HTC EXODUS 1. Funds are SSSafu

Jean-Baptiste Bédrune

Hong Kong - New York - Paris – San Francisco - Vierzon

Hardware wallet

- Appareil physique
- Protection des clés de l'utilisateur
- Calculs cryptographiques
  - Signature des transactions
  - Les clés ne sont jamais copiées sur un PC / un téléphone

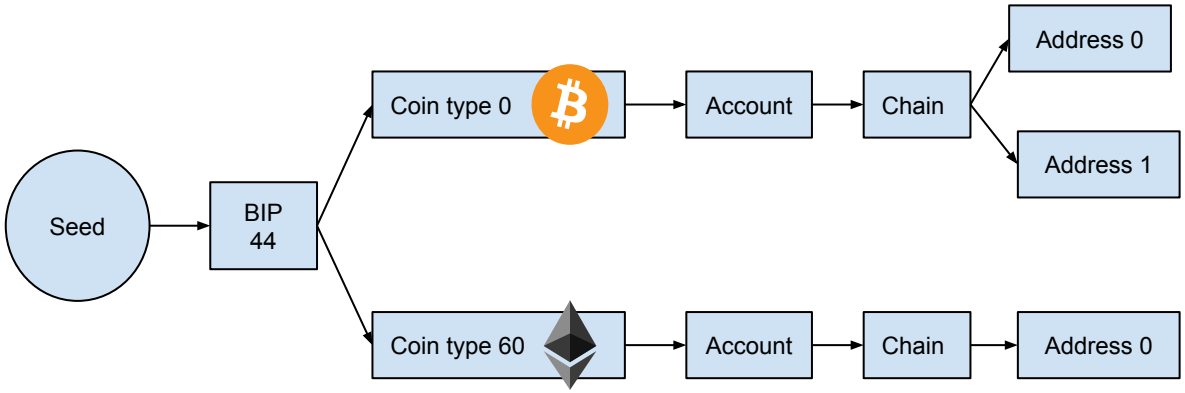
- Avantage : sécurité
- Inconvénients : prix, device supplémentaire

Portefeuille déterministe

Ledger

Une seed à partir de laquelle sont dérivées toutes les clés

- BIP 32 / BIP 44 : portefeuille hiérarchique déterministe

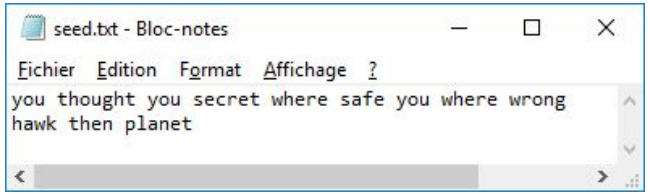


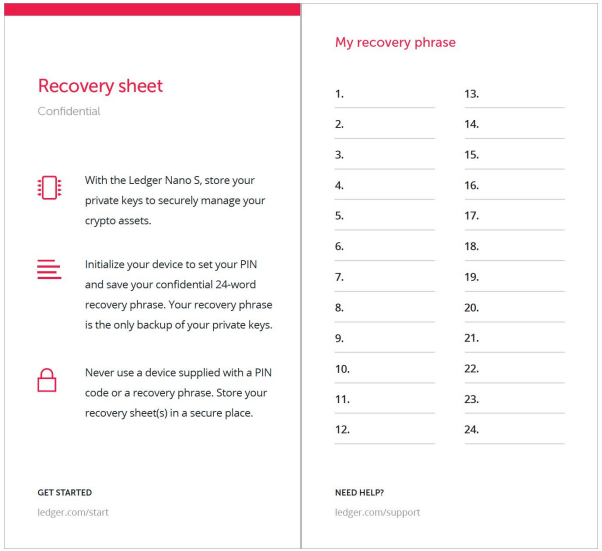
- BIP 39 : mnémoniques

```
all you base rare below toe bus
mesh width they best dice like then arrest
acoustic drastic plastic
```

Sauvegarde de la seed

- Objet dédié
- Recovery sheet
- Coffre fort
- Feuille plastifiée
- GPG
- Clé USB, CD ROM…
- password.txt
- etc.





HTC EXODUS 1

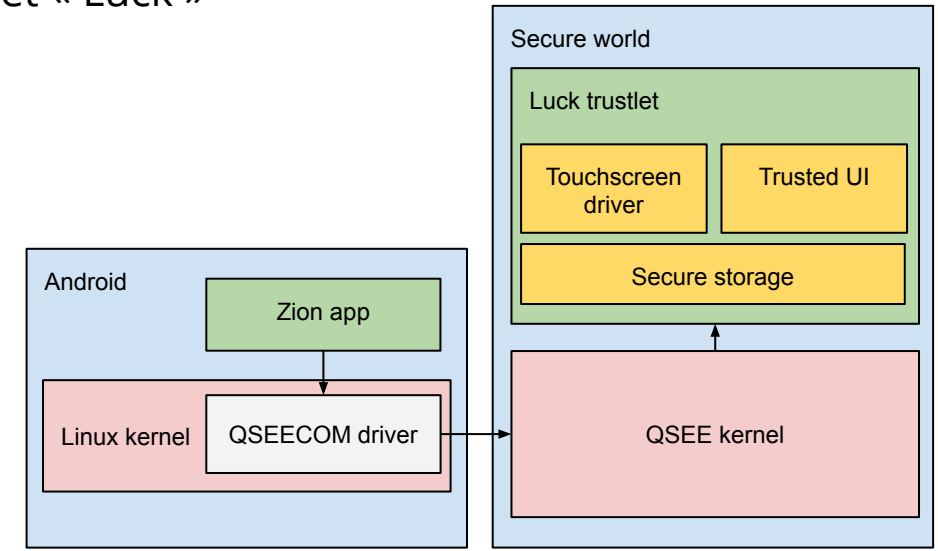
Smartphone « dédié à la blockchain »

- « Hardware Wallet »
- Enclave sécurisée
- Affichage sécurisé
- « Social Key Recovery »

SoC Qualcomm SnapDragon 845

Portefeuille matériel : Zion

- Application Android « Zion » + trustlet « Luck »
- Seed stockée dans l'OS sécurisé
- Périphériques sécurisés
  - Écran
  - Touchscreen
  - Capteur d'empreintes
- Signatures protégées par un PIN



Seed protégée même si le téléphone est rooté

Social Key Recovery

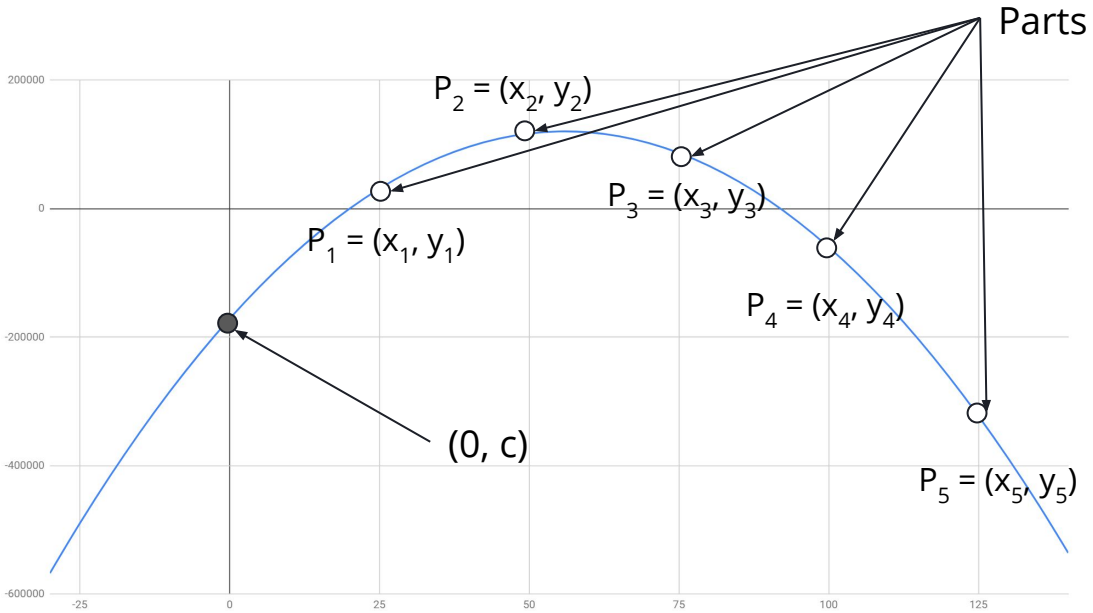
Partage de la seed auprès de contacts de confiance :

- Liste des contacts de confiance stockée dans Google Drive.
- Chaque contact reçoit une partie de la seed.
- Trois contacts sont nécessaires pour reconstruire la seed.
- Seed jamais sauvegardée à un seul endroit.

Contact de confiance :

- Doit installer l'application « Zion ».
- Partie de la seed stockée chiffrée, clé dans le Keystore Android (TEE).

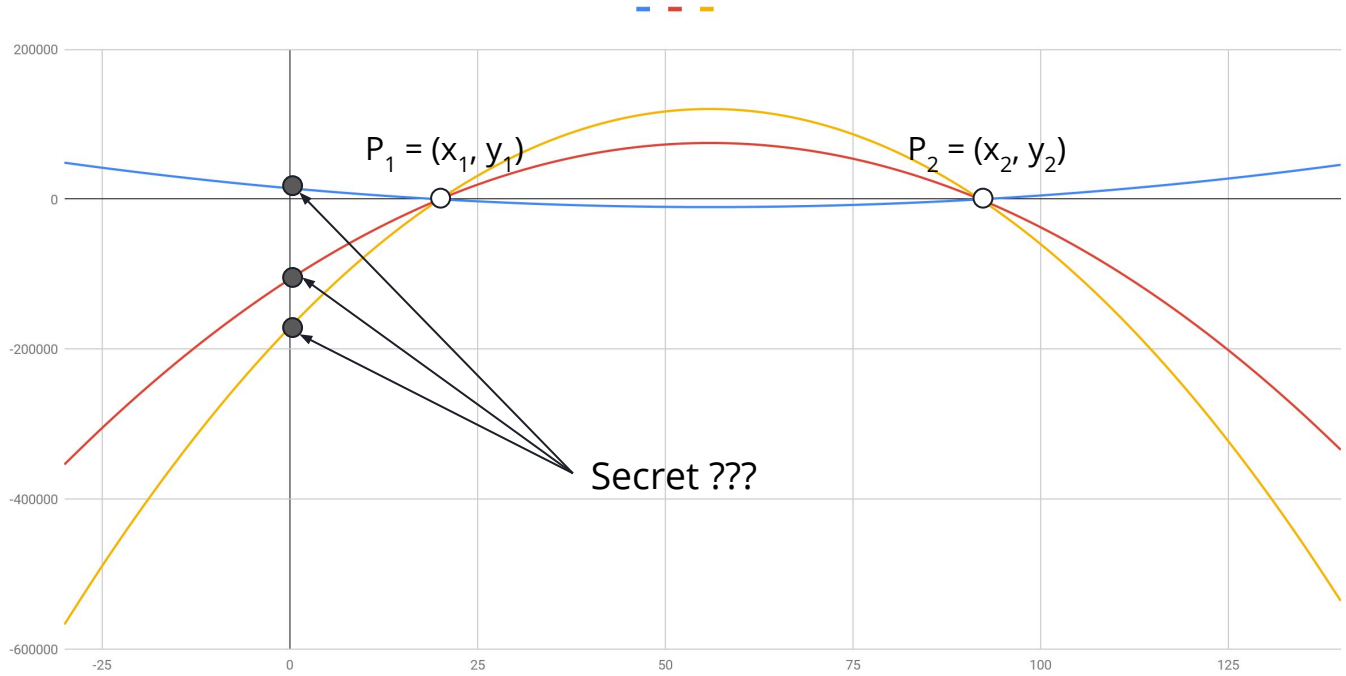
Partage de secret à seuil de Shamir (SSS)



$y = ax^2 + bx + c$

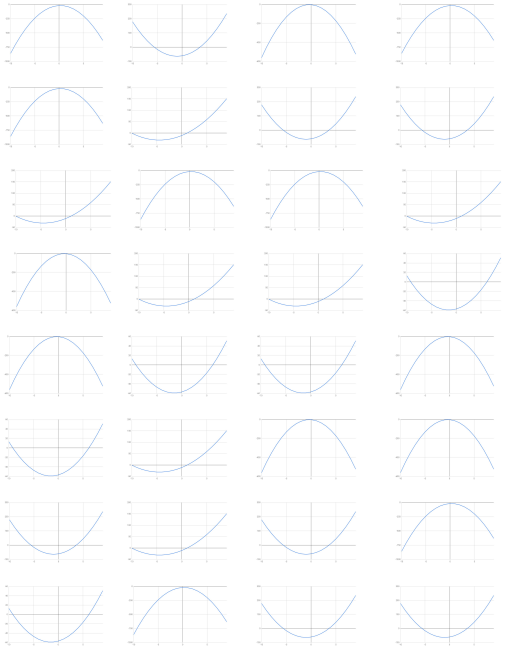
- Secret : c
- 3 parts nécessaires pour reconstruire c
- a, b non divulgués
- Parts : (x_i, y_i)

Partage de secret à seuil de Shamir (SSS)



Deux parts : aucune information sur le secret

Mise en œuvre par HTC



Chiffrement de la seed

- Seed chiffrée envoyée à tous les contacts
- Clé partagée avec SSS
- Bibliothèque open source
  - https://github.com/dsprenkels/sss

5 contacts de confiance, 3 nécessaires pour reconstruire la clé.

- Clé de 256 bits (32 octets)
- 32 polynômes de degré 2
- Coefficients dans GF(2^8)

Génération des secrets

- Clé de chiffrement : 256 bits
- Coefficients des polynômes
  - 32 polynômes de degré 2, $P_i(x) = a_i x^2 + b_i x$
  - Coefficients dans $GF(2^8)$, 8 bits par coefficient

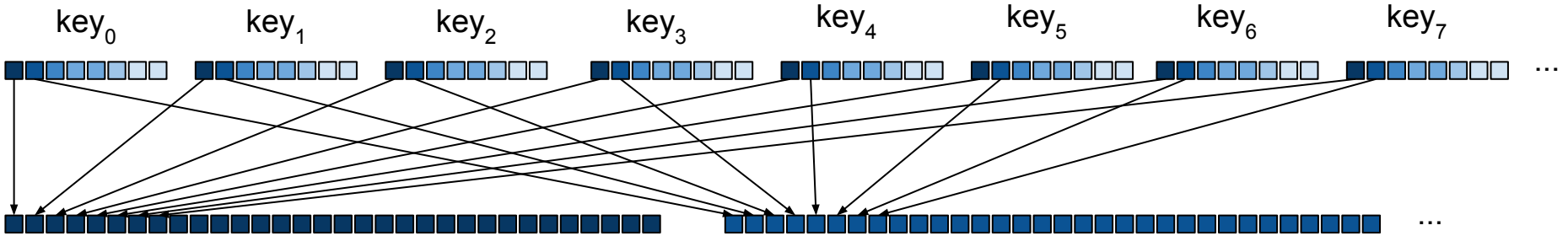
→ 256 + 256 bits

```
sss_rand(key, 32);
/* Put the secret in the bottom part of the polynomial */
bitslice(poly0, key);
/* Generate the other terms of the polynomial */
sss_rand(poly, sizeof(poly));
```

Bitslice

Opérations bit à bit

- Temps constant, bonnes performances



PRNG : initialisation

Ledger

```
void sss_update_secure_random_buffer(const uint8_t *entropy, size_t size) {
  SHA256_CTX ctx;
  uint8_t digest[SHA256_DIGEST_LENGTH];
  uint8_t *p = random_pool;

  sha256_Init(&ctx);
  sha256_Update(&ctx, entropy, size);
  sha256_Final(digest, &ctx);

  for (int i = 0; i < 4; i++) {
    memcpy(p, digest, SHA256_DIGEST_LENGTH);
    sha256_Init(&ctx);
    sha256_Update(&ctx, p, SHA256_DIGEST_LENGTH);
    sha256_Final(digest, &ctx);
    p += SHA256_DIGEST_LENGTH;
  }
}
```

PRNG : génération

```c
#define RANDOM_POOL_SIZE 128
static uint8_t random_pool[RANDOM_POOL_SIZE];


size_t sss_rand(uint8_t *data, size_t len) {
  if (len == 0) {
    return 0;
  }
  while (len > RANDOM_POOL_SIZE) {
    memcpy(data, random_pool, RANDOM_POOL_SIZE);
    data += RANDOM_POOL_SIZE;
    len -= RANDOM_POOL_SIZE;
  }
  memcpy(data, random_pool, len);
  return len;
}
```

- Période de 128 octets
- Fuite de l'état interne
- Etat interne pas modifié

Attaque ?

- Deux appels successifs au PRNG vont retourner les mêmes valeurs
- bitslice() est linéaire → Il existe une relation linéaire entre les bits du secret et les bits des coefficients des polynômes

```
sss_rand(key, 32);   <------------------------------------  c_i
/* Put the secret in the bottom part of the
polynomial */
bitslice(poly0, key);
/* Generate the other terms of the polynomial */
sss_rand(poly, sizeof(poly));   <------------------------  a_i, b_i
```

$P_i(x) = a_i x^2 + b_i x + c_i$

Attaque ?

- Deux appels successifs au PRNG vont retourner les mêmes valeurs
- bitslice() est linéaire → Il existe une relation linéaire entre les bits du secret et les bits des coefficients des polynômes

```
sss_rand(key, 32);
/* Put the secret in the bottom part of the
polynomial */
bitslice(poly0, key);
/* Generate the other terms of the polynomial */
sss_rand(poly, sizeof(poly));
```

c_i

a_i, b_i

~~$P_i(x) = a_i x^2 + b_i x + c_i$~~ $P_i(x) = L_i(c_0, c_1, ..., c_{31})\, x^2 + b_i x + c_i$

Avec trois parts

768 équations :

- 256 équations pour la première part
- 256 équations pour la seconde part
- 256 équations pour la troisième part

768 inconnues :

- 256 coefficients a, 256 coefficients b, 256 coefficients c

→ Résolution du système linéaire, une solution

Avec deux parts

512 équations :

- 256 équations pour la première part
- 256 équations pour la seconde part

768 inconnues :

- 256 coefficients a, 256 coefficients b, 256 coefficients c
- 2^{256} solutions possibles

Solutions du système linéaire

512 équations :

- 256 équations pour la première part
- 256 équations pour la seconde part

~~768 inconnues :~~

- ~~256 coefficients a, 256 coefficients b, 256 coefficients c~~
- ~~2^{256} solutions possibles~~

512 inconnues :

- 256 coefficients b, 256 coefficients c

Résolution

Ledger

Mise en œuvre avec SageMath.

Rang de la matrice : entre 504 et 511, selon les paires de parts.

→ Entre 2 et 256 solutions potentielles selon les paires.

Résolution immédiate :

- Calcul d'une solution particulière.
- Combinaison linéaire avec les éléments de la base pour trouver toutes les solutions du système.
- Test de la validité de chaque solution, une seule est la clé de chiffrement.
- Temps de calcul : ~0,2s.

Conséquences

2 parties sont suffisantes pour calculer la seed.

- Le seuil n’est pas celui attendu.
- Un contact malveillant doit compromettre / corrompre un seul autre contact pour reconstruire la seed.
- Idéalement, il ne devrait y avoir aucun groupe de 3 personnes se connaissant. parmi les contacts de confiance. Abaisser ce seuil à deux est un vrai risque.

Firmware vulnérables

- 1.47.2401.2 : Firmware d'origine ? ← Vulnérable
- 1.53.2401.2 : 18 décembre 2018 ← Vulnérable
- 1.57.2401.6 : 19 février 2019 ← ??
- 1.62.2401.7 : 25 mars 2019

Tous les partages réalisés avant le 19 février 2019 sont affaiblis.

Firmware 1.57.2401.6

`sss_update_secure_random_buffer` n’est jamais appelé.

Le pool d’entropie est initialisé à une valeur fixe.

- Tests reproductibles ?
- Compilation avec des options de test ?

→ Clé de chiffrement fixe

Conséquences

Ledger

Récupération de la seed avec un seul partage.

Tout contact de confiance peut calculer la seed et dérober les fonds.

```
secret_key = b"\x0e\x74\xcd\x69..."
box = nacl.secret.SecretBox(secret_key)
nonce = b"\x00" * 24
encrypted_seed = share1[1 + 32:]
seed = box.decrypt(nonce + encrypted_seed)[:16]
print(mnemonic.Mnemonic('english').to_mnemonic(seed))
```

Conclusion

Vulnérabilités remontées à HTC

- Pas de réponse de l'équipe sécurité.
- Contact avec EXODUS un mois plus tard. Prise en compte des problèmes.
- L'équipe sécurité analyse l'application et corrige d'autres bugs.

Remontée d'autres bugs : corruptions mémoire dans le driver TZ du touchscreen au travers d'I2C, dans la trusted UI, dans les fonctions de lecture des transactions ETH / BTC.

- Tous les problèmes ont été corrigés fin mars 2019.
- HTC EXODUS a lancé son programme de bug bounty le 5 avril 2019.