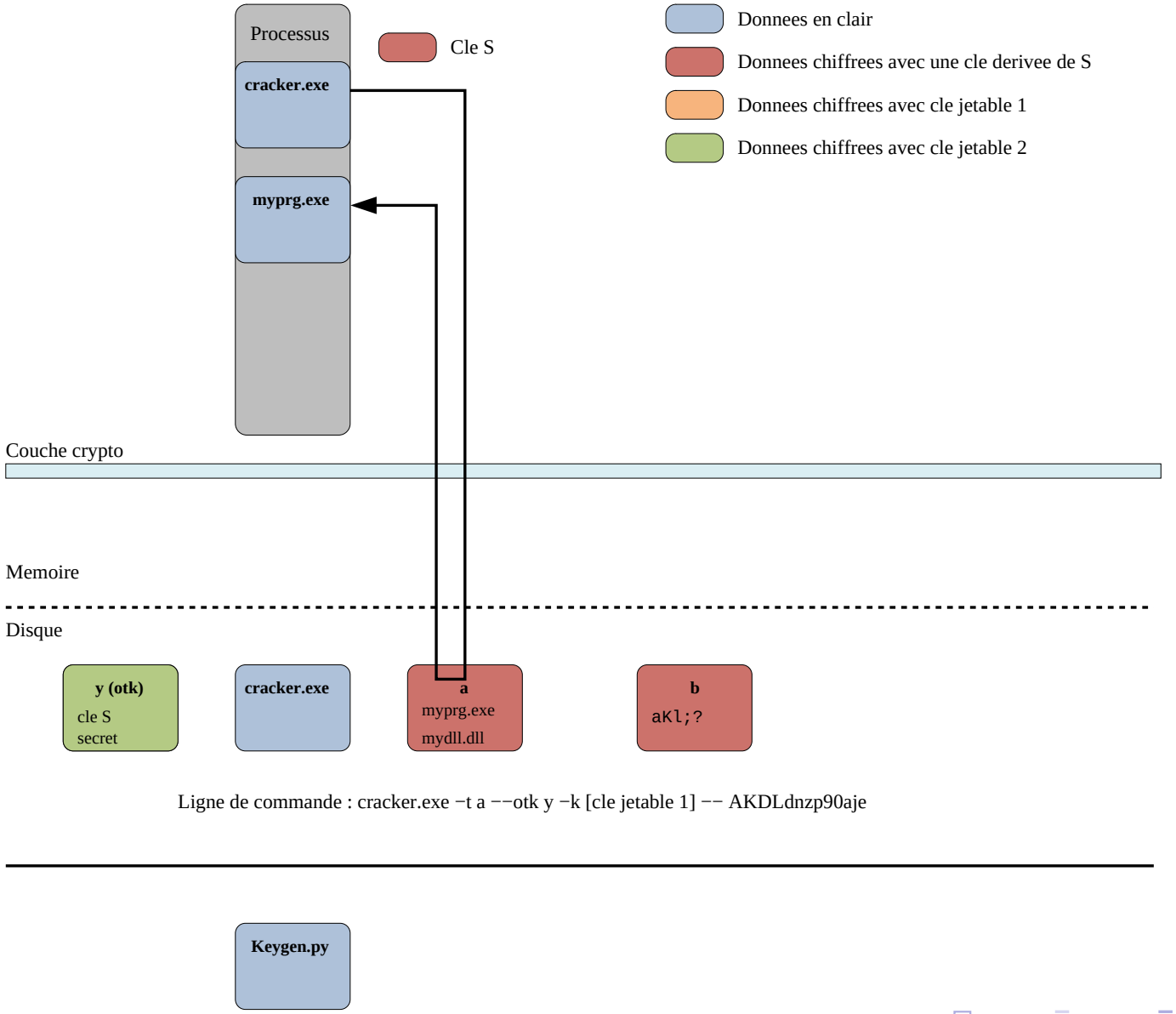
---



Principe de fonctionnement détaillé (3)

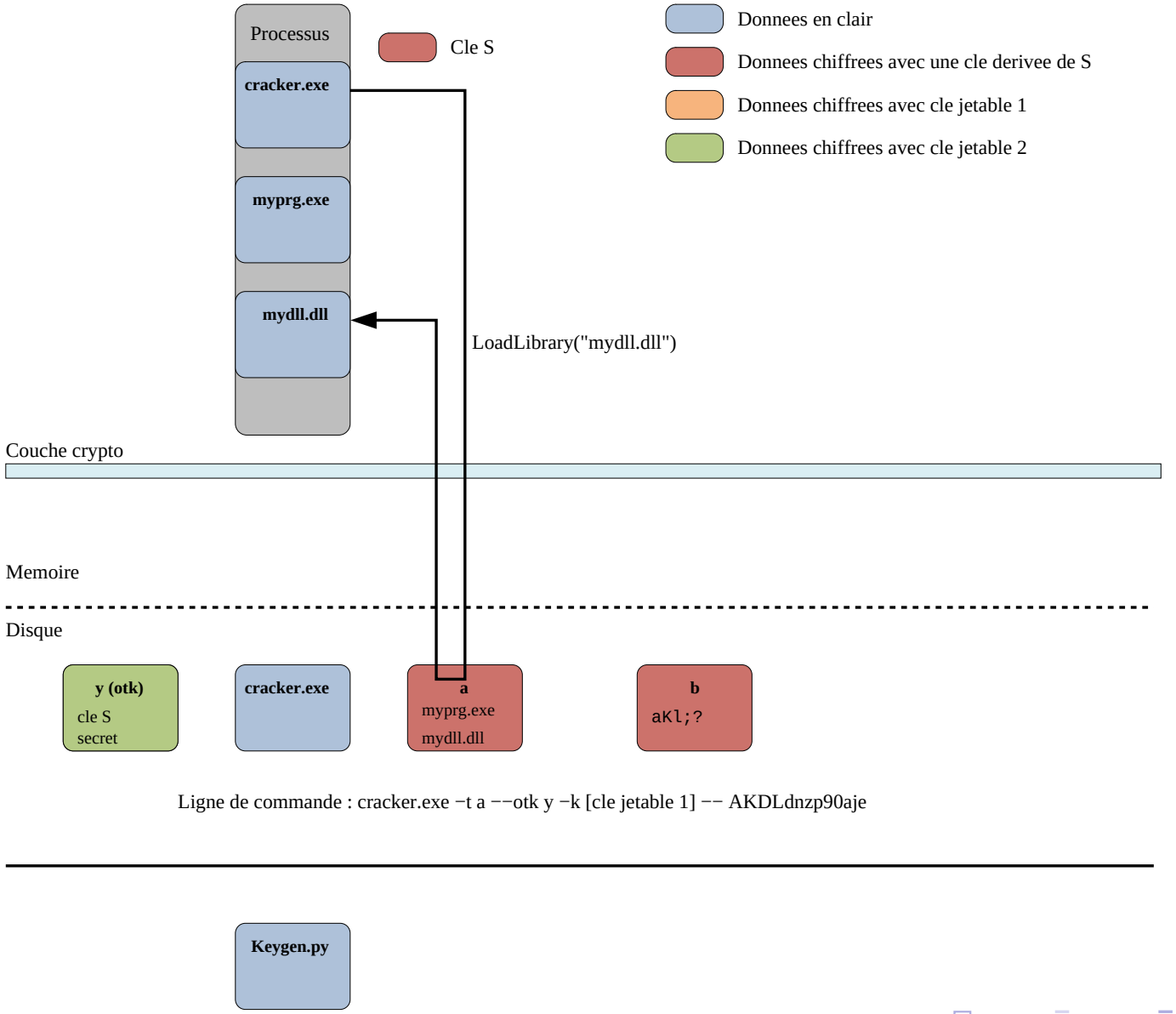


Principe de fonctionnement détaillé (3)



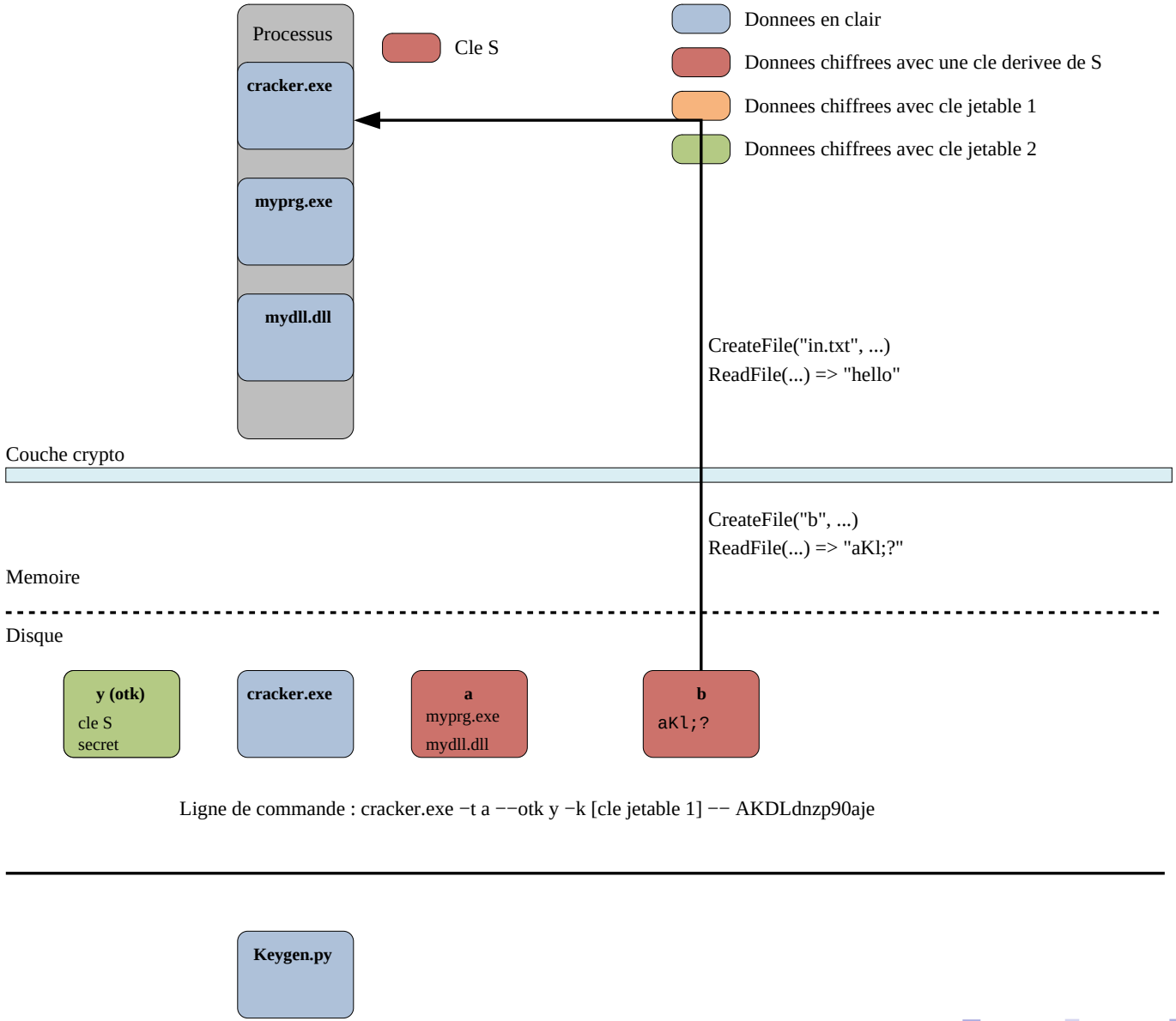
Ligne de commande : cracker.exe −t a −−otk y −k [cle jetable 1] −− AKDLdnzp90aje

Principe de fonctionnement détaillé (3)

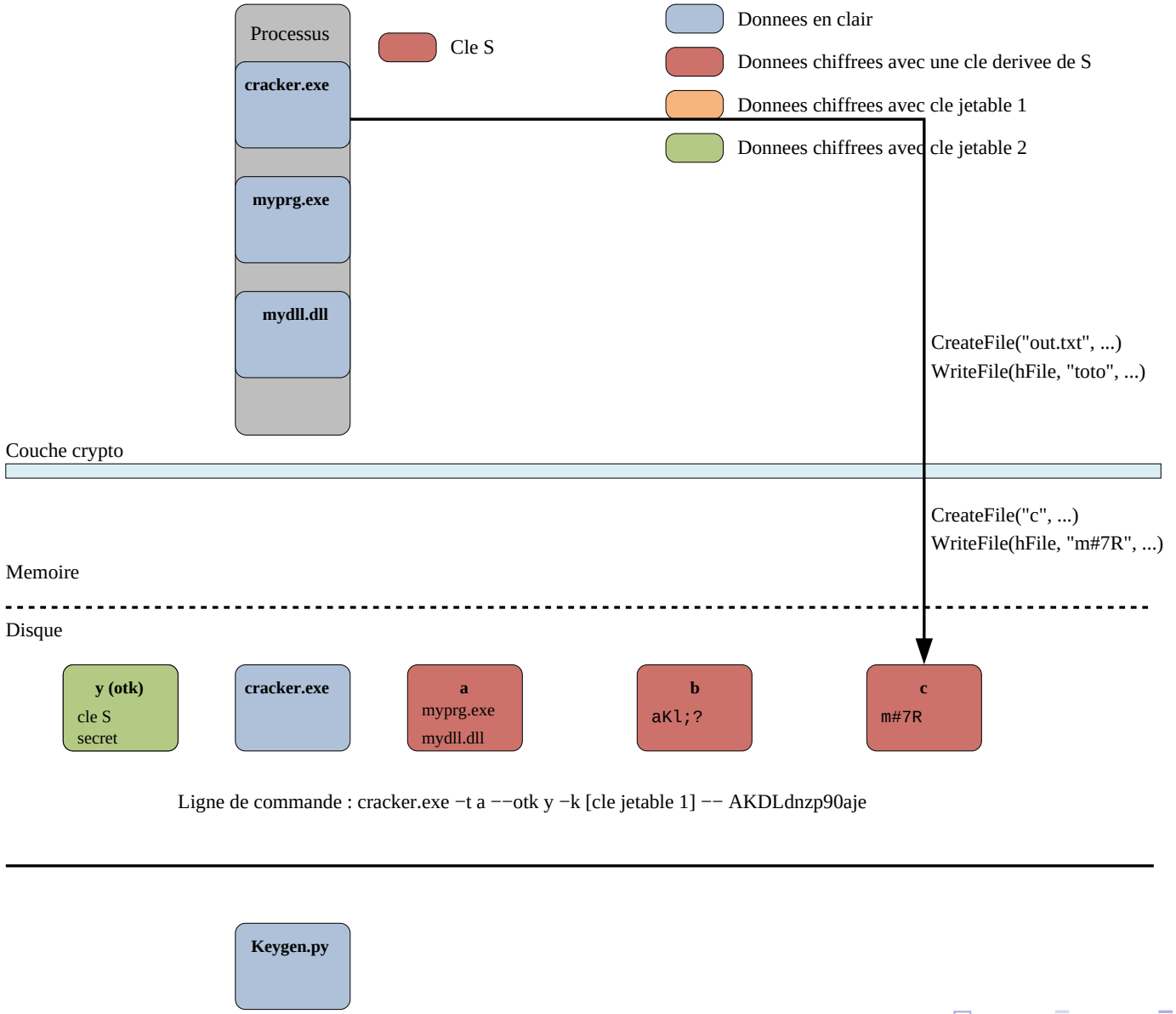


Ligne de commande : cracker.exe −t a −−otk y −k [cle jetable 1] −− AKDLdnzp90aje

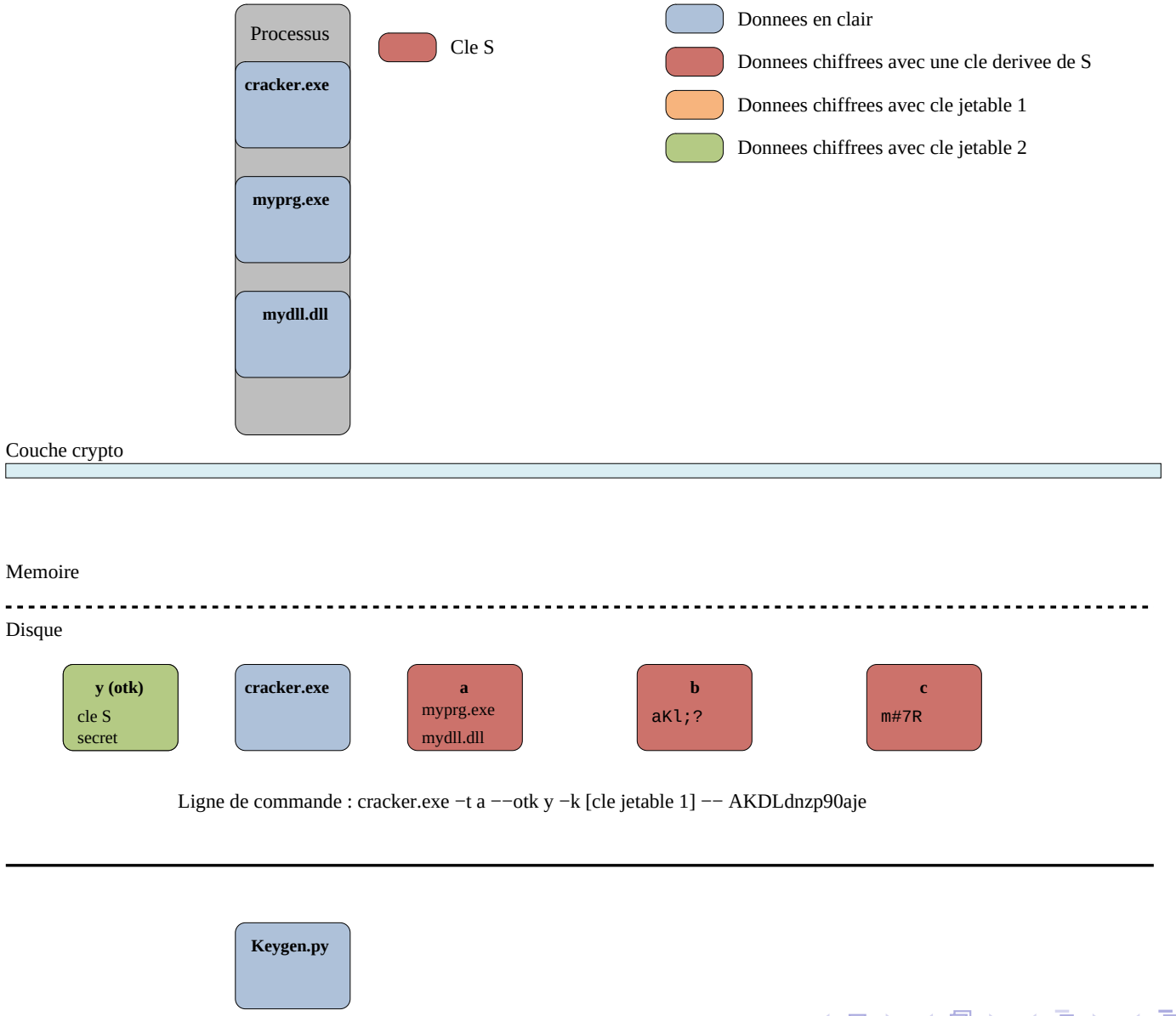
Principe de fonctionnement détaillé (3)



Principe de fonctionnement détaillé (3)



Principe de fonctionnement détaillé (3)



Démo sur programme de test

Description programme de test

- lecture des arguments ;
- lecture fichier .txt (fopen/fread/fseek/fclose)
- lecture fichier .txt (CreateFile/WriteFile/CloseHandle)
- affichage sortie standard (printf/fprintf)
- affichage sortie standard (WriteConsole)
- recherche de fichiers par pattern *.txt (FindFirstFile/FindNextFile)
- appel de fonction dans une dll (résolution à la compilation)
- appel de fonction dans une dll (résolution dynamique par nom et par ordinal LoadLibrary/GetProcAddress)

Démo

Pas le temps, mais ça marche (ou pas)

Conclusion

- Outil en version bêta, mais exécute la majorité des programmes (nmap, findstr, ...)
- Subsiste des problèmes/bugs : freeze lors de certains programmes (Word)
- Protection perfectible : par exemple, actuellement programme entièrement en clair en mémoire $\Rightarrow$ récupérable par simple dump
- Bientôt disponible sur mon site `http://benjamin.caillat.free.fr/`