Rapport M(alléable)EGA

SSE - Séminaire en sécurité

08.04.2024

Alexis Martins
HEIG
SSE

Table des matières

1. Introduction

En 2022, des chercheurs de l'ETHZ se sont attardés sur la manière dont MEGA gérait les données de ses utilisateurs. MEGA est un fournisseur de services de stockage cloud qui met en avant son intérêt pour la sécurité des données de ses utilisateurs. Ils indiquent que seuls les utilisateurs peuvent avoir accès à leurs données et que même MEGA ne peut pas y accéder. Malheureusement pour MEGA, ces quelques chercheurs remettent en question ces propos.

2. Cryptographie et architecture

Afin de comprendre les attaques qui vont suivre dans ce rapport, il est crucial de comprendre comment MEGA a décidé d'organiser la sécurité de ses clés. Comme on peut le voir, ils ont décidé d'avoir toutes les clés du système chiffrées avec `AES-ECB` et la même `Master Key`. Cette dernière n'étant protégée que par le mot de passe de l'utilisateur dérivé avec PBKDF.

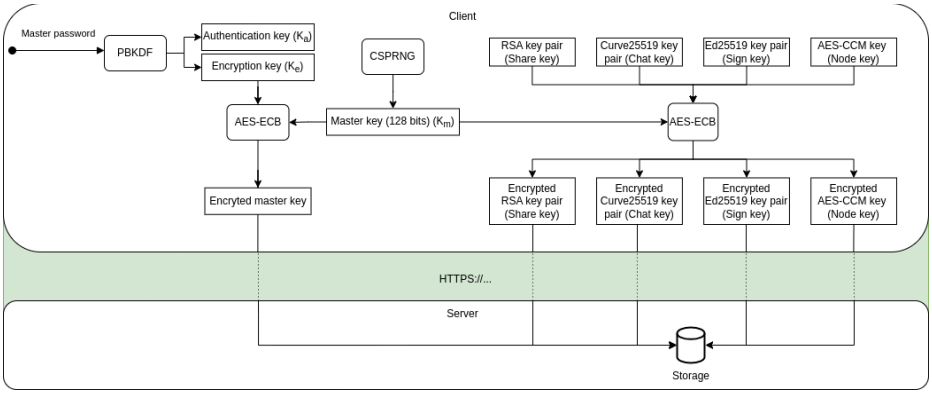

Fig. 1 — Architecture des clés

La connexion au compte de l'utilisateur est la seconde partie importante à maîtriser afin de pouvoir correctement comprendre les attaques. L'utilisateur envoie sa clé d'authentification au serveur et ce dernier lui envoie en retour la clé privée RSA chiffrée avec la Master Key, ainsi qu'un challenge qu'il doit déchiffrer. A noter que la clé privée n'est pas envoyé telle quelle, mais elle est séparée dans ses diverses composantes, ainsi qu'une valeur `u` utilisée dans le CRT qui vaut q^{-1}. Cette séparation semble étrange à première vue, mais elle est utilisée pour réaliser le déchiffrement avec le CRT (Théorème des restes chinois). On voit aussi que le challenge `m` qui est déchiffré n'est pas renvoyé tel quel, mais uniquement quelques bytes (43 bytes sur les 256 qui composent `m`).

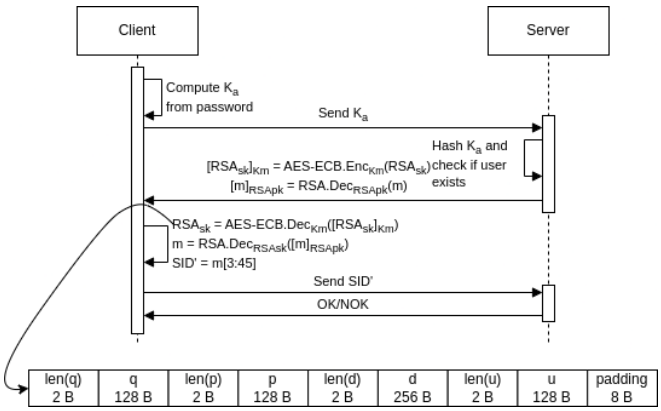

Fig. 2 — Diagramme des étapes de la connexion

3. Attaques

3.1. Récupération de la clé privée RSA

La première attaque part du principe que l'attaquant contrôle le serveur, donc il se fait passer pour un fournisseur de services malicieux ou il a compromis le serveur de MEGA. Pour cette attaque, cela signifie que l'attaquant peut impacter tout ce qui est sur le serveur, mais aussi tout ce qui est envoyé par le serveur comme le challenge de connexion ou la valeur de la clé privée. Le but de l'attaquant est de récupérer la valeur de la constante `q` de la clé privée en utilisant un `Case distinction oracle`. C'est-à-dire que lors de la connexion de l'utilisateur, selon la valeur qu'il va retourner à l'attaquant pour un challenge donné, l'attaquant va réussir à en tirer de l'information et ainsi isoler `q`.

Dans le schéma qui suit, on peut voir dans un premier temps la première modification apportée par l'attaquant à la clé RSA qui est envoyé à l'utilisateur lors de la connexion. Celle-ci est stockée sur le serveur et chiffrée avec AES-ECB. Le problème de ce mode opératoire est qu'il est **malléable**. Il est donc possible pour l'attaquant de modifier facilement uniquement le bloc où est stocké `u` afin que sa valeur soit différente de q^{-1}. Cela nous permettra lors de la réalisation du CRT de voir les deux cas apparaître.

Concernant les cas, ceux-ci surviennent selon le challenge que l'attaquant décide d'envoyer à l'utilisateur. Si l'attaquant envoie un challenge `m` plus petit que `q`, alors le client va renvoyer 43 bytes remplis de 0 à l'attaquant, car les bytes qui sont prélevés du challenge sont trop hauts pour impacter la valeur retournée. Dans le cas où `m` est plus grand ou égal à `q`, alors l'attaquant reçoit de l'aléatoire. Avec ces deux cas, un attaquant peut en environ 1023 connexions retrouver la valeur de `q`. Ainsi, il récupère la clé RSA et peut accéder à tout le contenu partagé à l'utilisateur (dossiers, fichiers et certains chats). L'attaque a été améliorée en utilisant des lattices pour descendre le nombre de connexions requises à 512, puis 6 et finalement 2 en ajoutant un second vecteur d'attaque.

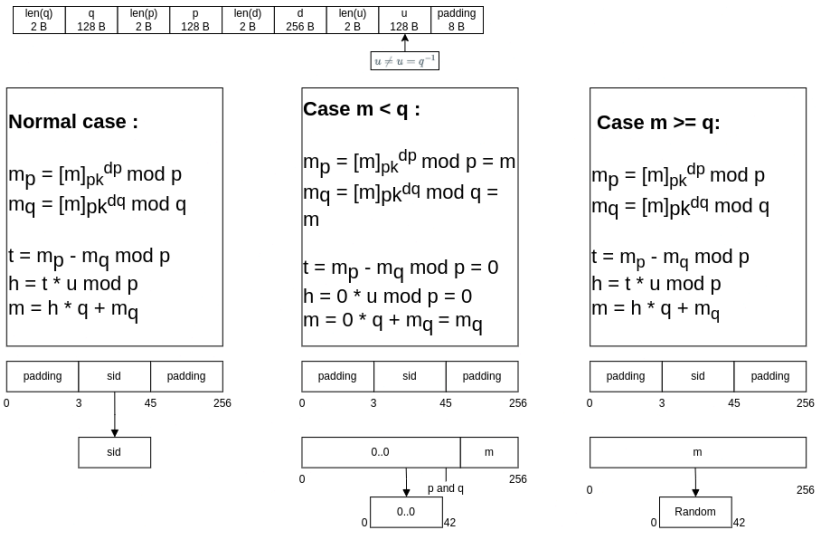

Fig. 3 — Schémas de la première attaque

3.2. Récupération de textes clairs

Cette seconde attaque se base sur la première, le modèle de l'attaquant est similaire, mais il possède maintenant la clé privée. Pour cette attaque, le but de l'attaquant va être de pouvoir déchiffrer n'importe quelle clé stockée sur le serveur en utilisant l'utilisateur comme oracle de déchiffrement et de récupérer le contenu protéger par cette clé.

Dans un premier temps comme le montre le schéma ci-dessous, il va falloir modifier la clé RSA chiffrée qui est envoyée à l'utilisateur. On va la séparer en 41 blocs de 16 bytes pour expliquer la manipulation. Le but de cette étape est de trouver un endroit où on peut mettre les clés que l'on souhaite déchiffrer sans compro-

mettre complètement l'exécution du CRT lors du déchiffrement de la clé RSA par l'utilisateur. L'endroit le plus adapté se trouve dans la composante `u`, on verra qu'avec les opérations qui vont se faire dans le CRT, on arrivera à retrouver notre `u` contenant la clé déchiffrée. Le bloc c_{33} contient une partie de la composante `d`, la longueur de `u` et le début de `u`, on ne peut donc pas l'utiliser. Par contre les deux blocs suivants sont totalement dans `u`. On va donc recréer un `u` qui aura ses 8 premiers bytes inchangés, puis les deux blocs contenant la clé (c_{34} et c_{35}) et finalement le reste de `u`.

Ensuite le challenge envoyé par l'attaquant sera soigneusement choisi pour que sa valeur soit $m = u \cdot q$, ainsi le CRT sera simplifié et la valeur finale qui en sort est $u' \cdot q$. L'attaque n'est cependant pas finie, car l'attaquant ne reçoit pas directement cette valeur, mais uniquement la partie `sid` (bytes 3 à 45). Il ne peut donc pas retrouver `u'` qui est maintenant déchiffrer.

L'attaquant connaît une partie de $m' = u' \cdot q$ et u', il va dans un premier bruteforce les premiers bytes qu'il ne connaît pas dans `m'` (2 bytes) et ensuite avec les 45 premiers bytes de `m'` et la valeur de `u'`, il va effectuer diverses transformations arithmétiques jusqu'à obtenir une égalité entre le début de `m'` et le début de `u'`. Il va tester cette égalité en boucle jusqu'à ce qu'elle soit vraie, sinon il refait un bruteforce sur les deux premiers bytes de `m'`. Cette attaque permet donc à un attaquant de récupérer toutes les clés chiffrées sur les serveurs de MEGA. Les mathématiques derrière cette manipulation sont trop complexes et n'apprtent rien à l'explication. Le plus important est de comprendre que l'attaque trouve une égalité entre les 43 bytes du sid qu'il connait (+ 2 bytes brouteforcés) et la valeur de u' qu'il a forgé.

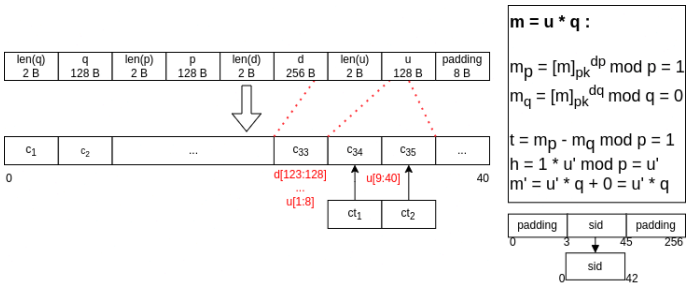

Fig. 4 — Schémas de la deuxième attaque

3.3. Attaques sur l'intégrité

Le modèle de l'attaquant est similaire aux précédents, mais il possède en plus l'oracle de déchiffrement que nous avons obtenu à l'attaque précédente. L'attaquant peut déjà effectuer beaucoup d'actions, mais pour cette attaque le but est de pouvoir mettre dans le drive de l'utilisateur de nouveaux fichiers qu'il n'a jamais mis lui-même.

La première étape est donc de réaliser la même opération qu'à l'attaque précédente en modifiant les blocs c_{34} et c_{35}, sauf que cette fois on va y mettre des valeurs aléatoires. Avec les valeurs aléatoires, le client va déchiffrer des clés qui n'existaient pas avant et ça sera les nouvelles clés pour nos nouveaux fichiers. Un détail que je n'ai pas précisé avant, c'est que le client ne déchiffre pas vraiment la clé directement, mais une version « obfusquée » de la clé. C'est un système qui a été mis en place par MEGA et une clé obfusquée contient en réalité une clé, un nonce et un tag. Ce qui veut dire que l'attaquant vient d'obtenir ces trois valeurs, le problème étant que l'attaquant ne connaît pas le fichier qui va avec le tag qu'il vient de générer.

Il va donc réaliser un MitM en partant du tag, il va prendre son fichier de base et il va choisir un chunk `j` auquel il peut ajouter un bloc AES de 128 bits (dans les headers par exemple). Ce bloc va permettre de faire le lien entre le début du calcul du tag et la fin. Il va donc calculer tout le tag de T_1 à T_{j-1} que l'on va appeler $T_{cond,j-1}$ après le chiffrement AES, mais aussi tout le tag de T_n à T_{j+1} que l'on va appeler $AES\text{-}ECB.Dec(T_{cond,j+1})$ après le déchiffrement. Pour calculer la valeur que doit avoir T_j, il faut donc faire $T_j = T_{cond,j-1} \oplus AES\text{-}ECB.Dec(T_{cond,j+1})$. Une fois que l'on possède T_j, il est assez facile de refaire la chaîne à l'envers pour le chiffrement des chunks et retrouver la valeur que devait avoir notre bloc AES. Avec cette attaque, l'attaquant peut facilement déposer des fichiers compromettants dans les drives de ses victimes.

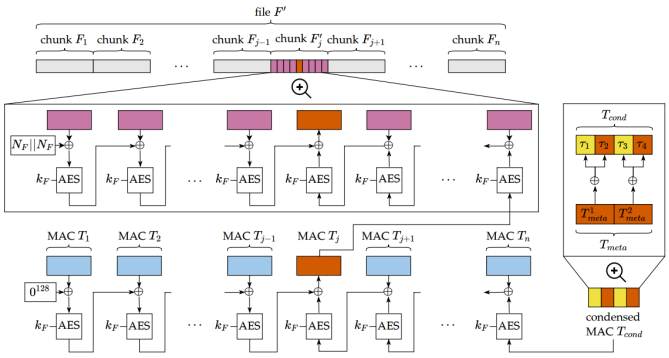

Fig. 5 — Schémas de la troisième attaque (pris du whitepaper)

3.4. GaP-Bleichenbacher

Cette dernière attaque permet de retrouver la valeur des clés échangées avec RSA comme les clés des nodes ou certaines clés pour les chats. Elle reprend grandement l'attaque de Bleichenbacher initiale sur RSA PKCS1 v1.5 où un attaquant va exploiter la manière dont les messages sont paddés pour retrouver la valeur de celui-ci. De base un message est paddé ainsi `00 || 02 || RAND_PAD || 00 || MESSAGE` et l'attaquant va prendre ce message et le multiplier par s^e où s est une valeur bien choisie, ce nouveau message s'appelle c'. Lorsque la victime déchiffre, l'attaquant pourra savoir si c' appartient ou non à l'intervalle des messages commençant par `0x0002` . Il va ensuite réajuster sa valeur s pour isoler le message. L'attaque de GaP-Bleichenbacher est donc similaire, sauf qu'elle possède plusieurs intervalles dans lesquelles le message serait valide. Cela est dû à un padding spécial que MEGA ont fait eux-mêmes pour RSA PKCS1 v1.5. Ce rapport ne contient pas plus de détails sur cette attaque, car elle ne rajoute pas grand chose à ce qu'un attaquant sait déjà faire et elle demande de faire un effort assez considérable au niveau des mathématiques et probabilités.

4. Contre-mesures

Les chercheurs de l'ETHZ ayant trouvé toutes ces attaques, ont aussi décidé de proposer à MEGA des potentielles solutions. Ils les ont séparées en 3 catégories selon la rapidité et l'efficacité de la mesure. Les premières mesures correspondent à ce qui doit être patcher de façon urgente et qui ne prend pas trop de temps. Plus on avance dans les catégories, plus la criticité diminue et cela correspond surtout à des bonnes pratiques.

Catégorie	**Action**
Immédiates	Ajout de protection sur l'intégrité des clés avec HMAC par exemple. / Mise en place de séparation des clés pour éviter que la master key chiffre tout (HKDF) / Padding RSA PKCS#1 v1.5 plus strict afin d'augmenter la difficulté de l'attaque
Minimales	Changer le chiffrement des clés (sauf clés des nodes) de AES-ECB+HMAC à AES-GCM / Changer RSA PKCS#1 v1.5 pour RSA-OAEP
Recommandées	Changer le chiffrement des clés des nodes et fichiers pour AES-GCM / Gérer l'authentification avec un PAKE tel que OPAQUE

5. Conclusion

Le plus important à retenir des ces attaques est de toujours suivre les recommandations cryptographiques lorsque l'on réfléchit à un nouveau système. MEGA a fait tout faux, car ils ont voulu utiliser des algorithmes pas sûrs, mais ils ont aussi apporté des modifications à des algorithmes ce qui les a rendus plus vulnérables.

Don't roll your own crypto