Projet Cryptographie : Génération de clés RSA embarquée

Laveus Mick-Wanderly

22 avril 2025

Introduction

Ce projet a pour objectif de mettre en œuvre un système embarqué capable de générer des clés RSA de manière sécurisée, en utilisant des sources d'entropie matérielle telles que la mémoire SRAM ou les mesures analogiques via l'ADC. L'idée est de fournir une base fiable pour un générateur de clés dans des contextes contraints, comme les cartes à puce ou les systèmes embarqués sans accès à un générateur aléatoire cryptographiquement sûr.

Structure du projet

Le projet est structuré en plusieurs fichiers sources :

- **entropie.c** : collecte les données d'entropie à partir de la mémoire ou du convertisseur analogique-numérique (ADC).
- **gener_alea.c** : applique un hachage Blake2s aux données d'entropie collectées.
- **blake2s.c** : implémentation complète de la fonction de hachage Blake2s.
- **primalite.c** : implémente le test de Miller-Rabin pour vérifier la primalité des nombres générés.
- **rsa.c** : implémente les opérations de chiffrement/déchiffrement RSA et la gestion des clés.
- **experiment.c** : simulateur pour tester l'interface APDU et les commandes dans un environnement local.
- **fake_avr.c/h** : simulation des registres et fonctions spécifiques AVR pour un environnement de test sur PC.
- **globals.c/h** : contient les variables partagées entre les modules.

Méthodologie de génération

L'entropie brute est collectée via deux sources :

- Initialisation aléatoire de la mémoire SRAM après reboot.
- Valeurs fluctuantes de l'ADC lues à différents intervalles.

Ces données sont ensuite combinées et hachées avec `blake2s` pour produire une séquence pseudo-aléatoire. Exemple :

```
collect_entropy(entropy, ENTROPY_LEN);
blake2s(out, outlen, entropy, ENTROPY_LEN, NULL, 0);
```

La sortie est convertie en entier 64 bits :

```
uint64_t number = candidate_to_uint64(candidate);
```

Test de primalité

Le test utilisé est le test probabiliste de Miller-Rabin avec 5 itérations :

```
if (miller_rabin(number, 5)) {
    printf("Premier probable\n");
}
```

Interface APDU

La communication avec le système s'effectue via des commandes APDU. Chaque instruction est traitée via un switch-case dans `main.c` ou `experiment.c` :

```
switch(ins) {
    case 0xF5: // Génération RSA
    case 0xB7: // Entropie NIST requise
    ...
}
```

Difficultés rencontrées

- **Portabilité AVR/PC** : la compilation croisée pour simulateur PC et carte AVR a nécessité une couche d'abstraction (`fake_avr.h`) pour simuler les registres matériels (MCUSR, OSCCAL, etc.).
- **Conflits de variables globales** : plusieurs fichiers déclaraient les mêmes variables (ex : `cla, ins, p1, ...`) sans utiliser `extern`, causant des erreurs de linkage.

- **Multiples points d'entrée** : deux fonctions `main` étaient présentes (dans `rsa.c` et `experiment.c`), nécessitant la suppression de l'une selon l'environnement de test.
- **Gestion de l'entropie** : les résultats de l'ADC étant trop stables, leur traitement devait être couplé à un XOR et un hash pour générer une entropie convenable.

Résultats et perspectives

Le programme permet de :

- Générer des clés RSA de manière pseudo-aléatoire.
- Vérifier leur primalité avec une probabilité suffisante.
- Tester les fonctionnalités via une interface de communication type carte à puce.

Les perspectives incluent :

- Ajouter la sauvegarde EEPROM des clés.
- Intégrer une interface SPI ou I2C pour la communication externe.
- Migrer l'ensemble vers une vraie carte Arduino ou AVR avec `avr-gcc`.

Conclusion

Ce projet a permis d'explorer en profondeur la génération d'entropie, les contraintes de l'embarqué, les algorithmes de primalité, et les bonnes pratiques de modularisation logicielle. Le simulateur PC permet un prototypage rapide avant déploiement sur cible réelle.