


Analyse forensique de la mémoire de GnuPG


SSTIC 2022

Qui sommes nous ?

Sylvain Pelissier

- Chercheur en sécurité
- Cryptographie appliquée
- Attaques matérielles
- Joueur de CDD
- @Pelissier_S 

Nils Amiet

- Chercheur en sécurité
- Confidentialité des données
- Traitement de données à large échelle
- Enthousiaste de Linux
- @tmlxs 




GnuPG

- GnuPG (GPG) est une implémentation du standard OpenPGP et une solution de chiffrement et signature de données
- Exemples:
 - Signature et chiffrement de courriel (cas le plus connu)
 - Signature de commits git
 - Signature de paquets pour différentes distributions Linux: Debian, Arch, RedHat
 - Authentification SSH via clé PGP
- GnuPG VS-Desktop est maintenant approuvé [1] par le gouvernement allemand pour sécuriser les données classées au niveau VS-NfD [2] (restreint)
- [1] <https://gnupg.org/blog/20220102-a-new-future-for-gnupg.html>
- [2] <https://de.wikipedia.org/wiki/Geheimhaltungsgrad>

Signature de commit avec git

```
sylvain:~/gpg/$ git commit -S -m "Fix tests"
```

[11868]@sylvain-laptop

 Veuillez entrer la phrase secrète pour déverrouiller la clef secrète OpenPGP :

« Utilisateur <utilisateur@protonmail.com> »
clef RSA de 3072 bits, identifiant 2657727DB950E96C,
créée le 2022-05-02.

Phrase secrète :


☐ Save in password manager

L'agent GPG

- **gpg-agent** est un daemon qui gère les phrases et les clés secrètes.
- C'est l'application dorsale de GPG.
- Les clés secrètes sont chiffrées et mises en cache dans la mémoire vive de gpg-agent, de manière à ce que les phrases secrètes ne soient demandées qu'une seule fois.

Modèle de menace de GPG

gnupg/agent/cache.c



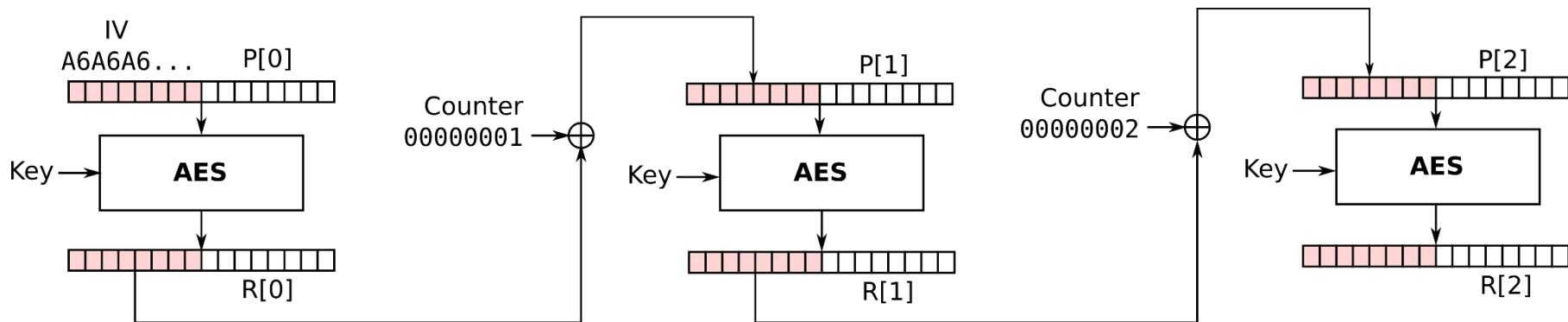
```
39 /* The encryption context. This is the only place where the
40    encryption key for all cached entries is available. It would be nice
41    to keep this (or just the key) in some hardware device, for example
42    a TPM. Libgcrypt could be extended to provide such a service.
43    With the current scheme it is easy to retrieve the cached entries
44    if access to Libgcrypt's memory is available. The encryption
45    merely avoids grepping for clear texts in the memory. Nevertheless
46    the encryption provides the necessary infrastructure to make it
47    more secure. */
48 static gcry_cipher_hd_t encryption_handle;
```

Chiffrement de la mémoire

- Les éléments en cache sont chiffrés dans la mémoire vive avec le mode d'opération key wrap d'AES
- La clé est générée aléatoirement au démarrage de gpg-agent
- La clé est stockée en clair quelque part dans la mémoire

AES key wrap

La première des six itérations:



Mémoire de Libgcrypt

```
39 [0x0698f2f0]> px 80
40 - offset -    0 1  2 3   4 5   6 7   8 9   A B   C D   E F   0123456789ABCDEF
41 0x0698f2f0   0000 0000 0000 0000 0000 0000 0000 0000 .....
42 0x0698f300   a6a6 a6a6 a6a6 a6a6 7665 7279 6c6f 6e67 .....verylong
43 0x0698f310   a6a6 a6a6 a6a6 a6a6 0000 0000 0000 0000 .....
44 0x0698f320   0000 0000 0000 0000 0000 0000 0000 0000 .....
45 0x0698f330   0000 0000 0000 0000 0000 0000 0000 0000 .....
```

Un instant ! La mémoire devrait être chiffrée ?


Rapport de bug GnuPG

- Les 8 premiers octets de la phrase secrète ne sont pas effacés de la mémoire
- Le problème vient de l'implémentation de AES key wrap dans Libgcrypt
- Bug rapporté à GnuPG: <https://dev.gnupg.org/T5597>
- Corrigé dans Libgcrypt 1.8.9 (2022-02-07): <https://dev.gnupg.org/T5467>

Récupération complète de la phrase secrète

Structure de la mémoire de GPG

gnupg/agent/cache.c



```
51 struct secret_data_s {
52     int totallen; /* This includes the padding and space for AESWRAP. */
53     char data[1]; /* A string. */
54 };
55
56 /* The cache object. */
57 typedef struct cache_item_s *ITEM;
58 struct cache_item_s {
59     ITEM next;
60     time_t created;
61     time_t accessed; /* Not updated for CACHE_MODE_DATA */
62     int ttl; /* max. lifetime given in seconds, -1 one means infinite */
63     struct secret_data_s *pw;
64     cache_mode_t cache_mode;
65     int restricted; /* The value of ctrl->restricted is part of the key. */
66     char key[1];
67 };
```

Recherche des horodatages

- Les éléments mis en cache contiennent deux dates de type **time_t**: date de création et de dernier accès.
- La durée de vie de l'élément est de 10 minutes par défaut (0x258 secondes)
- Donc l'expression régulière:

.{3}\x61\x00\x00\x00\x00.{3}\x61\x00\x00\x00\x00\x58\x02

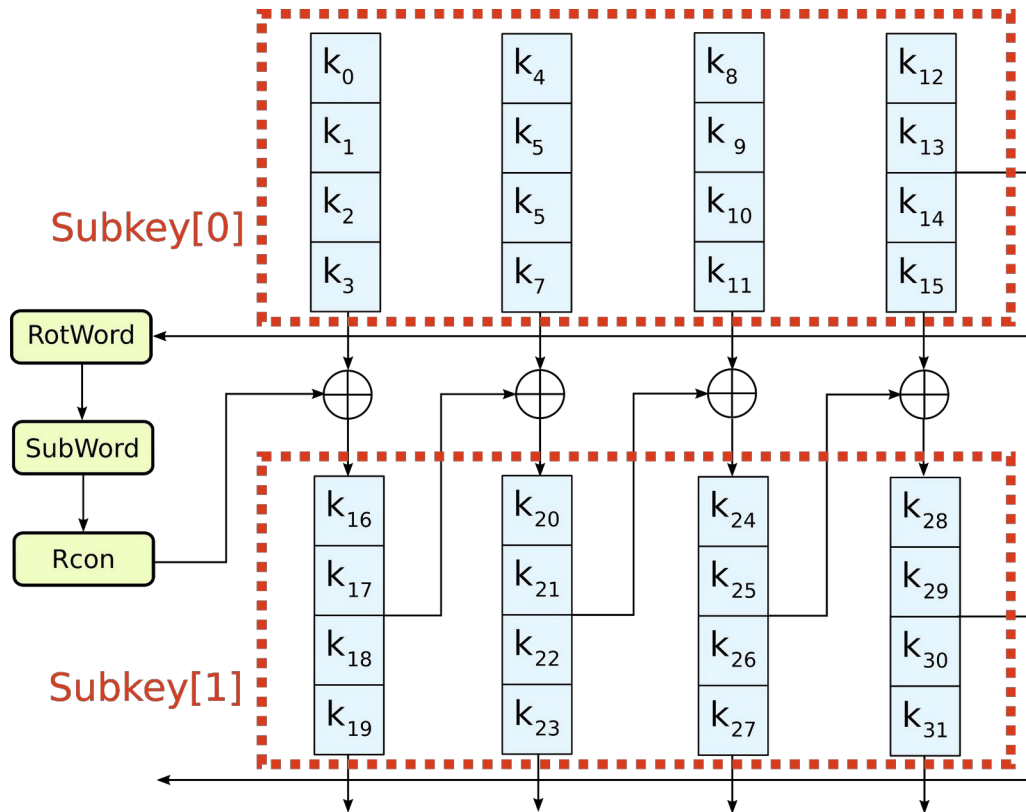
Détecte les éléments en cache créés après le 27 juillet 2021 à 12:45:52 et avant le 6 février 2022 à 17:06:08 avec un temps de vie de 10 minutes.

- On peut ensuite tester que la **date de création** <= **date du dernier accès**.

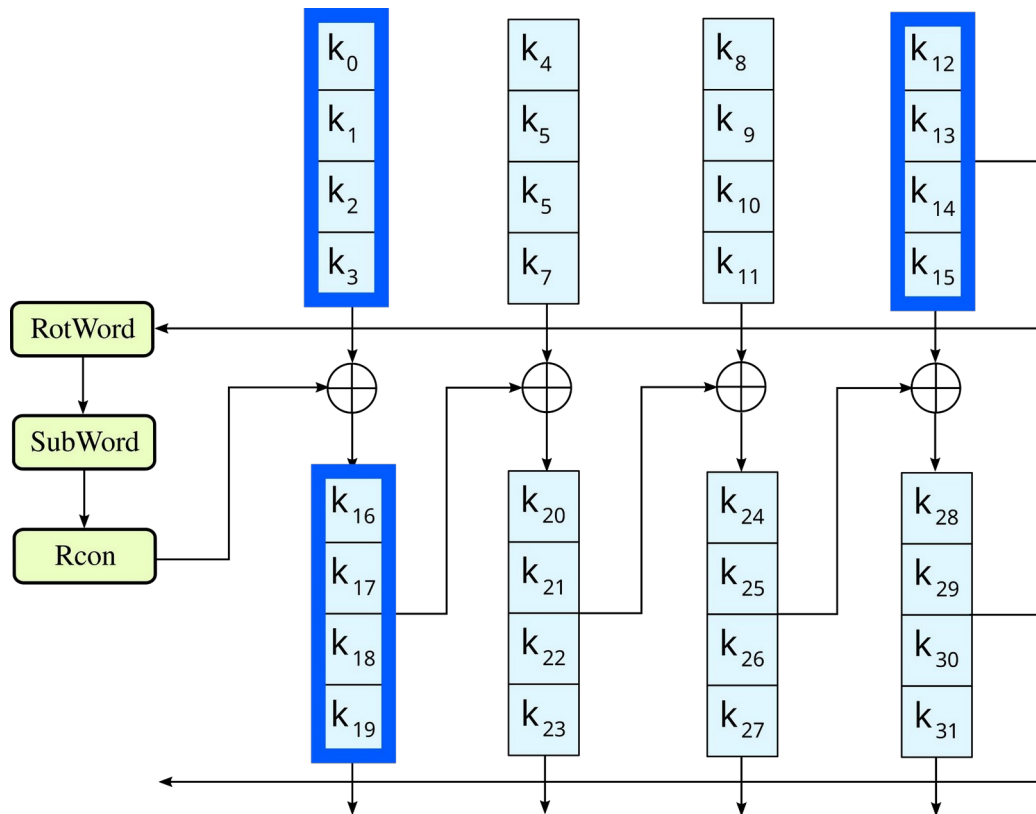
Recherche de clés AES

- Nous avons montré une méthode pour retrouver les éléments mis en cache.
- Maintenant comment retrouver la clé AES en mémoire ?

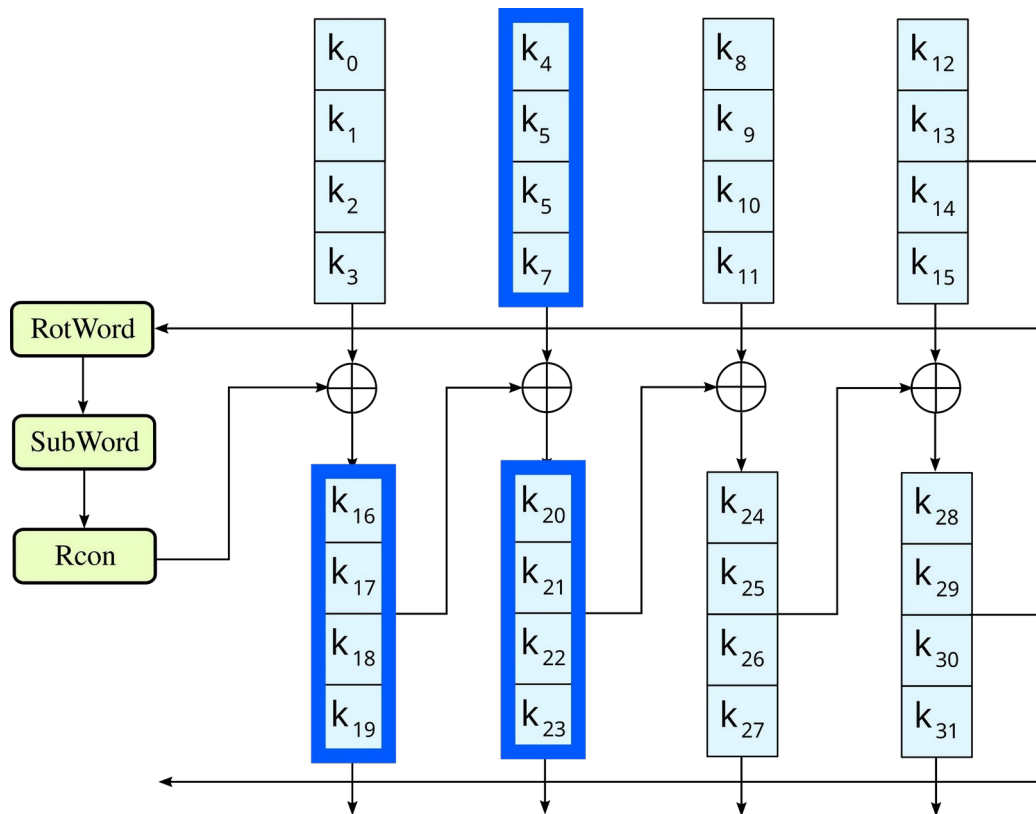
Recherche des clefs AES dérivées



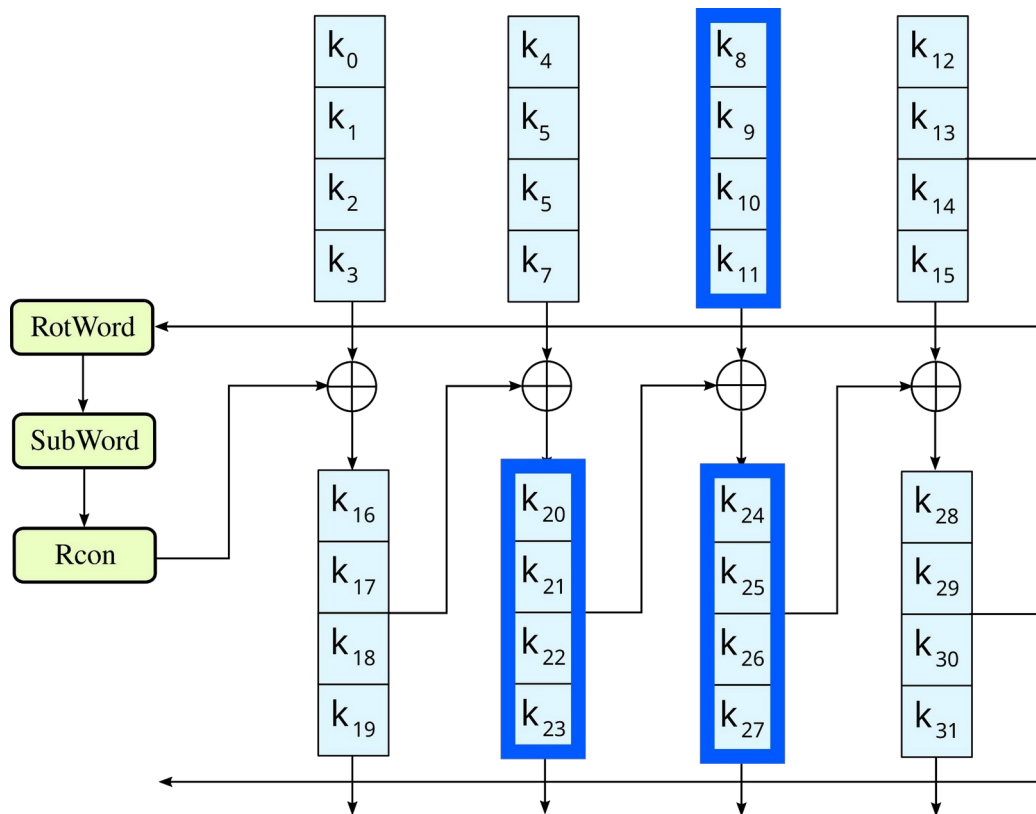
Recherche des clés AES dérivées



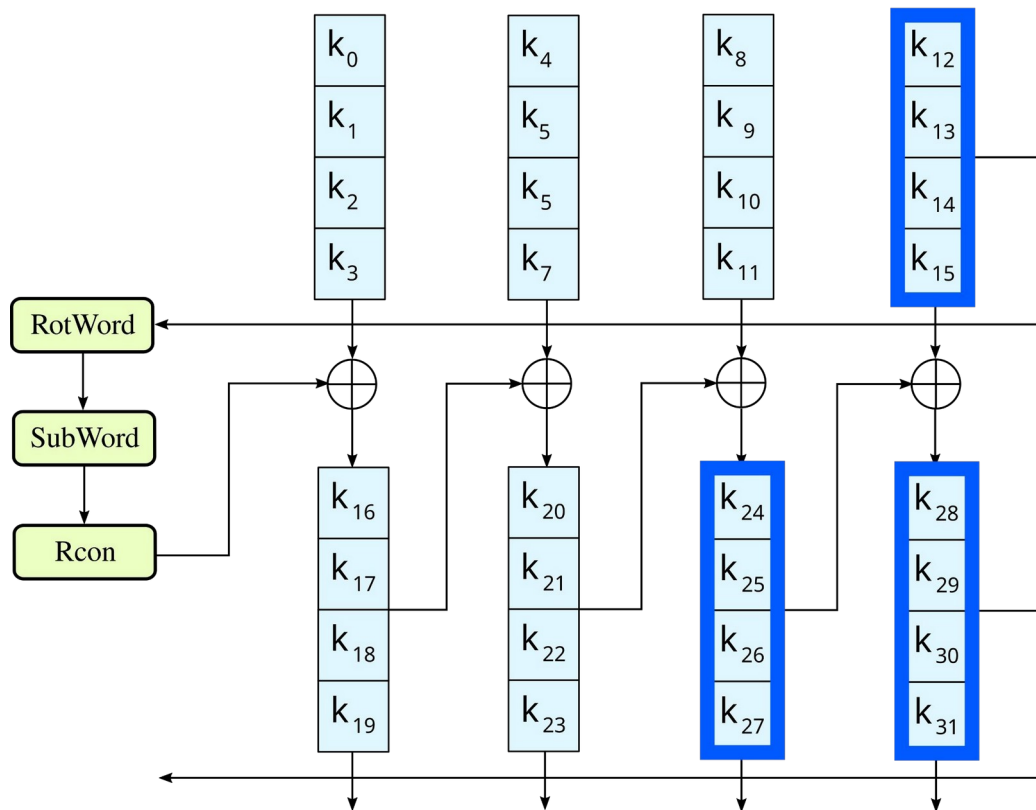
Recherche des clés AES dérivées



Recherche des clés AES dérivées



Recherche des clés AES dérivées



Récupération et déchiffrement d'un élément en cache

- La structure de l'élément en cache est trouvée avec l'expression régulière.
- La clé de AES key wrap est trouvée en mémoire.
- L'élément en cache est déchiffré et l'intégrité est vérifiée.
- Si la clé est valide, la phrase secrète déchiffrée est correcte

Implémentations pratiques

Volatility

- Volatility3 est un logiciel open-source, facilitant l'extraction d'information à partir de traces mémoire système
- Plusieurs cas d'utilisation
 - Analyse forensique
 - Récupération de clé privée
 - Récupération d'historique de commandes (par exemple Bash)
 - Analyse de maliciel
 - Analyse de trace mémoire d'un système en fonctionnement (processus, connexions, ...)
 - Permet d'analyser le statut à n'importe quel moment, même si la machine d'origine est arrêtée
- Système de modules d'extension

Modules d'extension pour Volatility3

- Module 1: Récupération (partielle) de phrase secrète de GPG (max 8 caractères)
- Module 2: Récupération complète de phrase secrète et texte en clair (sans limite de longueur)
- Les 2 modules d'extension sont open-source et publiés ici:
<https://github.com/kudelskisecurity/volatility-gpg>

Démonstration

Demo - Récupération partielle d'une phrase de passe



```
[16:05:10]-[nils ~/work/gpg-talk/volatility]
└─$ ~/git/volatility3/vol.py -f memdump-gpg-verylongpassphrasestarexclexcl -s symbols/ -p ~/git/gpg-
mem-forensics/volatility-gpg/ linux.gpg_partial
Volatility 3 Framework 2.0.0
Progress: 100.00          Stacking attempts finished
Offset  Partial GPG passphrase (max 8 chars)

0x7fb04caee2a0  verylong
```

Demo - Récupération complète d'une phrase de passe

```
[16:05:22]-[nils ~/work/gpg-talk/volatility]
└─>$ ~/git/volatility3/vol.py -f memdump-gpg-verylongpassphrasestarexclexcl -s symbols/ -p ~/git/gpg-mem-forensics/volatility-gpg/ linux.gpg_full --fast
Volatility 3 Framework 2.0.0
Progress: 100.00          Stacking attempts finished
Offset Private key      Secret size  Plaintext
0x7fb048002578 788dc61976e3ac8e9e10d7b80b3e7b40      32      verylongpassphrase*!!
0x7fb048002578 788dc61976e3ac8e9e10d7b80b3e7b40      32      verylongpassphrase*!!
```

Conclusions

- Nos modules d'extension peuvent être utiles dans plusieurs cas pratiques.
- La protection de la mémoire vive est difficile à implémenter en pratique.
- Certaines protections sont déjà en place dans GnuPG mais elles ne sécurisent pas entièrement la mémoire...
- Pour se prémunir de nos attaques il faudrait utiliser un *Trusted Platform Module* (TPM).

Merci ! Des questions ?