Options de fabrication et budget:

Cadrage de l'approvisionnement et de l'assemblage (BOM / équipements)

Dans le cadre du projet **Tillitis Key1 (TKey)**, il nous a été demandé de fournir une **liste de matériel** (BOM des composants + équipements/consommables nécessaires). Pour établir une liste réaliste, nous avons d'abord analysé plusieurs stratégies d'approvisionnement et d'assemblage, car le choix de l'approche conditionne directement la nature des achats (composants seuls, cartes nues, cartes assemblées) ainsi que les moyens techniques requis.

Analyse des options envisagées

Une première option consistait à **commander chaque composant séparément** (BOM) et à assembler la clé « from scratch ». Cette approche est techniquement exigeante car la TKey intègre des composants SMD à pas fin (notamment un **FPGA Lattice iCE40 en boîtier QFN**, une **Flash SPI**, un **connecteur USB-C**, ainsi que de nombreux petits passifs). L'assemblage de ce type de carte requiert des outils adaptés (station de soudage fine, pâte/flux, idéalement stencil, loupe/microscope, et souvent air chaud ou four à refusion) et une certaine expérience en **assemblage SMD / rework**, sinon le risque de défauts (ponts de soudure, mauvais alignement du FPGA, composants endommagés) devient important, avec un impact sur coûts et délais.

Une seconde option consistait à **reconstituer le schéma et le PCB sous KiCad**, puis à faire fabriquer le PCB auprès d'un fabricant, éventuellement avec une prestation d'assemblage (PCB assemblé). Cette approche peut améliorer la reproductibilité mais introduit des contraintes supplémentaires : coût potentiellement plus élevé (PCB + assemblage + stencil + frais de mise en production), délais liés aux validations (DRC/ERC) et à d'éventuelles itérations, et besoin d'informations très précises sur l'implantation et les contraintes d'assemblage, en particulier sur les zones sensibles (FPGA, USB-C).

Une troisième option consistait à **commander des TKey déjà assemblées**, afin de disposer rapidement d'une plateforme fonctionnelle pour les tests, la prise en main et les travaux d'analyse/développement (firmware, modèle de confiance, durcissement, scénarios d'attaque). Cette solution limite les risques liés à la fabrication et sécurise l'avancement du projet sur la partie technique.

Décision retenue : approche hybride (options 1 et 3)

Après échange et validation, l'approche retenue est une **hybridation des options 1 et 3**. D'une part, l'achat d'une ou plusieurs clés **déjà assemblées** garantit une base opérationnelle immédiate, permettant d'avancer sans dépendre des aléas de fabrication (délais, défauts d'assemblage) et de démarrer les tests et développements dès le début. D'autre part, afin de conserver une dimension matérielle forte et de monter en compétences, il est prévu de **commander les composants pour environ une dizaine d'unités** et de réaliser l'assemblage au sein du **lab**, avec accompagnement, en s'appuyant sur les moyens disponibles (outillage, instrumentation, assistance au SMD).

Estimation budgétaire : choix final "hybride options 1 et 3"

Suite au cadrage projet, l'option retenue consiste à **hybrider** :

- **Option 1** : achat des composants et **assemblage au lab** (avec accompagnement SMD) sur un **lot cible de 10 unités** ;
- **Option 3** : achat d'au moins une **TKey assemblée** ("tel quel") pour disposer rapidement d'une plateforme fonctionnelle et réduire le risque planning.

Clarification essentielle sur l'option 1 : composants seuls = impossible sans PCB

Dans le cas de l'option 1, **commander uniquement les composants ne suffit pas** : il faut obligatoirement **des PCB nus** sur lesquels souder. Le projet open-hardware fournit précisément :

- les **empreintes** (footprints),
- les **emplacements** (placement) et la conception routée, ce qui permet de générer les fichiers de fabrication (Gerbers) et les fichiers d'assemblage (Pick&Place). Le contour de carte du design correspond à une petite carte "clé USB" (≈ **27,7 mm × 13,9 mm** d'après le fichier PCB).

Hypothèses de chiffrage (option 1 - 10 unités)

Les estimations présentées ci-dessous reposent sur la **BOM générée depuis le design KiCad**. Les prix indiqués (lorsqu'ils sont chiffrés) correspondent à des **prix publics observés** avec un palier proche de **10 pièces**, hors TVA et hors frais de livraison. Les frais logistiques peuvent représenter une part significative du budget si l'approvisionnement est réparti sur plusieurs canaux.

Les composants listés dans le tableau "référencés" sont ceux dont la **référence est explicitement indiquée** dans la BOM (MPN) : FPGA, Flash, régulateurs, ESD, ferrite, etc. Les composants définis uniquement par **valeur/empreinte** (résistances, condensateurs, LEDs génériques) sont traités séparément car leur coût dépend fortement des MOQ et du fournisseur retenu.

BOM “composants référencés” : coût unitaire et coût pour 10 clés

Le design intègre notamment un **FPGA Lattice iCE40UP5K (U6)**, une **Flash Winbond (U8)**, un **contrôleur WCH CH552E (U3)**, un **connecteur USB-C (P1)**, une **protection ESD (U5)** et trois régulateurs (U1, U2, U9).
Dans la BOM (généré depuis KiCad en partant du dépôt github), plusieurs éléments sont explicitement exclus de l’achat : les **points de test** (TPx) et les **fiducials** (FIDx). Par ailleurs, **U11** (spring finger 1674954-1) est marqué “Excluded from board” et n’est donc pas considéré dans l’assemblage de référence.

Tableau de chiffrage (principaux composants)

Réf	Composant (MPN) et leur lien d’achat	Qté / clé	Prix unitaire (palier)	Devise	Coût pour 10
U6	iCE40UP5K-SG48ITR	1	$6.21 (10+)	USD	$62.10
U8	W25Q80DVUXIE	1	€0.6977 (10+)	EUR	€6.98
U5	USBLC6-2SC6	1	€0.449 (≈10)	EUR	€4.49
U1	MCP1824T-2502EOT (2.5V)	1	€0.482 (10+)	EUR	€4.82
U9	NCP752BSN33T1G (3.3V)	1	€0.245 (10+)	EUR	€2.45
FB3	BLM18KG300TN1D	1	€0.052 (10+)	EUR	€0.52
U3	CH552E	1	$0.3713 (10+)	USD	$3.71
U10	PT2043AT6	1	$0.0417 (10+)	USD	$0.42
P1	USB-C (U261-241N-4BS60)	1	$0.2563 (10+)	USD	$2.56

Sous-total (hors U2) – 10 unités :

- ≈ €19.26 (partie EUR)
- ≈ $69.37 (partie USD)

Point à compléter : U2 (MIC5258-1.2YM5)
La BOM contient U2 = MIC5258-1.2YM5 (LDO 1.2 V). Le coût dépend fortement de la disponibilité/stock et du canal d'achat.

Passifs "génériques" (sans MPN) : quantité par clé et impact coût

Le design comprend des passifs définis principalement par valeur + empreinte. Ces composants sont nombreux mais peu coûteux à l'unité ; en pratique, ce sont les MOQ (bandes 50/100/1000) et les frais logistiques qui structurent le budget.

Condensateurs (hors C8 exclu BOM)

D'après la BOM :

- **10 µF** : 7 pièces (C1, C2, C3, C4, C5, C6, C25)
- **0.10 µF** : 11 pièces (C7, C13, C16, C17, C18, C19, C20, C21, C22, C23, C26)
- **1 pF** : 1 pièce (C27)

Soit **19 condensateurs par clé**, donc **190 condensateurs** pour 10 unités.

Résistances

D'après la BOM :

- **10 kΩ** : 9 pièces
- **1 kΩ** : 4 pièces
- **5.1 kΩ** : 1 pièce
- **2 kΩ** : 1 pièce

Soit **15 résistances par clé**, donc **150 résistances** pour 10 unités.

LEDs "génériques"

- **D1 :** **LED bleue 0402** : 1 par clé
- **D3 :** **LED_ARGB** : 1 par clé

Estimation pratique : pour un lot de 10 cartes, un budget "passifs génériques + LEDs" de l'ordre de **quelques euros à ~10–15 €** est généralement réaliste, car l'achat est souvent contraint par les MOQ et non par le coût matière pur.

Coûts additionnels incontournables : PCB + stencil (option 1)

PCB nus (10 unités)

Le PCB étant de petite taille, il peut entrer dans des offres "prototype" à faible coût unitaire. Le coût final dépend toutefois des paramètres de fabrication (nombre de couches, finition, épaisseur, délai) et surtout du transport.

Stencil (fortement recommandé pour QFN + 0402)

Un stencil est fortement recommandé pour fiabiliser la dépose de pâte, en particulier à cause des composants QFN et des passifs très petits (0402). Le stencil devient un facteur de réduction des défauts d'assemblage (ponts de soudure, manque de pâte, reprises).

Total option 1 (10 unités) : synthèse "budget technique"

En combinant les postes :

- **Composants référencés (MPN explicites, hors U2)** : **≈ €19.26 + ≈ $69.37**
- **U2 (MIC5258-1.2YM5)** : **prix à confirmer** (et éventuelle substitution à documenter si nécessaire)
- **Passifs génériques (R/C/LED)** : **≈ 10–15 €** (selon MOQ/canal)
- **PCB + stencil + livraison** : dépend fortement du fabricant et du mode d'expédition

L'analyse montre que le budget de l'option 1 est dominé par le **FPGA** et par la **logistique d'approvisionnement** (MOQ, frais de port, disponibilité). Le coût des passifs est marginal en valeur absolue, mais leur achat peut nécessiter des conditionnements minimaux.

Les délais varient principalement selon :

- la provenance (distributeur européen vs fournisseur hors UE),
- la disponibilité en stock,
- le mode de livraison et les éventuelles formalités douanières.

En pratique, il est courant d'observer des délais de **quelques jours** via des distributeurs européens (si stock disponible), contre **1 à 3 semaines** lorsque des composants/PCB proviennent d'une chaîne de fabrication et d'expédition internationale.

<u>Option 3 : coût d'une clé assemblée</u>

Pour disposer d'un matériel **non provisionné** et conserver la maîtrise complète du **provisioning** (programmation du *bitstream* FPGA et initialisation des identifiants/secrets tels que **UDI/UDS**), nous privilégions l'achat d'une **TKey Unlocked**.

La boutique Tillitis indique un prix public de **880 SEK** (taxes incluses), hors frais de livraison. (Tillitis Shop)

Contrairement à une TKey “end-user”, une TKey Unlocked nécessite un outillage de programmation : la boutique précise qu’un **TKey Programmer Board** est requis pour programmer une TKey Unlocked. (Tillitis Shop) Le TKey Programmer Board est affiché à **500 SEK** (taxes incluses). (Tillitis Shop)

Contenu et accessoires à prévoir

- **TKey Unlocked** : le package comprend **1× TKey Unlocked** et **2× coques plastiques** (donc pas de coût additionnel pour le boîtier). (Tillitis Shop)
- **TKey Programmer Board (TP1)** : nécessaire pour programmer la TKey Unlocked ; il est basé sur une plateforme **Raspberry Pi Pico** et utilise un connecteur **USB micro** (donc il faut un câble micro-USB pour le relier au PC si le lab n’en a pas déjà). (Tillitis Shop)
- **Frais de livraison** : la boutique indique que la livraison est **calculée au moment du checkout** (donc à ajouter au total ci-dessous). (Tillitis Shop)

Coûts (hors livraison)

Les conversions EUR ci-dessous utilisent le taux de référence BCE : **1 EUR = 10,904 SEK** (19 décembre 2025). (European Central Bank)

Élement nécessaire	**Prix (SEK)**	**Équivalent (EUR)**
TKey Unlocked (taxes incluses) (Tillitis Shop)	880	~80,70 €
TKey Programmer Board / TP1 (taxes incluses) (Tillitis Shop)	500	~45,85 €
Total “kit minimal utilisable” (hors livraison)	**1380**	**~126,56 €**

-> Remarque importante (lecture budget) : le **TKey Programmer Board** est un coût “d’amorçage” : si l’on achète plusieurs TKey Unlocked, **un seul programmeur** peut suffire pour l’ensemble (donc le coût marginal par clé se rapproche de **880 SEK** + livraison, une fois le programmeur acquis). (Tillitis Shop)

SBOM

SBOM “générée de base” : sbom_source.json

Une première SBOM a été générée à partir du dépôt tillitis-key1 au format CycloneDX JSON à l'aide de l'outil Syft. Cette SBOM initiale (sbom_source.json) reflète l'intégralité de la composition logicielle détectable dans le répertoire scanné, incluant notamment certains éléments non directement liés au produit (ex. composants CI/CD et fichiers/outils présents localement).
Dans cette version, des composants issus des workflows GitHub Actions peuvent apparaître (par exemple actions/checkout, actions/cache), car Syft catalogue également les dépendances de la chaîne CI via les fichiers de workflows.
De plus, si l'outil Syft est présent dans le répertoire scanné (ex. ./bin/syft), ses propres dépendances peuvent être incluses dans la SBOM initiale, ce qui "pollue" l'interprétation lorsqu'on cherche à décrire la supply chain du projet lui-même.

```
└─$ ./bin/syft dir:. -o cyclonedx-json > sbom_source.json
 ✔ Indexed file system                                                                  .
 ✔ Cataloged contents                        cdb4ee2aea69cc6a83331bbe96dc2caa9a299d21329efb0336fc02a82e1839a8
   ├── ✔ Packages                        [283 packages]
   ├── ✔ Executables                     [6 executables]
   ├── ✔ File metadata                   [4 locations]
   └── ✔ File digests                    [4 files]
[0005]  WARN no explicit name and version provided for directory source, deriving artifact ID from the given path (

┌──(eliza9123㉿kali)-[~/Documents/tillitis-key1]
└─$ jq '.components | length' sbom_source.json

287
```

SBOM "clean" : sbom_source_clean.json

Afin d'obtenir une SBOM plus représentative du projet (et éviter d'inclure des artefacts externes ou locaux), une seconde SBOM a été générée en excluant explicitement le dossier contenant l'outil Syft (./bin/**) et les métadonnées Git (./.git/**). La SBOM obtenue (sbom_source_clean.json) est considérée comme la référence pour l'analyse, car elle réduit le bruit et améliore la traçabilité des composants réellement liés au dépôt étudié.

```
$ ./bin/syft dir:. -o cyclonedx-json \
--exclude ./bin/** \
--exclude ./.git/** \
> sbom_source_clean.json
```

La comparaison entre la SBOM brute et la SBOM nettoyée met en évidence l'importance du périmètre de scan. La SBOM brute capture tout ce qui est présent dans le répertoire scanné (y compris des outils ajoutés localement), alors que la SBOM nettoyée se concentre sur les composants pertinents pour le dépôt. La version nettoyée est donc utilisée comme base d'interprétation dans le rapport.
Enfin, il est important de noter que cette SBOM "source-level" décrit principalement la supply chain de développement (outils, CI/CD, dépendances déclaratives), ce qui est cohérent avec un contexte embarqué bare-metal : les artefacts firmware (.hex/.bin) ne contiennent pas de métadonnées de packaging permettant d'identifier automatiquement des bibliothèques au runtime.

Version	Fichier	Objectif	Limites
Clean	sbom_source.json	visibilité globale de tout le répertoire	inclut du bruit (outils locaux, CI, etc.)
Clean	sbom_source_clean.json	analyse “projet” plus fidèle	reste une SBOM “source-level”

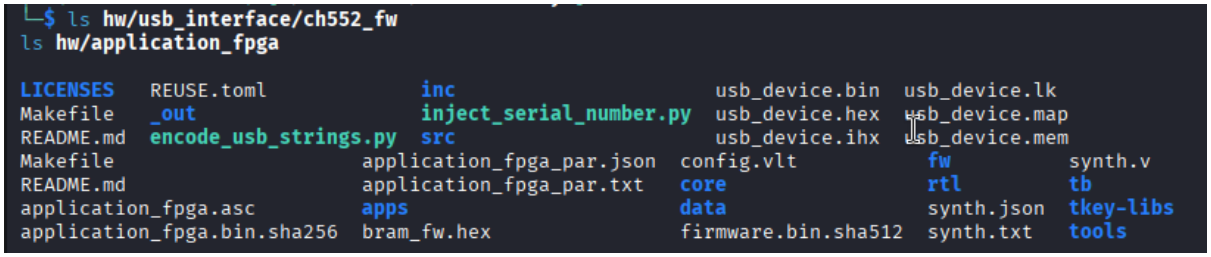
Pourquoi les SBOM automatiques ne suffisent pas

Les SBOM générées automatiquement (brute et nettoyée) décrivent principalement la supply chain logicielle de développement du projet (outils, bibliothèques tierces, CI/CD). Cependant, la Tillitis Key étant un système embarqué bare-metal, le firmware final est fourni sous forme de binaire (.hex/.bin) sans métadonnées de packaging. Il est donc impossible d’identifier automatiquement les composants logiciels réellement exécutés à partir du binaire seul.

Afin de décrire fidèlement la composition logicielle réelle du produit, une approche complémentaire a été adoptée : l’établissement d’une Product Bill of Materials (PBOM). La PBOM repose non pas sur l’analyse automatique du binaire, mais sur l’étude du processus de build, en reliant chaque artefact exécutable à son code source, sa commande de compilation et son empreinte cryptographique.

L’analyse du dépôt permet d’identifier deux composants logiciels exécutés :

- le firmware CH552 assurant l’interface USB,
- le bitstream FPGA implémentant la logique applicative.



Le firmware CH552 est construit à partir du code C situé dans hw/usb_interface/ch552_fw.
Le bitstream FPGA est généré à partir des sources RTL Verilog situées dans hw/application_fpga/rtl et hw/application_fpga/core.

Tableau PROM

Component	Type	Source path	Build command	Artifact	SHA-256
CH552	MCU firm	hw/usb_interface/ch552_fw	make -C hw/usb_int	usb_device.hex	270fd8dd2a466ae00ac31ef5539ccf79def227ed347529564721efde8b8ea9a6

firm ware	mw are		erface/ch55 2_fw		

La combinaison des SBOM automatiques (brute et nettoyée) et de la PBOM manuelle permet d'obtenir une vision réaliste de la composition logicielle de la Tillitis Key. Les SBOM décrivent la supply chain de développement, tandis que la PBOM garantit la traçabilité des composants réellement exécutés sur le produit.