

Travail de Bachelor 2023

Informatique de gestion

Evaluation du framework TamaGo

Etudiant :

Thomas Cheseaux

Professeur :

Jean-Luc Beuchat

Résumé

Le stockage, la génération et la gestion des clés de chiffrement sont cruciaux pour garantir la sécurité, la confidentialité et l'intégrité des systèmes qui les utilisent. Pour garantir cette protection, des parts de clés peuvent être stockées dans des hébergements en ligne distincts. A savoir qu'une gestion inadéquate peut entraîner de graves problèmes de sécurité et compromettre les données protégées par ces clés.

Ce travail porte sur l'analyse de l'utilisation de la clé *USB Armory Mk II* en tant que dispositif de sécurité complémentaire à des hébergements en ligne. Autrement dit, l'objectif de ce rapport vise à déterminer la faisabilité de l'utilisation du *framework* TamaGo pour le développement d'une application de démonstration en Go dans le but de l'employer comme moyen de sécurité additionnel. De plus, l'enjeu du développement sur ce périphérique représente un défi en raison de la volonté de ne pas utiliser un système d'exploitation pour exécuter notre application. Pour atteindre notre objectif, nous effectuerons une première recherche sur les *frameworks* existant, suivie d'une étape de développement utilisant une approche similaire à la méthodologie *Agile* avec des rencontres hebdomadaires pour le suivi du travail. Puis, nous présenterons notre application test utilisant un service *gRPC* et la documentation OpenApi et finalement, nous concluerons sur les limites rencontrées lors du développement et sur les perspectives de recherches possibles.

Mots clés : USB Armory Mk II, Multi-Party Computation, MPC, TamaGo, Go

Remerciements

Jean-Luc Beuchat, enseignant responsable du travail, pour son aide, ses relectures et ses modifications qui ont permis d'élaborer ce travail.

Louis Mayencourt, ingénieur en système embarqué, pour ses explications claires et précises de l'environnement du système embarqué.

Andrea Barisani, développeur du framework TamaGo, pour son aide, sa réactivité et sa disponibilité à répondre à mes nombreuses questions.

Joyce Carraud qui a pris le temps de relire ce travail. Merci pour ta patience, ta compréhension et ton soutien indéfectible.

Table des matières

Table des matières	iv
Table des figures	vi
Acronymes	vii
1 Introduction	1
1.1 Contexte	1
1.2 Objectifs	2
1.3 Sujet traité	3
1.4 Problématique	4
1.5 Méthodologie	5
1.6 Environnement de travail	5
2 Présentation des concepts et des technologies	7
2.1 Langage Go	7
2.2 USB Armory Mk II	8
2.3 Génération de nombres aléatoires en cryptographie	12
2.4 Multi-Party Computation (MPC)	13
2.5 Hardware Security Modules	14
3 État de l'art	15
3.1 Introduction	15
3.2 Langage bas niveau	15
3.3 Framework existants	21
3.3.1 Bare metal Rust	21
3.3.2 TamaGo	22
3.4 Critères de sélection	25
3.5 Détermination d'une solution	26
4 Mise en oeuvre	27
4.1 Mise en place d'un environnement de développement en Go	27
4.1.1 Installation de Golang	27
4.1.2 Installation de l'environnement de développement	27
4.1.3 Configuration pour la compilation et le déploiement sur macOS	27

4.2	Communication entre l'ordinateur et l'Armory Mk II	28
4.2.1	Installation sur une clé USB	28
4.2.2	Configuration du partage de connexion	29
4.2.2.1	macOS	29
4.2.2.2	Linux	31
4.2.3	Configuration du masquerading pour les connexions sortantes	34
4.2.3.1	macOS	34
4.2.3.2	Linux	35
4.2.4	Exécution des fonctionnalités de l'application	36
4.2.4.1	macOS	36
4.2.4.2	Linux	36
4.2.5	Lancement des serveurs HTTP de l'application	37
4.3	Implémentation d'une application de démonstration sur l'ordinateur	38
4.3.1	Serveurs REST et gRPC	38
4.3.2	OpenApi et Swagger	40
4.3.3	gRPC-Gateway	41
4.3.4	Structure du code	42
4.3.5	Code source de l'application de démonstration	42
5	Proof of concept	46
5.1	Code source TamaGo	46
5.2	Test de la qualité du <i>True Random Number Generator</i>	54
6	Alternatives	56
6.1	Hardware Security Modules	56
6.2	SoftHSM	56
6.3	Solution cloud	57
7	Conclusion	59
7.1	Synthèse	59
7.2	Limites du travail	59
7.3	Perspectives de recherches	60
	Bibliographie	62
	Glossaire	68

Table des figures

2.1	Image de l'Armory Mk II	9
2.2	Use case d'une solution Armory Drive	11
2.3	Architecture d'un OS traditionnel et d'une architecture TamaGo	11
2.4	Schéma de génération de nombres aléatoires	12
2.5	Exemple de tableau pour illustrer le concept de MPC	13
3.1	Comparaison des langages basés sur le contrôle et la simplicité d'utilisation	23
4.1	Extension Go pour Visual Studio Code	27
4.2	Résultat de la commande VBoxManage list usbhost	30
4.3	Résultat de la commande ifconfig	30
4.4	Affichage du statut inactif de l'interface après activation	30
4.5	Connexion SSH à l'Armory Mk II sur macOS	31
4.6	Configuration du port de l'Armory Mk II sur VirtualBox	32
4.7	Connexion SSH à l'Armory Mk II sur une machine virtuelle Ubuntu	33
4.8	Partage Internet activé	34
4.9	Configuration du partage de connexion sur macOS	35
4.10	Affichage des résultats des méthodes de l'application de démo sur macOS	36
4.11	Affichage des résultats des méthodes de l'application de démo sur une machine virtuelle Ubuntu	36
4.12	Menu d'affichage du serveur HTTP de l'Armory Mk II	37
4.13	Affichage de plots représentant les métriques de l'Armory Mk II	37
4.14	Exemple de fonctionnement d'un service gRPC	40
4.15	Exemple d'affichage d'un service API avec Swagger UI	41
4.16	Génération du serveur API en gRPC et en REST	41
4.17	Affichage de la méthode de génération de nombre aléatoire avec Swagger UI	45
5.1	Affichage du service gRPC-Gateway avec OpenApi	53
5.2	Affichage des résultats de la suite des tests statistiques du nombre aléatoire	54

Acronymes

CPU Central Processing Unit. 7, 16–18

DNS Domain Name System. 57

DNSSEC Domain Name System Security Extensions. 56, 57

FIPS Federal Information Processing Standard. 14, 57

gRPC google Remote Procedure Call. ii, 38, 40–44, 46, 47, 51, 53, 54, 60

HSM Hardware Security Modules. 1, 14, 55–58, 70

HTTP Hypertext Transfer Protocol. 38, 42, 50–52, 54

IP Internet Protocol. 30, 35, 57

JSON JavaScript Object Notation. 38, 39

KMS Key Management Service. 1, 57

MPC Multi-Party Computation. 1, 2, 13

OS Operating System. 3, 4, 24–26, 68

PME Petite ou Moyenne Entreprise. 14, 56

RNGB Random Number Generator. 9

SOAP Simple Object Access Protocol. 38

SQL Structured Query Language. 19

SSH Secure Shell. 10, 29, 31, 33, 37

TLS Transport Layer Security. 51

URI Uniform Resource Identifier. 38

URL Uniform Resource Locator. 53

USB Universal Serial Bus. ii, 1–3, 5, 7–11, 15, 25, 28, 29, 31, 34, 46, 48, 54

XML Extensible Markup Language. 38

1 | Introduction

1.1 Contexte

La gestion des clés cryptographiques est une tâche complexe. Cette gestion nécessite le choix d'une méthode de génération des clés et d'une réflexion sur le cycle de vie ([MINISTÈRE D'ÉTAT, 2018](#)). Il est important de noter que la gestion des clés cryptographiques requiert de très bonnes connaissances en la matière et induit une haute responsabilité. Plusieurs options permettent de gérer des clés cryptographiques. :

- Azure Key Vault ; La solution Microsoft Azure Key Vault est un service en ligne qui sauvegarde et gère l'accès à des données sensibles comme mots de passe, certificats, clés cryptographiques ainsi que tout autre secret confidentiel ([KEITHP, s. d.](#)).
- AWS Key Management Service (*KMS*) ; Comme la solution présentée ci-dessus, le *KMS* offre une gestion similaire des clés de chiffrement.
- *Hardware Security Modules (HSM)* ; Les *Hardware Security Module* sont des appareils informatiques destinés à gérer les clés de chiffrement. Ce type de matériel est considéré comme inviolable. Il stocke, protège et génère des clés cryptographiques. Les clés restent dans la zone de protection du module et les données sont traitées uniquement à l'intérieur de celui-ci. Par exemple, ces appareils peuvent être intégrés dans les solutions *Cloud*.
- *MPC* ; Le Multi-Party Computation est une solution de partage d'un secret en plusieurs parties à travers différentes entités. Ainsi, une clé de chiffrement peut être protégée sans l'utilisation d'un *HSM* ([DVORIN, 2019](#)). Cela permet à la fois d'éviter la complexité des modules *HSM* mais aussi les coûts qu'ils entraînent.

Dans notre cas, les clés de chiffrement sont stockées en plusieurs parts dans des hébergements en ligne distincts. Les opérations de chiffrement, de déchiffrement ou de signature sont effectuées sans reconstruire la clé cryptographique en un point unique. En d'autres termes, l'utilisation du *MPC* implique que la clé n'est jamais reconstruite. L'objectif à moyen terme serait d'utiliser notre travail pour stocker une part des clés de chiffrement sur une clé *USB* sécurisée en complément des hébergements en ligne. Nous décrivons plus tard dans le rapport le fonctionnement des *HSM* et du *MPC*.

Afin de compléter le dispositif de sécurité que constituent les clés cryptographiques, deux scénarios sont envisagés :

- L'installation du composant hardware *MPC* dans un *data center*.
- La matérialisation du composant en un périphérique *USB* sécurisé.

Ces deux scénarios permettent d'implémenter des solutions dont le but est de réaliser des opérations ou calculs sécurisés en préservant la confidentialité des données.

Avant de poursuivre, nous souhaitons nous arrêter sur deux concepts de la cryptographie ; le Secret Sharing et le *Multi-Party Computation (MPC)*.

- Le Secret Sharing appelé aussi Shamir est utilisé pour gérer la reconstruction de clés cryptographiques. La reconstitution d'un secret nécessite l'assemblage soit de la totalité des parts, soit d'une partie de celles-ci. Un seuil détermine le nombre de parts du secret nécessaire à la reconstitution de l'entièreté de la clé. En d'autres termes, le seuil définit le nombre minimum de parties requises pour reconstituer un secret ou une clé. Certains algorithmes peuvent définir ce seuil. Notons aussi que les parties du secret sont liées les unes aux autres.

Pour illustrer nos propos, prenons un exemple ; deux parts du secret sont stockées dans des serveurs en ligne et trois parts sur des dispositifs réparties entre des personnes autorisées à accéder au service. Si le seuil est fixé à quatre, il est nécessaire de rassembler quatre parts de la clé pour la reconstituer. Ici, les personnes qui détiennent une clé *USB* contenant une part du secret agissent comme un second facteur d'authentification. Ils garantissent un dispositif de contrôle supplémentaire en déterminant quelles personnes sont légitimes à utiliser la clé *USB*.

- L'autre concept est le *MPC*, celui-ci permet aux parties de réaliser des opérations sur des données tout en maintenant la confidentialité de leurs propres données. Aucune reconstruction de la clé ou du secret n'est effectuée.

En résumé, le Secret Sharing est un moyen spécifique de diviser un secret entre plusieurs parties, alors que le *MPC* permet à plusieurs personnes de collaborer sur des calculs tout en préservant la confidentialité de leurs données. En outre, le *MPC* ne reconstruit jamais la clé alors que le secret Sharing le fait.

L'objet de notre travail portera donc sur l'analyse de ce second scénario, c'est-à-dire l'utilisation de périphérique *USB* en tant que dispositif de sécurité complémentaire des clés cryptographiques.

Pour la réalisation de ce travail, nous utilisons le genre masculin générique pour représenter l'ensemble des personnes concernées.

1.2 Objectifs

Le périphérique *USB* considéré lors de la réalisation de ce travail est une clé *USB* Armory Mk II. Celle-ci est détaillée plus tard dans le rapport sous la section 2.2. Afin de parfaire l'utilisation de ce périphérique en tant que dispositif supplémentaire de sécurité, nous aurons pour mission de déterminer le *framework* rendant possible son application aux clés cryptographiques.

Pour ce faire, nous commencerons par répertorier, analyser et comparer les solutions pour le développement sur des appareils *bare metal*. Le développement *bare metal* n'utilise pas de système d'exploitation pour exécuter un programme. L'absence d'un OS réduit la surface d'attaque car moins de fonctionnalités et d'interfaces sont exposées. Il est important de noter que l'Armory Mk II pourrait contenir un OS et exécuter un programme en Go. Sur la base de ces observations, nous porterons une attention particulière à la compréhension et au développement d'une application test en Go. Plus précisément, nous évaluerons le *framework* TamaGo. Pour cela nous effectuerons tout d'abord un test de ses fonctionnalités et étudierons les spécificités de son implémentation sur la clé USB Armory Mk II. Dans un second temps, nous développerons une application de démonstration qui nous permettra d'éprouver concrètement l'implémentation de TamaGo au sein du périphérique USB.

Il est à noter que cette étude cherche à déterminer la complexité, mais surtout la faisabilité de l'exécution du *framework* TamaGo. De ce fait, le contenu de ce travail mettra en lumière les difficultés d'implémentation rencontrées, les solutions trouvées ainsi que les résultats obtenus.

1.3 Sujet traité

TamaGo¹ est un *framework* basé sur le langage Go. Ce cadre de développement offre la possibilité d'exécuter et de compiler sur des systèmes embarqués de type ARM ou RISC-V. Par système embarqué, nous entendons un système construit pour effectuer de manière autonome des tâches spécifiques et être intégré sur un périphérique. Le projet de développement d'une solution fonctionnant sans système d'exploitation découle de la volonté de réduire les dépendances à l'OS.

Les appareils qui n'utilisent pas d'OS pour exécuter leur programme sont appelés *bare metal*. Pour certaines applications, l'implémentation sur ce type d'appareil offre une exécution rapide, performante et consomme moins de mémoire. En effet, l'absence d'un OS permet de préserver de la mémoire. Cela diminue aussi le temps d'exécution de l'application étant donné que le programme peut s'exécuter directement sur les composants.

Le *framework* TamaGo se compose de la manière suivante (BARISANI, 2023a) :

- Une modification de la distribution du langage Go.
- De *packages* permettant l'exécution sur un système embarqué.
- De *packages* permettant la communication entre le périphérique et son environnement.

En s'exécutant sur des appareils *bare metal* - autonomes au sens où ils ne nécessitent l'utilisation d'aucun autre logiciel - TamaGo fortifie le dispositif de sécurité des données. Par ce biais, les vulnérabilités liées à l'OS telles que l'exécution non autorisée de programmes, le *privilege escalation*, les bugs et les vulnérabilités dites *zero-days* sont évitées.

1. <https://github.com/usarmory/tamago>

De plus, TamaGo contient uniquement les fonctionnalités minimales à son exécution. De par l'absence d'OS, le *framework* s'exécute uniquement en mode utilisateur. De cette manière, les potentialités pour un utilisateur malveillant de modifier ou d'exécuter son propre programme sont plus contraignantes. Malheureusement, le risque zéro n'existe pas, mais nous pouvons augmenter la protection de nos systèmes et programmes pour dissuader et décourager ces personnes.

1.4 Problématique

L'enjeu du développement sur une clé Armory Mk II se constitue d'une exécution sans système d'exploitation. En effet, l'absence de système d'exploitation réduit fortement les langages ou *framework* permettant l'exécution d'un programme autonome sur ce type d'appareil.

Un OS intégré dans un système embarqué permet l'abstraction de la complexité matérielle et logicielle (ROSSIER, 2014). La complexité matérielle regroupe les trois grandes catégories du matériel d'un ordinateur : la mémoire, le processeur ainsi que les entrées/sorties (souris, carte réseau, haut-parleur, clavier) (QKZK, 2020). La complexité logicielle s'applique quant à elle à la gestion d'une application à l'aide d'outils ou de services intégrés dans un système d'exploitation.

L'avantage d'un système d'exploitation installé sur une machine est qu'il offre à l'utilisateur un aperçu simplifié de la gestion des ressources matérielles de l'ordinateur. Le système d'exploitation est un outil qui facilite la manipulation et l'exécution d'une application en proposant à l'utilisateur un environnement graphique, un compilateur ou un terminal de commande pour gérer son espace. Le système d'exploitation peut être comparé à une couche d'abstraction qui affranchit l'utilisateur de tâches lourdes comme, la gestion de la mémoire, l'affichage des compteurs de temps ou encore la communication avec les périphériques (TANENBAUM, 2008).

Malgré les nombreux avantages d'un système d'exploitation, son absence peut également constituer une plus-value. Pour exemple, le fait qu'une clé Armory Mk II ne nécessite pas de système d'exploitation pour fonctionner présente les avantages suivants :

- Gestion efficace des ressources : sans système d'exploitation les ressources sont consacrées uniquement à la tâche dédiée du système embarqué. Dans ce cas-ci, l'efficacité du système est optimisée.
- Temps de démarrage réduit : sans système d'exploitation, les processus sont simplifiés et l'appareil peut démarrer plus rapidement. Le système n'a pas besoin d'effectuer les tâches de vérification au démarrage telles que l'initialisation des services (processeur, mémoire, périphérique) ou la vérification de l'intégrité du système. Aussi, la phase de chargement de l'OS est évitée, cela réduit le temps au démarrage.
- Sécurité améliorée : la vulnérabilité et la surface d'attaque sont réduites car ces systèmes ont moins de fonctionnalités et d'interface exposées. Par ce biais, l'intégrité et la sécurité de l'appareil sont renforcées, octroyant davantage de crédit à son utilisation pour des applications en cryptographies.

- Contrôle absolu : un développeur obtient un contrôle total sur le matériel et les fonctionnalités qu'il décide d'implémenter.

En conclusion, le périphérique Armory Mk II ne nécessite pas de système d'exploitation lors de l'exécution ce qui peut à la fois être un avantage pour les critères susmentionnés mais aussi un handicap lorsqu'il s'agit de trouver un langage susceptible (Go, Rust, Assembleur, C/C++) d'exécuter un programme autonome dans ces conditions. Présupposant que le *framework* TamaGo est adapté à cette configuration particulière, il s'agira de vérifier la faisabilité de son implémentation sur la clé Armory Mk II.

1.5 Méthodologie

Afin de bien organiser notre travail, une première recherche sur les *frameworks* existants est nécessaire de façon à déterminer les avantages, les inconvénients ainsi que les cas d'utilisation de chaque solution trouvée. Le développement sur une clé *USB* de type Armory nécessite une compréhension claire du fonctionnement de ce type d'appareil ainsi qu'une connaissance des langages compatibles à la compilation sur la clé. La réalisation de ce travail utilise les nombreuses ressources présentes en ligne dont notamment, site officiel, vidéos explicatives ainsi que des *repository Git*. Toutes ces ressources sont nécessaires au choix et au développement d'un service sur une clé *USB* Armory Mk II.

Ce rapport se subdivise en plusieurs parties : une première section de présentation des concepts et des langages compatibles aux développement *bare metals*, une seconde qui résume les étapes de mise en œuvre, une autre présentant le code source de notre application et finalement une dernière partie concernant les alternatives et les perspectives de recherche. La mise en place d'une méthode *Agile* complète n'est pas approprié car la taille du projet est relativement petite et les objectifs fixés sont déjà bien définis. De ce fait, une mise en place complète semble inadaptée. Néanmoins, il est convenu d'une rencontre hebdomadaire avec la partie prenante pour le suivi de l'avancement du rapport ainsi que des problèmes rencontrés. Ces rencontres hebdomadaires sont l'occasion de discuter d'éléments techniques concernant le développement de ce travail et des outils implémentés.

A ce moment du développement, nous ne savons pas encore si la mise en place est possible. Tout au long de l'implémentation, nous tenons un historique des difficultés rencontrées décrites dans la section 7.2.

1.6 Environnement de travail

Le code source de notre application de démonstration sur la machine et celle produite pour le périphérique *USB* se trouvent sur le Gitlab sous les dénominations THOMAS CHESEaux 2023² :

2. <https://gitlab.com/hesso-vs/business-information-technology/65-62-bachelor-thesis>

Chapitre 1. Introduction

- `gRPC Machine` contient le code source de l'application de démonstration exécutée sur une machine.
- `TamaGo` comprend l'application de démonstration pour l'Armory Mk II ainsi qu'un fichier README avec des explications complémentaires pour l'implémentation.
- `TamaGo - Thesis` regroupe l'entièreté de mon travail de Bachelor codé sur \LaTeX

2 | Présentation des concepts et des technologies

Dans ce chapitre, nous décrivons les concepts et les technologies mobilisés au cours de la rédaction du rapport. Premièrement, nous présentons le langage Go et son utilisation ainsi que ses spécificités. Dans un second temps, nous exposons les détails du périphérique *USB Armory Mk II* et son utilité. Puis, nous montrons l'importance de la génération de nombre aléatoire dans le cadre de la cryptographie. Ensuite, nous expliquons ce qu'est le *Multi-Party Computation*. Enfin, nous terminons avec la définition et l'utilisation des *Hardware Security Modules*.

2.1 Langage Go

Dans cette section, nous parlons du langage de programmation Go et de son importance dans le monde informatique.

Go ou Golang est un langage de programmation développé par l'entreprise Google en 2007. Depuis 2009, ce langage est accessible au public et connaît une forte popularité ([GOOGLE, 2020](#)).

Pour la suite de la rédaction du rapport, la dénomination Go sera utilisée pour mentionner ce langage de programmation.

Go est un langage compilé. Un langage compilé se décrit par une conversion directe en langage machine de son code afin que le processeur puisse l'exécuter. Il est donc directement exécuté par l'ordinateur ([GEEKSFORGEEKS, 2022](#)).

En quelques points, le langage Go rassemble les meilleures propriétés d'autres langages de programmation ([Go, s. d.](#)) :

- Malgré la ressemblance au langage C, la prise en main du langage est plus facile pour des utilisateurs ayant des connaissances de langage haut niveau comme Java ou C#. De plus, la gestion de la mémoire est traitée par le *garbage collector*.
- Le typage de Go est statique. Cette caractéristique permet une vérification dès la compilation et non uniquement durant l'exécution du programme. La valeur instanciée et son type ne peuvent pas être modifiés dans la suite du programme.
- Go est un langage *opensource*. Toute personne qui souhaite modifier, visualiser et améliorer le code peut le faire.
- Go a été pensé et conçu pour être utilisé sur des systèmes à plusieurs coeurs. Les ressources des *CPU* peuvent être pleinement exploitées sans être un casse-tête pour des développeurs.

- La manipulation de *threads* appelés aussi *Goroutines* peut être faite simplement avec le mot "go" placé devant une fonction. Leur taille et leur *stack* peuvent augmenter et se réduire selon les besoins de l'application. Les *Goroutines* utilisent les *Channels* pour communiquer. Les *Channels* sont un moyen de déplacer et d'échanger des données entre différents *Goroutines*.

```
go myList.Sort() // méthode myList.Sort() est lancée parallèlement
```

- De grandes entreprises telles que Netflix, Google et Meta l'utilisent dans le développement de leurs produits. La fréquence d'utilisation de ce langage par ces entreprises crédibilise davantage le Go.
- La création et l'instanciation des variables sont aisées. Il existe d'ailleurs différentes façons de créer ces variables.

```
1  var montagne string
2  montagne = "Catogne" // 1ère
3
4  var montagne string = "Catogne" // 2e
5
6  var montagne = "Catogne" // 3e
7
8  montagne := "Catogne" // 4e utilisation la plus courante
```

- Go est utilisé dans des applications diverses telles que GoKey, un gestionnaire de mots de passe dont l'avantage réside dans le fait qu'il ne stocke pas les mots de passe mais les dérive au fur à mesure de leur utilisation ([CLOUDFLARE, 2023](#)). Go est aussi utilisé dans la *blockchain* (GoCoin) ou encore la gestion des *containers* (Docker et Kubernetes).

Pour résumer, Go est un langage *opensource* et statiquement typé qui constitue un choix judicieux pour le développement back-end. Sans aucunement prétendre à l'exhaustivité, les facteurs exposés ci-dessus justifient le choix de ce langage dans la création et le déploiement de diverses applications.

2.2 USB Armory Mk II

Cette section décrit les propriétés d'une clé *USB Armory Mk II*, son champ d'application ainsi que la sécurité de ses composants.

Tout d'abord, cet appareil est développé par l'entreprise WithSecure spécialisée dans la cybersécurité. La version utilisée (Mk II) est le successeur du premier modèle développé (Mk I). Selon le site officiel de l'entreprise, cette clé *USB* représente le plus petit et le plus sûr des ordinateurs ([WITHSECURE, s. d.-a](#)).

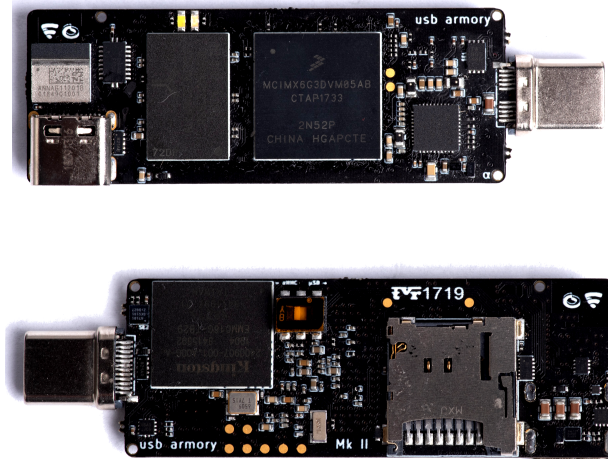


FIGURE 2.1 – Image de l'Armory Mk II

Source : à partir de <https://github.com/usbarmory/usbarmory/wiki/Mk-II-Introduction>

L'Armory Mk II peut :

- Sauvegarder des données sur sa mémoire embarquée *eMMC* de 16GB ou sur une carte micro SD.
- Exécuter des applications sans système d'exploitation.
- Exécuter des systèmes d'exploitation comme des images Debian ou Kali Linux.
- Prévenir des accès non-autorisés grâce au *secure boot* qui n'autorise que des programmes signés à s'exécuter.
- Configurer une communication entre la clé et l'utilisateur via bluetooth.

Cette clé *USB* peut être utilisée pour de nombreuses applications telles que les sauvegardes chiffrées, la gestion de clés cryptographiques, un pare-feu embarqué ou encore comme un coffre-fort pour des cryptomonnaies.

L'Armory Mk II est un outil flexible, minimal et ouvert. Ses configurations sont multiples allant d'un système d'exploitation à l'exécution de programme. Les caractéristiques de ce produit permettent de réduire le potentiel d'attaque ([WITHSECURE](#), s. d.-b).

Les principales fonctionnalités de sécurité de l'Armory Mk II sont les suivantes :

- L'utilisation d'un *True Random Number Generator* : un générateur aléatoire de nombres est essentiel en cryptographie pour construire des clés de chiffrement. Pour vérifier la qualité et l'efficacité de ses services, le générateur passe des tests statistiques ([SCHNEIER, 1996](#)). Cette fonctionnalité est possible car le processeur i.MX6ULZ installé sur la clé *USB* intègre un module *RNGB* ([NXP, 2018](#)). De plus ce module se retrouve dans les noyaux Linux et dans TamaGo ([BARISANI, 2023c](#)). Le *TRNG* se base sur la collecte de bits à partir d'une source physique de bruit aléatoire. Cette source de bruit provient d'un oscillateur

en anneau sensible au bruit aléatoire (variations de températures, variations de tensions, autres bruits aléatoires) au sein du dispositif dans lequel il est utilisé ([TECHSUPPORT, 2021](#)).

- La mise en place de la technologie *Secure Non-Volatile Storage* : une protection des données sensibles contre les accès non-autorisés, la modification directe et l'altération de l'intégrité d'un composant sans y avoir droit ([NXP, s. d.](#)).
- L'utilisation d'un *High Assurance Boot* : cette fonctionnalité assure l'intégrité et l'authenticité du processus de *boot*. Il est important de noter que cette sécurité protège contre la modification non autorisée ou l'exécution de logiciel malveillant ([DEVICES, 2016](#)).

Comme nous l'avons lu, l'Armory Mk II rassemble les composants et fonctionnalités nécessaires au développement d'applications nécessitant une sécurité informatique accrue. Pour ces raisons, quelques applications ont déjà été développées avec le *framework* TamaGo telles que GoKey et Armory Drive illustrant la diversité d'utilisation de l'appareil ([BARISANI, 2023a](#)). Ainsi, dans la partie ci-dessous, nous décrivons brièvement quelques-unes d'entre elles.

GoKey

GoKey est construit et fonctionne sur le *framework* TamaGo décrit ci-dessus. Cette solution imite la fonction des cartes appelées *smart card* traditionnelles. Ces cartes ont de nombreuses utilités telles que l'authentification *MFA* ou encore le paiement (Visa, MasterCard). Celles-ci offrent un emplacement pour des clés sécurisées et sont accessibles via une interface en ligne de commande comme *SSH*. Les clés sécurisées utilisées pour le chiffrement sont sauvegardées sur le périphérique *USB* ([BARISANI, 2023b](#)).

Armory Drive

L'application Armory Drive permet de renforcer la sécurité d'un périphérique *USB* en limitant l'utilisation de celui-ci par une couche de chiffrement supplémentaire et une authentification à plusieurs facteurs. Ce système autorise le déverrouillage d'une carte micro SD à l'aide d'une application mobile. Ce système est géré grâce à la fonctionnalité bluetooth intégrée sur la clé USB. D'une façon simple, l'application implémente une solution d'accès sécurisé au disque chiffré avec n'importe quelle carte microSD. Donc, un utilisateur en possession de cette clé ne pourrait pas lire son contenu sans l'aide d'une application tierce. Aussi, si les données sont écrites directement sur le périphérique, l'accès aux données est d'autant plus difficile en raison de la protection en cas d'altération physique de la mémoire. Le schéma ci-dessous montre le fonctionnement de la solution Armory Drive. Nous constatons que l'utilisateur débloque le contenu de la clé *USB* à l'aide d'une application de contrôle installée sur son smartphone ([BARISANI, 2022](#)).

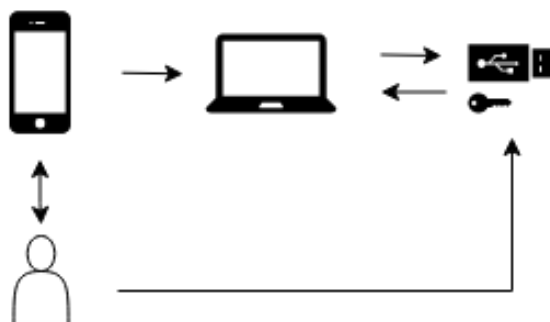


FIGURE 2.2 – Use case d'une solution Armory Drive

Source : adaptée à partir de <https://www.withsecure.com/content/dam/with-secure/en/resources/withsecure-usb-armory-brochure-en.pdf.coredownload.pdf>

Les deux applications GoKey et Armory Drive sont des exemples d'architectures orientées sur la sécurité soutenues et rendues possibles par l'Armory Mk II. Ainsi, ces illustrations nous laissent entrevoir l'étendue d'utilisation du petit ordinateur embarqué que constitue le périphérique Armory Mk II.

TamaGo

TamaGo est pris en charge par la clé *USB Armory Mk II*. Ce *framework* permet l'exécution d'application en Go dans un système embarqué muni d'un processeur de type *ARM*. Pour illustrer nos propos, l'image ci-dessous présente d'un côté les interactions entre les composants, applications et pilotes et un système d'exploitation ; puis, de l'autre, la relation entre les mêmes composants à TamaGo - en l'absence cette fois-ci de système d'exploitation.

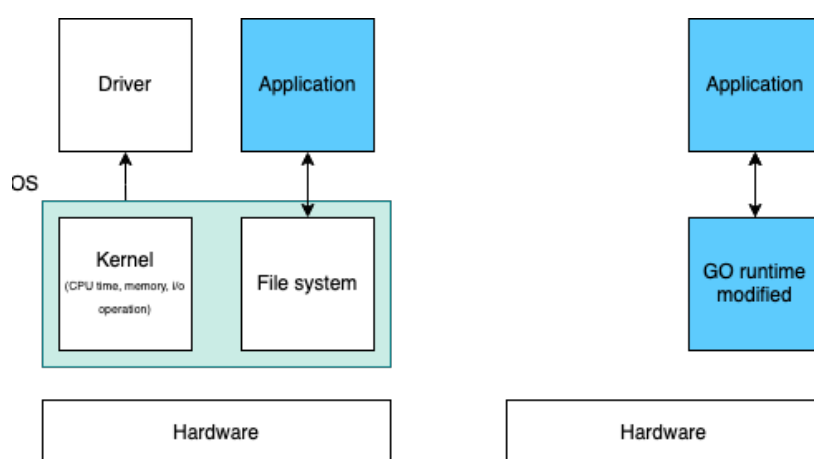


FIGURE 2.3 – Architecture d'un OS traditionnel et d'une architecture TamaGo

Source : adaptée à partir de <https://www.withsecure.com/content/dam/with-secure/en/resources/withsecure-usb-armory-brochure-en.pdf.coredownload.pdf>

2.3 Génération de nombres aléatoires en cryptographie

La protection de la vie privée, des communications et des données est un enjeu important pour les acteurs des secteurs industriels et privés. Une manière de protéger ces données se constitue autour de leur chiffrement. Pour ce faire, une attention particulière doit être portée à la qualité de séquençage des nombres aléatoires, qui conditionnent l'intégrité et la sécurité des protocoles cryptographiques (LAYAT, 2006).

La structure des générateurs de nombres aléatoires se divise en deux types :

- Les générateurs qui utilisent un phénomène physique appelé aussi *True Random Number Generator*.
- Ceux qui produisent des nombres aléatoires à partir d'algorithmes déterministes nommés Pseudo-RNG (SOUCARROS, 2006).

La figure ci-dessous, tirée de la thèse de Mathilde Soucarros, explique le fonctionnement des générateurs de nombres aléatoires. Comme mentionné, les *True Random Number Generator* sont produits à partir d'une source physique d'entropie. Ce mode de génération mobilise des phénomènes physiques (*bruit thermique* ou *effet photoélectrique*) pour générer des nombres aléatoires. Les signaux créés sont numérisés, puis un retraitement algorithmique est appliqué afin d'améliorer leur qualité. Le traitement numérique désigne l'application d'un algorithme comme des fonctions de hachages ou des algorithmes de chiffrement par blocs ou encore ceux qui se basent sur la théorie des nombres. Pour exemple nous pouvons citer Diffie-Hellman qui est un algorithme permettant l'échange de clés sans que personne ne puisse les intercepter (SOUCARROS, 2006). Les nombres pseudo-aléatoires ne bénéficient pas d'une source physique pour créer leur aléa. Comme leur nom l'indique, ces nombres semblent aléatoires alors qu'en réalité ils ne le sont pas. Pour ces derniers, un algorithme déterministe muni d'une graine aléatoire et d'un traitement numérique sont nécessaires à leur génération (LAYAT, 2006).

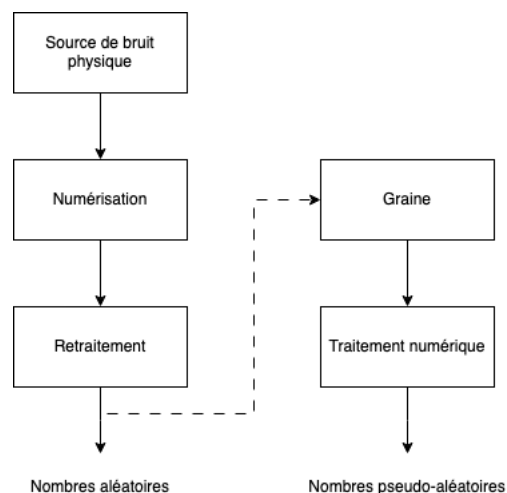


FIGURE 2.4 – Schéma de génération de nombres aléatoires

Source : adaptée à partir de <https://theses.hal.science/tel-00759976v3>

Une graine aléatoire est un nombre utilisé pour initialiser les générateurs de nombres pseudo-aléatoires (HENKEMANS et LEE, 2001).

L'entropie ou entropie de Shannon est un indicateur qui qualifie l'aléa d'une source. Les *True Random Number Generator* utilisent la variabilité imprévisible des données physiques pour produire des nombres aléatoires. Plus cette variabilité est grande, moins il est possible de prédire la suite, et cela se traduit par un haut niveau d'incertitude et d'imprévisibilité et donc d'une entropie de Shannon élevée (CHEVALIER, 2022).

De nos jours, les nombres aléatoires sont cruciaux dans la cryptographie. Ils garantissent l'intégrité et la sécurité des échanges, des protocoles et, plus largement, la protection des données. Imiter le hasard dans la génération de nombres aléatoires est un moyen sûr de protéger l'intégrité des clés de chiffrement et d'empêcher que des personnes malveillantes déterminent les moyens mis en place pour les générer.

2.4 Multi-Party Computation (MPC)

Le MPC, appelée aussi calcul multipartite sécurisé, est une branche de la cryptographie. « Le MPC a été proposé il y a plus de 30 ans, pour permettre à des utilisateurs, possédant chacun des données secrètes, d'effectuer un calcul commun et d'obtenir le résultat tout en gardant les entrées confidentielles » (POINTCHEVAL, 2021). Le MPC a pour objectif de permettre aux agents d'un réseau de communication de calculer conjointement une fonction sur leurs entrées, afin que les entrées restent privées et que le résultat soit exact .

Pour illustrer ces propos, nous présentons ci-après un exemple. Un groupe de quatre personnes, travaillant dans le même secteur, souhaite connaître la moyenne de leur salaire. Le but est de savoir si mon salaire se situe en-dessus ou en-dessous de la moyenne. Dans cet exemple, les personnes sont dénotées P1, P2, P3, P4. La personne P1 choisit quatre nombres "aléatoires" (400, 300, -200, 500) dont la somme équivaut à son salaire (1000). Ces nombres sont ensuite distribués aux autres participants (PARTISIA BLOCKCHAIN FOUNDATION, 2022).

		P1	P2	P3	P4
P1	1000	400	300	-200	500
P2	1500	1000	-500	700	300
P3	2300	1000	800	-500	1000
P4	3450	3000	500	-700	650
		5400	1100	-700	2450

FIGURE 2.5 – Exemple de tableau pour illustrer le concept de MPC
Source : adapté à partir de <https://www.youtube.com/watch?v=vRVudJADQLk>

Après distribution, il convient d'additionner l'ensemble des nombres reçus par chaque participant. De cette façon nous obtenons les montants suivants : 5400, 1100, -700, 2450, qui sont rendus publics. En additionnant ces quatre sommes, puis en les divisant par le nombre de participants, nous obtenons la moyenne des données fournies, soit 2'062,50 CHF.

Ce procédé permet de définir la moyenne des entrées fournies tout en garantissant la confidentialité des valeurs salariales de chaque participant. Ainsi, chaque participant peut comparer son salaire (qu'il est seul à connaître) et la moyenne des salaires du groupe, connue de tous.

Appliqué aux clés cryptographiques, le concept de Multi-party Computation permet, via la décomposition de celles-ci en plusieurs parts, de les partager en garantissant la confidentialité de leur valeur initiale.

2.5 Hardware Security Modules

Appelés aussi *HSM*, les Hardware Security Modules sont des équipements électroniques utilisés dans la cryptographie. Leur utilité est multiple : de la génération à la protection tout en passant par la sauvegarde de clés cryptographiques. Tant au niveau logiciel qu'au niveau matériel, les moyens de protection intégrés sont un élément clé. Ces moyens sont mis en place pour protéger l'appareil contre des tentatives d'intrusion ou d'altération physique ([CAMPANA et BÉDRUNE, 2019](#)).

Les *HSM* offrent des solutions pour la génération et la manipulation des clés de chiffrement. Toutes ces opérations sont internes à ces appareils, le confinement est total. De nombreuses entités utilisent ces outils dans le but de garantir une sécurité maximale à leurs services. Pour exemple, les autorités de certifications et le milieu bancaire font usage des *HSM* pour leur gestion de clés ([CAMPANA et BÉDRUNE, 2019](#)).

Il existe un critère universel pour définir le niveau de sécurité d'un appareil. Celui-ci se décompose en plusieurs sous-critères consignés dans le standard *FIPS-140-3* - Sécurité pour modules cryptographiques. Ce standard offre ainsi des critères d'évaluation des *HSM*, qu'ils soient matériels, logiciels ou une combinaison des deux ([CREN, 2001](#)).

Bien que les *HSM* soient considérés comme un moyen efficace de gérer, générer et stocker des clés cryptographiques, les coûts engendrés par leur mise en place restreint leur utilisation par les *PME*. Partant de ce constat, l'Armory Mk II s'érige en alternative plus abordable.

3 | État de l'art

Dans ce chapitre, sont tout d'abord présentées les solutions existantes pour le développement sur un système embarqué de type clé *USB Armory Mk II*. Seront ensuite résumées à l'aide d'un tableau comparatif les solutions considérées. Enfin, nous justifierons le choix de la méthode TamaGo pour la réalisation de ce travail.

3.1 Introduction

La section suivante consitute un passage en revue des différents langages et *framework* permettant l'exécution d'une solution sur la clé *USB Armory Mk II*. Avant de se lancer dans cette revue, il convient de se remémorer la spécificité de la clé *USB Armory Mk II* : à savoir l'exécution de programmes sans système d'exploitation. En conséquence, il convient de définir les avantages et inconvénients de chaque solution citée ci-dessous en regard de cette spécificité. Nous synthétiserons ensuite ces informations via un tableau récapitulatif et justifierons finalement la sélection du *framework* TamaGo.

3.2 Langage bas niveau

Les langages de programmation bas niveau sont au plus proche de la machine. Ils s'opposent aux langages haut niveau qui ne tiennent pas compte des caractéristiques de l'ordinateur pour exécuter leur programme. Précisons que le langage machine (suite de bits) et le langage assembleur sont les modèles des langages bas niveau. Ceux-ci proposent de manière explicite de traiter les registres, les instructions machines et tout ce qui touche aux adresses mémoires.

Ce type de langage est surtout utilisé dans les systèmes embarqués, la création des systèmes d'exploitation ou de pilotes ainsi que dans les entreprises industrielles. Leur utilisation est parfois nécessaire et inévitable pour des raisons de limitations du matériel (taille de la mémoire, vitesse du processeur).

Ce faisant, nous présentons ici deux langages orientés pour le développement *bare metal*, l'assembleur et le C.

Langage assembleur

Le langage assembleur est la dernière forme lisible par un humain. Celui-ci est ensuite transformé en langage machine binaire (suite de 0 et de 1). Ce langage bas niveau sert à l'écriture d'instructions pour un processeur spécifique. Pour l'exécuter, il est nécessaire d'utiliser un assembleur, c'est-à-dire un programme de traduction du langage d'assemblage en langage machine. Il existe un langage assembleur par type de processeur.

Le langage assembleur recoupe les caractéristiques suivantes. Elles sont énoncées ici pour permettre une meilleure assimilation des avantages et inconvénients de ce langage ([LONCHAMPS, 2017](#)).

- Le jeu d'instruction est le même que le langage machine. Ces instructions sont utiles pour des opérations basiques (addition, ET logique) ou plus complexes (division).
- Les opérations sont exprimées par abréviations comme **add**, **mul** ou **mov**.
- Pour gérer les données dont le processeur a besoin, le langage utilise des *registres*.
- Il est important de spécifier la base des valeurs constantes. Celles-ci sont exprimées en décimal, hexadécimal et en octal.
- Les branchements sont utilisés pour modifier le fil d'exécution d'un programme en sautant d'une partie d'un code à une autre. Lorsque nous utilisons une instruction de branchement comme **jmp**, nous spécifions l'étiquette vers laquelle nous souhaitons effectuer le saut.

Toutes ces particularités rendent l'utilisation du langage assembleur plus complexe pour programmer sur du *bare metal*. Il demande une compréhension précise du matériel et une gestion minutieuse des *registres* et des branchements pour assurer le bon fonctionnement d'un programme.

Avantages

Après avoir détaillé succinctement quelques caractéristiques du langage assembleur, nous allons maintenant nous pencher sur ses avantages ([KORNELIS, s. d.](#)).

- Utilisation complète : l'utilisation éventuelle de la capacité de totale l'appareil.
- Personnalisation accès mémoire : cela signifie que le langage accède directement aux *registres* et par conséquent à la mémoire physique offrant ainsi un contrôle granulaire de la mémoire.
- Performance : l'opportunité d'améliorer les performances d'un programme par rapport à d'autres langages. Le contrôle du matériel, la réduction de la taille du programme et l'utilisation efficace du matériel en fonction de sa plateforme offrent la possibilité de parfaire les performances.
- Accès aux flags du processeur : le langage assembleur offre l'opportunité d'accéder aux flags du *CPU*. La valeur du flag (0 ou 1) peut varier en fonction des résultats des opérations arithmétiques et logiques ([DE MEY, s. d.](#)). Par exemple, les flags peuvent indiquer si une opération a généré un overflow (dépassement de capacité) ou si un résultat est nul ([ROSSIER, 2017](#)).

Les avantages du langage permettent une personnalisation au plus proche du matériel et un gain en performance dans certains cas. La personnalisation du programme peut être vue comme un avantage ou un inconvénient, le programme fonctionne uniquement pour un type de processeur. Ainsi, un changement de processeur implique une modification du programme.

Inconvénients

La personnalisation et la proximité du langage vis-à-vis du matériel sont des avantages conséquents. En revanche, il existe des inconvénients que nous souhaitons mentionner ([X_EDITOR, 2023](#)).

- Portabilité : le principal inconvénient du langage assembleur est son manque de portabilité. En d'autres termes, ce langage est dépendant du *CPU* sur lequel il est exécuté.
- Courbe apprentissage : l'apprentissage du langage est plus difficile que des langages haut niveau. Ainsi, il faut du temps supplémentaire pour assimiler et maîtriser ce langage.
- Lisibilité : l'assembleur est beaucoup plus difficile à lire et à écrire que d'autres langages.

De plus, le temps de programmation pour exécuter une même portion de programme entre un langage bas niveau et haut niveau est significative. Les langages moins proches du langage machine sont développés pour faciliter la tâche du programmeur en proposant des instructions déjà faites. Par exemple, l'affichage dans une ligne de commande du fameux message "Hello World!" demande moins d'effort de la part du programmeur en langage C qu'en langage assembleur. Les exemples ci-dessous, tirés de sources internet^{1 2}, sont assez explicites.

```

1  #Langage C
2
3  #include <stdio.h>
4  int main() {
5      // printf() affiche la chaîne de caractères entre guillemets
6      printf("Hello World!");
7      return 0;
8  }
```

```

1  ; Langage assembleur
2  section .data
3      msg db Hello World!, 10
4      msgSize equ $ - msg
5
6  section .text
7      global _start
8
9  _start:
10     mov eax, 4          ; Déplacer la valeur 4 dans le registre EAX (fonction 4
11     → pour écrire sur la sortie standard)
12     mov ebx, 1          ; Déplacer la valeur 1 dans le registre EBX (descripteur
13     → de fichier pour la sortie standard)
14     mov ecx, msg        ; Déplacer l'adresse du message dans le registre ECX
15     mov edx, msgSize    ; Déplacer la taille du message dans le registre EDX
16     int 0x80            ; Appel du système d'interruption 0x80 pour exécuter la
17     → fonction 4 (écrire sur la sortie standard)
18
19     mov eax, 1          ; Déplacer la valeur 1 dans le registre EAX (fonction 1
20     → pour terminer le programme)
21     xor ebx, ebx        ; Effectuer un XOR entre la valeur de EBX et elle-même
22     → (mettre EBX à zéro, code de sortie 0)
```

1. <https://www.geeksforgeeks.org/c-hello-world-program/>
2. <https://ruuand.github.io/Assembleur/>

```
int 0x80 ; Appel du système d'interruption 0x80 pour exécuter la  
↪ fonction 1 (terminer le programme)
```

Les deux exemples de programme d'affichage du message "Hello World !" démontrent les tâches supplémentaires dont un programmeur doit s'acquitter. Cela n'est pas irréalisable mais le temps de développement est plus long. De nos jours, il est parfois judicieux d'utiliser des langages haut niveau pour faciliter le développement d'application comme le *Inline assembler* qui permet d'intégrer des instructions en assembleur dans un programme en C et C++ (TYLERMSFT, 2021).

Le manque de portabilité, la difficulté d'apprentissage et de lecture du langage sont des inconvénients à prendre en considération lorsque nous souhaitons développer une application.

Cas d'utilisation

Le langage assembleur s'utilise dans des cas spécifiques ainsi que pour le développement sur des systèmes embarqués. Souvent, ce langage est utilisé dans des configuration où le contrôle du matériel est nécessaire (KORNELIS, s. d.).

Nous le retrouvons dans le développement de systèmes d'exploitation et de pilotes. L'accès direct aux composants du matériel en font un langage adéquat. De plus, son utilisation pour les systèmes embarqués où les ressources sont limitées permet d'utiliser pleinement le matériel à disposition. Dans le reverse engineering, l'assembleur est utile pour comprendre le fonctionnement interne d'un programme. Nous utilisons aussi ce langage dans le développement de microcontrôleur où les ressources sont limitées.

Nous soulignons aussi que le langage assembleur permet le calcul arithmétique multi-précision. L'arithmétique multi-précision regroupe les opérations de base ; addition, soustraction, multiplication et division. Ce principe permet de manipuler des nombres de tailles variables sans que celui-ci ne soit limité par les capacités du *CPU*, mais plutôt par la taille de la mémoire du système (BAERISWYL, 2018). L'arithmétique multi-précision cherche à dépasser les limites physiques imposées par les *CPU* qui ne peuvent manipuler que des nombres de taille limitée (64 bits) (ZANOTTI, 2023). D'ailleurs, il existe une bibliothèque développée en assembleur pour effectuer ce type de calcul, la *GNU Multiple Precision Arithmetic Library*.

Il est important de noter que l'utilisation du langage assembleur est généralement limitée à des situations spécifiques où le contrôle précis des composants du matériel ou la performance et l'optimisation sont nécessaires.

Langage C

Après avoir discuté du langage assembleur, de ses avantages et inconvénients, nous allons nous pencher sur le langage de programmation C. Celui-ci aussi peut être utilisé pour le développement *bare metal*.

Le langage C est un langage de programmation développé par Dennis Ritchie dans le but de concevoir un système d'exploitation *Unix*. Le C a été rapidement adopté par la communauté de programmeurs pour le développement de logiciels embarqués notamment grâce à sa portabilité et à la facilité d'accéder au composant matériel ([BLOGGER, 2015](#)). Comme le langage assembleur, le C permet de gérer la mémoire et de contrôler le flux d'exécution. Ce langage propose une syntaxe claire et concise. Grâce à sa simplicité, la lecture du programme s'en trouve facilitée tout comme la détection des erreurs.

Avantages

Les avantages du langage C pour le développement *bare metal* sont les suivants : ([KADIONIK, 2018](#)).

- Compilé : le code source, lisible par le développeur, est traduit en code machine binaire.
- Universalité : ce langage est universel. Il n'est pas spécifique à un domaine d'application comme le *SQL*.
- Portabilité : la portabilité du C est un avantage considérable par rapport au langage assembleur. Il peut être exécuté sur n'importe quel système tant que celui-ci ait un *compilateur C*. Un *compilateur* est un programme dont la fonction principale est la traduction d'un langage de programmation en un code binaire lisible par une machine.
- Gestion de la mémoire : la manipulation des pointeurs et des bits est un atout majeur pour le développement *bare metal* dans le but d'optimiser au mieux le programme selon les contraintes (taille de la mémoire, performance).
- Langage typé : le C est un langage typé. Cette caractéristique favorise la sécurité et la cohérence du programme. A la lecture du programme, le *compilateur* identifie directement les types de variable et ses potentielles erreurs (non initialisation des variables, variable allouée incompatible).
- Extensibilité : le langage est extensible. Des bibliothèques sont ajoutées dans le but d'enrichir et de faciliter l'écriture d'un programme en récupérant des fonctions déjà écrites.

L'universalité du langage, la portabilité, la manipulation des composants matériels, l'extensibilité ainsi que la structure typée du langage C sont des avantages importants à prendre en considération lors du choix d'un langage de développement *bare metal*.

Inconvénients

Après avoir mentionnée les différents avantages du langage C, il convient de mettre en exergue ses limitations ([KADIONIK, 2018](#)).

- Lisibilité : comme dans tous les langages, il est possible d'utiliser des expressions plus compactes et plus efficaces. L'utilisation raccourcie d'expressions ne doit pas complexifier la compréhension et la lisibilité du programme.

- Portabilité : l'utilisation de bibliothèques spécifiques à un type de machine peut nuire à la portabilité du langage C. Par exemple, l'utilisation d'une fonction de lecture d'entrée et de sortie comme "scanf" est conforme à la norme *ANSI-C*. Si le programmeur décide d'utiliser une méthode spécifique à la machine, la portabilité n'est plus garantie pour le programme.
- Courbe d'apprentissage : comme pour le langage assembleur, la personnalisation et la gestion de l'utilisation des composants matériels demandent une bonne maîtrise du langage pour éviter de nombreux problèmes (libération de la mémoire, allocation de la mémoire heap, etc).
- Memory leaks : Le memory leak se produit lorsque le développeur alloue de la mémoire à un objet ou une variable et oublie de libérer cet espace. Cet oubli répété est susceptible de provoquer l'arrêt du système ou d'un programme ([GEEKSFORGEEKS, 2023](#)).

Les inconvénients décrits ci-dessus nous permettent d'observer que ce langage offre une abstraction supplémentaire par rapport au langage assembleur. Une bonne maîtrise du C est nécessaire pour garantir la sécurité, la lisibilité et l'exécution d'un programme.

Cas d'utilisation

De nombreux systèmes d'exploitation et de logiciels sont développés à l'aide du langage de programmation C. Minimaliste et proche de la machine, C se retrouve souvent dans les systèmes embarqués. Nous le trouvons aussi dans la conception des logiciels ou de jeux. Dans cette dernière catégorie, de nouveaux langages haut niveau plus faciles à prendre en main (C# et Javascript) sont apparus, rendant parfois désuète l'utilisation du langage C.

Comme mentionné dans le paragraphe ci-dessus, le langage C est utilisé dans le développement de systèmes d'exploitation. Au départ, il était exclusivement associé au système *Unix* ([RITCHIE, 2021](#)).

Finalement, nous retrouvons ce langage dans l'écriture de bibliothèques de fonction pour aider les programmeurs lors de l'écriture de programmes.

3.3 Framework existants

3.3.1 Bare metal Rust

Langage Rust

Rust est un langage statique et fortement typé. Il a été développé par Mozilla en 2010. Au moment de la compilation, tous les types des variables sont décrits avec un langage statique. Un langage fortement typé aide le développeur à être conscient des types de variable qu'il utilise et évite des erreurs ultérieures. Le succès de Rust en comparaison avec des langages comme C et C++ réside dans le fait que Rust est "memory safe". C'est-à-dire que chaque accès à la mémoire est vérifié. Il n'est pas possible de corrompre la mémoire par accident ([DONOVAN, 2017](#)).

Notons que Rust et Go sont des langages de programmation *opensource*. Cela veut dire que les développeurs sont libres de contribuer à l'évolution du langage.

Rust est utilisé pour sa sécurité, sa performance et sa gestion de la mémoire. Par défaut, l'espace mémoire est géré automatiquement sans l'aide d'un *garbage collector*. Rust se base le concept de propriété qui autorise une variable à avoir un seul propriétaire à la fois ([SASIDHARAN, 2019](#)). Cela est possible car Rust a comme règle principale que si un objet est hors de sa portée, il sera supprimé. Par suppression, nous entendons la libération de toutes les ressources associées à la variable ([MEAGHANLEWIS, s. d.](#)). La gestion de la mémoire est intrinsèque aux propriétés du langage. Celui-ci offre également une gestion manuelle des allocations mémoires par le biais de clonage, copie et référencement de l'adresse des variables. Rust et Go permettent de se lier avec d'autres langages de programmation tels que le C ([ROUNAK SHARMA, 2023](#)).

Avantages

Les avantages liés au langage Rust sont multiples. La liste ci-dessous décrit et explique les avantages identifiés :

- Sécurité à l'exécution des *threads* : le langage Rust est conçu pour garantir la sécurité et la prévisibilité de l'accès à la mémoire par un programme ([AKANOA, 2023](#)). Lors de la compilation, Rust vérifie que chaque emprunt ou référence à la mémoire est correct. Pour ainsi dire, Rust empêche les accès illégaux, la libération incorrecte de mémoire ainsi que la concurrence pour l'un accès à la mémoire.
- Économie de mémoire : disposer de la mémoire en Rust repose sur les compétences du développeur ainsi que sur sa connaissance des *threads*. La libération de la mémoire est automatique ([AKANOA, 2023](#)).
- Documentation complète : de nombreux sites, contenus multimédias et des Git offrent une documentation fournie pour tout type de développement.
- Message d'erreur du compilateur cohérent : à la compilation si des erreurs sont trouvées au sein du programme, le compilateur affiche un message d'erreur compréhensible.

- Gestionnaire de paquet intégré : Rust intègre un gestionnaire de paquets nommé *Cargo* qui gère les dépendances des *packages*, compile ceux-ci et les tient à jour.
- Absence du *garbage collector*

Tous ces avantages démontrent la puissance du langage Rust. Celui-ci peut s'exécuter sur des appareils de type *bare metal*.

Inconvénients

Bien que les avantages du langage Rust soient attrayants, il revêt quelques inconvénients.

Premièrement, comparé à d'autres langages, Rust a un temps de compilation relativement plus lent ([WICKRAMASINGHE, 2023](#)).

Deuxièmement, la taille des fichiers binaires Rust est plus volumineuse car le langage utilise des liaisons statiques pour compiler ses programmes et par conséquent toutes les bibliothèques sont compilés dans l'exécutable. Cela inclut aussi l'environnement d'exécution Rust ([ASTOPHER, 2016](#)).

Ensuite, Rust est un langage développé par Mozilla. Par conséquent, il pourrait être influencé par la vision et l'orientation de son créateur ; contrairement aux langages *opensource* qui sont soumis à la contribution d'un grand nombre d'acteurs. Cependant, il existe une fondation Rust dont le but est de développer et promouvoir le langage Rust ainsi que sa communauté ([RUST FOUNDATION, 2023](#)).

Pour finir, la courbe d'apprentissage du langage est plus longue que le Go. Une connaissance des langages C et C++ est nécessaire à sa bonne utilisation.

Cas d'utilisation

Un projet d'implémentation du *framework* Rust au sein du dispositif Armory Mk II est en cours de développement³. Ce projet actuel prévoit une application de démonstration comme pour TamaGo. Il est à noter que le développement avec le *framework* TamaGo est plus avancé que celui avec Rust.

3.3.2 TamaGo

Le framework

La première présentation du *framework* TamaGo a été faite en décembre 2019. TamaGo se compose de trois éléments principaux ([BARISANI, 2014](#)) :

3. <https://github.com/iqlusioninc/usbarmory.rs/tree/develop/firmware/usbarmory>

- Une enveloppe du langage Go qui permet le support du *GOOS* dans le but d'exécuter une application sur un appareil de type *bare metal*. *GOOS* représente le système d'exploitation sur lequel le langage Go peut être compilé. Cette enveloppe autorise l'exécution sur un système embarqué de type *ARM* et *RISC-V*.
- Un lot de *packages* Go dédié au support de fonctionnalité sur l'Armory Mk II.
- Un lot de *packages* Go pour le support des pilotes externes de l'appareil.

Ces trois éléments fournissent une base solide pour le développement *bare metal*. La motivation du développement de TamaGo réside dans la volonté de se détacher du langage de programmation C, de la dépendance d'un système d'exploitation pour s'exécuter mais aussi dans la réduction de sa surface d'attaque.

L'objectif est de limiter au maximum la complexité pour exécuter un programme en utilisant un langage de haut niveau tout en permettant une compilation et exécution directes sur l'appareil.

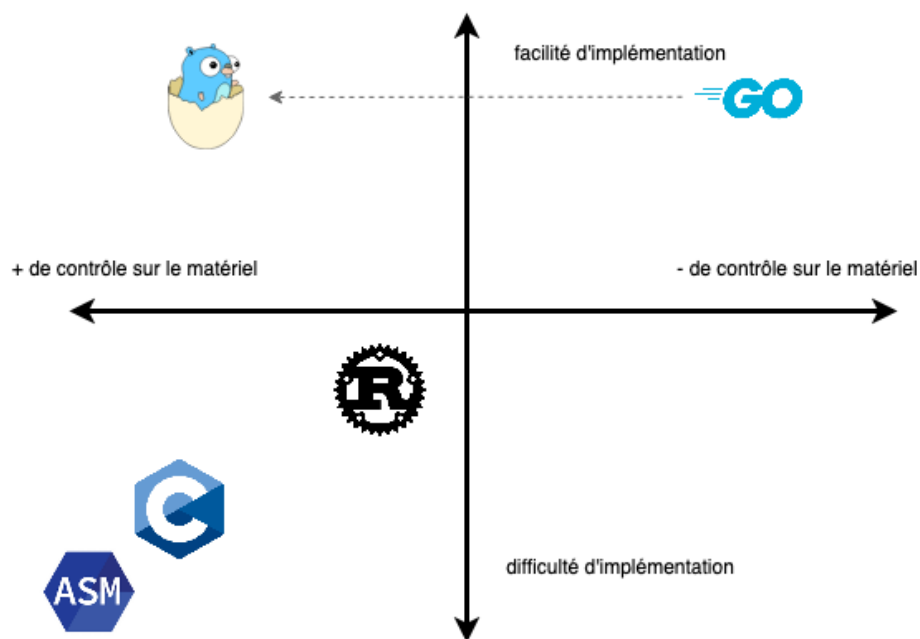


FIGURE 3.1 – Comparaison des langages basés sur le contrôle et la simplicité d'utilisation

Source : adaptée à partir

<https://github.com/abarisani/abarisani.github.io/blob/master/research/tamago/TamaGo.pdf>

L'image ci-dessus représente une comparaison de quatre types de langages (TamaGo, Rust, C et l'assembleur) qui peuvent être utilisés pour le développement sur un appareil *bare metal*. D'après le développeur de TamaGo, celui-ci suggère que l'implémentation de son *framework* est plus facile.

Avantages

Il est question ici de définir les différents avantages offerts par le développement avec un *framework* du type TamaGo. Le premier avantage de cette solution est qu’elle propose une enveloppe de la distribution du langage Go en apportant des modifications minimales. Ces modifications sont nécessaires pour permettre l’exécution *bare metal*. La simplicité a été mise au profit de la complexité dans le but d’obtenir un moyen simple de vérifier ce que l’on développe.

Le second avantage de cette solution réside dans le fait qu’elle permet non seulement d’être exécutée sur différentes architectures mais qu’elle réutilise au maximum la distribution Go. Tout le support et les *packages* liés au pilote sont écrits en Go.

De plus, lors de l’initialisation des composants matériels, le fichier binaire peut être exécuté directement par l’appareil sans l’aide d’un *bootloader* annexe. De ce fait, cela élimine toute dépendance externe à celle proposée par Go (BARISANI, 2023d).

Nous tenons à rappeler l’intérêt de développer avec TamaGo est que l’on peut se passer d’un système d’exploitation pour exécuter notre application Go. Le *framework* met à disposition des configurations pour faciliter l’exécution *bare metal*.

En résumé, toute l’implémentation et l’exécution sont gérées par le *framework*.

Inconvénients

Après avoir énoncé les avantages de la solution TamaGo, il convient de s’attarder sur les complications qui peuvent survenir lors d’une implémentation avec cette même solution.

Premièrement, il existe quelques imports qui ne sont pas possibles dans l’environnement TamaGo. Une liste exhaustive des réussites et des échecs d’importation figurent sur le repository GitHub⁴.

- `time/tzdata`⁵ : Ce *package* contient les données liées au temps (des dates et des fuseaux horaires). Grâce à cet ajout, la conversion des heures, le formatage et la détermination de l’heure actuelle sont également possibles.
- `runtime/cgo`⁶ : Ce *package* permet d’utiliser des fonctionnalités ou bibliothèques en C.
- `os/signal`⁷ : Ce *package* interagit avec le système d’exploitation autorisant l’application Go à gérer des signaux envoyés par l’OS. Un exemple de signal est l’exécution de la commande CTRL+C dans un terminal. Cette commande envoie un message au système de terminer ce qu’il est en train de faire.

4. <https://github.com/usbarmory/tamago/wiki/Import-report>

5. <https://pkg.go.dev/time>

6. <https://pkg.go.dev/runtime/cgo>

7. <https://pkg.go.dev/os/signal>

Deuxièmement, TamaGo ne propose pas de support pour les *threads* et la variable GOMAX-PROCS est limitée à 1. Cela veut dire que la variable limite le nombre de *threads* pouvant être exécutés simultanément pour du code Go (Go, 2023). La configuration de cette variable contrôle combien de fils d'exécution peuvent exécuter simultanément du code. Par exemple, si la variable est initialisée à 4, le programme s'exécute sur 4 *threads* du système d'exploitation en même temps, même si le code contient 500 *Goroutines* (Cox, 2015).

Par la suite, TamaGo n'utilise pas de système d'exploitation, pas de variable d'environnement et pas de gestion d'utilisateur. Ces limites sont liées à la nature intrinsèque du développement d'application *bare metal*. Plutôt que de considérer cet argument comme un désavantage, nous pouvons le voir comme une extension du *framework*.

Enfin, l'importation de bibliothèques liés à *cgo* dont l'appel implique le lien avec une librairie externe n'est pas supportée. Cependant, il est possible de compiler du code C en utilisant *cgo*, mais il doit être indépendant et ne doit pas faire appel à des fonctions externes ou à des appels système. (BARISANI, 2023d).

La liste des inconvénients du *framework* illustre les limites encore actuelles de celui-ci. Plusieurs acteurs ont estimé ces limites négligeables et se sont servi de TamaGo pour différentes utilisations.

Cas d'utilisation

Les cas d'utilisation de TamaGo avec le périphérique Armory Mk II ont déjà été présentés dans la section 2.2.

3.4 Critères de sélection

Les solutions recherchées et présentées ci-dessus doivent respecter différents critères pour être exécuté sur une clé *USB* Armory Mk II.

L'Armory Mk II, conçue par l'entreprise WithSecure, est un appareil de la taille d'une clé *USB* agissant comme un ordinateur.

Pour permettre une comparaison neutre, les critères suivants seront utilisés pour sélectionner la solution la plus adaptée :

- Absence d'*OS* : exécution sans l'aide d'un système d'exploitation.
- Documentation à jour : date de la dernière mise à jour.
- Documentation claire et précise.
- Cas d'utilisation d'implémentation avec une clé *USB* Armory Mk II.
- Simplicité du langage : le langage est facile à utiliser.

3.5 Détermination d'une solution

L'état de l'art de ce travail a permis de mettre en lumière quatre langages pour le développement *bare metal*, le TamaGo qui est *framework* basé sur le langage Go, le Rust développé par Mozilla, l'assembleur qui est un langage bas niveau spécifique et personnalisable pour un type de processeur et le langage C qui offre une abstraction du langage assembleur.

Sur la base des critères de sélection susmentionnés, nous réalisons une matrice de comparaison.

	Exécution sans OS	Doc. à jour	Clarté précision	Cas util. Armory Mk II	Simplicité
Assembleur	oui	23.06.2023	-	aucun	-
C	oui	23.06.2023	+	aucun	-
Rust	oui	17.03.2020	+	+	+
TamaGo	oui	23.06.2023	++	+++	++

D'après notre matrice de comparaison, les langages Assembleur et C permettent l'exécution d'un programme sans l'aide d'un système d'exploitation mais nous n'avons pas trouvé de cas d'utilisation avec notre appareil. Ainsi, ces deux langages sont écartés du choix final.

Le langage Rust propose une exécution sans OS et un cas d'utilisation avec l'Armory Mk II. Malgré cela, l'état actuel du développement du cas d'utilisation et la prise en main de Rust ne permettent pas de sélectionner cette option comme *framework* de développement.

Finalement, TamaGo répond complètement aux critères que nous avons mentionné dans notre matrice de comparaison. Les nombreux cas d'utilisation, la simplicité et la documentation à jour nous prouvent que cette solution mérite d'être testée et approfondie.

4 | Mise en oeuvre

La partie suivante se consacre à la mise en place et la préparation de l'environnement de travail pour le développement d'un programme en Go. La solution sélectionnée utilise TamaGo et nécessite pour cela d'installer préalablement des modules requis à son emploi. De plus, un fichier `readme.md` présenté sur le Git¹ apporte si besoin de plus amples informations.

4.1 Mise en place d'un environnement de développement en Go

4.1.1 Installation de Golang

L'installation rapide du langage est décrite sur le site officiel du langage². La version utilisée pour le développement du programme est la 1.20.5.

4.1.2 Installation de l'environnement de développement

L'environnement de développement utilisé pour l'écriture du programme est Visual Studio Code³. L'extension ci-dessous propose des fonctionnalités tels que l'IntelliSense, les tests, le débogage et bien d'autres encore.

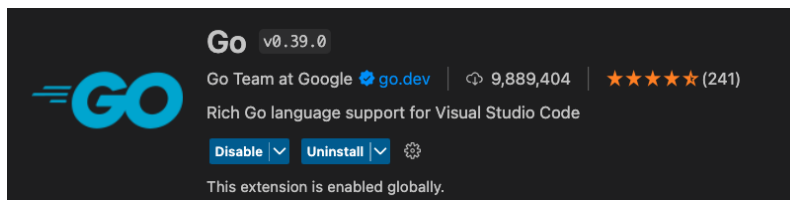


FIGURE 4.1 – *Extension Go pour Visual Studio Code*

4.1.3 Configuration pour la compilation et le déploiement sur macOS

Pour construire un programme dédié au système embarqué, il nous faut installer quelques outils.

La chaîne d'outils⁴ GNU Arm est une boîte à outils pour le développement sur des processeurs de type Arm Cortex-A, Arm-Cortex-M.

```
brew tap ArmMbed/homebrew-formulae
brew install arm-none-eabi-gcc
```

1. <https://gitlab.com/hesso-vs/business-information-technology/65-62-bachelor-thesis/thomas-cheseaux-2023-tamago>

2. <https://go.dev/doc/install>

3. <https://code.visualstudio.com/docs/languages/go>

4. <https://github.com/ARMmbed/homebrew-formulae>

Ensuite, nous installons l'outil U-Boot⁵. U-Boot est un chargeur de démarrage multiplateforme pour les systèmes embarqués. Il est utilisé comme chargeur de démarrage par défaut par plusieurs fournisseurs de cartes (ARM, x86, etc).

```
brew install u-boot-tools
```

4.2 Communication entre l'ordinateur et l'Armory Mk II

Dans ce chapitre, nous allons exposer et décrire les étapes nécessaires à l'implémentation et à l'exécution d'une application de démonstration.

Premièrement, nous expliquons le déroulement de l'installation sur l'Armory Mk II et des prérequis nécessaires. Ensuite, nous enchaînons avec la configuration de la connexion internet depuis une machine macOS et une machine Linux. Puis, nous présentons la solution installée et exécutons quelques fonctionnalités.

Pour chaque étape, nous décrivons les configurations pour une machine macOS et Linux. Malgré leur base *Unix* similaire, leur configuration mérite une distinction particulière.

4.2.1 Installation sur une clé USB

Étant donné que l'installation de la distribution Golang et des packages sont déjà faites, nous passons directement à la présentation de la première étape pour l'exécution de l'application de démonstration.

Pour compiler et créer l'image de l'application, nous devons construire le *compilateur*⁶.

```
# télécharger le dernier compilateur tamago depuis une certaine version, ici  
→ 1.20.5  
git clone -b tamago1.20.5 --single-branch  
→ https://github.com/usbarmory/tamago-go.git  
  
# se placer dans le dossier source et exécuter le fichier all.bash  
cd tamago-go/src && ./all.bash  
  
# se replacer dans le dossier /bin et exporter le chemin dans la variable  
→ d'environnement TAMAGO  
cd ../bin && export TAMAGO=`pwd`/go
```

Le *compilateur* exporté, nous pouvons construire l'image de type *ARM*. Cette image contient la configuration nécessaire pour se lancer automatiquement sur la clé *USB*. Pour créer l'image, il est primordial d'avoir téléchargé le répertoire de l'application¹.

5. <https://formulae.brew.sh/formula/u-boot-tools>

6. <https://github.com/usbarmory/tamago-example>

```
# se placer dans le repository
cd ./tamago-example

# création de l'image de type usbarmory
make imx TARGET=usbarmory
```

L'image de l'application est prête, la prochaine étape consiste en l'écriture sur la carte Micro SD. Pour ce faire, nous devons brancher la carte Micro SD à l'ordinateur et effectuer les commandes suivantes sur macOS. Le disk2 doit être modifié en fonction du nom du disque de la carte Micro SD de l'hôte.

```
# lister les disques présents/connectés à l'ordinateur
diskutil list

# choisir le disque représentant la carte Micro SD et le rendre inaccessible
↳ (unmount) pour l'ordinateur
sudo diskutil unmountDisk /dev/disk2

# copier l'image sur la carte Micro SD
sudo dd if=example.imx of=/dev/disk2 bs=512 seek=2
```

Cette étape nous a permis de télécharger le *compilateur* TamaGo, de l'exporter dans une variable d'environnement, de créer l'image exécutable sur l'Armory Mk II et de la copier sur une carte Micro SD.

4.2.2 Configuration du partage de connexion

Pour pouvoir se connecter à l'Armory Mk II par *SSH*, nous devons configurer l'interface virtuelle Ethernet de la clé *USB*. La marche à suivre présentée sur le site officiel ne fonctionne pas entièrement pour notre cas d'implémentation de l'application démo. La configuration De ce fait, nous proposons une alternative en ligne de commande.

4.2.2.1 macOS

Tout d'abord, nous trouvons l'interface correspondante à l'Armory Mk II et gardons en tête son numéro de série. Le numéro de série est une information précieuse pour trouver l'interface réseau liée à celle-ci. N'oublions pas qu'il faut avoir branché le périphérique à l'ordinateur.

```
# lister tous les appareils USB connectés à l'hôte
VBoxManage list usbhost # possible que cette commande ne fonctionne plus
# ou
system_profiler SPUSBDataType
```

Grâce au numéro de série affiché avec la commande précédente, nous pouvons retrouver l'interface liée à notre périphérique en exécutant la commande `ifconfig`. Le numéro de série est quasiment identique à celui de l'adresse mac de l'interface. Dans notre cas, l'interface `en9` représente notre périphérique avec l'adresse `1a :55 :89 :a2 :69 :42`.

```
UUID:                129b90be-b0f3-4439-a362-cc94a4d0ea2c
VendorId:             0x1209 (1209)
ProductId:            0x2702 (2702)
Revision:             0.1 (0001)
Port:                 0
USB version/speed:    0/High
Manufacturer:         WithSecure Foundry
Product:              CDC Ethernet _ECM_
SerialNumber:         1a:55:89:a2:69:41
Address:              p=0x2702;v=0x1209;s=0x00003890b8d9b867;l=0x14300000
Current State:        Busy
```

FIGURE 4.2 – Résultat de la commande `VBoxManage list usbhost`

```
en9: flags=8863<UP,BROADCAST,SMART,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    options=6467<RXCSUM, TXCSUM, VLAN_MTU, TS04, TS06, CHANNEL_IO, PARTIAL_CSUM, ZEROINVERT_CSUM>
    ether 1a:55:89:a2:69:42
    nd6 options=201<PERFORMNUD,DAD>
    media: autoselect
    status: inactive
```

FIGURE 4.3 – Résultat de la commande `ifconfig`

Ensuite, il est nécessaire de configurer l'interface réseau avec une adresse *IP* à chaque fois que nous branchons la clé. Le nom de l'interface réseau `en9` doit être modifié en conséquence selon les informations trouvées sur la machine hôte.

```
# lister les interfaces réseaux de l'hôte
ifconfig -a

# configurer et activer l'interface réseau avec une adresse IP
sudo ifconfig en9 10.0.0.2/24 up

# vérification de la configuration de l'interface
ifconfig en9
```

Notons qu'il est possible que l'interface affiche un statut inactif. Ce problème n'a malheureusement pas pu être résolu. Ci-dessous, nous exposons le résultat après l'activation de l'interface.

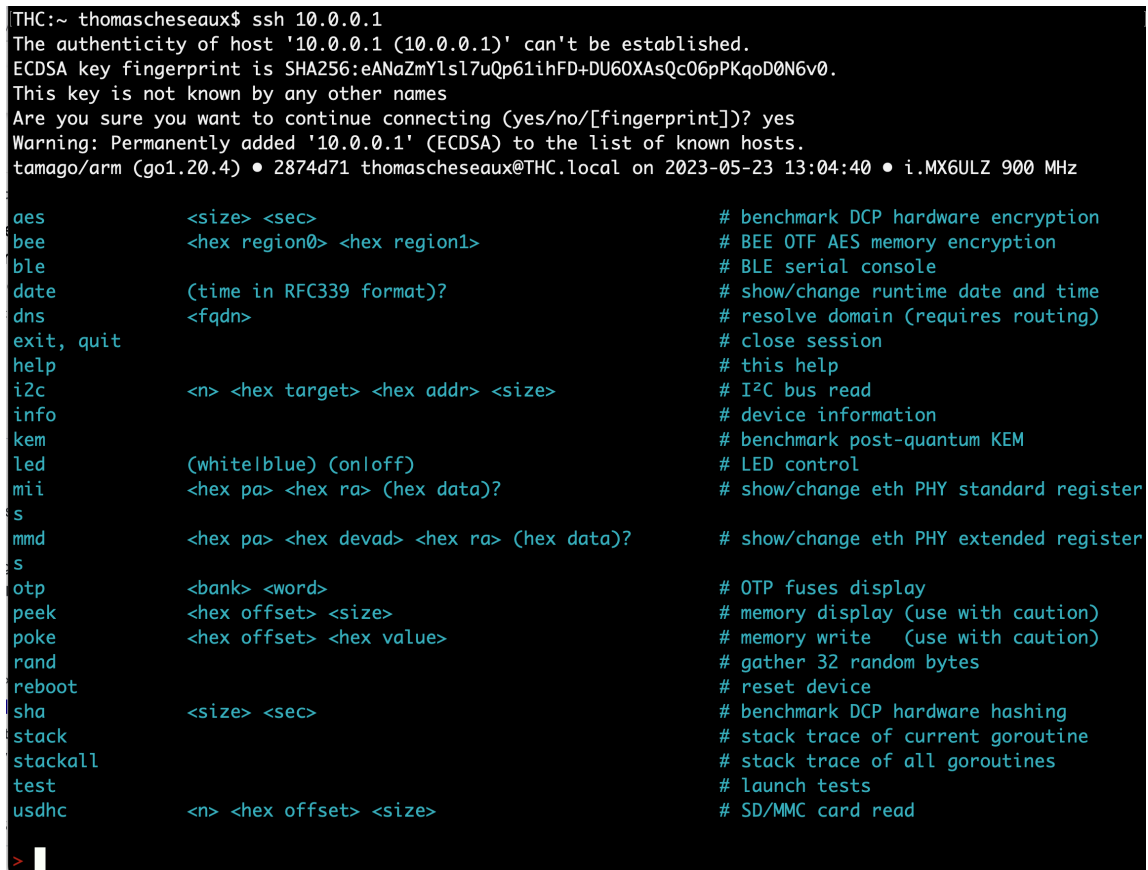
```
THC:~ thomascheseaux$ sudo ifconfig en9 10.0.0.2/24 up
Password:
THC:~ thomascheseaux$ ifconfig en9
en9: flags=8963<UP,BROADCAST,SMART,RUNNING,PROMISC,SIMPLEX,MULTICAST> mtu 1500
    options=6467<RXCSUM, TXCSUM, VLAN_MTU, TS04, TS06, CHANNEL_IO, PARTIAL_CSUM, ZEROINVERT_CSUM>
    ether 1a:55:89:a2:69:42
    inet 10.0.0.2 netmask 0xffffffff broadcast 10.0.0.255
    nd6 options=201<PERFORMNUD,DAD>
    media: autoselect
    status: inactive
```

FIGURE 4.4 – Affichage du statut inactif de l'interface après activation

Pour terminer, l'accès à l'appareil s'effectue par le biais d'une connexion *SSH*.

```
ssh 10.0.0.1
```

Le résultat final se présente comme sur l'image ci-dessous.



```
THC:~ thomascheseaux$ ssh 10.0.0.1
The authenticity of host '10.0.0.1 (10.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:eANaZmYls17uQp61ihFD+DU60XAsQc06pPKqoD0N6v0.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.0.0.1' (ECDSA) to the list of known hosts.
tamago/arm (go1.20.4) • 2874d71 thomascheseaux@THC.local on 2023-05-23 13:04:40 • i.MX6ULZ 900 MHz

aes          <size> <sec>                # benchmark DCP hardware encryption
bee          <hex region0> <hex region1> # BEE OTF AES memory encryption
ble          # BLE serial console
date         (time in RFC339 format)?    # show/change runtime date and time
dns          <fqdn>                      # resolve domain (requires routing)
exit, quit   # close session
help        # this help
i2c         <n> <hex target> <hex addr> <size> # I2C bus read
info        # device information
kem         # benchmark post-quantum KEM
led         (whitelblue) (onloff)        # LED control
mii         <hex pa> <hex ra> (hex data)?  # show/change eth PHY standard register
s
mmd         <hex pa> <hex devad> <hex ra> (hex data)? # show/change eth PHY extended register
s
otp         <bank> <word>                 # OTP fuses display
peek        <hex offset> <size>          # memory display (use with caution)
poke        <hex offset> <hex value>      # memory write (use with caution)
rand        # gather 32 random bytes
reboot      # reset device
sha         <size> <sec>                 # benchmark DCP hardware hashing
stack       # stack trace of current goroutine
stackall    # stack trace of all goroutines
test        # launch tests
usdhc       <n> <hex offset> <size>      # SD/MMC card read

> |
```

FIGURE 4.5 – Connexion SSH à l'Armory Mk II sur macOS

4.2.2.2 Linux

La configuration du partage de connexion pour un système Linux a été effectué avec VirtualBox. Les prérequis pour effectuer cette configuration⁷ sont l'installation de VirtualBox ainsi qu'une machine virtuelle dotée d'un système d'exploitation Ubuntu.

Notons qu'il faut impérativement démarrer Virtual Box en mode administrateur. Il suffit ensuite de se placer dans le dossier où se situe notre application et de lancer la commande `sudo virtualbox`. En parallèle, nous devons insérer manuellement le port *USB* à la machine virtuelle. Pour ce faire, il convient d'utiliser les informations fournies avec les commandes de la sous-section précédente 4.2.2.1.

7. <https://ubuntu.com/tutorials/how-to-run-ubuntu-desktop-on-a-virtual-machine-using-virtualbox#2-create-a-new-virtual-machine>

La configuration initiale du port s'effectue dans le menu configuration du gestionnaire de machines sur VirtualBox.

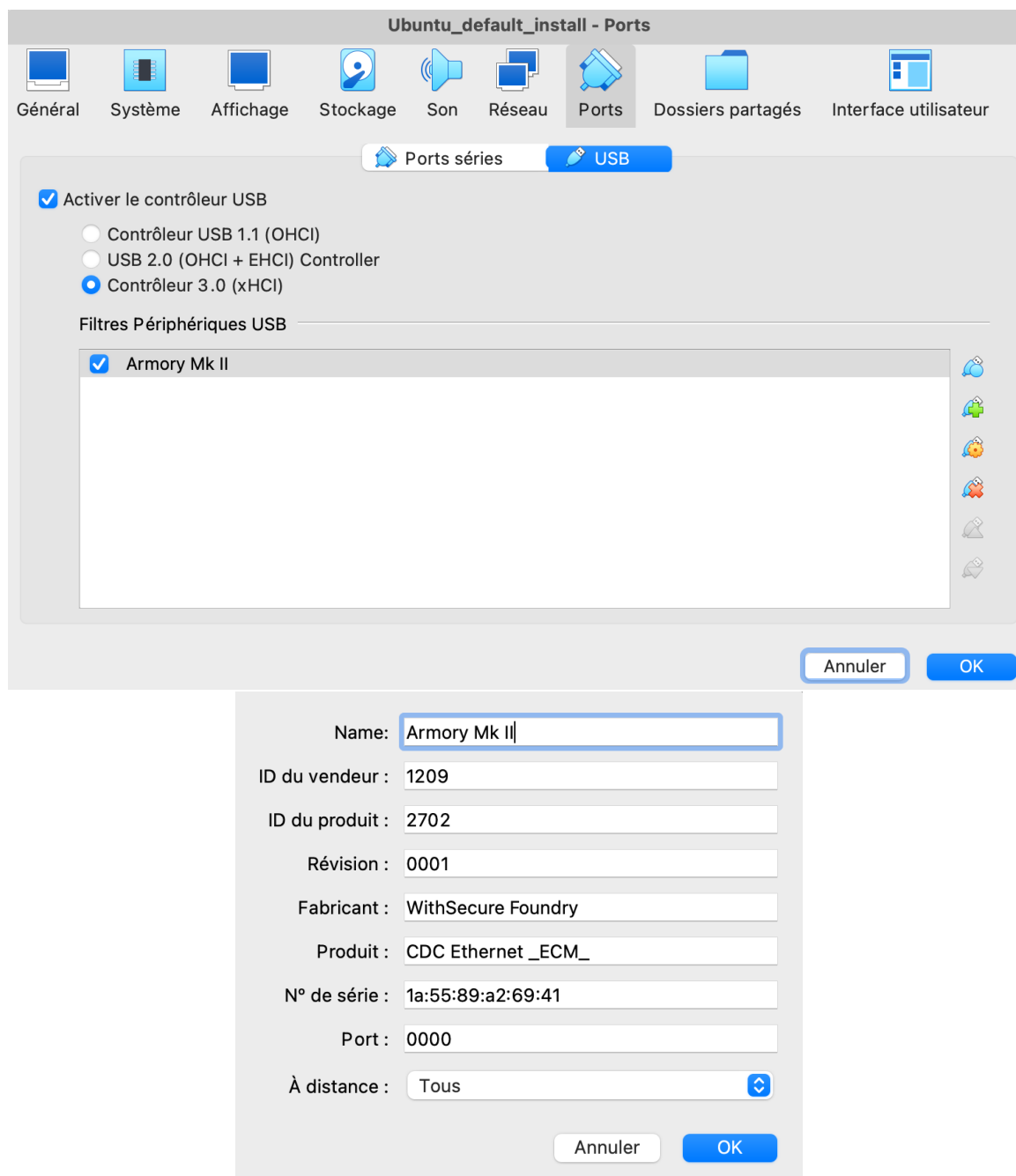


FIGURE 4.6 – Configuration du port de l'Armory Mk II sur VirtualBox

Comme énoncé sur le site officiel⁸, la suite de ces commandes permet l'activation de l'interface Ethernet. Les noms des interfaces, `usb0` pour l'Armory Mk II et `wlan0` pour l'interface réseau, sont adaptés en fonction de la machine hôte de l'utilisateur.

8. <https://github.com/usbarmory/usbarmory/wiki/Host-communication>


```
# afficher les interfaces de la machine
ifconfig -a

# mettre en place l'interface virtuelle Ethernet USB de l'Armory Mk II
sudo /sbin/ip link set usb0 up

# définir l'adresse IP de l'hôte
sudo /sbin/ip addr add 10.0.0.2/24 dev usb0

# activer le masquage pour les connexions sortantes vers l'interface wlan0
/sbin/iptables -t nat -A POSTROUTING -s 10.0.0.1/32 -o wlan0 -j MASQUERADE

# activer le transfert IP
echo 1 > /proc/sys/net/ipv4/ip_forward
```

L'image ci-dessous illustre le résultat final obtenu après la configuration de VirtualBox, de la machine virtuelle et de la connexion SSH.

```
thomas@thomas-ubuntu:~$ ssh 10.0.0.1
The authenticity of host '10.0.0.1 (10.0.0.1)' can't be established.
ECDSA key fingerprint is SHA256:1eVUKUpp2p3MsFhN+U0492+ej4v8fXuc/qWIYZZBiq4.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '10.0.0.1' (ECDSA) to the list of known hosts.
tamago/arm (go1.20.4) • 2874d71 thomascheseaux@THC.local on 2023-05-23 13:04:40 • i.MX6ULZ 900 MHz

aes                <size> <sec>                # benchmark DCP hardware encryption
bee                <hex region0> <hex region1>    # BEE OTF AES memory encryption
ble                # BLE serial console
date              (time in RFC339 format)?        # show/change runtime date and time
dns                <fqdn>                        # resolve domain (requires routing)
exit, quit        # close session
help              # this help
i2c               <n> <hex target> <hex addr> <size> # I²C bus read
info              # device information
kem               # benchmark post-quantum KEM
led               (white|blue) (on|off)           # LED control
mil               <hex pa> <hex ra> (hex data)?    # show/change eth PHY standard regis
ters
mmd               <hex pa> <hex devad> <hex ra> (hex data)? # show/change eth PHY extended regis
ters
otp               <bank> <word>                   # OTP fuses display
peek              <hex offset> <size>             # memory display (use with caution)
poke              <hex offset> <hex value>         # memory write (use with caution)
rand              # gather 32 random bytes
reboot            # reset device
sha               <size> <sec>                   # benchmark DCP hardware hashing
stack             # stack trace of current goroutine
stackall          # stack trace of all goroutines
test              # launch tests
usdhc             <n> <hex offset> <size>          # SD/MMC card read

> dns www.sion.ch
;; opcode: QUERY, status: NOERROR, id: 47366
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.sion.ch.      IN      ANY

;; ANSWER SECTION:
www.sion.ch.      19064   IN      CNAME   webext02.i-web.ch.
```

FIGURE 4.7 – Connexion SSH à l'Armory Mk II sur une machine virtuelle Ubuntu

4.2.3 Configuration du masquerading pour les connexions sortantes

Cette section développe la configuration du fichier de table de routage interne d'une machine cliente dans le but de rediriger et masquer l'adresse de la clé *USB* sur le routeur.

4.2.3.1 macOS

Pour permettre à l'appareil d'utiliser la connexion internet via un routeur, nous devons exécuter les quelques commandes ci-dessous⁹.

```
# Activer le transfert IP
sudo sysctl -w net.inet.ip.forwarding=1

# Activer le pare-feu PF
sudo pfctl -e

# Option 1 : ajouter une règle NAT après la mise en service de en9 (l'USB
→ armory est déjà branché et démarré)
echo "nat on en0 from en9:network to any -> (en0)" | sudo pfctl -f -

# Option 2 : ajouter une règle NAT avant de brancher l'USB armory, en
→ spécifiant son réseau
echo "nat on en0 from 10.0.0.0/8 to any -> (en0)" | sudo pfctl -f -
```

Avant de continuer, nous devons nous assurer que le partage internet est activé pour autoriser la redirection de l'adresse de l'appareil.

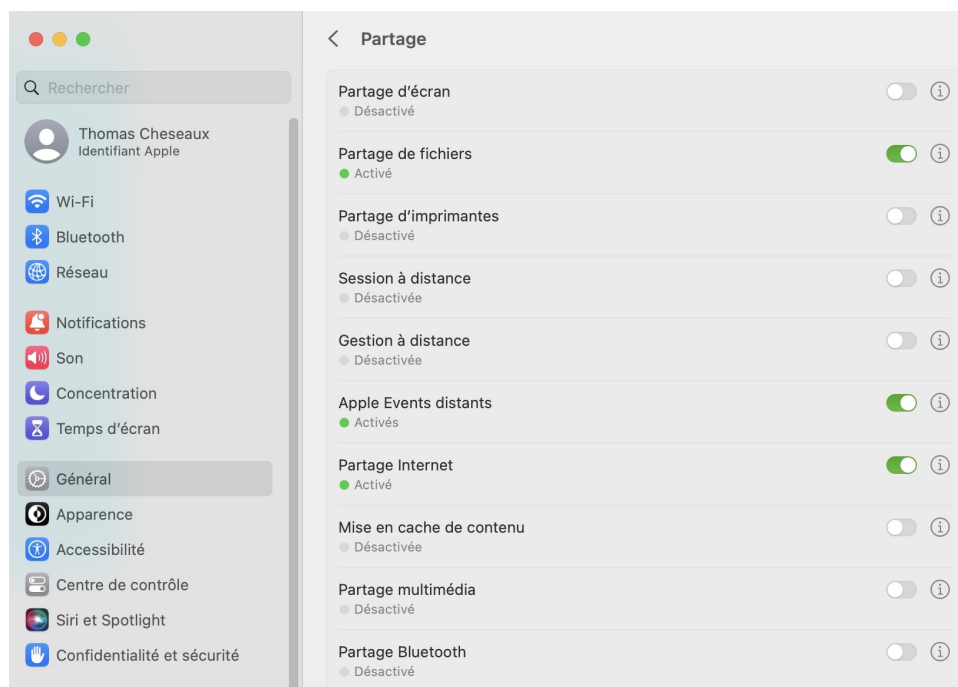


FIGURE 4.8 – *Partage Internet activé*

9. <https://github.com/usbarmory/usbarmory/wiki/Host-communication>

Dans ce cas-ci, nous choisissons de partager notre connexion par l'intermédiaire du *Wi-Fi* pour l'appareil CDC Ethernet.

● Partage Internet : activé

Le partage Internet permet à d'autres ordinateurs de partager votre connexion Internet. Les ordinateurs connectés au secteur ne seront pas en veille lorsque le partage Internet est activé.

Partager votre connexion depuis : Wi-Fi

Aux ordinateurs via :

Activé	Ports
<input type="checkbox"/>	USB iPhone
<input type="checkbox"/>	TomTom
<input type="checkbox"/>	USB 10/100/1000 LAN
<input type="checkbox"/>	Pont Thunderbolt
<input checked="" type="checkbox"/>	CDC Ethernet (ECM)

FIGURE 4.9 – Configuration du partage de connexion sur macOS

4.2.3.2 Linux

De la même façon que pour la configuration effectuée sur macOS, nous devons mettre en place la transmission et la redirection des adresses *IP*. Ces commandes sont fournies par le site officiel ¹⁰.

```
# activer le masquage pour les connexions sortantes vers l'interface sans  
→ fil  
sudo /sbin/iptables -t nat -A POSTROUTING -s 10.0.0.1/32 -o wlan0 -j  
→ MASQUERADE  
  
# change d'utilisateur et passer en mode root  
sudo su  
  
# activer la transmission des adresse IP  
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Pour résumer notre configuration, ces actions sont nécessaires pour permettre à l'appareil d'utiliser la connexion internet du routeur en se faisant passer pour un appareil du réseau privé.

10. <https://github.com/usbarmory/usbarmory/wiki/Host-communication>

4.2.4 Exécution des fonctionnalités de l'application

Ici, nous exécutons quelques fonctionnalités (recherche par nom de domaine, *True Random Number Generator*, affichage des informations de l'appareil et changement de la date) présentes sur l'appareil pour démontrer que l'application de démonstration fonctionne.

4.2.4.1 macOS

```
> dns www.sion.ch
;; opcode: QUERY, status: NOERROR, id: 17702
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.sion.ch.      IN      ANY

;; ANSWER SECTION:
www.sion.ch.      21600   IN      CNAME   webext02.i-web.ch.

> rand
b7c4134b7c75b43b64fbd5b80145deeea2cabbefbd1f197cfe347e3336a581
> date
1970-01-01T00:04:33Z
> date 2023-05-31T16:00:01Z
2023-05-31T16:00:01Z
> info
Runtime .....: go1.20.4 tamago/arm
RAM .....: 0x80000000-0x8ff00000 (255 MiB)
Board .....: UA-MKII-y
SoC .....: i.MX6ULZ 900 MHz
SDP .....: false
Secure boot ..: false
Boot ROM hash : 1727a0f46dbde555b583e9a138ae359389974b7be4369ffd4a252a8730f7e59b
Unique ID ....: DC986D5DD7294238
Temperature ...: 80.517242
```

FIGURE 4.10 – Affichage des résultats des méthodes de l'application de démo sur macOS

4.2.4.2 Linux

```
> dns www.sion.ch
;; opcode: QUERY, status: NOERROR, id: 33093
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;www.sion.ch.      IN      ANY

;; ANSWER SECTION:
www.sion.ch.      21406   IN      CNAME   webext02.i-web.ch.

> rand
39ec22dc8522e879caa39d11c718a2023a065968f53b86d9844249dac6f4e4ae
> date
1970-01-01T00:07:11Z
> date 2023-05-31T16:00:01Z
2023-05-31T16:00:01Z
> info
Runtime .....: go1.20.4 tamago/arm
RAM .....: 0x80000000-0x8ff00000 (255 MiB)
Board .....: UA-MKII-y
SoC .....: i.MX6ULZ 900 MHz
SDP .....: false
Secure boot ..: false
Boot ROM hash : 1727a0f46dbde555b583e9a138ae359389974b7be4369ffd4a252a8730f7e59b
Unique ID ....: DC986D5DD7294238
Temperature ...: 76.896553
```

FIGURE 4.11 – Affichage des résultats des méthodes de l'application de démo sur une machine virtuelle Ubuntu

4.2.5 Lancement des serveurs HTTP de l'application

Après avoir réussi à se connecter via *SSH*, il est possible d'écrire sur un navigateur web le lien suivant `10.0.0.1:80`. La figure ci-dessous affiche le résultat obtenu lorsque l'on se connecte au serveur.

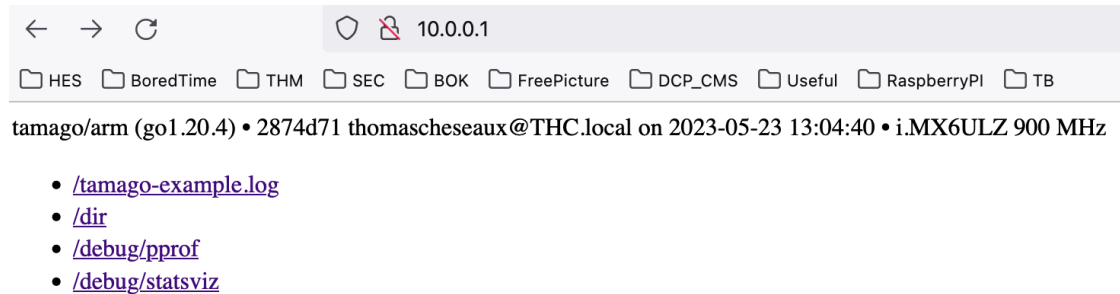
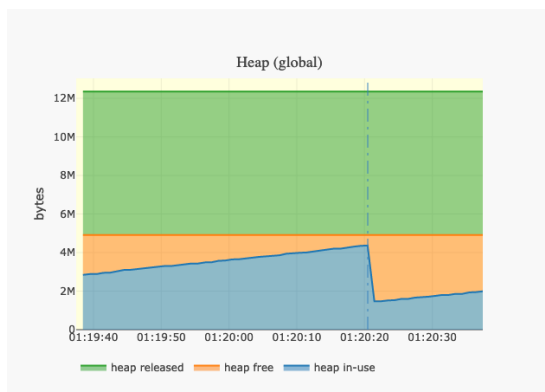
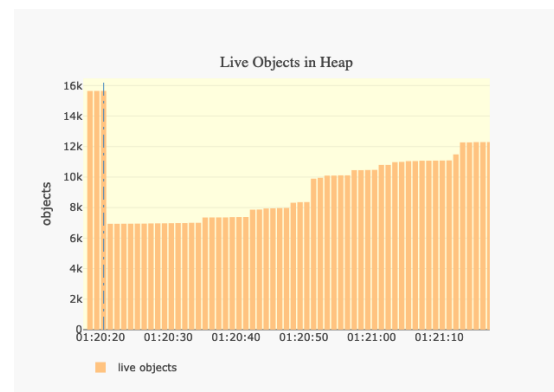


FIGURE 4.12 – Menu d'affichage du serveur HTTP de l'Armory Mk II

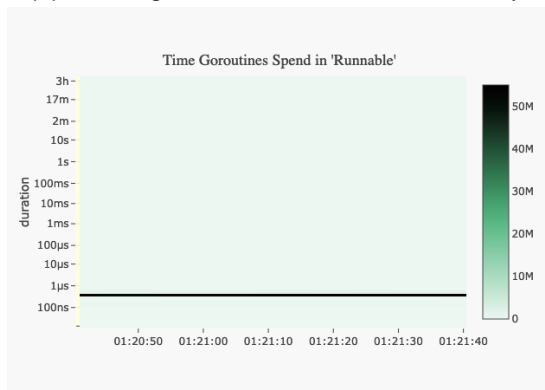
Toutes les fonctionnalités ne sont pas affichées. Nous avons choisi de présenter ici le lien `/debug/statsviz/`. Ce lien offre une vue graphique des performances et de l'état de l'Armory Mk II en temps réel. De cette façon, nous pouvons visualiser en temps réel les métriques d'exécution d'un programme Go (RAINONE, 2023).



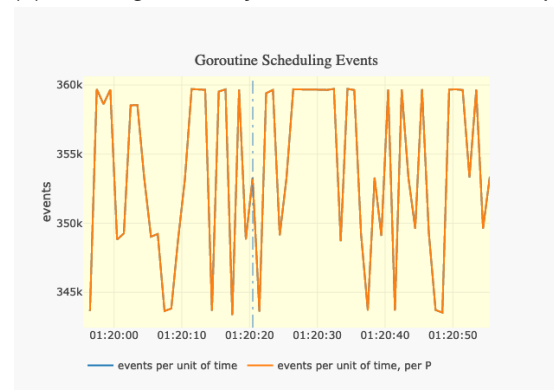
(a) Affichage de l'état de la mémoire Heap



(b) Affichage des objets dans la mémoire Heap



(c) Temps par goroutine en mode 'Runnable'



(d) Etat des events par goroutine

FIGURE 4.13 – Affichage de plots représentant les métriques de l'Armory Mk II

4.3 Implémentation d'une application de démonstration sur l'ordinateur

4.3.1 Serveurs REST et gRPC

Avant de présenter la structure du code de notre application, il convient d'aborder les services *REST* et *gRPC*.

Une *API REST* représente une interface de programmation d'application contrainte de respecter le style de l'architecture *REST* ([RED HAT, 2020](#)). En d'autres termes, ces contraintes ne sont ni un protocole ni une norme. Pour construire une *API REST*, celle-ci doit répondre à des critères particuliers. La liste suivante n'est pas exhaustive mais illustre sa structure ([RED HAT, 2020](#)).

- Les communications entre le client et le serveur sont *stateless*.
- Les requêtes effectuées sur le service sont gérées via *HTTPs*.
- L'architecture du service se divise en couches pour séparer la gestion, la sécurité ainsi que l'équilibrage de charge de l'*API*.
- L'utilisation d'un format standard de données comme *JSON* ou *XML* pour échanger des informations.

L'utilisation d'un serveur *REST* est plus facile que l'implémentation d'un service *SOAP* car sa conception comprend l'utilisation des méthodes telles que GET, POST, PUT et DELETE. De plus, il comprend également le format léger de données comme JSON et présente une indépendance aux protocoles, ne le limitant pas à des règles strictes spécifiques ([JOURNAL DU NET, 2022](#)). Celle-ci est souvent choisie pour le développement d'applications mobiles et idéalement pour l'*IoT*. Son implémentation est simple et rapide. Le suivi des versions des services API peut recourir à des stratégies différentes. Une des plus connues est le *versioning* via le chemin *URI*. L'exemple ci-après montre explicitement quelle version le développeur utilise : <https://www.tbTamago.com/api/v1/get-random>.

D'une autre part, il existe aussi des serveurs que l'on appelle *gRPC*. Ce protocole de communication est une plateforme *opensource* qui simplifie les échanges entre services en exposant certaines fonctionnalités pour des clients externes.

Concernant le type de format d'échange d'information, *REST* utilise pour sa part le format *JSON* ([GYORI, 2022](#)). Basé sur HTTP/2, *gRPC* utilise par défaut les *protocol buffers* appelés aussi *protobuf* comme structure de ces données transmissibles. Elle spécifie les méthodes appelées à distance, les paramètres autorisés ainsi que le type de retour. Du côté du serveur, celui-ci implémente le *gRPC* qui traite les requêtes des clients. Du côté du client, celui-ci offre les mêmes méthodes que celles du serveur.

Avant de poursuivre, nous allons présenter brièvement les différences entre le format *JSON* et le *protobuf*.

4.3 Implémentation d'une application de démonstration sur l'ordinateur

La grande différence entre les deux types est leur structure. Ci-dessous, nous pouvons observer que le format *JSON* se base sur un format de données textuel.

```
{
  "students": [
    {
      "name": "Alice",
      "age": 22,
      "email": "alice@students.hevs.ch"
    },
    {
      "name": "Bob",
      "age": 21,
      "email": "bob@students.hevs.ch"
    }
  ]
}
```

JSON est plus adapté pour les types textuels que numériques et celui-ci est utilisé quasiment par tous les langages.

En revanche, le *protobuf* suit une structure différente du format *JSON*. Il utilise le format de message binaire ce qui permet aux applications d'échanger des données même si celles-ci sont codées dans d'autres langages de programmation. Le format *JSON* autorise aussi cette interopérabilité entre différents systèmes. Il met également à disposition des modificateurs comme optional, required ou repeated pour définir la forme de la structure d'une donnée (IONOS, 2020).

```
{
  message Student {
    string name = 1;
    int32 age = 2;
    string email = 3;
  }

  message StudentList {
    repeated Person people = 1;
  }

  StudentList studentList = {
    students: [
      { name: "Alice", age: 22, email: "alice@students.hevs.ch" },
      { name: "Bob", age: 21, email: "bob@students.hevs.ch" },
    ]
  };
}
```

Les deux exemples proposés définissent les mêmes données mais dans un format différent. Le format *JSON* utilise une représentation clé-valeur alors que *protobuf* structure les données avec des "messages" et des champs numérotés. D'une part *JSON* est plus lisible par les

humains et est largement utilisé par la communauté de développeurs, de l'autre *protobuf* est plus compact et offre de meilleures performances pour la sérialisation et la désérialisation d'objets (IONOS, 2020).

Nous pouvons ainsi revenir sur le serveur *gRPC* et observer sur la figure ci-dessous son fonctionnement de base. Par exemple, il est facile de créer un serveur en Java ou C++ et par la même occasion des clients utilisant Android ou Ruby comme code source. Les méthodes créées sont définies dans un fichier du type *.proto*. La définition de ce service doit être disponible sur le serveur et sur le client.

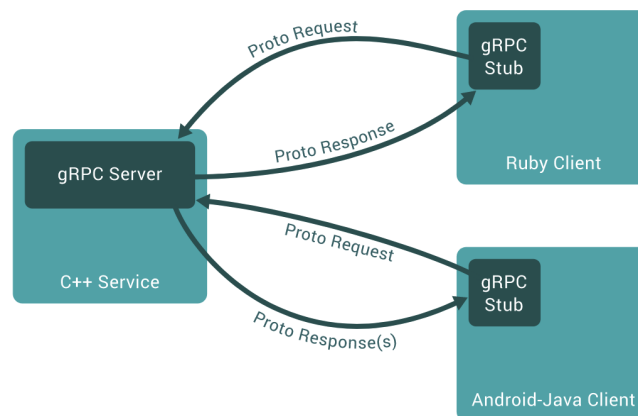


FIGURE 4.14 – Exemple de fonctionnement d'un service *gRPC*

Source : à partir de <https://grpc.io/docs/what-is-grpc/introduction/#overview>

4.3.2 OpenApi et Swagger

Pour le développement de notre programme, nous allons implémenter un serveur *gRPC*-gateway avec la documentation d'*OpenApi*. Le *gRPC*-gateway est expliqué dans la sous-section suivante. La fonctionnalité que nous proposons est la génération d'un *True Random Number Generator*. Cette méthode est accessible à travers les spécifications *OpenApi*, connue aussi sous le nom de *Spécification Swagger*. Ce service permet de décrire le format de notre *API*.

- Description des opérations possibles (GET, PULL, PUSH, DELETE)
- Méthode d'authentification
- Information de contact du support du serveur, licence et termes d'utilisation
- Paramètres autorisés ainsi que les entrées et sorties de chaque méthode

La figure ci-dessous montre un exemple d'affichage d'un service *API* avec les spécifications *OpenApi*.

4.3 Implémentation d'une application de démonstration sur l'ordinateur

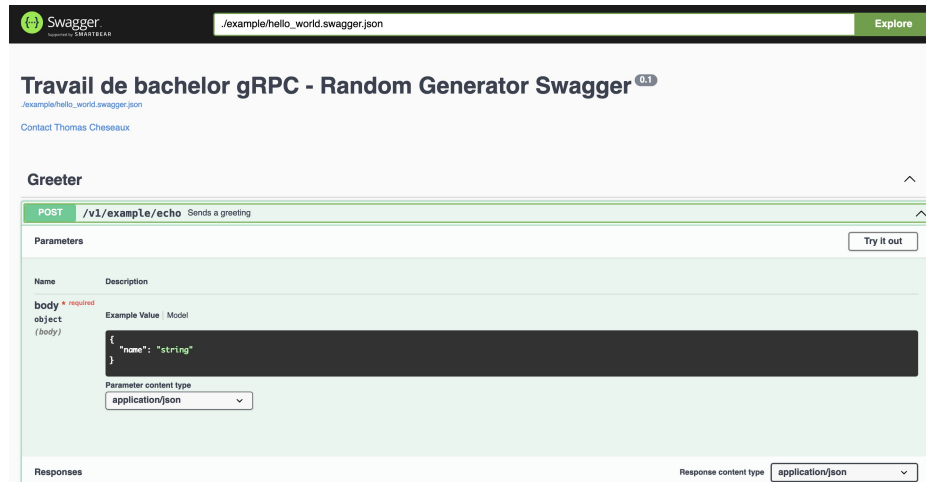


FIGURE 4.15 – Exemple d'affichage d'un service API avec Swagger UI

4.3.3 gRPC-Gateway

Le *gRPC-Gateway* est un plugin du compilateur Google nommé aussi *protoc*. Ce plugin lit les configurations du fichier *protobuf* et génère un serveur *reverse-proxy*. Son but est de traduire notre *API REST* en *gRPC*. Les annotations `google.api.http` permettent de spécifier la manière dont les méthodes de notre service sont associées aux opérations RPC (Remote Procedure Call) du serveur *gRPC* ([GitHub](https://github.com/grpc-ecosystem/grpc-gateway), 2023).

La schéma ci-dessous illustre la création du *gRPC-gateway* et du service *gRPC* à l'aide du fichier *.proto*. Pour résumer, notre programme de démonstration se compose d'un serveur *gRPC* et d'un serveur *REST*. Cela est possible grâce au plugin *gRPC-gateway*. L'interface visuelle pour l'utilisateur est détaillée par le fichier de configuration *Swagger*. La méthode de démonstration génère un *True Random Number Generator* grâce au composant présent sur l'Armory Mk II.

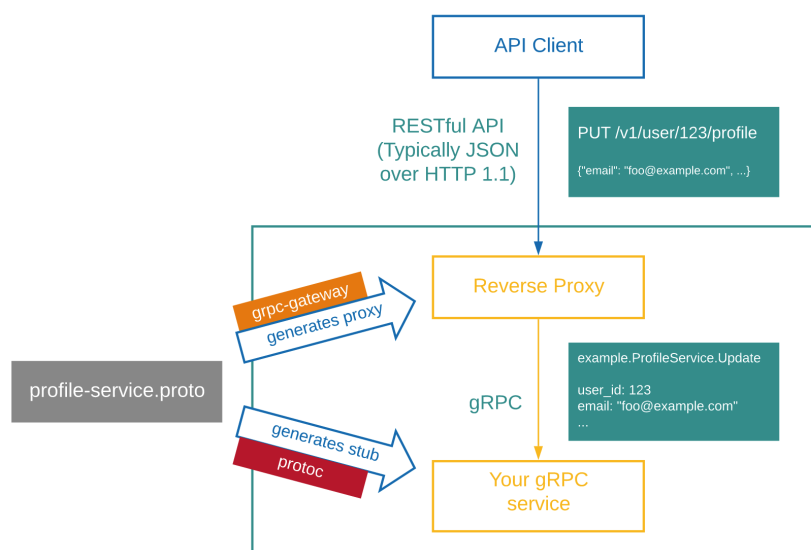


FIGURE 4.16 – Génération du serveur API en gRPC et en REST

Source : à partir de <https://github.com/grpc-ecosystem/grpc-gateway>

4.3.4 Structure du code

Avant de présenter le code source, nous exposons la structure de notre programme ainsi que les fichiers nécessaires à l'exécution de celui-ci.

```
root
├── google
│   └── api
├── proto
│   ├── buf.gen.yaml
│   ├── buf.yaml
│   ├── helloworld
│   └── hello_world.proto
├── protoc-gen-openapiv2
├── swagger
├── go.mod
├── go.sum
└── main.go
```

Dans le dossier `/google/api` nous retrouvons les fichiers nécessaires pour définir les annotations spécifiques à *gRPC* et aux services *HTTP*. De même pour le dossier `protoc-gen-openapiv2`, nous trouvons les fichiers `openapiv2.proto` ainsi que `annotation.proto` pour générer la documentation *OpenApi*. Le dossier `/swagger` regroupe les fichiers pour l'affichage graphique du service *OpenApi*. Les fichiers de configuration `go.mod` et `go.sum` sont utilisés de la manière suivante. Le premier est utilisé pour définir le module Go. Il contient le nom du module ainsi que les dépendances du programme. Le deuxième fichier garantit l'intégrité des dépendances dans le projet. Ensuite le fichier `main.go` contient le code source de notre programme de démonstration.

Pour finir, le dossier `/proto` contient les 2 fichiers `buf.gen.yaml` et `buf.yaml`. Ils sont utiles pour la gestion des définitions du protocole, la compilation ainsi que la génération du code dans notre projet.

4.3.5 Code source de l'application de démonstration

Pour le développement de notre application de démonstration, nous avons utilisé deux répertoires Git^{11 12}. De plus, l'implémentation de la documentation *OpenApi* a été possible grâce au site¹³ de Blain Smith.

Il convient de s'attarder sur quatre fichiers qui nécessitent une attention plus particulière. Commençons tout d'abord par parcourir les deux fichiers `buf.yaml` et `buf.gen.yaml`, enchaînons ensuite avec le fichier `hello_world.proto` et terminons cette section par le fichier `main.go`.

11. https://grpc-ecosystem.github.io/grpc-gateway/docs/tutorials/generating_stubs/using_buf/

12. <https://github.com/grpc-ecosystem/grpc-gateway>

13. <https://blainsmith.com/articles/go-grpc-gateway-openapi/>

4.3 Implémentation d'une application de démonstration sur l'ordinateur

Le premier fichier, `buf.yaml`, est nécessaire lors de l'exécution de la commande `buf generate` qui construit les dépendances du projet grâce au fichier `hello_world.proto`.

```
version: v1 # Version du fichier de configuration Buf

name: buf.build/myuser/myrepo # Nom du dépôt

deps: # Liste des dépendances
- buf.build/googleapis/googleapis # Dépendance des API Google
- buf.build/grpc-ecosystem/grpc-gateway # Dépendance gRPC Gateway
```

Le fichier ci-dessous, `buf.gen.yaml`, indique les plugins à utiliser et définit la manière de générer le code source selon le fichier `.proto`.

```
version: v1
plugins:
- plugin: go
  out: ./
  opt: paths=source_relative
- plugin: go-grpc
  out: ./
  opt: paths=source_relative,require_unimplemented_servers=false
- plugin: grpc-gateway
  out: ./
  opt: paths=source_relative
- name: openapi2
  out: ../../swagger/swagger-ui/example/
```

`hello_world.proto` est le fichier de configuration des méthodes et définit les arguments et le type de retour de la méthode de génération de nombre aléatoire.

```
service Greeter {
  rpc RndGenerator(RndRequest) returns (RndReply) {
    // google.api.http spécifie méthode RPC accessible via une requête HTTP GET.
    option (google.api.http) = {
      get: "/v1/example/rndGen"
    };
  }
}

// paramètre d'entrée pour la méthode RndGenerator.
message RndRequest {}

// valeur de retour pour la méthode RndGenerator.
message RndReply {string message = 1;}
```

Pour terminer, nous présentons le code source du fichier `main.go` qui définit le lancement de notre serveur *gRPC* ainsi que l'affichage du *Swagger*. Dans le code source ci-dessous, toutes les importations ne sont pas affichées. L'entièreté du programme est sauvegardée sur Git¹⁴.

14. <https://gitlab.com/hesso-vs/business-information-technology/65-62-bachelor-thesis/thomas-cheseaux-2023-grpc-machine>

```
import (
    "crypto/rand"
    ...
)

type server struct{
    // implémentation du service 'GreeterServer' depuis fichier pb.
    pb.UnimplementedGreeterServer
}

func NewServer() *server {
    // retourne nouvelle instance du serveur
    return &server{}
}

func (s *server) RndGenerator(ctx context.Context, _ *pb.RndRequest)
↳ (*pb.RndReply, error) {
    // Génération tableau de bytes aléatoire
    buf := make([]byte, 32)
    rand.Read(buf)

    return &pb.RndReply{Message: fmt.Sprintf("%x", buf)}, nil // conversion
↳ tableau en hexadécimale
}
```

Ci-dessous, nous retrouvons la fonction *main()*, celle-ci crée les objets nécessaires pour le serveur *gRPC*. La configuration des ports d'écoute du service sont mis en place. A la fin de la méthode, le service *gRPC*-Gateway est lancé.

```
func main() {
    // Création du listener sur le port 8080
    lis, err := net.Listen("tcp", ":8080")
    if err != nil {
        log.Fatalln("Failed to listen:", err)
    }

    // Création objet gRPC serveur
    s := grpc.NewServer()
    // Service 'GreeterServer' attaché au serveur
    pb.RegisterGreeterServer(s, &server{})

    // Démarrage en arrière plan du serveur dans un goroutine
    go func() {
        log.Fatalln(s.Serve(lis))
    }()

    // Création connexion cliente avec server gRPC
    conn, _ := grpc.DialContext(
        context.Background(),
        "0.0.0.0:8080",
        grpc.WithBlock(),
        grpc.WithTransportCredentials(insecure.NewCredentials()),
    )

    // Création de la passerelle gRPC
    gwmux := runtime.NewServeMux()
    // Implémentation de la gestion des requêtes liées à Swagger
    muxSwagger := http.NewServeMux()
```

4.3 Implémentation d'une application de démonstration sur l'ordinateur

```
muxSwagger.Handle("/", gwmux) //routes par défaut gérées par le gwmux

// Enregistrement du service 'Greeter' avec gwmux
err = pb.RegisterGreeterHandler(context.Background(), gwmux, conn)
// Activation de swagger si on trouve le fichier swagger.json
if _, err := os.Stat(pathSwaggerJson); err == nil {
    muxSwagger.HandleFunc("swagger/swagger-ui/example/swagger.json", func(w
        ↪ http.ResponseWriter, r *http.Request){
        // Indique au serveur de renvoyer le contenu du swagger.json quand le
        ↪ fichier est appelé
        http.ServeFile(w, r, pathSwaggerJson)
    })

    fs := http.FileServer(http.Dir("swagger/swagger-ui")) // racine swagger-ui
    // Appel l'url "swagger-ui", affichage du contenu du dossier
    muxSwagger.Handle("/swagger-ui/", http.StripPrefix("/swagger-ui/", fs))
}

// Création server HTTP pour Swagger UI et configuration port 8090
gwServer := &http.Server{
    Addr:      ":8090",
    Handler:   muxSwagger,
}

// Lancement serveur gRPC-Gateway
log.Fatalln(gwServer.ListenAndServe())
}
```

Après l'exécution du service, nous obtenons le résultat suivant.

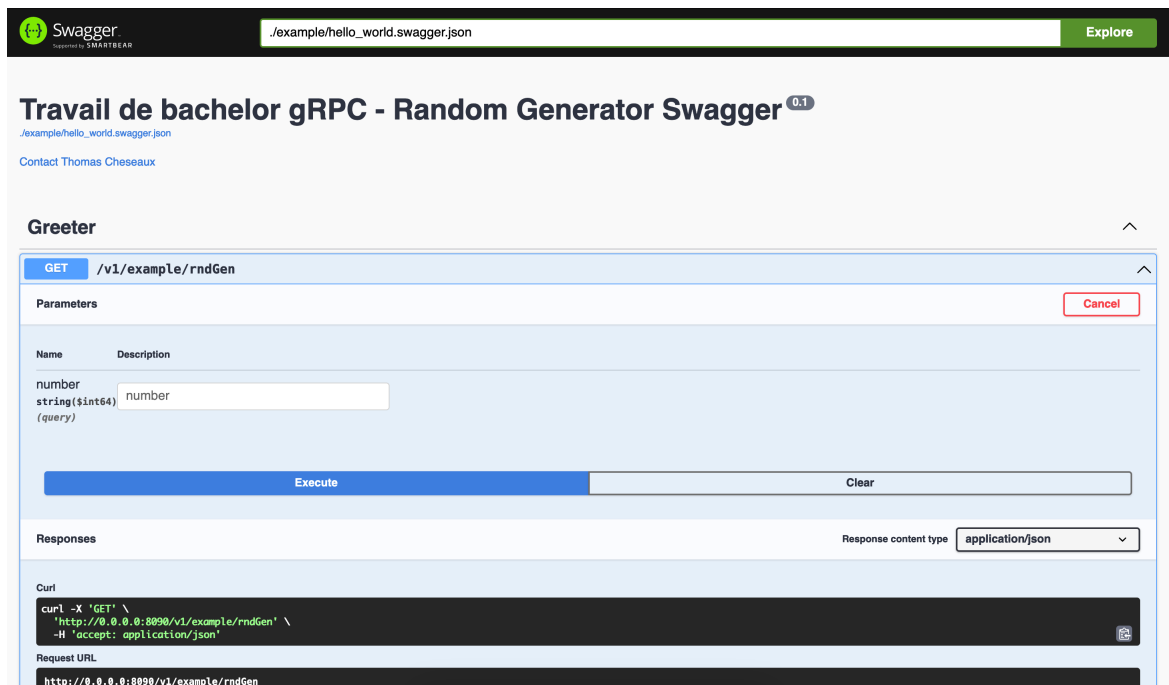


FIGURE 4.17 – Affichage de la méthode de génération de nombres aléatoires avec Swagger UI

5 | Proof of concept

Nous arrivons à la dernière partie de notre travail. Après avoir créé un programme offrant un service *gRPC*, il est temps d'intégrer notre programme de démonstration au périphérique. Nous effectuons ensuite quelques adaptations expliquées plus bas dans cette section. Celles-ci sont nécessaires pour atteindre notre objectif, qui est de faire fonctionner notre programme de démonstration sur le périphérique.

Pour le développement de notre application, nous utilisons le code source de base de l'application de démonstration du Git¹. Notons qu'il n'existe malheureusement pas de versioning ou de tag. Ainsi, notre programme se base sur l'ID du commit (4e59101d294fc22aae7931e84f17661de9f82058) du 11 juillet 2023.

5.1 Code source TamaGo

Dans cette section, nous exposons les parties du code source de l'application de démonstration TamaGo. Seuls quelques fichiers clés sont présentés. L'entièreté du code source que nous avons modifié se trouve sur ce répertoire Git².

Pour commencer, nous présentons les fichiers de configuration `buf.yaml`, `buf.gen.yaml` et `bachelor.proto`. Ces fichiers permettent la configuration et la génération automatique du service *gRPC*. Ensuite, nous expliquons les étapes de lancement de l'application décrites dans le fichier `main.go`. Puis, nous enchaînons avec le fichier `imx_usbnet.go` qui démarre et configure la connexion *USB*. Enfin, nous terminons cette partie avec le fichier `web_server_swagger.go` qui englobe le démarrage et la configuration du serveur Web.

Le fichier `buf.yaml` utilise le fichier `bachelor.proto` pour construire les dépendances liées au projet. En complément du fichier précédent, `buf.gen.yaml` liste les plugins nécessaires à l'application et mentionne l'emplacement de ceux-ci.

```
version: v1 # Version du fichier de configuration Buf

name: buf.build/myuser/myrepo # Nom du dépôt

deps: # Liste des dépendances
# Dépendance des API Google
- buf.build/googleapis/googleapis
# Dépendance sur le dépôt gRPC Gateway
- buf.build/grpc-ecosystem/grpc-gateway
```

1. <https://github.com/usbarmony/tamago-example>

2. <https://gitlab.com/hesso-vs/business-information-technology/65-62-bachelor-thesis/thomas-cheseaux-2023-tamago>

```

version: v1
plugins:
- plugin: go
  out: ./
  opt: paths=source_relative
- plugin: go-grpc
  out: ./
  opt: paths=source_relative,require_unimplemented_servers=false
- plugin: grpc-gateway
  out: ./
  opt: paths=source_relative
- name: openapiv2
  out: ../network/static/

```

Le fichier de configuration `bachelor.proto` regroupe les méthodes du serveur *gRPC*, les paramètres d'entrée et de retour ainsi que des informations annexes utiles à l'affichage du service *gRPC*.

```

syntax = "proto3";

package proto;
option go_package = "../proto";

import "thirdParty/google/api/annotations.proto";
import "thirdParty/protoc-gen-openapiv2/options/annotations.proto";

option (grpc.gateway.protoc_gen_openapiv2.options.openapiv2_swagger) = { info:
↪ {
  title: "Travail de bachelor gRPC - TamaGo et Swagger";
  version: "0.1";
  contact: {
    name: "Thomas Cheseaux";
    email: "thomas.cheseaux@students.hevs.ch";
  };
};
};

// Définition du service Greeter avec chaque fonction que le serveur gère
service Greeter {
  rpc RndGenerator(RndRequest) returns (RndReply) {
    // google.api.http spécifie méthode RPC accessible via une requête HTTP
    option (google.api.http) = {
      get: "/api/v1/example/rndGen"
    };
  }
}

// paramètre d'entrée pour la méthode RndGenerator
message RndRequest {}

// valeur de retour pour la méthode RndGenerator
message RndReply {
  string message = 1;
}

```

Chapitre 5. Proof of concept

L'application de démonstration contient le fichier `main.go` situé à la racine du projet. Celui-ci appelle la fonction principale *main* et lance ainsi le programme.

```
func main() {
    var usb *usb.USB
    var eth *enet.ENET

    // Ouverture du fichier de journalisation en mode écriture
    logFile, _ := os.OpenFile("/tamago-example.log",
        ↪ os.O_WRONLY|os.O_CREATE|os.O_TRUNC, 0600)
    // Définition de la sortie de journalisation pour écrire sur la sortie
    ↪ standard et dans le fichier de journalisation
    log.SetOutput(io.MultiWriter(os.Stdout, logFile))

    // Vérification de la disponibilité des interfaces USB et Ethernet
    hasUSB, hasEth := cmd.HasNetwork()

    console := &cmd.Interface{}

    // Démarrage de l'interface USB/Ethernet
    if hasUSB {
        usb = network.StartUSB(console.Start, logFile)
    }

    if hasEth {
        eth = network.StartEth(console.Start, logFile)
    }

    cmd.NIC = eth

    if hasUSB || hasEth {
        // Configuration des ressources Web statiques
        network.SetupStaticWebAssets(cmd.Banner)
        // Démarrage du gestionnaire d'interruption avec les interfaces
        ↪ USB/Ethernet
        network.StartInterruptHandler(usb, eth)
    } else {
        // Utilisation de la console série en cas d'absence d'interfaces
        ↪ USB/Ethernet
        cmd.SerialConsole(console)
        semihosting.Exit()
    }
}
```

Après l'exécution de la fonction principale, le programme prendra en compte le type de connexion *USB* ou *Ethernet* pour déterminer la suite des opérations. Ici, le programme appelle la fonction *StartUSB()* située dans le fichier `imx-usbnet.go`.

```
const hostMAC = "1a:55:89:a2:69:42"

...

// Démarrage de l'interface USB
func StartUSB(console consoleHandler, journalFile *os.File) (port *usb.USB) {
    port = imx6ul.USB1 // Sélection du port USB (imx6ul.USB1)

    // Initialisation de l'interface réseau USB
    iface, err := usbnet.Init(IP, MAC, hostMAC, 1)
```



```

    if err != nil {
        log.Fatalf("could not initialize USB networking, %v", err)
    }

    port.Device = iface.NIC.Device

    // Activation de la prise en charge ICMP (Internet Control Message
    ↪ Protocol)
    iface.EnableICMP()

    if console != nil {
        // Initialisation du serveur SSH sur l'interface réseau USB
        listenerSSH, err := iface.ListenerTCP4(22)

        if err != nil {
            log.Fatalf("could not initialize SSH listener, %v", err)
        }

        go StartSSHServer(listenerSSH, console)
    }

    // Initialisation des listeners
    listenerHTTP, err := iface.ListenerTCP4(80)

    if err != nil {
        log.Fatalf("could not initialize HTTP listener, %v", err)
    }

    listenerHTTPS, err := iface.ListenerTCP4(443)

    if err != nil {
        log.Fatalf("could not initialize HTTP listener, %v", err)
    }

    listenerHTTPgrpc, err := iface.ListenerTCP4(8080)

    if err != nil {
        log.Fatalf("could not initialize HTTP listener, %v", err)
    }

    go startWebServer(listenerHTTP, IP, 80, false)
    go startWebServer(listenerHTTPS, IP, 443, true)
    go startWebServerBasic(listenerHTTPgrpc, IP, 8080, false)

    journal = journalFile

    // Initialisation et configuration du port USB
    port.Init()
    port.DeviceMode()

    ...

    // Branchement de l'interface dans le runtime Go
    net.SocketFunc = iface.Socket

    return
}

```

Chapitre 5. Proof of concept

Le dernier fichier présenté ici est le `web_server_swagger.go`. Au sommet de celui-ci, nous listons les importations nécessaires à l'exécution des différentes fonctions du fichier.

```
import (  
    "context"  
    "crypto/rand"  
    "crypto/tls"  
    "embed"  
    "fmt"  
    "log"  
    "net"  
    "net/http"  
  
    "github.com/grpc-ecosystem/grpc-gateway/v2/runtime"  
    bachelorpb "github.com/usbarmory/tamago-example/proto/bachelor"  
    "google.golang.org/grpc"  
    "google.golang.org/grpc/credentials/insecure"  
)
```

Nous créons tout d'abord un système de fichiers embarqués à l'aide de `embed.FS` qui met à disposition du programme des ressources lorsque l'application est démarrée.

```
1  var (  
2      //go:embed static/*  
3      static embed.FS // Système de fichiers embarqués pour les ressources  
4      ↪ statiques  
5      swaggerFilePath = "/static/bachelor/bachelor.swagger.json"  
6  )
```

Notre programme propose une méthode de génération de vrais nombres aléatoires. Cela étant possible grâce aux composants de l'Armory Mk II expliqués dans la partie 2.2. Comme mentionné dans la partie "Génération de nombres aléatoires en cryptographie", un vrai nombre aléatoire est un nombre totalement imprévisible qui ne suit aucun modèle ou règle prévisible. C'est-à-dire qu'il n'est pas possible de deviner le prochain nombre aléatoire généré (PETURA, 2019).

```
1  func (s *server) RndGenerator(ctx context.Context, _ *bachelorpb.RndRequest)  
2  ↪ (*bachelorpb.RndReply, error) {  
3      buf := make([]byte, 32) // Génération tableau de bytes aléatoire  
4      rand.Read(buf)  
5      return &bachelorpb.RndReply{Message: fmt.Sprintf("%x", buf)}, nil  
6  }
```

Ensuite, la méthode `startWebServerBasic()` configure soit le serveur `HTTP`s, soit le serveur `HTTP` selon les arguments de la méthode.

```
1  func startWebServerBasic(listener net.Listener, addr string, port uint16,  
2  ↪ https bool) {  
3      var srv *http.Server  
4      var err error
```

```

5      if https {
6          srv, err = configureTLSServer(addr, port)
7      } else {
8          srv, err = configureHTTPServer(addr, port, listener)
9      }
10
11     log.Println("Serving gRPC-Gateway on http://10.0.0.1:8090")
12     // Lancement serveur gRPC-Gateway
13     log.Fatalln(srv.ListenAndServe())
14
15     log.Fatal("server returned unexpectedly ", err)
16 }

```

La première configuration *HTTP*s utilise la génération des certificats *TLS* et la création de clés *X.509*.

```

1  // Fonction configure le serveur avec TLS et retourne le serveur configuré
2  func configureTLSServer(addr string, port uint16) (*http.Server, error) {
3      // Génération des certificats TLS
4      TLSCert, TLSKey, err := generateTLSCerts(net.ParseIP(addr))
5      if err != nil {
6          return nil, fmt.Errorf("TLS cert|key error: %v", err)
7      }
8
9      ...
10
11     // Création de la paire de clés X.509 pour le certificat TLS
12     certificate, err := tls.X509KeyPair(TLSCert, TLSKey)
13     if err != nil {
14         return nil, fmt.Errorf("X509KeyPair error: %v", err)
15     }
16
17     // Configuration du serveur HTTP avec prise en charge du TLS
18     srv := &http.Server{
19         Addr: addr + ":" + fmt.Sprintf("%d", port),
20         TLSConfig: &tls.Config{
21             Certificates: []tls.Certificate{certificate},
22         },
23     }
24
25     return srv, nil
26 }

```

La seconde configuration crée un serveur *gRPC*, établit une connexion interne vers le serveur (10.0.0.1), configure un gestionnaire de requête *HTTP*, implémente les fichiers de configuration *OpenApi* et démarre finalement le serveur *gRPC*-Gateway.

```

1  func configureHTTPServer(addr string, port uint16, listener net.Listener)
2  ↪ (*http.Server, error) {
3      // Création du serveur gRPC
4      grpcServer := grpc.NewServer()
5      bachelorpb.RegisterGreeterServer(grpcServer, &server{})
6      log.Println("Serving gRPC on 10.0.0.1:8080")
7      go func() {
8          log.Fatalln(grpcServer.Serve(listener))
9      }()

```

```
10 // Etablissement d'une connexion gRPC vers le serveur 10.0.0.1
11 conn, err := grpc.DialContext(
12     context.Background(),
13     "10.0.0.1:8080",
14     grpc.WithBlock(),
15     grpc.WithTransportCredentials(insecure.NewCredentials()),
16 )
17 ...
18
19 // Création d'un multiplexeur de requêtes HTTP. ServeMux peut gérer
20 ↪ plusieurs routes
21 gwmux := runtime.NewServeMux()
22 muxSwagger := http.NewServeMux()
23 configureSwaggerRoutes(muxSwagger)
24 err = bachelorpb.RegisterGreeterHandler(context.Background(), gwmux, conn)
25 ...
26 // Configuration des routes pour le serveur HTTP
27 muxSwagger.Handle("/", gwmux)
28
29 // Création du serveur HTTP avec les routes configurées
30 srv := &http.Server{
31     Addr:    ":8090",
32     Handler: muxSwagger,
33 }
34
35 return srv, nil
}
```

A la ligne 22 du code source ci-dessus, l'implémentation de la documentation *OpenApi* s'exécute à travers la méthode *configureSwaggerRoutes()*. Nous créons un système de fichiers *HTTP* basé sur la variable *static* contenant les ressources requises à l'affichage et à la gestion des requêtes.

```
1 // Fonction de configuration des routes Swagger pour le ServeMux HTTP
2 func configureSwaggerRoutes(muxSwagger *http.ServeMux) {
3     // Création d'un gestionnaire de fichier http
4     fileServer := http.FileServer(http.FS(static))
5     muxSwagger.Handle("/bachelor/", http.StripPrefix("/bachelor/",
6     ↪ fileServer))
7
8     // Vérification bachelor.swagger.json existe dans la variable static
9     if _, err := static.ReadFile("static/bachelor/bachelor.swagger.json"); err
10     ↪ == nil {
11         log.Println("Swagger configuration found")
12
13         // Configuration des routes et des gestionnaires pour le serveur HTTP
14         muxSwagger.HandleFunc(swaggerFilePath, func(w http.ResponseWriter, r
15         ↪ *http.Request) {
16             file, err :=
17             ↪ static.ReadFile("static/bachelor/bachelor.swagger.json")
18             ...
19             // Définition du type de contenu de la réponse HTTP
20             w.Header().Set("Content-Type", "application/json")
21             w.Write(file)
22         })
23     }
24 }
```

Le résultat final de notre implémentation est illustré ci-dessous. Le service *gRPC* utilise la documentation *OpenApi* pour afficher les méthodes disponibles.

The screenshot displays the Swagger UI for the 'Travail de bachelor gRPC - TamaGo et Swagger' API. The interface is organized into several sections:

- Header:** Includes the Swagger logo, the API title, and a version indicator '0.1'.
- Method:** A 'GET' request to '/api/v1/example/rndGen' is shown, with a description: 'Générateur de TRNG'.
- Parameters:** A section indicating 'No parameters'.
- Responses:** A section showing the response for a 200 status code. The response body is a JSON object:


```
{
  "message": "F3181405bd4a85e554244c866a3b4d83c32ad830cc61edc8f5477faa2df3c896"
}
```
- Example Value:** A section showing an example value for the response body:


```
{
  "message": "string"
}
```

FIGURE 5.1 – Affichage du service *gRPC*-Gateway avec *OpenApi*

L'URL ci-après nous permet d'accéder localement au service : `http://10.0.0.1:8090/bachelor/static/#/Greeter/Greeter_RndGenerator`

5.2 Test de la qualité du *True Random Number Generator*

Pour vérifier la qualité du générateur de nombres aléatoires de l'Armory Mk II, nous allons implémenter une solution graphique trouvée sur ce répertoire Git³. Cette application permet d'évaluer un nombre aléatoire binaire généré et d'effectuer plusieurs tests statistiques sur celui-ci. Pour obtenir ce nombre binaire, nous avons créé un petit programme en Python qui exécute une requête *HTTP* vers la clé *USB* et sauvegarde la valeur binaire du nombre aléatoire dans un fichier texte à la racine de notre application Python. La figure ci-dessous illustre les résultats obtenus pour la valeur suivante :

57a06f3f2c241293785dd371cfb2133f53284f9c5334d459eae36772082ab501

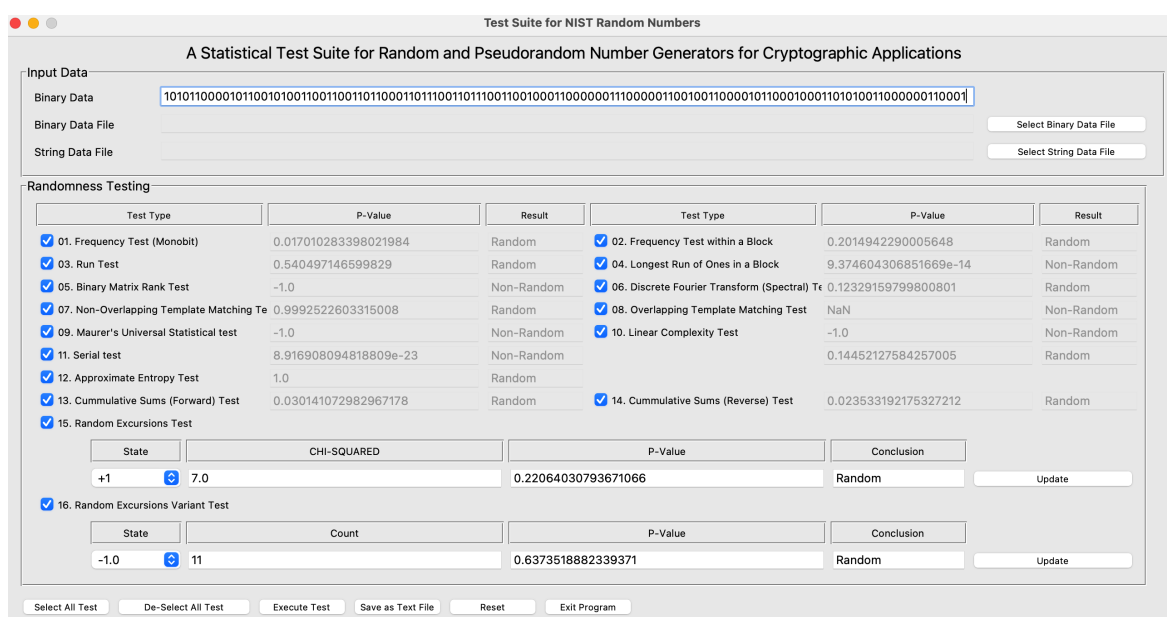


FIGURE 5.2 – Affichage des résultats de la suite des tests statistiques du nombre aléatoire

Les résultats affichés illustrent que le nombre aléatoire fabriqué n'atteint pas totalement tous les tests statistiques effectués.

Néanmoins, la réussite de l'implémentation d'un serveur *gRPC* et l'affichage des résultats à l'aide de la documentation *OpenApi*, nous font conclure que l'utilisation de l'Armory Mk II comme un périphérique de sécurité complémentaire aux hébergements en ligne est envisageable. De plus, l'Armory Mk II propose un générateur *True Random Number Generator* qui est crucial dans le secteur de la cryptographie car il garantit l'imprévisibilité et la sécurité des clés compliquant ainsi leur prédiction et rendant par la même occasion les systèmes de chiffrement plus robustes contre les attaques. Bien sûr, nous sommes conscients que le chemin est encore long pour une implémentation dans un environnement de production.

3. https://github.com/stevenang/randomness_testsuite/blob/master/README.md

En somme, notre travail a su positionner l'Armory Mk II comme un dispositif complémentaire à l'amélioration future de la gestion des clés de chiffrement et comme une potentielle alternative aux *HSM* selon l'emploi et le niveau de sécurité requis. Dans la section suivante, nous nous attarderons sur les alternatives existantes pour la gestion des clés de chiffrement.

6 | Alternatives

Dans cette dernière section, nous discutons de quelques alternatives pour la gestion des clés cryptographiques. Les solutions alternatives considérées sont les *HSM*, les *SoftHSM* et les solutions *Cloud*. Une des solutions possibles que nous avons aussi étudiée est l'utilisation de l'Armory Mk II.

6.1 Hardware Security Modules

Comme mentionné dans la partie 2. "Concepts et Technologies", les *HSM* sont un dispositif physique de sécurité pour gérer, stocker, et générer des clés cryptographiques. La réussite de leur mise en place dépend de la présence de personnel qualifié, faute de quoi tous les efforts déployés seront vains. En ce qui concerne leur aspect physique, ceux-ci sont vendus sous la forme de cartes PCI Express. L'achat d'un tel dispositif nécessite la prise en compte des coûts mentionnées ci-dessous :

- L'achat du module de départ.
- L'installation et la mise en place correcte du système de sécurité.
- Les coûts liés au déploiement.
- La maintenance du dispositif et son renouvellement.

Pour l'heure, il est difficile d'estimer le coût total lié à ces modules à cause des nombreuses inconnues relatives au personnel nécessaire. N'oublions pas que pour l'intégration, l'installation et la maintenance de ces modules, une entreprise doit avoir les ressources nécessaires tant au niveau humain que financier.

En conclusion, les *HSM* sont un dispositif efficace pour la génération, le stockage et la protection des clés cryptographiques. Cependant, leurs coûts demeurent une barrière pour encore de nombreuses *PME*.

6.2 SoftHSM

Après avoir présenté la solution des modules de sécurité matériels *HSM*, nous nous concentrons sur une alternative logicielle telle que les *SoftHSM*. Les *SoftHSM* ont été développés dans le but de se familiariser avec les spécifications *PKCS#11* et de simuler l'utilisation d'un véritable *HSM* (D'CRUZ, 2020). Cette alternative s'intègre dans un projet de plus ample envergure nommé OpenDNSSEC dont l'objectif est de favoriser l'adoption du *DNSSEC* appelée aussi Domain Name System Security Extensions et de renforcer de manière générale la sécurité d'Internet (OPENDNSSEC, s. d.).

L'abréviation anglaise *DNSSEC* appelée aussi extensions de sécurité du système de noms de domaine est une signature cryptographique ajoutée au *DNS*. Le *DNS* ou Domain Name System est un système qui traduit les noms de domaine en adresses *IP*. Celui-ci permet ainsi aux utilisateurs d'accéder à un site web en utilisant des adresses textuelles `www.google.ch` plutôt que des adresses numériques `172.217.168.67`.

Il est important de noter que l'utilisation de cette alternative n'est pas conseillé dans un environnement de production car celle-ci n'offre pas un stockage sûr pour les clés cryptographiques. En effet, cette option donne une fausse impression de protection et de sécurité ([AMAZON, s. d.](#)).

En somme, cette alternative est utile pour se familiariser avec le standard *PKCS#11* et reproduire une utilisation réelle d'un *HSM*. Tout ce procédé de familiarisation et de simulation vise à intégrer et adopter ces nouveaux standards pour une future implémentation dans nos propres systèmes et programmes.

6.3 Solution cloud

Pour les solutions *Clouds* de *HSM*, nous avons décidé de nous concentrer ici sur le service Cloud HSM de Google. Nous sommes conscients qu'il existe d'autres solutions en ligne mais nous en avons uniquement choisi une pour illustrer nos propos.

Comme les modules physiques de *HSM*, ce service "Cloud HSM" génère, stocke et protège les clés de chiffrement. Concernant les standards de sécurité, cette solution est certifiée *Federal Information Processing Standard (FIPS)-140-3*. En plus d'offrir un haut niveau de sécurité, ce service offre certains avantages comme ([ADIT et IL-SUNG, 2021](#)) :

- Une disponibilité et une séparation stricte des clés dans le monde entier : l'utilisateur peut appliquer des restrictions à l'utilisation des clés disponibles selon des zones géographiques.
- Un moyen d'accéder au service via une *API* unifiée.
- Une gestion centrale des clés de chiffrement : différentes ressources basées sur *HSM* peuvent être gérées en même temps depuis une interface commune.
- L'abstraction du matériel *HSM* : le code source pour interagir avec les *HSM* est intégré dans la solution en ligne. Pour utiliser le service, le client transmet des requêtes via l'*API* Cloud *KMS*.
- La responsabilité de la sécurité physique du matériel incombe aux fournisseurs du service.

Certes, une solution *Cloud* a de nombreux avantages comme la passation de la responsabilité quant à la sécurité des infrastructures en cas de vol ou d'attaques physiques. Cependant, il existe aussi des inconvénients comme ([ENTRUST, s. d.](#)) :

- Le client n'est plus maître de ses infrastructures physiques.

- La sécurité physique des clés de chiffrement dépend d'un fournisseur tiers.
- La localisation de l'emplacement des clés n'est pas forcément sur un site unique.
- Les configurations sont effectuées par le client et cela comporte un risque en cas de mauvaises manipulations.
- L'utilisation de ce service n'est pas forcément applicable pour tous car certaines entreprises sont soumises à la souveraineté des données. En d'autres termes, une entreprise doit respecter certaines normes ou lois sur la sécurité et la protection des données qui l'obligent à stocker nationalement ces informations.

En résumé, une solution en ligne simplifie l'utilisation et la mise en place d'un module *HSM* car celui-ci est géré par le fournisseur. La sécurité des infrastructures en cas de vol ou d'attaques sur place incombe au fournisseur. L'accès au service s'effectue à l'aide d'une *API*. En revanche, le client n'a plus la souveraineté de ses infrastructures et de ses données et dépend d'un prestataire externe.

En guise de conclusion des alternatives possibles, nous avons découvert que nous pouvions gérer des clés de chiffrement à l'aide d'un module physique *HSM*. Celui-ci requiert d'importants moyens financiers et humains ainsi que des connaissances accrues en informatique et en cryptographie. Puis, nous nous sommes aperçus que la solution SoftHSM ne peut malheureusement pas être utilisée dans un environnement de production car celle-ci n'offre pas une protection sûre mais plutôt une familiarisation et une reproduction d'une utilisation réelle d'un *HSM*. Finalement, la solution *Cloud* simplifie l'utilisation et la gestion d'un module *HSM* via un fournisseur mais entraîne une perte de la souveraineté sur les infrastructures et les données clientes.

7 | Conclusion

Dans ce dernier chapitre, nous présentons une synthèse du travail effectué depuis le 1^{er} mai 2023 jusqu'au 28 juillet de la même année. Nous enchaînons en exposant les limitations rencontrées tout au long de l'implémentation de l'application de démonstration. Finalement, nous proposons d'éventuelles axes de recherches et de développement sur l'Armory Mk II.

7.1 Synthèse

Ce travail nous a permis de prendre connaissance du langage Go que nous n'avions pas pu pratiquer lors de notre cursus scolaire, approfondir nos connaissances sur les concepts et les technologies liés à l'environnement des clés de chiffrement, explorer les avantages et les inconvénients des différents langages de programmation autorisant l'exécution sur des composants *bare metal*, évaluer, documenter et implémenter une application de démonstration sur Git et finalement l'adapter selon nos besoins et l'exécuter avec succès sur l'Armory Mk II.

De plus, ce rapport a été une opportunité unique de travailler sur un système embarqué de type Armory Mk II.

7.2 Limites du travail

Comme mentionné dans la méthodologie, nous avons gardé une trace des limitations et embûches rencontrées lors de l'implémentation du programme de démonstration.

La première limitation rencontrée se situe au niveau des tags du Git principal. Les tags sont associés au commit, ceux-ci sont utiles pour créer un marqueur dans l'historique du Git. A chaque version déployable, il est judicieux d'ajouter un tag à son commit. Cela facilite la gestion des versions et permet aux programmeurs de naviguer plus facilement à travers l'historique du répertoire. En développant notre programme de démonstration, nous nous sommes aperçus que le code source principal avait évolué depuis notre première copie du répertoire. Entre la mi-mai et la fin juin, des modifications ont été faites. Celles-ci ne devaient pas causer de problèmes à notre programme. Cependant, les nouvelles fonctionnalités apportées perturbent notre ancienne version. En d'autres termes, après avoir mis à jour à nouveau le répertoire de base, notre code source ne fonctionnait plus.

La deuxième limitation concerne la documentation. Bien que celle-ci soit généralement utile, nous avons rencontré des difficultés dans un cas précis : lors de la mise en place de la communication entre l'ordinateur et le périphérique. Pour réaliser cette opération, nous avons passé de nombreux jours à configurer l'interface du périphérique sans succès. Après deux semaines infructueuses, nous avons décidé de contacter le propriétaire du répertoire Git. Après

quelques échanges de mails, une solution a rapidement pu être trouvée. Cela suggère que nous n'étions probablement pas les seuls à avoir rencontré ce problème. Concernant l'affichage du statut inactif de l'interface de l'Armory Mk II, nous n'avons pas réussi à comprendre pourquoi celle-ci ne se mettait pas à jour. L'erreur d'affichage du statut a ainsi été l'une des causes de nos difficultés.

Au cours du développement, nous avons rencontré des difficultés à implémenter notre serveur *gRPC*. La raison de cet échec était l'utilisation d'une ancienne version incompatible avec notre application. Malgré nos efforts durant deux semaines, nous ne sommes pas parvenus à faire fonctionner notre service. Heureusement, une nouvelle version (1.20.5) a été déployée sur le site de TamaGo ¹, permettant l'établissement d'une connexion *gRPC* vers notre serveur, une fonctionnalité qui était jusque-là impossible à réaliser.

En résumé, le développement du programme de démonstration a été confronté à plusieurs limitations qui se concentrent autour des tags du Git principal, de la documentation et de l'implémentation du serveur *gRPC*. Ces obstacles ont mis en évidence l'importance de la gestion des versions, de la communication avec le propriétaire du répertoire, de la compatibilité des outils ainsi que de la qualité et la structure de la documentation proposée. Malgré ces difficultés, nous avons réussi à surmonter les obstacles grâce à notre persévérance.

7.3 Perspectives de recherches

Nous sommes conscients que le travail effectué ne présente pas tout le potentiel de l'Armory Mk II et qu'il existe encore des axes de recherche encore inexplorés.

Tout d'abord, notre première idée d'élargissement consisterait à configurer une image de base d'un système d'exploitation permettant aux utilisateurs d'effectuer des actions sensibles comme des transactions bancaires dans un environnement spécialement conçu. Nous supposons que la portabilité du périphérique et la sécurité des composants de l'Armory Mk II seraient des caractéristiques utiles à la mise en œuvre d'une telle solution.

Deuxièmement, nous suggérons d'explorer les solutions existantes d'un gestionnaire de mots de passe ainsi que sa mise en œuvre sur le périphérique. Ce travail de recherche pourrait être intégré à l'avenir avec notre première proposition - le développement et la recherche d'un système d'exploitation dédié aux transactions sensibles.

Finalement, nous proposons d'étudier la possibilité d'implémenter un processus de cryptage et de décryptage des données ou de signature numérique. Cette recherche serait l'occasion de comprendre et d'implémenter ce processus tout en offrant à l'utilisateur le choix de sélectionner librement l'algorithme de chiffrement et les données. La signature numérique est un moyen mathématique de vérifier l'intégrité et l'authenticité du contenu d'un document électronique

1. <https://github.com/usbarmory/tamago-go/tree/tamago1.20.5>

(OFFICE FÉDÉRAL DE L'INFORMATIQUE ET DE LA TÉLÉCOMMUNICATION OFIT, s. d.). Ceci contribuerait à renforcer la protection des données sensibles et d'une manière globale la sécurité informatique.

Bibliographie

- ADIT, S. & IL-SUNG, L. (2021, juin). Architecture Cloud HSM. Récupérée le 18 juillet 2023, à partir de <https://cloud.google.com/docs/security/cloud-hsm-architecture?hl=fr>
- AKANOA. (2023, février 20). Les raisons d'être de Rust. Récupérée le 23 mai 2023, à partir de <https://lafor.ge/rust/pourquoi/>
- AMAZON. (s. d.). Module 7 : Simulating Hardware Security Integration - AWS IoT Greengrass. Récupérée le 17 juillet 2023, à partir de <https://docs.aws.amazon.com/greengrass/v1/developerguide/console-mod7.html>
- ASTOPHER. (2016, janvier 4). Why are Rust executables so huge ? Récupérée le 22 juillet 2023, à partir de <https://stackoverflow.com/questions/29008127/why-are-rust-executables-so-huge>
- BAERISWYL, J. (2018). *Next Generation Digital Arithmetic Posit dans GNU/Linux*. Haute École d'ingénierie et de gestion du canton de Vaud.
- BARISANI, A. (2014, avril 10). CSE - Conception des systèmes embarqués. Récupérée le 22 mai 2023, à partir de <https://github.com/abarisani/abarisani.github.io/blob/master/research/tamago/TamaGo.pdf>
- BARISANI, A. (2022, août 23). Armory-Drive. Récupérée le 24 juillet 2023, à partir de <https://github.com/usbarmory/armory-drive>
- BARISANI, A. (2023a, juillet 17). GitHub - USbarmory/tamago : TamaGo - ARM/RISC-V Bare Metal Go. Récupérée le 20 juillet 2023, à partir de <https://github.com/usbarmory/tamago>
- BARISANI, A. (2023b, mai 24). GoKey. Récupérée le 24 juillet 2023, à partir de <https://github.com/usbarmory/GoKey>
- BARISANI, A. (2023c, mai 25). Hardware security features (Mk II). Récupérée le 21 juillet 2023, à partir de [https://github.com/usbarmory/usbarmory/wiki/Hardware-security-features-\(Mk-II\)](https://github.com/usbarmory/usbarmory/wiki/Hardware-security-features-(Mk-II))
- BARISANI, A. (2023d, mai 8). Internals. Récupérée le 22 mai 2023, à partir de <https://github.com/usbarmory/tamago/wiki/Internals>
- BLOGGER, P. (2015, juin 23). Embedded C : un langage optimisé pour l'embarqué - Pentalog. Récupérée le 16 juin 2023, à partir de <https://www.pentalog.fr/blog/systemes-embarques/embedded-c-un-langage-optimise-pour-les-objets-connectes/>

- CAMPANA, G. & BÉDRUNE, J.-B. (2019, juin 5). Everybody be cool, this is a robbery ! Récupérée le 16 mai 2023, à partir de https://www.sstic.org/media/SSTIC2019/SSTIC-actes/hsm/SSTIC2019-Article-hsm-campana_bedrune_neNSDyL.pdf
- CHEVALIER, P. (2022, avril 6). L'entropie sous Linux et la génération de nombres aléatoires. Récupérée le 21 juillet 2023, à partir de <https://www.deltasight.fr/entropie-linux-generation-nombres-aleatoires/>
- CLOUDFLARE. (2023, avril 28). A simple vaultless password manager in Go. Récupérée le 6 juin 2023, à partir de <https://github.com/cloudflare/gokey>
- COX, R. (2015, mai). Go 1.5 GOMAXPROCS Default. Récupérée le 22 mai 2023, à partir de https://docs.google.com/document/d/1At2Ls5_fhJQ59kDK2DFVhFu3g5mATSXqqV5QrxinasI/preview
- CREN. (2001, novembre 5). Corporation for Research & Educational Networking - Hardware Security Modules. Récupérée le 10 juillet 2023, à partir de <https://archive.wikiwix.com/cache/index2.php?url=http%3A%2F%2Fwww.cren.net%2Fcrenca%2Fonepagers%2Fhsm2.html#federation=archive.wikiwix.com&tab=url>
- D'CRUZ, C. (2020, avril 27). A dive into SoftHSM - Clyde D'Cruz - Medium. Récupérée le 17 juillet 2023, à partir de <https://clydedcruz.medium.com/a-dive-into-softhsm-e4be3e70c7bc>
- DE MEY, L. (s. d.). Le CPU. Récupérée le 21 juillet 2023, à partir de <https://www.courstechinfo.be/Techno/CPU.html>
- DEVICES, B. (2016, août 24). High Assurance Boot (HAB) for dummies. Récupérée le 23 mai 2023, à partir de <https://boundarydevices.com/high-assurance-boot-hab-dummies/>
- DONOVAN, S. (2017). Introduction - A Gentle Introduction to Rust. Récupérée le 23 mai 2023, à partir de <https://stevedonovan.github.io/rust-gentle-intro/readme.html>
- DVORIN, T. (2019, mars 20). Choosing Your Crypto Asset Protection : HSMs, Multi-Sig, and MPC Explained. Récupérée le 1 juin 2023, à partir de <https://tovadvorin.medium.com/choosing-your-crypto-asset-protection-hsms-multi-sig-and-mpc-explained-a375d6829684>
- ENTRUST. (s. d.). Qu'est-ce qu'un HSM basé sur le cloud ? Récupérée le 18 juillet 2023, à partir de <https://www.entrust.com/fr/resources/hsm/faq/what-is-cloud-hsm>
- GEEKSFORGEEKS. (2022, octobre 3). Difference between Compiled and Interpreted Language. Récupérée le 12 mai 2023, à partir de <https://www.geeksforgeeks.org/difference-between-compiled-and-interpreted-language/>

Bibliographie

- GEEKSFORGEEKS. (2023, juin 22). What is memory leak ? How can we avoid ? Récupérée le 22 juillet 2023, à partir de <https://www.geeksforgeeks.org/what-is-memory-leak-how-can-we-avoid/>
- GITHUB. (2023, juillet 21). GitHub - GRPC-ecosystem/GRPC-Gateway : GRPC to JSON Proxy Generator following the GRPC HTTP spec. Récupérée le 22 juillet 2023, à partir de <https://github.com/grpc-ecosystem/grpc-gateway>
- GO. (s. d.). Frequently Asked Questions (FAQ) - The Go Programming Language. Récupérée le 12 mai 2023, à partir de https://go.dev/doc/faq#Is_Go_an_object-oriented_language
- GO. (2023, mai 2). runtime package - runtime - Go Packages. Récupérée le 22 mai 2023, à partir de <https://pkg.go.dev/runtime>
- GOOGLE. (2020, août 27). Using Go at Google. Récupérée le 12 mai 2023, à partir de <https://go.dev/solutions/google/>
- GYORI, L. (2022, mai 31). gRPC vs. REST : Getting Started With the Best API Protocol. Récupérée le 2 juillet 2023, à partir de <https://www.toptal.com/grpc/grpc-vs-rest-api>
- HENKEMANS, D. & LEE, M. (2001). *C++ Programming for the Absolute Beginner 2nd Edition*. Premier Press.
- IONOS. (2020, juin 23). Protobuf : un code structuré avec protocol buffers. Récupérée le 22 juillet 2023, à partir de <https://www.ionos.fr/digitalguide/sites-internet/developpement-web/presentation-de-protocol-buffers/>
- JOURNAL DU NET. (2022, septembre 12). REST vs SOAP : quelles différences ? Récupérée le 22 juillet 2023, à partir de <https://www.journaldunet.fr/web-tech/developpement/1202749-soap-vs-rest-les-principales-differences/>
- KADIONIK, P. (2018, août 29). Le langage C pour l'embarqué. Récupérée le 16 juin 2023, à partir de <https://www.electronique-mixte.fr/wp-content/uploads/2018/08/Cours-Microcontr%C3%B4leur-microprocesseur-49.pdf>
- KEITHP, K. (s. d.). Questions fréquentes (FAQ) - HSM dédié Azure. Récupérée le 1 juin 2023, à partir de <https://learn.microsoft.com/fr-fr/azure/dedicated-hsm/faq>
- KORNELIS, A. F. (s. d.). Bixoft - Pourquoi écrire en Assembleur ? Récupérée le 16 juin 2023, à partir de <https://bixoft.com/francais/why.htm>
- LAYAT, K. (2006). *Modélisation et validation des générateurs aléatoires cryptographiques pour les systèmes embarqués*. Université de Grenoble.

- LONCHAMPS, J. (2017). *Introduction aux systèmes informatiques*. Dunod.
- MEAGHANLEWIS, M. (s. d.). En savoir plus sur les emprunts - Training. Récupérée le 23 mai 2023, à partir de <https://learn.microsoft.com/fr-fr/training/modules/rust-memory-management/2-learn-about-borrowing>
- MINISTÈRE D'ETAT. (2018, juillet 2). Règles et recommandations concernant la gestion des clés cryptographiques utilisées dans l'ensemble des mécanismes cryptographiques. Récupérée le 1 juin 2023, à partir de <https://journaldemonaco.gouv.mc/content/download/165173/3860441/file/JO%208.390%20AM2018-637%20Gestion%20des%20cl%C3%A9s%20cryptographiques.pdf>
- NXP. (s. d.). MCUXpresso SDK API Reference Manual : SNVS : Secure Non-Volatile Storage. Récupérée le 18 mai 2023, à partir de https://mcuxpresso.nxp.com/api_doc/dev/1523/a00257.html
- NXP. (2018, novembre 1). i.MX 6ULZ Applications Processors for Consumer Products. Récupérée le 21 juillet 2023, à partir de <https://www.nxp.com/docs/en/data-sheet/IMX6ULZCEC.pdf>
- OFFICE FÉDÉRAL DE L'INFORMATIQUE ET DE LA TÉLÉCOMMUNICATION OFIT. (s. d.). Signature électronique. Récupérée le 24 juillet 2023, à partir de <https://www.bit.admin.ch/bit/fr/home/themes/elektronische-signatur/elektronische-signatur.html>
- OPENDNSSEC. (s. d.). OpenDNSSEC » SoftHSM. Récupérée le 17 juillet 2023, à partir de <https://www.opendnssec.org/softhsm/>
- PARTISIA BLOCKCHAIN FOUNDATION. (2022, août 11). Multi-Party Computation simplified : Ivan Damgård, Co-founder/Chief Cryptographer-Partisia Blockchain. Récupérée le 15 mai 2023, à partir de <https://www.youtube.com/watch?v=vRVudJADQLk>
- PEŤURA, O. (2019). *True random number generators for cryptography : Design, securing and evaluation. Micro and nanotechnologies/Microelectronics*. Université de Lyon.
- POINTCHEVAL, D. (2021, mars 16). L'agrégation confidentielle de données : le chiffrement fonctionnel, 1ère partie. Récupérée le 21 juillet 2023, à partir de <https://www.lemonde.fr/blog/binaire/2021/03/16/lagregation-confidentielle-de-donnees-1ere-partie/>
- QKZK. (2020, octobre 12). OS : trois concepts clés. Récupérée le 9 juin 2023, à partir de https://qkzk.xyz/docs/nsi/cours_premiere/os/0_concepts_cles/
- RAINONE, A. (2023, avril 30). Statsviz. Récupérée le 9 juin 2023, à partir de <https://github.com/ar/statsviz>

Bibliographie

- RED HAT. (2020, mai 8). Une API REST, qu'est-ce que c'est ? Récupérée le 2 juillet 2023, à partir de <https://www.redhat.com/fr/topics/api/what-is-a-rest-api>
- RITCHIE, D. M. (2021, novembre 2). Chistory. Récupérée le 22 juillet 2023, à partir de <https://archive.wikiwix.com/cache/index2.php?url=http%3A%2F%2Fcm.bell-labs.com%2Fwho%2Fdmr%2Fchist.html#federation=archive.wikiwix.com&tab=url>
- ROSSIER, D. (2014, avril 10). CSE - Conception des systèmes embarqués. Récupérée le 18 mai 2023, à partir de https://reds.heig-vd.ch/share/cours/CSE/CSE_Design_software_v2014.2.0.pdf
- ROSSIER, D. (2017). Programmation assembleur (ASM) - Instructions de traitement. Récupérée le 21 juillet 2023, à partir de https://reds.heig-vd.ch/share/cours/ASM/pdf/ASM_04_Instructions_traitement_v2017.1.6.pdf
- ROUNAK SHARMA, R. (2023, janvier 27). Why is Rust Language Becoming Popular and Should You Learn it ? Récupérée le 23 mai 2023, à partir de <https://emeritus.org/blog/coding-rust-programming-language/>
- RUST FOUNDATION. (2023, juillet 19). Rust foundation. Récupérée le 22 juillet 2023, à partir de <https://foundation.rust-lang.org/>
- SASIDHARAN, D. K. (2019, novembre 7). My first impressions of rust. Récupérée le 22 juillet 2023, à partir de <https://deepu.tech/first-impression-of-rust/>
- SCHNEIER, B. (1996). *Self-Sovereign Identity*. John Wiley et Sons.
- SOUCARROS, M. (2006). *Analyse des générateurs de nombres aléatoires dans des conditions anormales d'utilisation*. Université de Grenoble.
- TANENBAUM, A. (2008). *Modern Operating Systems, 3rd edition*. Pearson Education France.
- TECHSUPPORT, N. (2021, juin 8). How do the on chip true random number generators work ? Récupérée le 23 juillet 2023, à partir de <https://community.nxp.com/t5/Kinetis-Microcontrollers/How-do-the-on-chip-True-Random-Number-Generators-work/td-p/1287513>
- TYLERMSFT. (2021, août 3). Inline Assembler Overview. Récupérée le 21 juillet 2023, à partir de <https://learn.microsoft.com/en-us/cpp/assembler/inline/inline-assembler-overview?view=msvc-170>
- WICKRAMASINGHE, S. (2023, janvier 13). Rust vs Python : Lequel est le meilleur pour votre projet ? Récupérée le 22 juillet 2023, à partir de <https://kinsta.com/fr/blog/rust-vs-python/#avantages-et-inconvnients-de-lutilisation-de-rust>

WITHSECURE. (s. d.-a). USB armory. Récupérée le 14 mai 2023, à partir de <https://www.withsecure.com/en/solutions/innovative-security-hardware/usb-armory>

WITHSECURE. (s. d.-b). USB armory brochure. Récupérée le 14 mai 2023, à partir de <https://www.withsecure.com/content/dam/with-secure/en/resources/withsecure-usb-armory-brochure-en.pdf.coredownload.pdf>

X_EDITOR. (2023, janvier 18). Langage d'assemblage. Récupérée le 15 juin 2023, à partir de <https://tech-lib.fr/langage-dassemblage/>

ZANOTTI, J. (2023, mars 14). Calcul Multiprécision. Récupérée le 22 juillet 2023, à partir de <https://zanotti.univ-tln.fr/ALGO/II/Multiprecision.html#sec:presentation>

Glossaire

Agile La méthodologie agile est une approche itérative et collaborative de gestion de projet. Celle-ci favorise la flexibilité, l'adaptation aux changements et le déploiement continu. Elle met l'accent sur la communication, la collaboration entre les membres de l'équipe et l'ajustement des objectifs en fonction de l'avancement du projet. ii, 5

ANSI-C Le terme "ANSI-C" fait référence à la norme du langage de programmation C. La norme ANSI-C définit les caractéristiques du langage C, y compris la syntaxe, les types de données, les opérations, la bibliothèque standard et d'autres aspects du langage. 20

API Une API ou interface de programmation d'application est un ensemble de composants qui permettent à un logiciel de fournir des services à d'autres logiciels. 38, 40, 41, 57, 58, 69

ARM ARM, Advanced RISC Machine est un type de processeur informatique. 3, 11, 23, 28

Bare metal L'appellation bare metal fait référence à un système informatique utilisé sans un système d'exploitation. 3, 5, 15, 16, 18, 19, 22–26, 59

Blockchain La blockchain est un système dans lequel des transactions sont conservées sous la forme de blocs connectés en chaîne. Chaque bloc contient un ensemble de transactions validées. Après avoir été ajouté à la chaîne, celui-ci devient immuable et ne peut plus être modifié. 8

Boot Le terme boot décrit la marche à suivre pour démarrer un ordinateur. 10

Bootloader Un bootloader est un logiciel qui permet de lancer un ou plusieurs systèmes d'exploitation. 24

Bruit thermique Le bruit thermique est un type de bruit électrique ou électronique généré par l'agitation thermique des particules dans un conducteur électrique. 12

Channel Le Channel est un type de conduit par lequel des informations peuvent être envoyées ou reçues. C'est un moyen de déplacer et d'échanger des données entre différents *Goroutines*. 8

Cloud Le cloud est une solution qui permet d'accéder à des services informatiques à travers Internet, sans avoir à gérer physiquement les infrastructures. 1, 56–58

Compilateur Un compilateur est un programme dont la fonction principale est la traduction d'un langage de programmation en un code lisible par une machine. 19, 28, 29

Container Un container est un environnement permettant d'isoler une application et ses dépendances d'un OS. 8

Effet photoélectrique L'effet photoélectrique décrit l'émission d'électrons par un matériau sous l'action de la lumière. 12

eMMC Le terme eMMC veut dire embedded Multi Media Card. Une mémoire de ce type est un moyen de stocker les données internes sur l'équipement. La mémoire eMMC est fixée sur la carte mère et ne peut pas être remplacée. 9

- Framework** Le mot framework peut être traduit comme une "infrastructure de développement" ou "cadre de développement". Celui-ci propose des bibliothèques de fonctionnalités pour aider le développement de programmes et d'applications. Il peut imposer un cadre de travail sur l'architecture du logiciel. ii, 2–5, 10, 11, 15, 22–26
- Garbage collector** Le garbage collector est un système exécuté en arrière-plan pour gérer la mémoire automatiquement. 7, 21, 22
- Git** Le Git est un logiciel de gestion qui permet de sauvegarder, suivre les versions, l'évolution du code et de stocker celui-ci d'une manière organisée. Ce type de logiciel permet de revenir en arrière en cas de besoin. 5
- Go Operating System** GOOS représente le système d'exploitation sur lequel le langage Go peut être compilé. 23
- Goroutines** Selon le site de développement Go, un goroutine est un fil d'exécution ou thread géré par l'environnement Go. En d'autres termes, les goroutines sont des méthodes qui s'exécutent en concurrence avec d'autres fonctions. 8, 25
- HAB** Le High Assurance Boot est un processus de démarrage sécurisé garantissant l'intégrité du système d'exploitation. Celui-ci empêche les attaques de logiciels malveillants dès le démarrage de l'ordinateur. 10
- IoT** L'IoT, appelé aussi l'Internet des Objets, est un réseau d'objets connectés entre eux et avec d'autres systèmes. Ils ont la possibilité d'échanger des données dans leur réseau. 38
- Multi-factor authentication** Le multi-factor authentication ou authentification à plusieurs facteurs est un mécanisme de sécurité demandant à l'utilisateur de fournir plusieurs éléments de vérification (mot de passe ou empreinte digitale) pour accéder à un compte ou à un système. 10
- OpenApi** La documentation OpenAPI Swagger UI est une interface conviviale et interactive qui permet de visualiser et de tester facilement les méthodes d'une API. 40, 42, 51–54
- Opensource** Le terme *opensource* promouvait l'utilisation de logiciels de manière publique en rendant son code source accessible et modifiable. Celui-ci est régi par une licence permettant de visualiser, modifier et améliorer le logiciel ou le code librement sous certaines conditions. 7, 8, 21, 22, 38
- Package** Un *package* est un dossier qui regroupe des classes, des modules et des ressources reliés entre eux. 3, 22–24
- PKCS** Le PKCS ou Public Key Cryptography Standards est un ensemble de normes définissant les protocoles et les formats utilisés pour la cryptographie, notamment pour les certificats, le chiffrement et les signatures électroniques. 56, 57
- Privilege escalation** Le privilege escalation est une technique qui permet à un utilisateur d'acquérir des droits et des privilèges supérieurs à ceux qui lui ont été initialement attribués. 3

Protocol buffers Le *protocol buffers* est un format de sérialisation. Celui-ci est utilisé pour structurer et échanger des données entre des systèmes informatiques. 38

Registre Un registre est un emplacement de stockage interne à un processeur. Il s'agit d'une zone de stockage très rapide et de petite taille qui peut contenir des données, des adresses mémoire ou des instructions. 16

REST REST ou Representational State Transfer est un style d'architecture logicielle. Il définit un ensemble de contraintes utilisées pour créer des services web et effectuer toute sorte d'opérations comme *GET*, *PUSH*, *PULL* et *DELETE*. 38, 41

Reverse-proxy Un reverse-proxy est un serveur qui agit comme intermédiaire entre les clients et les serveurs en recevant les requêtes et en les transmettant aux serveurs cibles. Celui-ci permet de protéger, d'équilibrer le volume des requêtes et de gérer de manière centralisée les requêtes entrantes. 41

Risc-v RISC-V est une architecture de jeu d'instruction utilisé dans certains micro-processeurs. Son utilisation est libre pour l'enseignement, l'industrie et la recherche. 3, 23

Secure boot Le secure boot est un bootloader dont l'authentification requiert une clé publique embarquée directement sur le périphérique. 9

Smart card Une smart card, en français appelée carte à puce, est une carte munie d'une puce intégrée qui sert de jeton de sécurité. Nous l'utilisons pour stocker des informations comme un code PIN ou des numéros d'accès par exemple. 10

SNVS Le SNVS est un module créé pour garantir l'intégrité et la sécurité des données sauvegardées dans un espace de stockage. 10

SoftHSM Les SoftHSM ont été développés dans le but de se familiariser avec les spécifications PKCS#11 et de simuler l'utilisation d'un véritable *HSM*. 56

stack En programmation, la stack fait référence à un emplacement de la mémoire pour le stockage temporaire de données lorsque un thread est exécuté. 8

Stateless Dans le contexte d'une API, stateless décrit un type d'échange de messages où les données du client ne sont pas enregistrées pendant la communication client-serveur. 38

Swagger Swagger est un ensemble d'outils open source permettant de concevoir, de documenter et de tester des API de manière interactive. 41, 43

Threads Le terme thread désigne une unité d'activité exécutée sur un processeur. 8, 21, 25

True Random Number Generator Les *TRNG* sont des générateurs de vrais nombres aléatoires totalement imprévisibles et ne suivent aucun modèle ou règle prévisible. C'est-à-dire que il n'est pas possible de deviner le prochain nombre aléatoire généré. Leur aléa est généré grâce à l'utilisation de phénomènes physiques. v, 9, 12, 13, 36, 40, 41, 54, 55

Unix Le système Unix est un système d'exploitation multi-utilisateurs et multi-tâches. Ce système est puissant et flexible, largement utilisé dans les environnements informatiques. 19, 20, 28

Versioning Le versionning est un moyen de contrôler et apporter des changements à un programme à travers le temps. Celui-ci permet de suivre et de gérer les changements, les améliorations et les compatibilités entre les différentes itérations. 38

Zero-day La vulnérabilité zero-day est une faille de sécurité informatique trouvée par des personnes avant que celle-ci ne soit découverte ou corrigée par des développeurs de la solution. 3

Informations sur ce travail

Informations de contact

Auteur : Thomas Cheseaux

HES-SO Valais-Wallis

E-mail : *thomas.cheseaux@students.hevs.ch*

Déclaration sur l'honneur

Je déclare, par ce document, que j'ai effectué le travail de bachelor ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de bachelor, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail.

Lieu, date : _____

Signature : _____