Armadillo-IoT ゲートウェイ G4
セキュリティガイド

Version 1.3.2
2023/10/27

株式会社アットマークテクノ [https://www.atmark-techno.com]

Armadillo サイト [https://armadillo.atmark-techno.com]

Armadillo-IoT ゲートウェイ G4 セキュリティガイド

株式会社アットマークテクノ

Version 1.3.2
2023/10/27

目次

1. セキュリティ機能の概要 8
2. 本書について 9
2.1. 本書の目的 9
2.2. 本書の構成 9
2.3. 表記について 10
2.3.1. フォント 10
2.3.2. コマンド入力例 10
2.3.3. アイコン 10
3. セキュリティの考え方 12
3.1. セキュリティの費用対効果 12
3.2. Armadiilo-IoT ゲートウェイ G4 のセキュリティ領域 12
3.3. Armadillo-IoT ゲートウェイ G4 で実現できるセキュリティ機能と効果 14
3.4. モデルユースケース 14
3.4.1. ネットワーク機器 14
3.4.2. クラウドサービスへの接続機器 15
3.4.3. 決済処理が可能な機器 15
4. 鍵の保護 17
4.1. EdgeLock SE050 を利用したキーストレージ 17
4.1.1. EdgeLock SE05x Plug & Trust Middleware 17
4.1.2. EdgeLock SE050 を有効にする 18
4.1.3. Plug & Trust Middleware のインストール 18
4.1.4. Plug & Trust Middleware を活用する 20
4.1.4.1. EdgeLock SE050 の保存された鍵を利用する 20
4.1.4.2. ユーザーが生成した鍵を保存して利用する 21
4.1.5. 補足説明 23
4.1.5.1. 事前書き込みされた鍵と X.509 証明書 23
4.1.5.2. リファレンスキー 23
4.1.5.3. se05x-tools 23
5. セキュアブート 25
5.1. セキュアブートとチェーンオブトラスト 25
5.2. HAB とは 26
5.3. 署名環境を構築する 26
5.3.1. 署名環境について 27
5.3.2. ATDE を準備する 27
5.3.3. 署名ツール (CST) を準備する 27
5.3.4. アップデートファイル作成ツール (mkswu) を準備する 28
5.3.5. Armadillo Base OS イメージの取得 29
5.3.6. 署名環境のディレクトリツリー構成 30
5.3.7. 署名鍵を生成する 30
5.3.7.1. 署名鍵の生成 31
5.3.7.2. 生成された署名鍵の確認 31
5.3.7.3. 鍵のハッシュ値の確認 33
5.4. セキュアブート機能の選択 33
5.5. セキュアブートのセットアップ 34
5.5.1. uboot-imx のセキュアブートの実装仕様 34
5.5.2. セットアップの流れ 34
5.5.3. 署名済みブートローダーの作成 35
5.5.4. 署名済み Linux カーネルイメージの作成 35
5.5.5. ルートファイルシステムのビルド 36
5.5.6. セキュアブートセットアップ用 SD boot ディスクの作成 36

5.6. 暗号化セキュアブートのセットアップ 37
5.6.1. uboot-imx の暗号化セキュアブートの実装仕様 37
5.6.2. セットアップの流れ 38
5.6.3. 署名済みの暗号化ブートローダーの作成 38
5.6.4. 暗号化 Linux カーネルイメージの生成 39
5.6.5. 暗号化セキュアブート対応 swu の作成 40
5.6.6. ルートファイルシステムのビルド 41
5.6.7. セキュアブートセットアップ用 SD boot ディスクの作成 41
5.7. ストレージの暗号化のセットアップ 42
5.7.1. 実装仕様の概要 42
5.7.2. セットアップの流れ 43
5.7.3. 署名済みブートローダーの作成 43
5.7.4. 署名済み Linux カーネルイメージの作成 44
5.7.5. セキュアブート対応 swu の作成 45
5.7.6. ルートファイルシステムのビルド 46
5.7.7. セキュアブートセットアップ用 SD boot ディスクの作成 47
5.8. Armadillo-IoT ゲートウェイ G4 上の作業 48
5.8.1. Armadillo-IoT ゲートウェイ G4 の準備 48
5.8.2. SRK ハッシュの書き込み 48
5.8.3. close 処理 49
5.8.4. ファームウェアの書き込みをはじめる 50
5.8.5. 作業が中断した場合 50
5.9. 動作確認 51
5.9.1. ストレージ暗号化の確認 51
5.9.2. セキュアブートの確認 51
5.10. セットアップ完了後のアップデートの運用について 53
5.11. 補足情報 53
5.11.1. secureboot.conf の設定方法 53
5.11.2. 署名済みイメージと展開先 54
5.11.3. セキュアブートのフロー 56
5.11.4. ビルドのプロセスフロー 59
5.11.5. ファームウェアアップデートのフロー 60
5.11.6. SRK の無効化と切り替え 60
5.11.6.1. SRK の無効化 (revocation) 60
5.11.6.2. SRK の切り替え 61
6. ソフトウェア実行環境の保護 63
6.1. OP-TEE 63
6.1.1. Arm TrustZone と TEE の活用 63
6.1.2. OP-TEE とは 63
6.2. OP-TEE の構成 64
6.2.1. Armadillo Base OS への組み込み 65
6.3. OP-TEE を利用する前に 65
6.3.1. 鍵の更新 65
6.4. CAAM を活用した TEE を構築する 66
6.4.1. ビルドの流れ 66
6.4.2. ビルド環境を構築する 67
6.4.3. ブートローダーを再ビルドする 67
6.4.4. imx-optee-client をビルドする 68
6.4.5. アプリケーションをビルドする 68
6.4.6. imx-optee-test をビルドする 69
6.4.7. ビルド結果の確認と結果の収集 70
6.4.8. OP-TEE を組み込む 73
6.5. パフォーマンスを測定する 78

6.6. Edgelogck SE050 を活用した TEE を構築する 78
6.6.1. OP-TEE 向け plug-and-trust ライブラリ 79
6.6.2. ビルドの流れ 79
6.6.3. ビルド環境を構築する 80
6.6.4. OP-TEE 向け plug-and-trust をビルドする 80
6.6.5. imx-optee-os のコンフィグの修正 81
6.6.6. uboot-imx の修正 82
6.6.7. imx-optee-os の imx-i2c ドライバの修正 83
6.6.8. ビルドとターゲットボードへの組み込み 84
6.6.9. xtest の制限 84
6.7. imx-optee-os 技術情報 85
6.7.1. ソフトウェア全体像 85
6.7.2. フロー 87
6.7.3. メモリマップ 88
7. 市場出荷に向けてデバッグ機能を閉じる 90
7.1. JTAG を無効化する 90
7.2. SD boot を無効化する 91
7.3. BOOT_CFG_LOCK について 93
7.4. u-boot プロンプトを無効にする 94
8. 悪意のある攻撃者への対策 96
8.1. KASLR 96
8.1.1. KASLR の有効化 96
8.1.2. KASLR の有効化確認 97

Armadillo-IoT ゲートウェイ G4 セキュリティガイド

図目次

3.1. Armadillo-IoT ゲートウェイ G4 のセキュリティ領域 12
3.2. Armadillo-IoT ゲートウェイ G4 の実行環境の領域 13
3.3. セキュリティ技術のカバーする範囲 14
4.1. Plug & Trust Middleware の周辺のソフトウェアスタック 17
5.1. チェーンオブトラスト 25
5.2. ブートローダーの署名済みイメージ 55
5.3. 暗号化ブートローダーの署名済みイメージ 55
5.4. Linux カーネルの署名済みイメージ 56
5.5. ストレージ暗号化に対応した Linux カーネルの署名済みイメージ 56
5.6. SPL セキュアブートのフロー 57
5.7. u-boot セキュアブートのフロー 58
5.8. セキュアブートのビルドフロー 59
5.9. ファームウェアアップデートのフロー 60
6.1. Armadillo Base OS と OP-TEE の担当範囲 65
6.2. SE050 向け OP-TEE の起動シーケンス図 82
6.3. OPTEE のシステム図 86
6.4. i.MX 8M Plus の物理メモリマップ 88

表目次

2.1. 使用しているフォント 10
2.2. 表示プロンプトと実行環境の関係 10
4.1. Plug & Trust Middle のパッケージ 18
4.2. Plug & Trust Middleware のサポート 18
4.3. SE050 ena pin 18
5.1. 署名環境 27
5.2. セキュアブート用の鍵と証明書 30
5.3. セキュアブートの派生機能 33
5.4. セキュアブート用の鍵の種類 54
6.1. OP-TEE メモリマップ 88

1. セキュリティ機能の概要

Armadillo-IoT ゲートウェイ G4 には用途の異なる 2 つの暗号処理アクセラレータを搭載します。Armadillo Base OS と組み合わせることで、豊富なセキュリティ機能を素早く簡単に実現することが可能です。

NXP Semiconductors (以下、NXP) EdgeLock SE050

- クラウドサービスの接続認証にも利用可能な事前書き込みされたチップ固有鍵や証明書
- 外部に秘密鍵が露出しないキーストレージ
- OpenSSL からの利用に対応 (OpenSSL engine) [1]
- EdgeLock SE05x Plug & Trust Middleware のビルド済みパッケージの提供 [2]
- 鍵の読み書きコマンドのビルド済みパッケージの提供 [3]

NXP i.MX 8M plus 内蔵の暗号処理アクセラレータ CAAM

- Cryptographic Acceleration and Assurance Module (以下、CAAM) による高速暗号処理
- セキュアブート (High Assurance Boot, 以下、HAB)
- ブートローダーの暗号化
- ストレージの暗号化
- セキュアブート、ブートローダーの暗号化、ストレージの暗号化に対応したビルドスクリプトの提供
- ファイルの暗号化・復号コマンドのビルド済みパッケージの提供 [4]

Arm TrustZone

- ソフトウェア実行環境の隔離 (OP-TEE) [5]

[1] RSA, EC, RNG のみ。OpenSSL 1.1 に対応。3.0 は非対応となります
[2] NXP からリリースされるライブラリのビルド済みパッケージを Alpine Linux と Debian Linux 向けにリリースしています
[3] アットマークテクノが開発したコマンドを Alpine Linux と Debian Linux 向けにリリースしています
[4] Black key blob を利用した暗号処理を Alpine Linux 向けにリリースしています
[5] 工場出荷状態や製品アップデートに含まれる Armadillo Base OS では OP-TEE は機能しません。詳しくは 「6.3. OP-TEE を利用する前に」 を参照してください

2. 本書について

2.1. 本書の目的

本ガイドには２つ目的があります。

- Armadillo-IoT ゲートウェイ G4 と Armadillo Base OS のもつ機能を用いて、利用者の情報資産に適したセキュリティ対策をみつける
- Armadillo Base OS とツール群を用いて、Armadillo-IoT ゲートウェイ G4 上に構築したシステムのセキュリティを確保していくための具体的な手順を説明する

2.2. 本書の構成

「3. セキュリティの考え方」	・「3.1. セキュリティの費用対効果」 では、セキュリティ機能を導入する導入しないに関わらず、立ち返って製品開発にもたらすセキュリティの影響について考えます ・「3.2. Armadiilo-IoT ゲートウェイ G4 のセキュリティ領域」では、攻撃の入り口や攻撃者の特徴、ハードウェア機能から Armadillo-IoT ゲートウェイ G4 をいくつかのセキュリティの領域として分類して説明します ・「3.3. Armadillo-IoT ゲートウェイ G4 で実現できるセキュリティ機能と効果」では、セキュリティ機能の導入を考えるときに、Armadillo-IoT ゲートウェイ G4 と Armadillo Base OS の組み合わせで実現できる、利用者に適したセキュリティ機能をみつけます ・「3.4. モデルユースケース」 では、いくつかのモデルユースケースとそのユースケースで必要と考えられるセキュリティ機能を紹介します
「4. 鍵の保護」	・「4.1. EdgeLock SE050 を利用したキーストレージ」 では、Armadillo-IoT ゲートウェイ G4 に搭載されている NXP 製 EdgeLock SE050 をキーストレージとして利用する方法について説明します
「5. セキュアブート」	・「5.5. セキュアブートのセットアップ」では、CAAM を用いて正規ソフトウェアが起動することを保証するための機能であるセキュアブートを有効にする方法について説明します ・「5.6. 暗号化セキュアブートのセットアップ」では、暗号化されたブートローダーでセキュアブートする方法を説明します ・「5.7. ストレージの暗号化のセットアップ」では、ストレージを暗号化する方法を説明します
「6. ソフトウェア実行環境の保護」	・「6.4. CAAM を活用した TEE を構築する」 では、Arm TrustZone テクノロジーを利用した TEE 実装である OP-TEE

	の構築方法について説明します。CAAM を利用した高速な暗号処理が可能になります
	・「6.6. Edgelogck SE050 を活用した TEE を構築する」 では、EdgeLock SE050 を利用した OP-TEE の構築方法について説明します。この節では EdgeLock SE050 のキーストレージを活用した暗号処理が可能になります
「7. 市場出荷に向けてデバッグ機能を閉じる」	・製品開発を終えて出荷に向けて実施すべき作業について説明します

2.3. 表記について

2.3.1. フォント

本書では以下のような意味でフォントを使いわけています。

表 2.1 使用しているフォント

フォント例	説明
本文中のフォント	本文
[PC ~]$ ls	プロンプトとユーザ入力文字列
text	編集する文字列や出力される文字列。またはコメント

2.3.2. コマンド入力例

本書に記載されているコマンドの入力例は、表示されているプロンプトによって、それぞれに対応した実行環境を想定して書かれています。「 / 」の部分はカレントディレクトリによって異なります。各ユーザのホームディレクトリは「 ~ 」で表します。

表 2.2 表示プロンプトと実行環境の関係

プロンプト	コマンドの実行環境
[ATDE ~/]$	ATDE 上の一般ユーザで実行
[armadillo /]#	Armadillo 上 Linux の root ユーザで実行
[container /]#	Podman コンテナ内で実行

2.3.3. アイコン

本ドキュメントでは以下のようにアイコンを使用しています。

注意事項を記載します。

警告事項を記載します。

重要事項を記載します。

補足情報を記載します。

参考情報を記載します。

3. セキュリティの考え方

3.1. セキュリティの費用対効果

製品開発に費やすコストには様々なものがありますが、セキュリティのような機能性につながらないコストは軽視されがちです。しかし、一旦セキュリティ問題が発生してしまうと、業務停止など直接的な影響だけでなく、社会的な信用の低下、損害賠償など様々な被害につながってしまう可能性があります。守るべき情報資産が明確に存在するのであれば、セキュリティは重要視されるべきです。しかし、コスト度外視でありったけの対策を盛り込むのではコストが膨らみます。適切かつ適度なセキュリティ対策を講じることが大切です。製品開発の早い段階で守るべき情報資産を認識し、それら情報資産に対する脅威とリスクを分析して、リスクを評価する作業を行うことをお勧めします。そのうえで、費用対効果に見合ったセキュリティ技術を選択することが大切です。

3.2. Armadiilo-IoT ゲートウェイ G4 のセキュリティ領域

Armadillo-IoT ゲートウェイ G4 は、高性能 SoC を登載した組み込み機器のプラットフォームとしての利用はもちろん、ネットワークインターフェースやセキュアエレメント EdgeLock SE050 を登載するので、IoT デバイス、IoT ゲートウェイといったネットワーク機器の実現にも適したプラットフォームです。様々な利用形態と様々な側面からの攻撃の可能性があります。製品セキュリティを考えるうえで、システムを大まかに俯瞰して分析し、考えられる攻撃の入り口や攻撃の特徴から、それらをセキュリティの領域として分類します。分類することでセキュリティを考えやすくすることができます。

ネットワーク
: イーサネットや無線 LAN などの通信インターフェースを介した遠隔からの攻撃ベクターが考えられます。攻撃者は世界中に存在する可能性があり、ネットワーク接続時は常に攻撃を受ける可能性があります。この場合、Linux、セキュリティライブラリ、プロトコルスタックなどソフトウェアの脆弱性を利用した攻撃などがあります。

ハードウェア
: I/O インターフェースやメモリなどハードウェアに対する攻撃ベクターが考えられます。正規に購入したデバイスや盗難などによって攻撃者はデバイスを入手して、手元にデバイスがある状態でデバイスに対して物理的に直接攻撃を行います。たとえば、バスのプローブ、メモリのダンプなどの攻撃があり、また、製品開発のためのデバッグ機能が利用されることもあります。

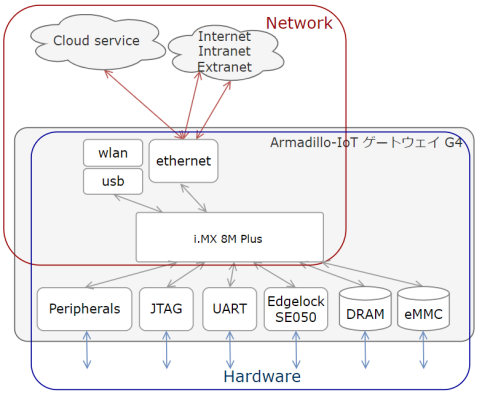

図 3.1 Armadillo-IoT ゲートウェイ G4 のセキュリティ領域

また、Armadillo-IoT ゲートウェイ G4 内のセキュリティに関する実行環境として以下のように分類することもできます。

REE (Rich Execution Environment)	この領域では Armadillo Base OS を構成する Linux、コンテナ、アプリケーションが動作します。ネットワークとハードウェアの双方の攻撃ベクターによる攻撃を受ける可能性があります。基本的には Linux のセキュリティの仕組みによって保護されるので、適切な設定を行うことが大切になってきます。また、Linux に限らず OSS の世界では頻繁に脆弱性が発見され、その対策が素早く行われています。セキュリティパッチを取り込むために、日々情報を収集することが大切で、必要な場合は早急にソフトウェアの更新を行わなければなりません。
TEE (Trusted Execution Environment)	i.MX 8M Plus に内蔵する ARM Cortex-A53 には TrustZone 技術によってリソースへのアクセスを制限する機能があります。この機能を利用してソフトウェアの実行環境を隔離し、情報資産やその処理を保護することができます。Armadillo Base OS では OP-TEE を採用しています。ソフトウェア側からの攻撃ベクターには、まず REE に足掛かりが必要になります。しかし、たとえ突破されたとしてもリソースへのアクセスは制限されるので直接攻撃は困難です。そのため、REE に公開されている TEE 向けの API を利用した攻撃などが考えられます。
CAAM (Cryptographic Acceleration and Assurance Module)	Armadillo-IoT ゲートウェイ G4 に搭載される i.MX 8M plus 内部には暗号処理アクセラレータである CAAM が内蔵されています。CAAM は SoC 内部にあるのでセキュアであり、また、高速に暗号処理を実行することができます。CAAM での暗号処理は隔離されるので CAAM への直接攻撃は困難です。ソフトウェア側からの攻撃ベクターとしては、API と入出力結果や鍵といったペイロードを利用した攻撃などが考えられます。
EdgeLock SE050	Armadillo-IoT ゲートウェイ G4 にはセキュアエレメント EdgeLock SE050 が搭載されています。RSA や ECC といった暗号処理はもちろん、内部にフラッシュメモリを内蔵するため、キーストレージとして利用することができます。いったん鍵を書き込むと外部に全く露出しないままで暗号処理に利用できます。また、チップ製造時にチップ固有の RSA と ECC の鍵が NXP の署名付きでいくつか事前にインストールされた状態で出荷されます。それらをクラウド接続や署名などに利用することができます。ソフトウェア側からの攻撃ベクターとしては、API と入出力結果や鍵といったペイロードを利用した攻撃などが考えられます。

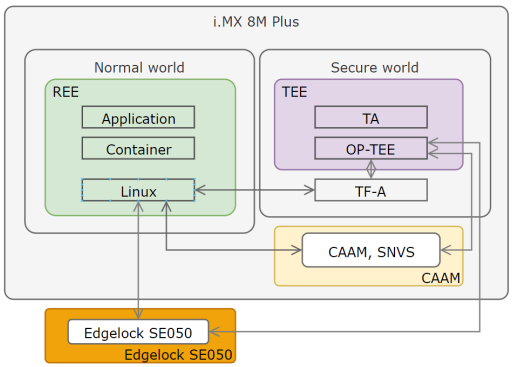

図 3.2 Armadillo-IoT ゲートウェイ G4 の実行環境の領域

3.3. Armadillo-IoT ゲートウェイ G4 で実現できるセキュリティ機能と効果

幅広い利用者のユースケースに沿った製品セキュリティの実現のため、Armadillo Base OS に様々なセキュリティ技術を組み込むことが可能です。次に各セキュリティ技術のカバーする範囲を示します。セキュリティ技術を採用する際には、採用する技術によってどこまでの範囲が守られるのかを把握することが大切です。設定次第でそれぞれの技術は高いセキュリティ性能を実現するものになり得ますが、単独で利用するだけではその効果は限定的で回避可能なものになってしまいます。そのため、いくつかの技術を適切に組み合わせて利用することで、より広範囲をカバーする回避困難なセキュリティを実現できます。そして、カバーする範囲が重なりあうことで突破困難なセキュリティを実現します。

技術 / 要件	OP-TEE	セキュア FWアップデート (SWUpdate)	アプリケーションの難読化	Edgelock SE050 セキュアエレメント	i.MX 8M Plus セキュアブート (HAB)	i.MX 8M Plus ストレージ暗号化	i.MX 8M Plus セキュリティ機能 (CAAM, SNVS)	JTAG ポートの無効化と利用認証
ソフトウェアのハッキング対策								
正規ソフトウェア以外を起動させない					✓			✓
ソフトウェアを改竄させない			✓		✓	✓		✓
ソフトウェアの実行環境を守る	✓							✓
不正なソフトウェアを書き込ませない		✓						✓
データのハッキング対策								
ストレージから抜かせない、盗聴させない	✓			✓		✓	✓	✓
RAMから抜かせない、盗聴させない	✓			✓			✓	✓
FWアップデートから抜かせない、盗聴させない	✓	✓						✓
証明書や鍵のハッキング対策								
ストレージから抜かせない、盗聴させない	✓			✓			✓	✓
事後対策								
インシデント発生時に鍵をリボークできる					✓			
インシデント発生時に鍵を変更する	✓	✓		✓	✓	✓		

図 3.3 セキュリティ技術のカバーする範囲

3.4. モデルユースケース

Armadillo-IoT ゲートウェイ G4 は様々な製品、様々な環境で利用されることが想定されます。それぞれのユースケースによって考慮すべき脅威が存在します。それらの脅威を正しく把握して、適切なセキュリティ技術を選択、組み合わせることが大切です。組み合わせの例として以下のモデルユースケースを示します。

- ネットワーク機器
- クラウドサービスへの接続機器
- 決済処理が可能な機器

3.4.1. ネットワーク機器

ユースケース　ネットワーク経由の攻撃からデバイスを保護しなければいけないケース。インターネットに接続する機器を保護するケースだけでなく、社内ネットワークへの接続であっても内部からの攻撃のリスクを考慮するケースもあります。

セキュリティの方針
- ネットワークインターフェースからの侵入を防ぐ
- ハードウェアへの直接攻撃は考慮しない

セキュリティ技術	Armadillo Base OS には最低限の Linux のセキュリティの仕組みが組み込まれています。適切な設定を実施することでセキュリティの確保が実現可能です。詳しくは、Armadillo-IoT ゲートウェイ G4 製品マニュアルの「 セキュリティ [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.8.0/ch09.html]」を参考に作業を行ってください。

また、ハードウェアへの直接攻撃は考慮しないとはいえ、Linux コンソールへのアクセスや JTAG が有効なままでは、スクリプトキディといった遊び半分の攻撃の対象になりかねません。出荷/稼働前にはデバッグ機能を無効にすることをお勧めします。

3.4.2. クラウドサービスへの接続機器

ユースケース	クラウドサービスとテレメトリなど情報資産のやりとりや、デバイス認証のための鍵や証明書を守りたいケース。また、デバイスだけでなくその先にあるサービスに対する攻撃も考慮する。
セキュリティの方針	・ ネットワークインターフェースからの侵入を防ぐ ・ ネットワーク経由の侵入を許してもデバイス内の鍵や証明書、情報資産は守る ・ ハードウェアへの直接攻撃は想定しない ・ 情報資産(暗号処理や鍵)のみピンポイントで守る
セキュリティ技術	ネットワーク機器のセキュリティ技術に以下を加える。 ・ キーストレージとして EdgeLock SE050 を利用する ・ ファイルに機密情報を保存して暗号アクセラレータで暗号化処理、復号処理を行う ・ CAAM を利用した OP-TEE でソフトウェアの実行環境を隔離する ・ 出荷/稼働前にはデバッグ機能を無効にする

3.4.3. 決済処理が可能な機器

ユースケース	決済端末やそれに相当する処理など、処理を行う経路全体を守らなければいけないケース。
セキュリティの方針	・ ネットワークインターフェースからの侵入を防ぐ ・ ネットワーク経由の侵入を許してもデバイス内の鍵や証明書、情報資産は守る ・ デバイスへの直接攻撃からデバイスを守る (ただし、ラボレベルの攻撃は想定しない)
セキュリティ技術	ネットワーク機器のセキュリティ技術に以下を加える。 ・ EdgeLock SE050 を利用した OP-TEE (鍵のストレージは EdgeLock SE050) ・ CAAM によるセキュアブート (HAB) とチェーンオブトラスト ・ ストレージの暗号化 ・ メモリの完全性チェック

・ 出荷/稼働前にはデバッグ機能を無効にする

4. 鍵の保護

この章では、Armadillo-IoT ゲートウェイ G4 と Armadillo Base OS を利用して鍵を保護する方法と保護された鍵を利用する方法について説明します。

4.1. EdgeLock SE050 を利用したキーストレージ

NXP の EdgeLock SE050 は IoT アプリケーション向けのセキュアエレメントです。フラッシュメモリを内蔵しており、保存された秘密鍵を外部に露出することなく暗号処理に利用できます。ここでは Egelock SE050 をキーストレージとして利用する方法を説明します。

EdgeLock SE050 の詳細については以下のデータシートを参照してください。

SE050 datasheet

https://www.nxp.com/docs/en/data-sheet/SE050-DATASHEET.pdf

4.1.1. EdgeLock SE05x Plug & Trust Middleware

EdgeLock SE050 を利用するためのソフトウェアとして、NXP から ミドルウェア EdgeLock SE05x Plug & Trust Middleware (以下、Plug & Trust Middleware) が提供されています。ただし、ビルド済みバイナリは提供されないため利用環境に合わせてビルドの必要があります。

Armadillo Base OS では Alpine Linux と Debian GNU/Linux 向けに Plug & Trust Middleware のビルド済みパッケージを提供しています。以下は Plug & Trust Middleware 周辺のソフトウェアスタックです。提供中のビルド済みパッケージは赤い四角のブロックになります。

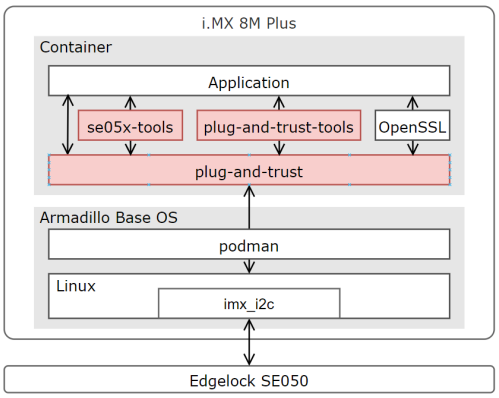

図 4.1 Plug & Trust Middleware の周辺のソフトウェアスタック

提供しているパッケージは以下のとおりです。

表 4.1 Plug & Trust Middle のパッケージ

パッケージ	内容
plug-and-trust	Plug & Trust Middleware のビルド済みライブラリ群。OpenSSL engine を含む。
plug-and-trust-dev	Plug & Trust Middleware の開発用ファイル。アプリケーション開発に利用できます。
plug-and-trust-tools	Plug & Trust Middleware に含まれるサンプルアプリケーション。最小限の動作確認ツール (se05x_Minimal) や機能や固有情報を取得するツール (se05x_GetInfo) など。
se05x-tools	非対称暗号向けの鍵の読み書きツール。

Armadillo Base OS が対応するディストリビューションとバージョンは以下のとおりです。

表 4.2 Plug & Trust Middleware のサポート

ディストリビューション	バージョン
Alpine Linux	3.15
Alpine Linux	3.16
Debian GNU/Linux	10 (buster)
Debian GNU/Linux	11 (bullseye)

本ガイド作成時点では利用したバージョンは以下のとおりです。

- Plug & Trust Middleware : 04.01.00
- plug-and-trust : 4.1.0
- se05x-tools : 1.0.0

4.1.2. EdgeLock SE050 を有効にする

Armadillo-IoT ゲートウェイ G4 は消費電力の削減のため EdgeLock SE050 を Deep Power-down モードに設定してパワーゲーティングしています。EdgeLock SE050 を利用するためには、ENA ピンをアサートして Deep Power-down モードを解除する必要があります。

表 4.3 SE050 ena pin

SE050 PIN	i.MX8MP port	initial port status
ENA	GPIO1_IO12	GPIO input

gpioset コマンドを利用して GPIO1_IO12 を出力ポートに変更して high にする。

```
[armadillo ~]# gpioset gpiochip0 12=1
```

4.1.3. Plug & Trust Middleware のインストール

Debian と Alpine Linux に対応しますが、ここでは Alpine Linux で作業を進めます。

コンテナを立ち上げます。

```
[armadillo ~]# podman run -it --name=plug_and_trust --device=/dev/i2c-2 ¥
-v /etc/apk:/etc/apk:ro docker.io/alpine /bin/sh
```

Debian を利用する場合は、at-debian の利用をお勧めします。at-debian はアットマークテクノによる動作確認済みの環境です。apt-get install を利用してインストールしてください。

Armadillo-IoT ゲートウェイ G4 製品マニュアルの「アットマークテクノが提供するイメージを使う [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.8.0/ch09.html#sct.podman-use-at-debian-image]」を参考にしてください。

パッケージをインストールします。

```
[container ~]# apk add se05x-tools plug-and-trust-tools
```

利用のために環境変数を設定します。

```
[container ~]# export OPENSSL_CONF=/etc/plug-and-trust/openssl11_sss_se050.cnf
[container ~]# export EX_SSS_BOOT_SSS_PORT=/dev/i2c-2:0x48
```

インストールと環境設定が終わったら、se05x_GetInfo で EdgeLock SE050 の動作確認を確認します。アクセスに成功すると以下のようなログが出力されます。

```
[container ~]# se05x_GetInfo
App   :INFO :PlugAndTrust_v04.01.01_20220112
App   :INFO :Running se05x_GetInfo
App   :INFO :Using PortName='/dev/i2c-2:0x48' (CLI)
:（省略）
App   :WARN :#####################################################
App   :INFO :Applet Major = 3
App   :INFO :Applet Minor = 1
App   :INFO :Applet patch = 1
App   :INFO :AppletConfig = 6FFF
App   :INFO :With    ECDAA
App   :INFO :With    ECDSA_ECDH_ECDHE
App   :INFO :With    EDDSA
App   :INFO :With    DH_MONT
App   :INFO :With    HMAC
App   :INFO :With    RSA_PLAIN
App   :INFO :With    RSA_CRT
App   :INFO :With    AES
App   :INFO :With    DES
App   :INFO :With    PBKDF
App   :INFO :With    TLS
App   :INFO :With    MIFARE
App   :INFO :With    I2CM
:（省略）
```

以上で Plug & Trust Middleware を利用するための準備を終わります。

4.1.4. Plug & Trust Middleware を活用する

ここからは実際に Plug & Trust Middleware を用いて EdgeLock SE050 を利用する方法を説明していきます。

4.1.4.1. EdgeLock SE050 の保存された鍵を利用する

plug-and-trust パッケージには OpenSSL engine のライブラリが含まれており、OpenSSL を利用して間接的に EdgeLock SE050 を操作することができます。ここでは例として、EdgeLock SE050 のチップ製造時に書き込まれた鍵を使って OpenSSL で署名と署名の検証を行います。

まず、次のコマンドでリファレンスキーを取得します。

- keyid = 0xF0000100
- Cloud connection key 0 (prime256v1)

```
[container ~]# se05x_getkey 0xF0000100 refkey.pem /dev/i2c-2:0x48
```

チップ製造時に書き込まれた鍵について

「4.1.5.1. 事前書き込みされた鍵と X.509 証明書」 を参照してください。

リファレンスキーについて

「4.1.5.2. リファレンスキー」 を参照してください。

次のコマンドで署名します。message.txt は任意のファイルを用意してください。取得したリファレンスキーを利用します。

```
[container ~]# openssl dgst -sha256 -sign refkey.pem -out sig.bin message.txt
ssse-flw: EmbSe_Init(): Entry
App   :INFO :Using PortName='/dev/i2c-2:0x48' (ENV: EX_SSS_BOOT_SSS_PORT=/dev/i2c-2:0x48)
sss   :INFO :atr (Len=35)
      00 A0 00 00    03 96 04 03    E8 00 FE 02    0B 03 E8 08
      01 00 00 00    00 64 00 00    0A 4A 43 4F    50 34 20 41
      54 50 4F
sss   :WARN :Communication channel is Plain.
sss   :WARN :!!!Not recommended for production use.!!!
ssse-flw: Version: 1.0.5
ssse-flw: EmbSe_Init(): Exit
ssse-flw: Control Command EMBSE_LOG_LEVEL; requested log level = 4
```

署名が正しいかどうかを確認するために署名を検証します。

```
[container ~]# openssl dgst -sha256 -prverify refkey.pem -signature sig.bin message.txt
ssse-flw: EmbSe_Init(): Entry
```

```
App   :INFO :Using PortName='/dev/i2c-2:0x48' (ENV: EX_SSS_BOOT_SSS_PORT=/dev/i2c-2:0x48)
sss   :INFO :atr (Len=35)
      00 A0 00 00    03 96 04 03    E8 00 FE 02    0B 03 E8 08
      01 00 00 00    00 64 00 00    0A 4A 43 4F    50 34 20 41
      54 50 4F
sss   :WARN :Communication channel is Plain.
sss   :WARN :!!!Not recommended for production use.!!!
ssse-flw: Version: 1.0.5
ssse-flw: EmbSe_Init(): Exit
ssse-flw: Control Command EMBSE_LOG_LEVEL; requested log level = 4
Verified OK
```

クラウドサービスに接続するために事前書き込みされた鍵と証明書を利用する方法は以下を参照してください。

EdgeLock SE050 を使用して AWS IoT Core へ接続する

https://armadillo.atmark-techno.com/howto/connect_to_aws_iot_core_for_aiot_g4

EdgeLock SE050 を使用して Azure IoT Hub へ接続する

https://armadillo.atmark-techno.com/howto/connect_azure_iot_hub_for_aiot_g4

4.1.4.2. ユーザーが生成した鍵を保存して利用する

EdgeLock SE050 ではユーザーが生成した鍵を保存することもできます。一度保存された鍵は、そのまま暗号処理に利用することができるようになります。ここでは例として、ユーザー鍵を保存して openssl cms で暗号化と復号を実行してみます。

まず、ECC の曲線 prime256v1 の鍵を生成します。証明書の情報はお任せします。

```
[container ~]# openssl req -x509 -nodes -days 3650 -newkey ec ¥
-pkeyopt ec_paramgen_curve:prime256v1 -keyout key.pem -out cert.pem
```

生成した鍵と自己証明書を EdgeLock SE050 に保存します。keyid は 1 から 0x7BFFFFFF の範囲が利用者に開放されています。

```
[container ~]# se05x_setkey -f 0x10 key.pem /dev/i2c-2:0x48
[container ~]# se05x_setkey -f 0x11 cert.pem /dev/i2c-2:0x48
```

Secure Object についての詳しい情報は以下を参照してください。

SE050A/B/C/D/F APDU Specification

https://www.nxp.com/search?keyword=AN12413

秘密鍵を EdgeLock SE050 に保存に成功したあとは、利用する際に EdgeLock SE050 にアクセスすることになります。そのため、ファイルシステム上の key.pem は基本的に必要ありません。セキュリティ上、削除することをお勧めします。

リファレンスキーを取得します。

```
[container ~]# se05x_getkey 0x10 refkey.pem /dev/i2c-2:0x48
```

リファレンスキーについて

「4.1.5.2. リファレンスキー」 を参照してください。

自己証明書に含まれる公開鍵を使って暗号化します。message.txt は任意のファイルを用意してください。

```
[container ~]# openssl cms -encrypt -binary -aes256 -in message.txt ¥
-out message.enc cert.pem
```

以下はあくまで例ですが、暗号化されたファイルは以下のような内容になります。

```
[container ~]# cat message.enc
MIME-Version: 1.0
Content-Disposition: attachment; filename="smime.p7m"
Content-Type: application/pkcs7-mime; smime-type=enveloped-data; name="smime.p7m"
Content-Transfer-Encoding: base64

MIIBmQYJKoZIhvcNAQcDoIIBijCCAYYCAQIxggERoYIBDQIBA6BRoU8wCQYHKoZI
zj0CAQNCAARdx2h5MMEe0e7MgYHg179QPxxJvuLTOPONM9NF10V7/3AjgxE1D2XS
Fjgt/btA17HU9l13f6NVRFbZzNVHtxX2MBgGCSuBBRCGSD8AAjALBglghkgBZQME
AS0wgZowgZcwazBTMQswCQYDVQQGEwJKUDEMMAoGA1UECAwDTi9BMQwwCgYDVQQH
DANOL0ExDDAKBgNVBAoMA04vQTEaMBgGA1UEAwwRYXRtYXJrIHRlY2hubyxpbmMC
FA2ES6jQVjVE5fv9KjfD4QqM5hrLBChc/Z1uATls1oxl6aPyYcqf1tns3pR41gko
sturG2/iRRjjQNbwaa2gMGwGCSqGSIb3DQEHATAdBglghkgBZQMEASoEEMl596FU
hh1Rs4sHBcqwmTyAQPyYZLqHHSr1np9CnCxtzcRztheo0gtkC+8elS97GPzKcbpU
gWoOECwNNwyOpTRGxEJ9Mx+C4yW7J7Jiz/XvgDs=
```

EdgeLock SE050 のキーストレージにある秘密鍵を使って復号します。

```
[container ~]# openssl cms -decrypt -in message.enc -out message.dec ¥
-inkey refkey.pem
```

4.1.5. 補足説明

4.1.5.1. 事前書き込みされた鍵と X.509 証明書

EdgeLock SE050 のフラッシュメモリにはチップ製造時に書き込まれたチップ固有な鍵と NXP に署名された X.509 証明書を保持しています。それらの鍵や証明書は Armadillo-IoT ゲートウェイ G4 を購入後にすぐにそのままの状態でクラウドサービスなどの PKI に利用できます。事前に書き込みがされた状態で空き容量はあるので、ユーザーの鍵や証明書を追加することも可能です。

事前書き込みされた鍵と証明書については以下のドキュメントを参照してください。利用可能な keyid、鍵の種類、想定する用途が記載されています。

なお、Armadillo-IoT ゲートウェイ G4 に搭載されている EdgeLock SE050 は Variant C です。

SE050 configuration

https://www.nxp.com/search?keyword=AN12436

4.1.5.2. リファレンスキー

EdgeLock SE050 は秘密鍵が外部に露出しません。これは、いったん鍵を EdgeLock SE050 に書き込むと秘密鍵を抜き出すことができない (削除や書き換えは可能) という仕様で実現されています。攻撃者がチップの中にある、どこかのサイトの認証資格などを有する証明書を得るためには、物理的にチップ自体も必要ということになってきます。

秘密鍵を抜き出すことができない仕様で、どのように秘密鍵を利用するかというと、リファレンスキーと呼ばれる特殊なファイルで代用します。リファレンスキーは秘密鍵のフォーマットで保存されますが、公開鍵部分のみ有効で、それ以外の値は管理用やダミーの情報になっています。リファレンスキーをファイルシステムに置いておいて、EdgeLock SE050 を利用する時にそのファイルを秘密鍵のように使うと、Plug & Trust Middleware に含まれる OpenSSL の engine のライブラリがフックして EdgeLock SE050 にアクセスします。

4.1.5.3. se05x-tools

se05x-tools は EdgeLock SE050 をキーストレージとして利用するためのツールです。アットマークテクノから Armadillo Base OS 向けにリリースされています。se05x-tools の内部では Plug & Trust Middleware を利用しています。バージョン 1.0.0 ではサポートする Secure Object は非対称暗号鍵のみです。

対応する非対称暗号鍵の種類は以下のとおりです。

se05x-tools でサポートされる ECC カーブ

- prime192v1, secp224r1, prime256v1, secp384r1, secp521r1

se05x-tools でサポートされる RSA 鍵長

- 1024bit, 2048bit, 3072bit, 4096bit

Secure Object についての詳しい情報は以下を参照してください。

SE050A/B/C/D/F APDU Specification

https://www.nxp.com/search?keyword=AN12413

5. セキュアブート

この章ではセキュアブートを有効にする方法について説明します。また、セキュアブートをベースにしたいくつかの技術についても紹介します。

5.1. セキュアブートとチェーンオブトラスト

組み込みデバイスへの攻撃は様々な方向から行われます。ある方向のセキュリティ対策が強固な場合、攻撃者は回避可能な別の方向がないのか模索します。攻撃者のコードを何らかの方法でデバイスに組み込んで対策を回避するのが、単純ですが有効な方法でしょう。IoT デバイスはネットワーク上のサービスとデータのやり取りを行います。通信路の暗号化、サーバーとデバイスの相互認証などの対策を講じたとしても、IoT デバイス上にあるソフトウェアに攻撃者のコードを組み込むことで対策を回避してシステムに侵入される可能性があるのです。

セキュアブートは、起動ソフトウェアのディジタル署名を用いて正規ソフトウェアであることを確認してから起動する処理のことです。攻撃者によって作られた不正なコードを実行前に検出することができます。セキュアブートはチェーンオブトラスト (chain of trust) と表裏一体に実装されます。チェーンオブトラストとは、その名の通り、信頼を繋いでいく形態のことを指します。ルートオブトラストと呼ばれる基礎となる情報から枝葉のように繋がれた情報を認証していくことで、繋がれた個々のコンポーネントだけでなくシステムを信頼できるものにしてくれます。セキュアブートは、起動時にソフトウェアを順番に認証することで信頼を次に繋いでいるのです。セキュアブートの範囲をどこまでにするかによりますが、起動時に認証されたソフトウェアで別の情報を認証すれば、チェーンオブトラストを繋げていくことができます。また、IoT デバイスとクラウドサービスから構成されるような広範囲に及ぶシステムは特に信頼が必要になります。信頼できるセキュリティ基盤を構築するためには、構成するソフトウェアが正規のリリース物であることを確認することが重要になります。チェーンオブトラストを採用することでより信頼できる IoT システムとなり得るのです。

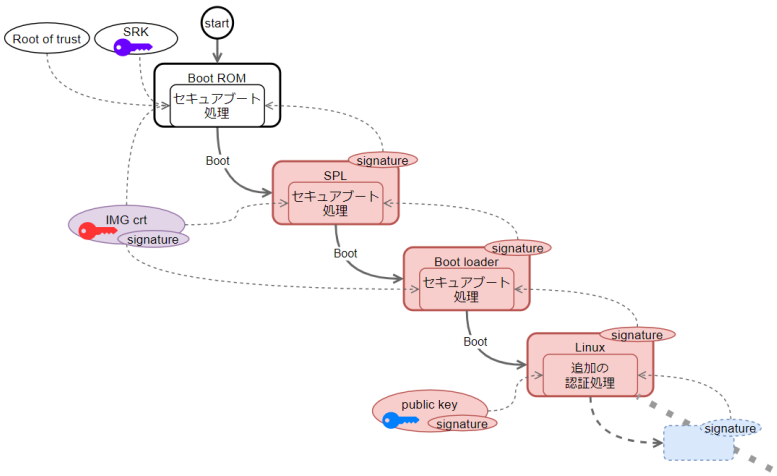

図 5.1 チェーンオブトラスト

では、どういったケースでセキュアブートを採用するべきでしょうか。セキュアブートはセキュアな組み込みデバイスを実現する上で最初のステップにするべき技術です。しかし、導入するのであればコスト面にも配慮することが必要でしょう。まず、製造時の追加コストが必要です。鍵等を書き込む工程が必要になります。ただ書き込めばよいわけではなく、漏洩があってはいけないので物理的な隔離などセキュアに書き込む必要があります。メンテナンスにも追加のコストが必要です。ソフトウェアのリリー

ス時にはソフトウェアの署名が必要になります。こちらも、物理的な隔離などの漏洩、汚染対策が必要になります。また、どこまでやるかによりますが、定期的な鍵の更新、インシデントや製品寿命による鍵のリボーケーションなどのメンテナンスコストが必要になってきます。費用対効果を検討してからの導入をお勧めします。

5.2. HAB とは

HAB (High Assurance Booting) は、NXP が提供するセキュアブートの実装です。i.MX 8M Plus の BootROM には HABv4 が組み込まれます。デフォルトでは無効な状態になっていますが、一度、eFuse に情報を書き込むことでセキュアブートが有効になり、それ以降、有効な状態のまま変更不可能になります。HABv4 で規定する仕様では次の情報をブートローダーイメージに追加することで BootROM が起動時にブートローダーの認証を行います。

- CSF (Command Sequence File)、IVT (Image Vector Table)
- SRK (Super Root Key)、CSF、IMG 署名確認鍵
- イメージの署名

NXP は署名ツールとして CST (Code Signing Tool) をリリースしています。本来、署名範囲などは環境によって様々な実装がなされるべきなので署名ツール自体にはその辺りの仕様が含まれません。ブートローダーの実装仕様に依存します。Armadillo Base OS で採用される uboot-imx のセキュアブート処理では、Trusted Firmware-A (ATF)、OP-TEE OS、Linux カーネルイメージまでが認証の対象になります。署名に関する概要は以下のとおりです。

- 署名確認用の鍵は X.509 証明書に対応する
- 署名は RSA/ECC に対応する
  - RSA 1024, 2048, 3072, 4096 bits
  - ECC NIST P-256, NIST P-384, NIST P-521
- 署名のダイジェストは SHA256 のみ

HAB の詳しい仕様は以下を参照してください。

i.MX Secure Boot on HABv4 Supported Devices

https://www.nxp.com/search?keyword=AN4581

Encrypted Boot on HABv4 and CAAM Enabled Devices

https://www.nxp.com/search?keyword=AN12056

5.3. 署名環境を構築する

まず、セキュアブートをセットアップするために必要な環境を構築していきます。

5.3.1. 署名環境について

ここで利用する署名環境の構成は以下のとおりです。

表 5.1 署名環境

ツール/パッケージ	説明
ATDE(Atmark Techno Development Environment)	提供元 : アットマークテクノ Armadillo シリーズの開発環境として VMware イメージとして配布されます
CST(Code Signing Tool)	提供元 : NXP HAB の署名を行うツールや鍵を生成するツールを含みます
mkswu パッケージ	提供元 : アットマークテクノ Armadillo Base OS のファームウェアアップデートの仕組みである SWUpdate 用のアップデートファイルを生成するツールを含みます
build-rootfs-[VERSION].tar.gz	提供元 : アットマークテクノ Armadillo Base OS のルートファイルシステムを生成するツールを含みます Alpine Linux をベースにアットマークテクノのパッケージを加えた構成になります
imx-boot-[VERSION].tar.gz	提供元 : アットマークテクノ ブートローダーのソースコードや、セキュアブート用のイメージをつくるスクリプトを含みます
baseos-[VERSION].tar.zst	提供元 : アットマークテクノ Armadillo Base OS のルートファイルシステムのビルド済みバイナリ

ビルドの全体的なプロセスフローについては「5.11.4. ビルドのプロセスフロー」に図があります。

5.3.2. ATDE を準備する

Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「開発/動作確認環境の構築 [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.8.0/ch04.html#sct.setup-environment]」を参考に作業をしてください。

5.3.3. 署名ツール (CST) を準備する

NXP から署名ツールをダウンロードします

以下のサイトからダウンロードします。

CST の最新版

https://www.nxp.com/search?keyword=IMX_CST_TOOL_NEW

・ ダウンロードには NXP サイトへのユーザー登録が必要になります

- 本書作成時点では v3.3.1 を利用しました。2023 年 6 月時点で最新の v3.3.2 では、ブートローダーの暗号化が利用出来ませんので、必要な場合はサポートにご連絡ください。

ツールを展開します。

```
[ATDE ~]$ tar -xaf cst-3.3.1.tgz
```

5.3.4. アップデートファイル作成ツール (mkswu) を準備する

mkswu は SWUpdate に対応したアップデートファイルを生成するツールです。ATDE にデフォルトでインストールされていないので、手動でインストールする必要があります。

Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「SWU イメージの作成 [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.8.0/ch09.html#sct.mkswu]」を参考に次の作業をしてください。

- mkswu の取得
- 最初に行う設定 (mkswu --init)

セキュアブートに対応する mkswu は 4.0-1 以上になります。

SWU イメージの暗号化

SWU イメージはデフォルト設定では暗号化されません。通信路は TLS で守られても、ファイルで保管されているときには平文なので SWU イメージ内にある機密情報が漏洩する可能性があります。SWU イメージに漏洩すると問題のある情報資産を含めるときには必ず暗号化するべきです。

Armadillo-IoT ゲートウェイ G4 の製品マニュアル内の「最初に行う設定」を実施するときには、以下の質問では y と答えてください。

アップデートイメージを暗号化しますか？ (N/y)

バージョンの確認方法は以下のとおりです。

```
[ATDE ~]$ dpkg -l mkswu
ii  mkswu        4.0-1
```

5.3.5. Armadillo Base OS イメージの取得

署名や暗号化の対象であるブートローダーや Linux は、アットマークテクノからビルド済みイメージが配布されています。アットマークテクノのサイトから最新イメージをダウンロードしてください。

Armadillo-IoT ゲートウェイ G4 向け Armadillo Base OS のコンテンツのビルド済みバイナリ

「Armadillo Base OS アーカイブ」を取得してください。

https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/baseos

Armadillo-IoT ゲートウェイ G4 ブートローダー

「ブートローダー ソース（u-boot 等）」を取得してください。

https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/boot-loader

Armadillo-IoT ゲートウェイ G4 開発用ツール

「Alpine Linux ルートファイルシステムビルドツール」を取得してください。

https://armadillo.atmark-techno.com/resources/software/armadillo-iot-g4/tools

本書作成時点で利用したパッケージは以下のとおりです。

- imx-boot-2020.04-at8.tar.gz
- baseos-x2-3.15.4-at.7.tar.zst
- build-rootfs-v3.15-at.7.tar.gz

Armadillo Base OS を展開します。

```
[ATDE ~]$ mkdir -p baseos
[ATDE ~]$ tar xaf baseos-x2-[VERSION].tar.zst -C baseos ❶
```

❶ [VERSION] はバージョンによって変化します

ブートローダーを展開します。

```
[ATDE ~]$ tar xaf imx-boot-[VERSION].tar.gz ❶
```

❶ [VERSION] はバージョンによって変化します

ビルドツールを展開します。

```
[ATDE ~]$ tar xaf build-rootfs-[VERSION].tar.gz ❶
```

❶ [VERSION] はバージョンによって変化します

5.3.6. 署名環境のディレクトリツリー構成

ディレクトリ構成の一部抜粋は以下のとおりです。

```
├── baseos
├── build-rootfs-[VERSION] ❶
├── cst_3.3.1
│   ├── crts ❷
│   ├── keys ❸
│   └── linux64/bin ❹
├── imx-boot-[VERSION] ❺
├── mkswu ❻
└── secureboot ❼
```

❶ [VERSION] はそれぞれのバージョンによって変化します
❺

❷ ブートローダーに組み込む自己証明書群が配置される

❸ 鍵の生成ツール、生成された鍵が配置される

❹ ビルド済み署名ツールが配置される

❻ swu 作成に利用される。

❼ ビルド結果が配置される。ビルド前にはないディレクトリ

5.3.7. 署名鍵を生成する

セキュアブート用の PKI tree を作っていきます。生成する鍵とその証明書は以下のとおりです。

表 5.2 セキュアブート用の鍵と証明書

name	description	file name
CA 鍵ペア	ルート CA	CA1_sha256_[alg]_v3_ca_crt
SRK 鍵ペア	Super Root Key 用の鍵ペア	SRK[n]_sha256_[alg]_v3_ca_key
CSF 鍵ペア	CSF 用の鍵ペア	CSF[n]_sha256_[alg]_v3_usr_key
IMG 鍵ペア	イメージ用の鍵ペア	IMG[n]_sha256_[alg]_v3_usr_key
SRK 証明書	CA 鍵によって署名された SRK 公開鍵を含んだ証明書	SRK[n]_sha256_[alg]_v3_ca_crt
CSF 証明書	SRK によって署名された CSF 公開鍵を含んだ証明書	CSF[n]_sha256_[alg]_v3_usr_crt
IMG 証明書	SRK によって署名された IMG 公開鍵を含んだ証明書	IMG[n]_sha256_[alg]_v3_usr_crt

- [alg]: アルゴリズム (RSA: 1024/2048/3072/4096, ECC: prime256v1/secp384r1/secp521r1)
- [n]: n = 1,2,3,4

署名鍵のデフォルト設定は ECC secp521r1 となっています。鍵の種類などの設定は imx-boot/secureboot.conf にあります。変更する場合は「5.11.1. secureboot.conf の設定方法」 を参照してください。

5.3.7.1. 署名鍵の生成

実際に署名鍵を生成していきます。ここでは既存の CA 証明書を利用せずに鍵を生成します。以下のコマンドを実行してください。

```
[ATDE ~]$ ./imx-boot/secureboot.sh init_srk
```

秘密鍵のパスワードを求められるので入力してください。

```
Passphrase used for key generation (will be written in plain text on filesystem)
```

CST の詳細の情報は cst 内の docs ディレクトリにある CST_UG.pdf をご参照ください。

5.3.7.2. 生成された署名鍵の確認

ファイルがあるかどうかで鍵が生成されたことを確認してください。以下はデフォルト設定である ECC secp521r1 の鍵を生成した場合の結果になります。

```
cst-3.3.1
     ├──── crts
     │     ├──── CA1_sha256_secp521r1_v3_ca_crt.der
     │     ├──── CA1_sha256_secp521r1_v3_ca_crt.pem
     │     ├──── CSF1_1_sha256_secp521r1_v3_usr_crt.der
     │     ├──── CSF1_1_sha256_secp521r1_v3_usr_crt.pem
     │     ├──── CSF2_1_sha256_secp521r1_v3_usr_crt.der
     │     ├──── CSF2_1_sha256_secp521r1_v3_usr_crt.pem
     │     ├──── CSF3_1_sha256_secp521r1_v3_usr_crt.der
     │     ├──── CSF3_1_sha256_secp521r1_v3_usr_crt.pem
     │     ├──── CSF4_1_sha256_secp521r1_v3_usr_crt.der
     │     ├──── CSF4_1_sha256_secp521r1_v3_usr_crt.pem
     │     ├──── IMG1_1_sha256_secp521r1_v3_usr_crt.der
     │     ├──── IMG1_1_sha256_secp521r1_v3_usr_crt.pem
     │     ├──── IMG2_1_sha256_secp521r1_v3_usr_crt.der
     │     ├──── IMG2_1_sha256_secp521r1_v3_usr_crt.pem
     │     ├──── IMG3_1_sha256_secp521r1_v3_usr_crt.der
     │     ├──── IMG3_1_sha256_secp521r1_v3_usr_crt.pem
     │     ├──── IMG4_1_sha256_secp521r1_v3_usr_crt.der
     │     ├──── IMG4_1_sha256_secp521r1_v3_usr_crt.pem
     │     ├──── SRK1_sha256_secp521r1_v3_ca_crt.der
     │     ├──── SRK1_sha256_secp521r1_v3_ca_crt.pem
     │     ├──── SRK2_sha256_secp521r1_v3_ca_crt.der
```

```
│   ├── SRK2_sha256_secp521r1_v3_ca_crt.pem
│   ├── SRK3_sha256_secp521r1_v3_ca_crt.der
│   ├── SRK3_sha256_secp521r1_v3_ca_crt.pem
│   ├── SRK4_sha256_secp521r1_v3_ca_crt.der
│   ├── SRK4_sha256_secp521r1_v3_ca_crt.pem
│   ├── SRK_1_2_3_4_fuse.bin
│   └── SRK_1_2_3_4_table.bin
└── keys
    ├── CA1_sha256_secp521r1_v3_ca_key.der
    ├── CA1_sha256_secp521r1_v3_ca_key.pem
    ├── CSF1_1_sha256_secp521r1_v3_usr_key.der
    ├── CSF1_1_sha256_secp521r1_v3_usr_key.pem
    ├── CSF2_1_sha256_secp521r1_v3_usr_key.der
    ├── CSF2_1_sha256_secp521r1_v3_usr_key.pem
    ├── CSF3_1_sha256_secp521r1_v3_usr_key.der
    ├── CSF3_1_sha256_secp521r1_v3_usr_key.pem
    ├── CSF4_1_sha256_secp521r1_v3_usr_key.der
    ├── CSF4_1_sha256_secp521r1_v3_usr_key.pem
    ├── IMG1_1_sha256_secp521r1_v3_usr_key.der
    ├── IMG1_1_sha256_secp521r1_v3_usr_key.pem
    ├── IMG2_1_sha256_secp521r1_v3_usr_key.der
    ├── IMG2_1_sha256_secp521r1_v3_usr_key.pem
    ├── IMG3_1_sha256_secp521r1_v3_usr_key.der
    ├── IMG3_1_sha256_secp521r1_v3_usr_key.pem
    ├── IMG4_1_sha256_secp521r1_v3_usr_key.der
    ├── IMG4_1_sha256_secp521r1_v3_usr_key.pem
    ├── SRK1_sha256_secp521r1_v3_ca_key.der
    ├── SRK1_sha256_secp521r1_v3_ca_key.pem
    ├── SRK2_sha256_secp521r1_v3_ca_key.der
    ├── SRK2_sha256_secp521r1_v3_ca_key.pem
    ├── SRK3_sha256_secp521r1_v3_ca_key.der
    ├── SRK3_sha256_secp521r1_v3_ca_key.pem
    ├── SRK4_sha256_secp521r1_v3_ca_key.der
    └── SRK4_sha256_secp521r1_v3_ca_key.pem
```

生成した鍵は今後も利用するものです。壊れにくい、セキュアなストレージにコピーしておくことをお勧めします。

鍵の更新は計画性を持って行ってください。たとえば開発時のみ利用する鍵、運用時に利用する鍵を使い分ける。また、鍵は定期的に更新が必要です。以下を参考にしてください。

BlueKrypt Cryptographic Key Length Recommendation

https://www.keylength.com/

5.3.7.3. 鍵のハッシュ値の確認

生成が終わったら、次のコマンドを実行して鍵のハッシュ値を確認して、結果を控えておいて下さい。後ほどデバイス上で efuse を書く際に利用します。

```
[ATDE ~]$ ./imx-boot-[VERSION]/secureboot.sh print_srk_fuse
```

以下のように表示されます。**あくまで例でハッシュ値は生成された鍵毎に異なります。このまま利用しないでください。**

```
Use the following commands in uboot to set root key hash in OTP:
fuse prog -y 6 0 0xC04FA990 0x6C4BCFCC 0x90DAC78E 0x6C6CED49
fuse prog -y 7 0 0x535C7B3E 0x2695B4D4 0x9B3D028E 0x9FF5EFFB
fuse prog -y 0 0 0x200

For testing it is possible to temporarly override fuses instead,
but this only validates the last part of the boot:
fuse override 6 0 0xC04FA990 0x6C4BCFCC 0x90DAC78E 0x6C6CED49
fuse override 7 0 0x535C7B3E 0x2695B4D4 0x9B3D028E 0x9FF5EFFB
```

以上で、環境構築を終わります。

5.4. セキュアブート機能の選択

実際にセットアップする前にセキュアブートのオプションについて説明します。

セキュアブートを実現することによって、セキュアブートのチェーンオブトラストをベースとした様々な機能を入れられるようになります。Armadillo-IoT ゲートウェイ G4 と Armadillo Base OS を利用することで、以下のようなセキュリティ機能を実現することができます。セキュリティの要件に合わせて機能を選択してください。

表 5.3 セキュアブートの派生機能

機能	概要	期待される効果
セキュアブート (基本) 「5.5. セキュアブートのセットアップ」	ブートローダーと Linux kernel の署名確認	ブートローダーと Linux Kernel の改竄を検出することができます。また、 署名とその検証によって、正規ソフトウェアのみの起動を強制させることができます。
暗号化セキュアブート 「5.6. 暗号化セキュアブートのセットアップ」	セキュアブート ＋ブートローダーの暗号化	ブートローダーがフラッシュメモリに保存されている間は暗号化によって ブートローダー内にある情報資産の漏洩や、解析から保護することができます。 基本ブートローダーに情報資産はないですが、情報資産を守るための鍵をもつ場合が あるので、そういったケースでの活用が考えられます。
ストレージの暗号化 「5.7. ストレージの暗号化のセットアップ」	セキュアブート ＋ブートローダーの暗号化 ＋ブロックデバイスの暗号化	ファイルがフラッシュメモリに保存されている間は暗号化によって ファイルは保護されます。 通常はアプリケーションやデータなどはファイルとして保存されるので、 そういったファイルに情報資産が含まれるケースでの活用が考えられます。

5.5. セキュアブートのセットアップ

ここではセキュアブートを有効にする方法を説明します。

5.5.1. uboot-imx のセキュアブートの実装仕様

セキュアブートの実装は uboot-imx のガイドに準拠しています。詳しい仕様は以下を参照してください。取得したソースツリー内にドキュメントがあります。

i.MX8M, i.MX8MM Secure Boot guide using HABv4

imx-boot/uboot-imx/doc/imx/habv4/guides/mx8m_secure_boot.txt

セキュアブートのフローは「5.11.3. セキュアブートのフロー」 を参照してください。

イメージの詳しいフォーマットと展開先については「5.11.2. 署名済みイメージと展開先」 を参照してください。

アップデートのフローの詳細については 「5.11.5. ファームウェアアップデートのフロー」を参照してください。

5.5.2. セットアップの流れ

セキュアブートのセットアップの流れは以下のとおりです。

PC 上の作業

事前準備

1. 「5.3. 署名環境を構築する」
2. 「5.3.7. 署名鍵を生成する」

ビルド

3. 「5.5.3. 署名済みブートローダーの作成」
4. 「5.5.4. 署名済み Linux カーネルイメージの作成」
5. 「5.5.5. ルートファイルシステムのビルド」
6. 「5.5.6. セキュアブートセットアップ用 SD boot ディスクの作成」

Armadillo-IoT ゲートウェイ G4 上の作業

8. 「5.8. Armadillo-IoT ゲートウェイ G4 上の作業」
9. 「5.8.4. ファームウェアの書き込みをはじめる」
10. 「5.10. セットアップ完了後のアップデートの運用について」

ビルドの全体的なプロセスフローについては「5.11.4. ビルドのプロセスフロー」に図があります。

5.5.3. 署名済みブートローダーの作成

セキュアブートに対応したブートローダーをビルドします。Armadillo-IoT ゲートウェイ G4 製品マニュアルの「ブートローダーをビルドする [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.8.0/ch09.html#sct.build-imx-boot]」を参考にビルドしてください。

以下のファイルが作られていれば、ビルドに成功しています。

```
[ATDE ~]$ ls imx-boot-[VERSION]
imx-boot_armadillo_x2
```

次に、ビルドしたイメージを署名して、イメージに署名を付加していきます。セキュアブートに対応するために、特別な修正は必要ありません。アットマークテクノからリリースされているブートローダーをそのまま利用できます。既にビルドされたブートローダーを利用します。

署名をブートローダーイメージに付加していきます。

```
[ATDE ~]$ cd imx-boot-[VERSION]
[ATDE ~/imx-boot-[VERSION]]$ ./secureboot.sh imxboot
```

ビルド後に以下の２つのイメージが作られれば成功です。

```
[ATDE ~/imx-boot-[VERSION]]$ ls ../secureboot
secureboot/imx-boot_armadillo_x2.signed
```

imx-boot_armadillo_x2.sign は SRK の異なるボードには当たり前ですが利用できません。誤って書き込まないようご注意ください。

5.5.4. 署名済み Linux カーネルイメージの作成

secureboot.conf を開発環境に合わせて更新していきます。

```
[ATDE ~]$ vi imx-boot-[VERSION]/secureboot.conf
```

Linux カーネルとデバイスツリーブロブがある場所を参照するように変更します。

```
LINUX_IMAGE="$SCRIPT_DIR/../baseos/boot/Image"
LINUX_DTB="$SCRIPT_DIR/../baseos/boot/armadillo_iotg_g4.dtb"
```

設定の詳細については 「5.11.1. secureboot.conf の設定方法」 を参照してください。

最後に、Linux カーネルイメージを作ります。

```
[ATDE ~]$ cd imx-boot-[VERSION]
[ATDE ~/imx-boot-[VERSION]]$ ./secureboot.sh linux
```

ビルド後に以下のイメージがあれば成功です。

```
[ATDE ~/imx-boot-[VERSION]]$ ls ../secureboot
Image.signed
```

5.5.5. ルートファイルシステムのビルド

まず、ルートファイルシステムに組み込むために、署名された Linux カーネルのイメージをビルド用のリソースディレクトリに配置します。

```
[ATDE ~]$ mkdir -p build-rootfs-[VERSION]/ax2/resources/boot
[ATDE ~]$ cp secureboot/Image.signed build-rootfs-[VERSION]/ax2/resources/boot/Image
```

組み込むファイルの準備が終わったので、次にルートファイルシステムをビルドします。

```
[ATDE ~]$ cd build-rootfs-[VERSION]
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_rootfs.sh
```

以下のファイルがあればルートファイルシステムのビルドは成功しています。以下は例なので、ファイル名などはバージョンや日付によって変化します。

```
[ATDE ~/build-rootfs-[VERSION]]$ ls
baseos-[VERSION].[DATE].tar.zst
```

5.5.6. セキュアブートセットアップ用 SD boot ディスクの作成

SD boot するための起動イメージを作ります。

```
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh
```

以下のファイルができていれば成功です。ファイルに含まれる日付やバージョンは変化します。

```
[ATDE ~/build-rootfs-[VERSION]]$ ls
baseos-[VERSION].[DATE].img
```

次に、上記の起動イメージを使って SD boot で初回書き込みを行うためのディスクイメージを作成します。上記のイメージで SD boot して、これからつくるイメージを eMMC に書き込んでいきます。

同じコマンドを使って引数を変えることで可能です。--installer には上記のイメージを指定します。--boot には生成したブートローダーを指定します。

```
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh ¥
--installer baseos-[VERSION].[DATE].img ¥
--boot ../secureboot/imx-boot_armadillo_x2.signed
```

以下のファイルができていれば成功です。ファイルに含まれる日付やバージョンは変化します。

```
[ATDE ~/build-rootfs-[VERSION]]$ ls
baseos-[VERSION].[DATE]-installer.img
```

上記で作成した SD boot イメージを、Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「ブートディスクの作成 [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.8.0/ch09.html#sct.make-bootdisk]」を参考にして SD カードに書き込んでください。

SD boot 用の SD カードの作成が終わったら、次からは実際に「5.8. Armadillo-IoT ゲートウェイ G4 上の作業」を行います。

5.6. 暗号化セキュアブートのセットアップ

ここでは暗号化されたブートローダーによるセキュアブートを有効にする方法を説明します。

5.6.1. uboot-imx の暗号化セキュアブートの実装仕様

暗号化セキュアブートは uboot-imx の Encrypted boot で実現しています。実装仕様は uboot-imx のガイドに準拠しています。詳しい仕様は以下を参照してください。取得したソースツリー内にドキュメントがあります。

i.MX8M family Encrypted Boot guide using HABv4

imx-boot/uboot-imx/doc/imx/habv4/guides/mx8m_encrypted_boot.txt

セキュアブートのフローは「5.11.3. セキュアブートのフロー」 を参照してください。

イメージの詳しいフォーマットと展開先については「5.11.2. 署名済みイメージと展開先」 を参照してください。

アップデートのフローの詳細については 「5.11.5. ファームウェアアップデートのフロー」を参照してください。

5.6.2. セットアップの流れ

暗号化セキュアブートのセットアップの流れは以下のとおりです。

PC 上の作業

事前準備

1. 「5.3. 署名環境を構築する」
2. 「5.3.7. 署名鍵を生成する」

ビルド

3. 「5.6.3. 署名済みの暗号化ブートローダーの作成」
4. 「5.6.4. 暗号化 Linux カーネルイメージの生成」
5. 「5.6.5. 暗号化セキュアブート対応 swu の作成」
6. 「5.6.6. ルートファイルシステムのビルド」
7. 「5.6.7. セキュアブートセットアップ用 SD boot ディスクの作成」

Armadillo-IoT ゲートウェイ G4 上の作業

8. 「5.8. Armadillo-IoT ゲートウェイ G4 上の作業」
9. 「5.8.4. ファームウェアの書き込みをはじめる」
10. 「5.10. セットアップ完了後のアップデートの運用について」

ビルドの全体的なプロセスフローについては「5.11.4. ビルドのプロセスフロー」に図があります。

5.6.3. 署名済みの暗号化ブートローダーの作成

セキュアブートと暗号化に対応したブートローダーをビルドします。Armadillo-IoT ゲートウェイ G4 製品マニュアルの「ブートローダーをビルドする [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.8.0/ch09.html#sct.build-imx-boot]」を参考にビルドしてください。

以下のファイルが作られていれば、ビルドに成功しています。

```
[ATDE ~]$ ls imx-boot-[VERSION]
imx-boot_armadillo_x2
```

ビルドしたイメージを署名して、イメージに署名を付加していきます。セキュアブートに対応するために、特別な修正は必要ありません。アットマークテクノからリリースされているブートローダーをそのまま利用できます。既にビルドされたブートローダーを利用します。

署名をブートローダーイメージに付加していきます。

```
[ATDE ~]$ cd ./imx-boot-[VERSION]
[ATDE ~/imx-boot-[VERSION]]$ ./secureboot.sh imxboot
[ATDE ~/imx-boot-[VERSION]]$ ./secureboot.sh imxboot_enc
```

ビルド後に以下のイメージが作られれば成功です。

```
[ATDE ~/imx-boot-[VERSION]]$ ls ../secureboot
secureboot/imx-boot_armadillo_x2.enc
secureboot/imx-boot_armadillo_x2.signed
secureboot/imx-boot_armadillo_x2.dek_offsets
```

imx-boot_armadillo_x2.enc は暗号化セキュアブート専用になります。この手順以外では基本的には利用できません。誤って書き込まないようご注意ください。

また、imx-boot_armadillo_x2.sign は SRK の異なるボードには当たり前ですが利用できません。誤って書き込まないようご注意ください。

5.6.4. 暗号化 Linux カーネルイメージの生成

secureboot.conf を開発環境に合わせて更新していきます。

```
[ATDE ~]$ vi imx-boot-[VERSION]/secureboot.conf
```

Linux カーネルとデバイスツリーブロブがある場所を参照するように変更します。

```
LINUX_IMAGE="/home/atmark/baseos/boot/Image"
LINUX_DTB="/home/atmark/baseos/boot/armadillo_iotg_g4.dtb"
```

設定の詳細については 「5.11.1. secureboot.conf の設定方法」 を参照してください。

Linux カーネルイメージを作ります。

```
[ATDE ~]$ cd imx-boot-[VERSION]
[ATDE ~/imx-boot-[VERSION]]$ ./secureboot.sh linux
```

ビルド後に以下のイメージがあれば成功です。

```
[ATDE ~/imx-boot-[VERSION]]$ ls ../secureboot
Image.signed
```

5.6.5. 暗号化セキュアブート対応 swu の作成

暗号鍵 (dek.blob) を組み込み処理を行う SWUpdate ファイル (swu)を作っていきます。

まず、dek.blob の組み込み処理を記述した desc ファイルの用意します。desc ファイルの修正のためにファイルをコピーします。

```
[ATDE ~]$ cp /usr/share/mkswu/examples/encrypted_imxboot_update.desc ./mkswu
```

以下のように変更してください。

```
[ATDE ~]$ vi mkswu/encrypted_imxboot_update.desc
```

```
swdesc_boot_enc "../secureboot/imx-boot_armadillo_x2.enc" ¥
"../secureboot/armadillo_x2.dek_offsets"
```

desc ファイル

SWUpdate ではファームウェアアップデート処理中にユーザー定義の追加処理を行うことが可能です。ユーザー定義の追加処理は desc ファイルに記述します。desc ファイルは独自のシンタックスをもつアットマークテクノ製ツール (mkswu) 用のスクリプトファイルです。Armadillo Base OS に最適化されています。mkswu パッケージのインストール先に desc ファイルの例があります。

desc のシンタックスの詳細仕様は Armadillo-IoT ゲートウェイ G4 製品マニュアルの「mkswu の desc ファイル [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.8.0/ch09.html#sct.mkswu-desc]」を参照してください。

実際に swu ファイルを作ります。

```
[ATDE ~]$ cd mkswu
[ATDE ~/mkswu]$ mkswu initial_setup.desc encrypted_imxboot_update.desc ¥
-o initial_setup_encboot.swu
```

以下のファイルができていれば成功です。

```
[ATDE ~/mkswu]$ ls
initial_setup_encboot.swu
```

initial_setup_encboot.swu は暗号化セキュアブート専用になります。この手順以外では基本的には利用できません。誤って書き込まないようご注意ください。

5.6.6. ルートファイルシステムのビルド

まず、ルートファイルシステムに組み込むために、署名された Linux カーネルのイメージをビルド用のリソースディレクトリに配置します。

```
[ATDE ~]$ mkdir -p build-rootfs-[VERSION]/ax2/resources/boot
[ATDE ~]$ cp secureboot/Image.signed build-rootfs-[VERSION]/ax2/resources/boot/Image
```

生成した暗号化セキュアブート対応 swu を配置します。

```
[ATDE ~]$ cp mkswu/initial_setup_encboot.swu build-rootfs-[VERSION]/ax2/installer
```

組み込むファイルの準備が終わったので、次にルートファイルシステムをビルドします。

```
[ATDE ~]$ cd build-rootfs-[VERSION]
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_rootfs.sh
```

以下のファイルがあればルートファイルシステムのビルドは成功しています。以下は例なので、ファイル名などはバージョンや日付によって変化します。

```
[ATDE ~/build-rootfs-[VERSION]]$ ls
baseos-[VERSION].[DATE].tar.zst
```

5.6.7. セキュアブートセットアップ用 SD boot ディスクの作成

SD boot するための起動イメージを作ります。

```
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh
```

以下のファイルができていれば成功です。ファイルに含まれる日付やバージョンは変化します。

```
[ATDE ~/build-rootfs-[VERSION]]$ ls
baseos-[VERSION].[DATE].img
```

次に、上記の起動イメージを使って SD boot で初回書き込みを行うためのディスクイメージを作成します。上記のイメージで SD boot して、これからつくるイメージを eMMC に書き込んでいきます。

同じコマンドを使って引数を変えることで可能です。--installer には上記のイメージを指定します。--boot には生成したブートローダーを指定します。

```
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh ¥
--installer baseos-[VERSION].[DATE].img ¥
--boot ../secureboot/imx-boot_armadillo_x2.signed
```

以下のファイルができていれば成功です。ファイルに含まれる日付やバージョンは変化します。

```
[ATDE ~/build-rootfs-[VERSION]]$ ls
baseos-[VERSION].[DATE]-installer.img
```

上記で作成した SD boot イメージを、Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「ブートディスクの作成 [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.8.0/ch09.html#sct.make-bootdisk]」を参考にして SD カードに書き込んでください。

SD boot 用の SD カードの作成が終わったら、次からは実際に「5.8. Armadillo-IoT ゲートウェイ G4 上の作業」を行います。

5.7. ストレージの暗号化のセットアップ

ここではストレージを暗号化する方法を説明します。

5.7.1. 実装仕様の概要

Armadiilo Base OS では、ストレージの暗号化に LUKS2 (Linux Unified Key Setup)を採用しました。CAAM を用いた方法もありますが、CAAM は AES XTS モードをサポートしておらず、AES CBC/ECB モードのみサポートされています。パフォーマンスが極端に悪いので、CAAM を利用せずに LUKS2 で ARM Cryptographicextension を利用しています。

LUKS2 に与える鍵は CAAM の Black key 技術によって保護されます。Black key はデバイス固有の鍵を利用してファイルを暗号化することができます。デバイス固有の鍵を利用しているので、Black key を別のボートにコピーしても、鍵を利用することはできません。Black key をフル機能で利用するためには、セキュアブートが必須なので、暗号化セキュアブートも利用します。

Armadillo Base OS ではストレージの２つの領域の暗号化をサポートします。

1. アプリケーションを配置するコンテナのファイルシステム
2. Armadioo Base OS のルートファイルシステム

build-rootfs のスクリプト (build_image.sh) では、1 (USERFS) か 1+2 (ALL) の設定が可能です。

セキュアブートのフローは「5.11.3. セキュアブートのフロー」 を参照してください。

イメージの詳しいフォーマットと展開先については「5.11.2. 署名済みイメージと展開先」 を参照してください。

アップデートのフローの詳細については 「5.11.5. ファームウェアアップデートのフロー」を参照してください。

5.7.2. セットアップの流れ

ストレージ暗号化のセットアップの流れは以下のとおりです。

PC 上の作業

事前準備

1. 「5.3. 署名環境を構築する」
2. 「5.3.7. 署名鍵を生成する」

ビルド

3. 「5.7.3. 署名済みブートローダーの作成」
4. 「5.7.4. 署名済み Linux カーネルイメージの作成」
5. 「5.7.5. セキュアブート対応 swu の作成」
6. 「5.7.6. ルートファイルシステムのビルド」
7. 「5.7.7. セキュアブートセットアップ用 SD boot ディスクの作成」

Armadillo-IoT ゲートウェイ G4 上の作業

8. 「5.8. Armadillo-IoT ゲートウェイ G4 上の作業」
9. 「5.8.4. ファームウェアの書き込みをはじめる」
10. 「5.10. セットアップ完了後のアップデートの運用について」

ビルドの全体的なプロセスフローについては「5.11.4. ビルドのプロセスフロー」に図があります。

5.7.3. 署名済みブートローダーの作成

セキュアブートと暗号化に対応したブートローダーをビルドします。Armadillo-IoT ゲートウェイ G4 製品マニュアルの「ブートローダーをビルドする [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.8.0/ch09.html#sct.build-imx-boot]」を参考にビルドしてください。

以下のファイルが作られていれば、ビルドに成功しています。

```
[ATDE ~]$ ls imx-boot-[VERSION]
imx-boot_armadillo_x2
```

次に、ビルドしたイメージを署名して、イメージに署名を付加していきます。セキュアブートに対応するために、特別な修正は必要ありません。アットマークテクノからリリースされているブートローダーをそのまま利用できます。既にビルドされたブートローダーを利用します。

署名をブートローダーイメージに付加していきます。

```
[ATDE ~]$ cd ./imx-boot-[VERSION]
[ATDE ~/imx-boot-[VERSION]]$ ./secureboot.sh imxboot
[ATDE ~/imx-boot-[VERSION]]$ ./secureboot.sh imxboot_enc
```

ビルド後に以下の３つのファイルが作られれば成功です。

```
[ATDE ~/imx-boot-[VERSION]]$ ls ../secureboot
secureboot/imx-boot_armadillo_x2.enc
secureboot/imx-boot_armadillo_x2.signed
secureboot/imx-boot_armadillo_x2.dek_offsets
```

imx-boot_armadillo_x2.enc は暗号化セキュアブート専用になります。この手順以外では基本的には利用できません。誤って書き込まないようご注意ください。

また、imx-boot_armadillo_x2.sign は SRK の異なるボードには当たり前ですが利用できません。誤って書き込まないようご注意ください。

5.7.4. 署名済み Linux カーネルイメージの作成

まず、initrd 用のルートファイルシステムをビルドします。

```
[ATDE ~]$ cd build-rootfs-[VERSION]
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_rootfs.sh
```

以下のファイルがあればルートファイルシステムのビルドは成功しています。以下は例なので、ファイル名などはバージョンや日付によって変化します。

```
[ATDE ~/build-rootfs-[VERSION]]$ ls
baseos-[VERSION].[DATE].tar.zst
```

次に、initrd の作成します。ストレージ暗号化に対応するためには、通常の Armadillo Base OS にはない、暗号鍵などの設定を行うための initrd が必要になります。

```
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_initrd.sh
```

以下のファイルがあれば initrd のビルドは成功しています。以下は例なので、ファイル名などはバージョンや日付によって変化します。

```
[ATDE ~/build-rootfs-[VERSION]]$ ls
initrd-baseos-[VERSION].[DATE].zst
```

次に、secureboot.conf を開発環境に合わせて更新していきます。

```
[ATDE ~]$ vi imx-boot/secureboot.conf
```

Linux カーネルとデバイスツリーブロブがある場所を参照するように変更します。

```
LINUX_IMAGE="/home/atmark/baseos/boot/Image"
LINUX_DTB="/home/atmark/baseos/boot/armadillo_iotg_g4.dtb"
```

上記で作った initrd を参照するように設定を変更します。

```
LINUX_INITRD="../build-rootfs-[VERSION]/initrd-baseos-[VERSION].[DATE].zst"
```

設定の詳細については 「5.11.1. secureboot.conf の設定方法」 を参照してください。

最後に、Linux カーネルイメージを作ります。

```
[ATDE ~]$ cd imx-boot-[VERSION]
[ATDE ~/imx-boot-[VERSION]]$ ./secureboot.sh linux
```

ビルド後に以下のイメージが更新されていれば成功です。

```
[ATDE ~/imx-boot-[VERSION]]$ ls ../secureboot
Image.signed
```

5.7.5. セキュアブート対応 swu の作成

ストレージ暗号化対応の SWUpdate 用のファイル (swu) を作っていきます。

まず、設定ファイルの準備します。swu を作成するために必要なファイルを集めます。desc ファイルは swu を作成するアットマークテノ製ツール mkswu の入力となる設定ファイルです。修正のためにファイルをコピーします。

```
[ATDE ~]$ cp /usr/share/mkswu/examples/enable_disk_encryption.desc ./mkswu
[ATDE ~]$ cp /usr/share/mkswu/examples/encrypted_imxboot_update.desc ./mkswu
```

以下のように変更してください。

```
[ATDE ~]$ vi mkswu/encrypted_imxboot_update.desc
```

```
swdesc_boot_enc "../secureboot/imx-boot_armadillo_x2.enc" ¥
"../secureboot/armadillo_x2.dek_offsets"
```

desc ファイル

SWUpdate ではファームウェアアップデート処理中にユーザー定義の追加処理を行うことが可能です。ユーザー定義の追加処理は desc ファイルに記述します。desc ファイルは独自のシンタックスをもつアットマークテクノ製ツール (mkswu) 用のスクリプトファイルです。Armadillo Base OS に最適化されています。mkswu パッケージのインストール先に desc ファイルの例があります。

desc のシンタックスの詳細仕様は Armadillo-IoT ゲートウェイ G4 製品マニュアルの「mkswu の desc ファイル [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.8.0/ch09.html#sct.mkswu-desc]」を参照してください。

実際に swu ファイルを作ります。

```
[ATDE ~]$ cd mkswu
[ATDE ~/mkswu]$ mkswu initial_setup.desc enable_disk_encryption.desc ¥
encrypted_imxboot_update.desc -o initial_setup_encryption.swu
```

以下のファイルができていれば成功です。

```
[ATDE ~/mkswu]$ ls
initial_setup_encryption.swu
```

initial_setup_encryption.swu はストレージ暗号化専用になります。この手順以外では基本的には利用できません。誤って書き込まないようご注意ください。

5.7.6. ルートファイルシステムのビルド

まず、ルートファイルシステムに組み込むために、署名された Linux カーネルのイメージをビルド用のリソースディレクトリに配置します。

```
[ATDE ~]$ mkdir -p build-rootfs-[VERSION]/ax2/resources/boot
[ATDE ~]$ cp secureboot/Image.signed build-rootfs-[VERSION]/ax2/resources/boot/Image
```

生成したストレージ暗号化対応 swu も配置します。

```
[ATDE ~]$ cp mkswu/initial_setup_encryption.swu build-rootfs-[VERSION]/ax2/installer
```

組み込むファイルの準備が終わったので、次にルートファイルシステムをビルドします。

```
[ATDE ~]$ cd build-rootfs-[VERSION]
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_rootfs.sh
```

以下のファイルがあればルートファイルシステムのビルドは成功しています。以下は例なので、ファイル名などはバージョンや日付によって変化します。

```
[ATDE ~/build-rootfs-[VERSION]]$ ls
baseos-[VERSION].[DATE].tar.zst
```

5.7.7. セキュアブートセットアップ用 SD boot ディスクの作成

SD boot するための起動イメージを作ります。

```
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh
```

以下のファイルができていれば成功です。ファイルに含まれる日付やバージョンは変化します。

```
[ATDE ~/build-rootfs-[VERSION]]$ ls
baseos-[VERSION].[DATE].img
```

次に、上記の起動イメージを使って SD boot で初回書き込みを行うためのディスクイメージを作成します。上記のイメージで SD boot して、これからつくるイメージを eMMC に書き込んでいきます。

同じコマンドを使って引数を変えることで可能です。--installer には上記のイメージを指定します。--boot には生成したブートローダーを指定します。

```
[ATDE ~/build-rootfs-[VERSION]]$ sudo ./build_image.sh ¥
--encrypt all ¥
--installer baseos-[VERSION].[DATE].img ¥
--boot ../secureboot/imx-boot_armadillo_x2.signed
```

ストレージの暗号化を行う範囲は２通り選択可能です。

--encrypt オプション

- userfs : アプリケーション領域
- all : アプリケーション領域と Linux rootfs 領域

以下のファイルができていれば成功です。ファイルに含まれる日付やバージョンは変化します。

```
[ATDE ~/build-rootfs-[VERSION]]$ ls
baseos-[VERSION].[DATE]-installer.img
```

上記で作成した SD boot イメージを、Armadillo-IoT ゲートウェイ G4 の製品マニュアルの「ブートディスクの作成 [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.8.0/ch09.html#sct.make-bootdisk]」を参考にして SD カードに書き込んでください。

SD boot 用の SD カードの作成が終わったら、次からは実際に「5.8. Armadillo-IoT ゲートウェイ G4 上の作業」を行います。

5.8. Armadillo-IoT ゲートウェイ G4 上の作業

以下は作業の流れです。セキュアブート、暗号化セキュアブート、ストレージの暗号化で共通の作業になります。

1. 「5.8.1. Armadillo-IoT ゲートウェイ G4 の準備」
2. 「5.8.2. SRK ハッシュの書き込み」
3. 「5.8.4. ファームウェアの書き込みをはじめる」

以降はアップデート処理によって自動的にセットアップが行われます。

5.8.1. Armadillo-IoT ゲートウェイ G4 の準備

Armadillo IoT G4 の電源を投入する前に 以下の作業を行ってください。

- CON1(SD インターフェース)に作った SD カードを挿入する
- JP1 ジャンパーをショート(SD ブートに設定)します。

また、minicom などのシリアル通信ソフトウェアを起動して、デバッグ入出力を受け付けられるようにしておいてください。

5.8.2. SRK ハッシュの書き込み

電源を投入するとすぐに以下のように uboot からデバッグ出力されます。**その間にシリアル通信ソフトウェア上で何らかのキーを押して、uboot の プロンプトに入ってください。**

```
Hit any key to stop autoboot:  2
u-boot=>
```

uboot プロンプトへの遷移に間に合わなかった場合は、そのままの状態(SD カードを挿したまま、JP1 をショート)で電源を切って再投入後に再試行して下さい。

セキュアブートを有効にするためには、i.MX 8M Plus の eFuse に SRK のハッシュ値を書く必要があります。i.MX 8M Plus の OCOTP (On-chip One-Time Programmable Element Controller) のレジスタ経由で書き込むことになります。uboot-imx には OCOTPA へのアクセスを行う fuse コマンドがあります。

以下は手順のはじめに出てきた secureboot.sh print_srk_fuse の結果です。**あくまで例なのでハッシュ値は生成された鍵毎に異なります。このまま入力しないでください。**

```
u-boot=> fuse prog -y 6 0 0xC04FA990 0x6C4BCFCC 0x90DAC78E 0x6C6CED49
u-boot=> fuse prog -y 7 0 0x535C7B3E 0x2695B4D4 0x9B3D028E 0x9FF5EFFB
u-boot=> fuse prog -y 0 0 0x200
```

eFuse は OTP なので一度作業を行うと変更はできません。注意して作業してください。

ハッシュ値は生成された鍵毎にことなります。例の値をそのまま書かないで下さい。

次に、これまでのビルド作業や実機上の作業を簡単に確認するために、HAB の状態を確認します。次のステップである close 処理を行うとセキュアブートに対応したファームウェアでしか起動できなくなります。まだ、close 処理されていない状態ではセキュアブートの必要がないので、問題の解析がしやすくなります。

まず、再起動します。

```
u-boot=> reset
```

uboot のプロンプトで以下のコマンドで確認してください。以下は正常時のログになります。

```
u-boot=> hab_status

Secure boot enabled

HAB Configuration: 0xcc, HAB State: 0x99
No HAB Events Found!
```

問題が発生した場合

原理確認などを行っているケースでは作業内容に何かしらの問題がある可能性があります。ログで今までの作業内容を見直してください。製造ラインなどで同じ作業を繰り返しているケースでは問題は発生しないはずですが、個体不良の解析のためにここでチェックすることをお勧めします。

5.8.3. close 処理

SRK ハッシュを書いただけでは署名確認に失敗しても起動自体は継続されます。close 処理を行うことで署名確認に失敗すると起動に失敗するようになります。

開発フェーズなどでセキュアブート処理を確認する場合は、この close 処理をスキップすることをすることをお勧めします。

i.MX 8M plus 内にある SNVS (Secure Non-Volatile Storage) で利用される鍵は、close 処理をしないとテスト用の鍵が使われます。テスト用の鍵はデバイス間で共通なため、そのままでは簡単に復号できてしまいます。close 処理を行うことで、デバイスに固有な鍵を利用するようになります。

close 処理は以下のコマンドで efuse を書き込むことで行います。

```
u-boot=> fuse prog -y 1 3 0x2000000
```

5.8.4. ファームウェアの書き込みをはじめる

次に、**アップデート作業をはじめるためにリセットします。**今後の作業は全て自動的に行われます。

```
u-boot=> reset
```

以下のログが表示されたら、**電源が入った状態で JP1 ジャンパーをショート(SD ブートに設定)を解除してください。**

```
mkfs.fat 4.2 (2021-01-31)
Finished writing mmc. powering off now
```

最終的に Linux のログインプロンプトが表示できたら作業は完了です。

- 問題が発生して作業が中断した場合は「5.8.5. 作業が中断した場合」を参照してください
- 問題なくログインプロンプトが表示できたら「5.9. 動作確認」進んでください

アップデートのフローの詳細については 「5.11.5. ファームウェアアップデートのフロー」を参照してください。

5.8.5. 作業が中断した場合

万が一、電源断などで作業が中断してしまった場合に作業を完了させる方法について説明します。

まず、電源を切った状態で 「5.8.1. Armadillo-IoT ゲートウェイ G4 の準備」 が維持されていることを確認してください。そのうえで、状況に応じて以下の作業を行ってください。

- SRK ハッシュを書いていない場合は、「5.8.2. SRK ハッシュの書き込み」から作業をやり直します
- close 処理を完了していない場合は、SRK ハッシュの書き込みをスキップしてください
- ファームウェアの書き込みが完了していない場合は、SRK ハッシュの書き込みと close 処理をスキップします。具体的な手順としては、電源を投入した直後 (だいたい数秒後) に電源が入った状態で JP1 ジャンパーのショートを解除して、処理が完了するのを待ってください

最終的に立ち上がってきたら、次の動作確認に進んでください。

5.9. 動作確認

5.9.1. ストレージ暗号化の確認

ストレージの暗号化が成功したかどうかは、Armadillo IoT G4 上で以下のコマンドを実行することで確認できます。TYPE が crypt となっているパーティションが暗号化されています。

```
[armadillo ~] # lsblk
NAME          MAJ:MIN RM  SIZE RO TYPE  MOUNTPOINTS
mmcblk2       179:0    0 14.8G  0 disk
|-mmcblk2p1   179:1    0  301M  0 part
| `-rootfs_0  253:0    0  300M  0 crypt /live/rootfs
|-mmcblk2p2   179:2    0  301M  0 part
|-mmcblk2p3   179:3    0   50M  0 part
| `-mmcblk2p3 253:1    0   49M  0 crypt /var/log
|-mmcblk2p4   179:4    0  200M  0 part
| `-mmcblk2p4 253:2    0  199M  0 crypt /opt/firmware
`-mmcblk2p5   179:5    0   14G  0 part
  `-mmcblk2p5 253:3    0   14G  0 crypt /var/tmp
                                       /var/app/volumes
                                       /var/app/rollback/volumes
                                       /var/lib/containers/storage_readonly
```

5.9.2. セキュアブートの確認

Linux カーネルが立ち上がることを確認してください。Linux カーネルまで立ち上がらない場合、uboot-imx のコマンドで HAB の状態を確認することができます。

Linux 起動まで正常な起動ログ

```
:（省略）
Booting from mmc ...

## Checking Image at 40480000 ...
Unknown image format!
53522 bytes read in 22 ms (2.3 MiB/s)

Authenticate image from DDR location 0x40480000...

Secure boot enabled ❶
```

```
HAB Configuration: 0xcc, HAB State: 0x99
No HAB Events Found! ❷

## Flattened Device Tree blob at 45000000
   Booting using the fdt blob at 0x45000000
   Using Device Tree in place at 0000000045000000, end 0000000045010111

Starting kernel ...
:（省略）
```

❶　セキュアブートが有効な場合に表示されます

❷　問題がない場合はイベントが表示されません

ブートローダーに問題がある場合の起動ログ

```
:（省略）
spl: ERROR:  image authentication unsuccessful
### ERROR ### Please RESET the board ###
:（省略）
```

Linux カーネルイメージに問題がある場合の起動ログ

```
:（省略）
Authenticate image from DDR location 0x40480000...
bad magic magic=0x14 length=0xa1 version=0x0
bad length magic=0x14 length=0xa1 version=0x0
bad version magic=0x14 length=0xa1 version=0x0
Error: Invalid IVT structure

Allowed IVT structure:
IVT HDR      = 0x4X2000D1
IVT ENTRY    = 0xXXXXXXXX
IVT RSV1      = 0x0
IVT DCD      = 0x0
IVT BOOT_DATA = 0xXXXXXXXX
IVT SELF      = 0xXXXXXXXX
IVT CSF      = 0xXXXXXXXX
IVT RSV2      = 0x0
Authenticate Image Fail, Please check ❶
:（省略）
```

❶　認証に失敗しています

u-boot コマンドの hab_status

問題がない場合

```
u-boot=> hab_status

Secure boot enabled
```

```
HAB Configuration: 0xcc, HAB State: 0x99
No HAB Events Found!
```

Linux カーネルイメージの署名確認で問題がある場合

```
u-boot=> hab_status

Secure boot disabled

HAB Configuration: 0xf0, HAB State: 0x66

--------- HAB Event 1 -----------------
event data:
        0xdb 0x00 0x14 0x45 0x33 0x0c 0xa0 0x00
        0x00 0x00 0x00 0x00 0x40 0x1f 0xdd 0xc0
        0x00 0x00 0x00 0x20

STS = HAB_FAILURE (0x33)
RSN = HAB_INV_ASSERTION (0x0C)
CTX = HAB_CTX_ASSERT (0xA0)
ENG = HAB_ENG_ANY (0x00)
```

5.10. セットアップ完了後のアップデートの運用について

コンテナ内のアプリケーションのアップデートはセットアップとは関係なく、とくに特別な対応なしで通常どおりの方法でアップデートが可能です。Armadillo-IoT ゲートウェイ G4 製品マニュアルの「Armadillo のソフトウェアをアップデートする [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.8.0/ch09.html#sct.build-imx-boot]」を参考にしてください。

ブートローダーや Linux カーネルを更新するのは若干の違いがあります。以下のオプションをつけることが必要になります。

ブートローダーの更新 (encrypted_imxboot_update.desc)

```
[ATDE ~/mkswu]$ mkswu encrypted_imxboot_update.desc -o update_boot.swu
```

Linux カーネルの更新 (encrypted_rootfs_linux_update.desc)

```
[ATDE ~/mkswu]$ mkswu encrypted_rootfs_linux_update.desc -o update_linux.swu
```

5.11. 補足情報

5.11.1. secureboot.conf の設定方法

secureboot.conf はセキュアブートイメージ生成スクリプト secureboot.sh の設定ファイルです。以下はコンフィグの一部です。

CST_ECC　既存の CA 証明書を利用するかどうかを設定する

- y: EC を利用する

・ n: RSA を利用する

CST_KEYLEN, CST_KEYTYPE　鍵長を設定する

表 5.4 セキュアブート用の鍵の種類

Algorithm	CST_KEYLEN	CST_KEYTYPE
RSA 1024	1024	1024_65537
RSA 2048	2048	2048_65537
RSA 3072	3072	3072_65537
RSA 4096	4096	4096_65537
EC NIST P-256	p256	prime256v1
EC NIST P-384	p384	secp384r1
EC NIST P-521	p521	secp521r1

CST_SOURCE_INDEX　SRK は最大４本まで持つことができます。この設定ではどの SRK を利用するのか設定します。設定値は 0 はじまりで、0 がデフォルトです。

CST_UNLOCK_SRK　SRK のリボーク時に利用します。デフォルト設定では攻撃や事故の防止のために、リボークができない状態になっています。#CST_UNLOCK_SRK=y のコメントを外すことでリボークが可能な状態になります。

LINUX_IMAGE,LINUX_DTB　FIT イメージに組み込むための Linux kernel イメージとデバイスツリーブロブ (DTB) の絶対パス。

LINUX_DTB_OVERLAYS　Linux 向けの Device tree オーバーレイ用ファイルを組み込むために利用する。いったん、Linux の device mapper を利用してルートファイルシステムを暗号化してしまうと、ブートローダーでは鍵がないとファイルを読むことができない。そのために FIT イメージに組み込んでおくためのオプション。/boot/overlays.conf にも同様の修正を加えてください。以下は例です。組み込みたいファイルを追加してください。

```
LINUX_DTB_OVERLAYS=(
        armadillo_iotg_g4-nousb.dtbo
        armadillo_iotg_g4-sw1-wakeup.dtbo
)
```

LINUX_INITRD　暗号化されたルートファイルシステムを復号するための処理を行うための initrd の絶対パス。encrypted boot を利用しない場合には設定は不要です。

5.11.2. 署名済みイメージと展開先

以下にブートローダーの署名済みイメージと展開先についての例を示します。緑色の部分が BootROM によって署名検証される部分、橙色の部分は SPL によって署名検証される部分になります。

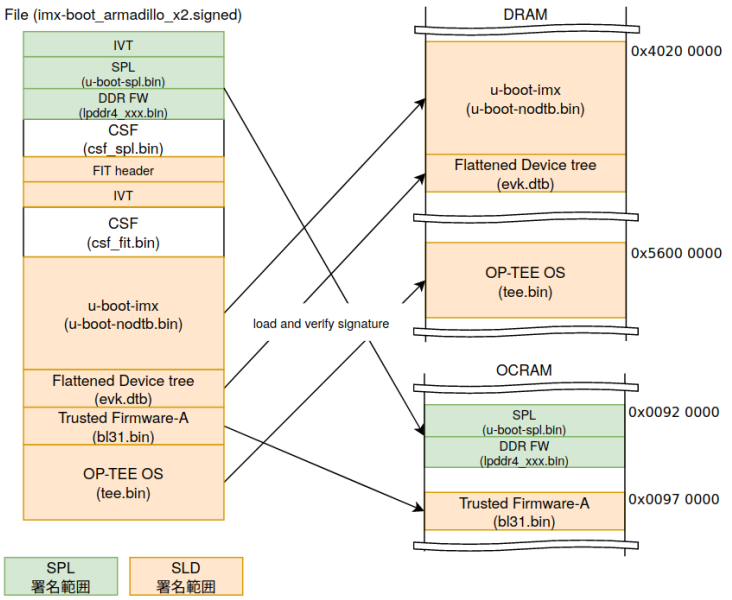

図 5.2 ブートローダーの署名済みイメージ

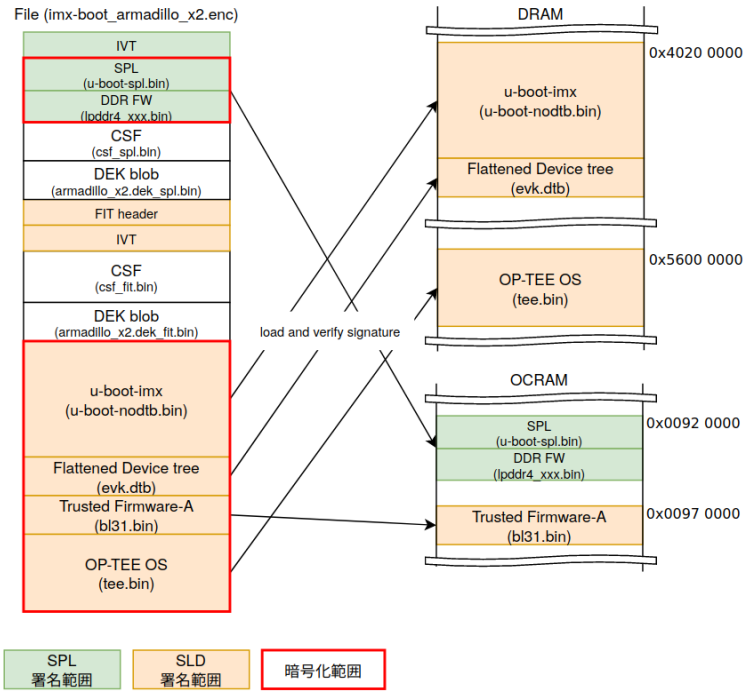

図 5.3 暗号化ブートローダーの署名済みイメージ

Linux の署名済みイメージと展開先の例は以下のとおりです。水色の部分は u-boot によって署名検証される部分になります。

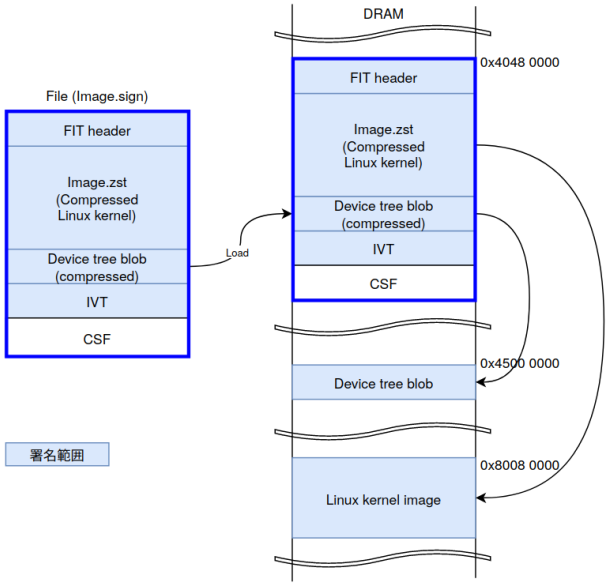

図 5.4 Linux カーネルの署名済みイメージ

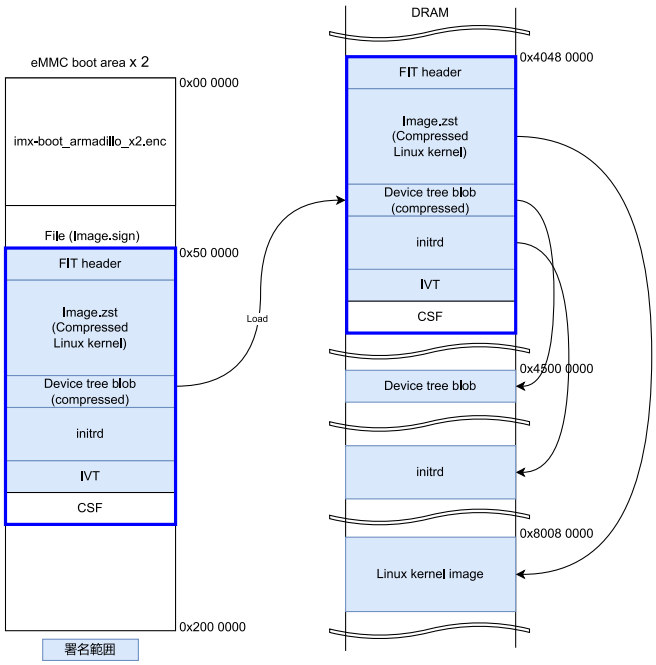

図 5.5 ストレージ暗号化に対応した Linux カーネルの署名済みイメージ

5.11.3. セキュアブートのフロー

以下に SPL (Secondary Program Loader) のブートフローの概要を示します。点線で囲っている部分はセキュアブートで有効になる処理です。

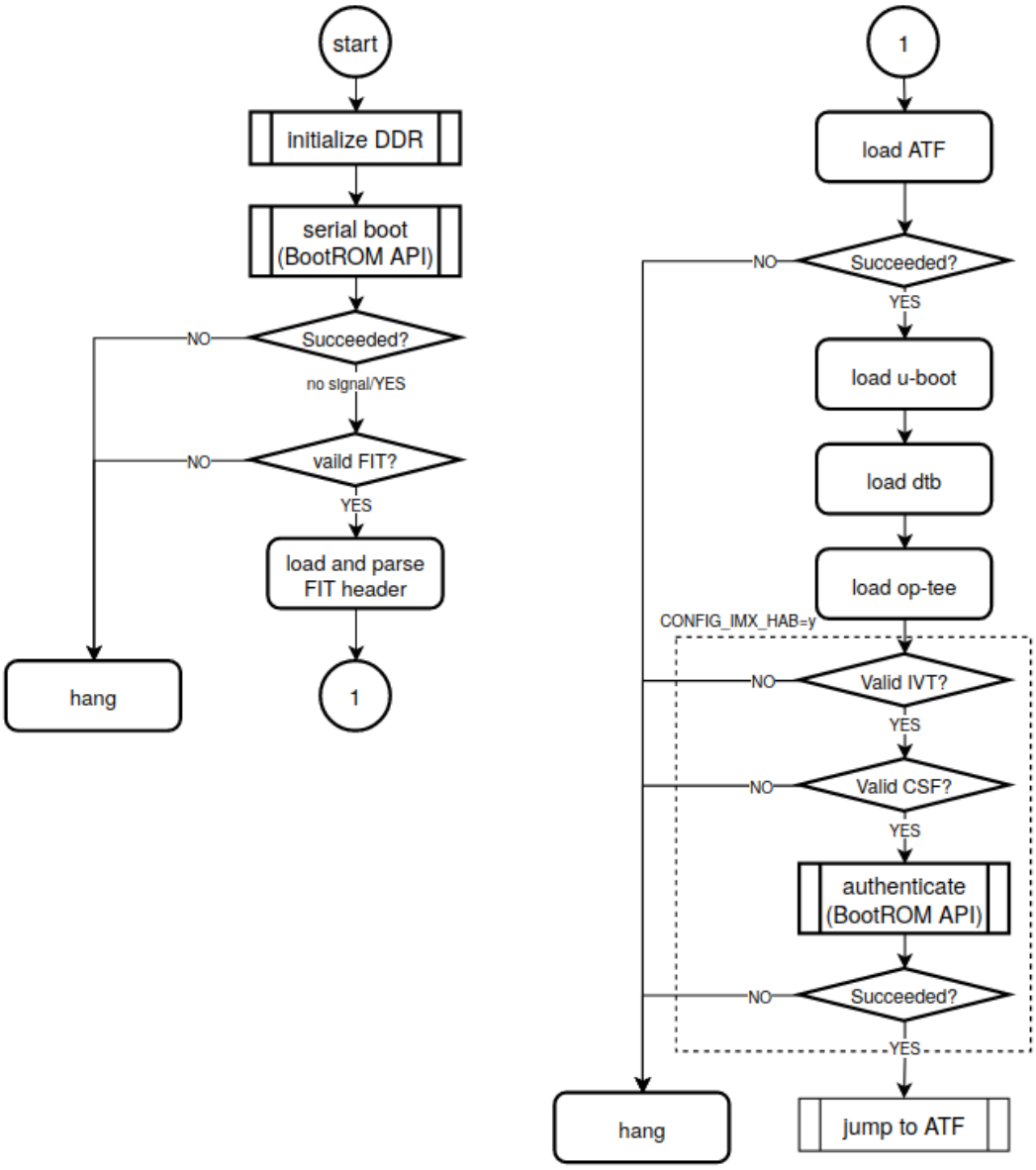

図 5.6 SPL セキュアブートのフロー

u-boot のブートフローの概要は以下のとおりです。SPL と同様に点線で囲っている部分はセキュアブートで有効になる処理です。

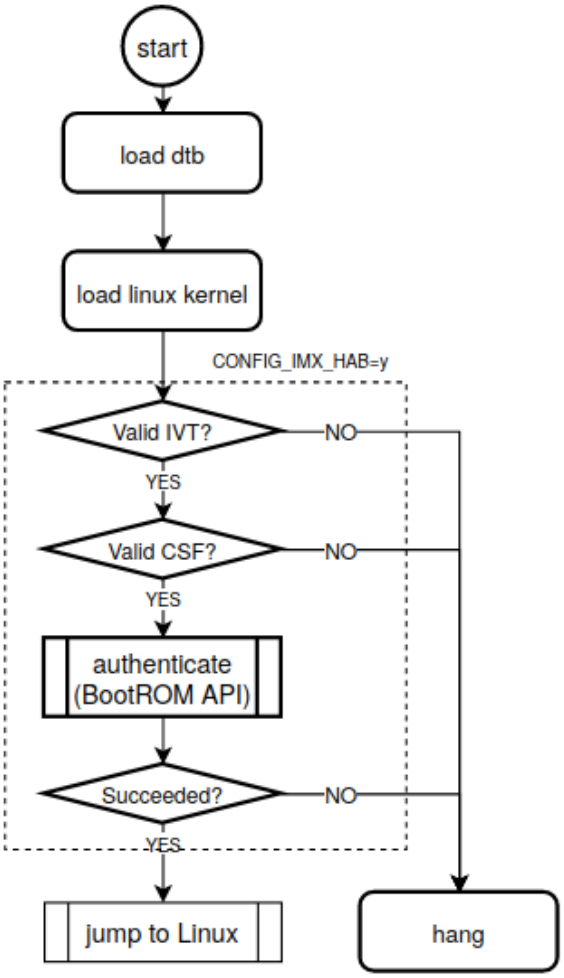

図 5.7 u-boot セキュアブートのフロー

5.11.4. ビルドのプロセスフロー

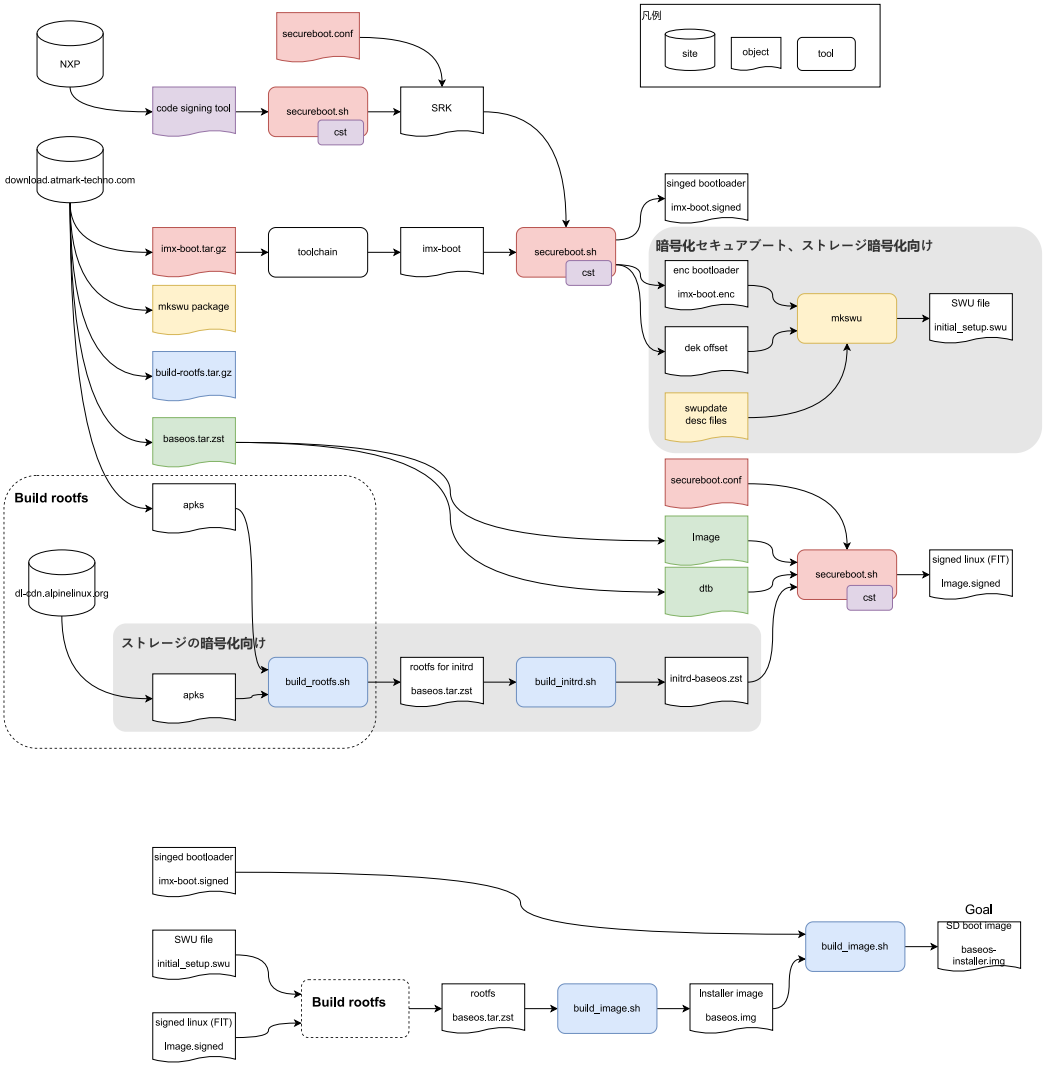

図 5.8 セキュアブートのビルドフロー

5.11.5. ファームウェアアップデートのフロー

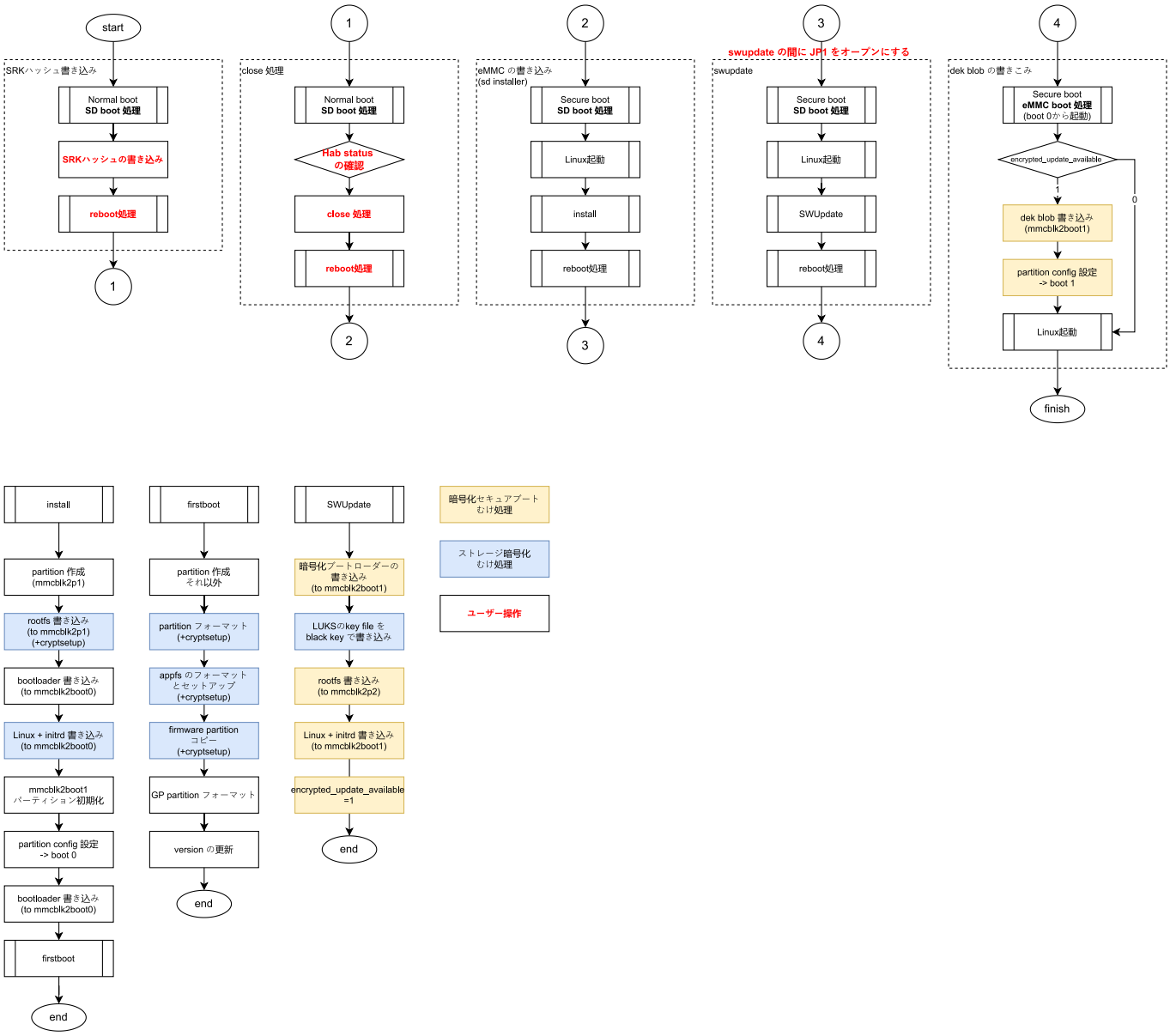

図 5.9 ファームウェアアップデートのフロー

5.11.6. SRK の無効化と切り替え

何らかのインシデント対応による鍵更新、また、鍵の定期更新などが必要な場合、その時点で利用している鍵を無効化して、別の鍵に切り替えることが可能です。ただし、その場合は複数 (i.MX 8M Plus の場合、最大４つ) の SRK が書かれていることが前提となります。

5.11.6.1. SRK の無効化 (revocation)

ここでは SRK1 (index 0) から SRK2 (index 1) に変更する例を説明します。

1. secureboot.conf の revocation のロックを解除する設定を有効にします

デフォルトでは eFuse の revoke レジスタはロックされているので書き込みできません。ロックは HAB の設定で解除することができます。常にロックを解除すると攻撃者に悪用される可能性があるので通常はロックされるべきです。

imx-boot-[VERSION]/secureboot.conf を開いて、CST_UNLOCK_SRK=y のコメントを外してください。secureboot.conf の詳細については 「5.11.1. secureboot.conf の設定方法」 を参照してください。

```
[ATDE ~]$ vi imx-boot-[VERSION]/secureboot.conf
```

2. 署名済みイメージを書き込む

   環境に合わせて、署名済みか、暗号化＋署名済みのイメージを作成して、イメージを書き込んでください。

3. 再起動

4. Unlock を確認する

   再起動時の uboot-imx のプロンプトを立ち上げてレジスタ値を確認します。以下のコマンドを実行してください。bit1 (SRK_REVOKE_LOCK) が落ちていると Unlock 状態です。

```
u-boot=> md 0x30350050 1
30350050: 00007dbc ❶
```

   ❶ 7dbc の bit 1 が落ちているので unlock 状態

5. SRK を無効化する

   ビットフィールドはビットは 0 はじまりで、鍵の番号は 1 はじまり (1,2,3,4) になります。bit0 が SRK1、bit1 が SRK2、bit2 が SRK3、bit3 が SRK4 です。

以下のコマンドはあくまで例なので、そのまま実行しないで下さい。

   SRK1 を無効化する場合は以下のコマンドを実行してください。最終引数が無効化する鍵の設定値です。

```
u-boot=> fuse prog 9 3 1
```

5.11.6.2. SRK の切り替え

ここでは SRK1 (index 0) から SRK2 (index 1) に変更する例を説明します。

1. SRK の変更

secureboot.conf の CST_SOURCE_INDEX を 0 から 1 に変更してください。secureboot.conf の詳細については 「5.11.1. secureboot.conf の設定方法」 を参照してください。

```
[ATDE ~]$ vi imx-boot-[VERSION]/secureboot.conf
```

2. 再署名する

   環境に合わせて、署名済みか、暗号化＋署名済みのイメージを作成して、イメージを書き込んでください。

6. ソフトウェア実行環境の保護

この章ではソフトウェア実行環境を守るセキュリティ技術を適用する方法を説明します。 説明するセキュリティ技術は以下のとおりです。

- OP-TEE
  - CAAM を利用した OP-TEE
  - SE050 を利用した OP-TEE

6.1. OP-TEE

6.1.1. Arm TrustZone と TEE の活用

Linux を利用するシステムでは、自社開発したソフトウェアだけでなく複数の OSS が導入されるケースがあります。このような状況下では、自社開発したタスクだけでなく、同時に OSS のタスクが実行されることになり、システムのどこかに悪意のあるコードが含まれるのか、その可能性を排除することは困難です。仮にすべての OSS が信頼できるものであったとしても、不具合がないことを保証することはほぼ不可能であり、結局のところどのソフトウェアが脆弱性のきっかけになるかは分からないのです。

そういった背景から Arm は TrustZone 技術を導入しました。リソースアクセス制限によって敵対的なソフトウェアからソフトウェア実行環境を隔離することができます。TrustZone の導入によって、自社開発以外のソフトウェアが多数動作する状況においても、通常はセキュアワールドにその影響が及びません。TrustZone を利用したものとして、GlobalPlatform の TEE (Trusted Execution Environment) を実現するソフトウェアがいくつか存在します。TEE を活用すると secure world において信頼できるソフトウェアだけを実行させることが可能です。

では、どういったケースで TEE を採用するべきでしょうか。一概には言えませんが、情報資産とその資産を処理するライブラリなどをまとめて保護するケースで利用することが考えられます。具体的には、電子決済処理、証明書の処理、有料コンテンツの処理、個人情報の処理などで利用するケースが考えられます。

6.1.2. OP-TEE とは

商用、OSS の TEE などいくつかの TEE 実装が存在します。Armadillo Base OS では OP-TEE を採用します。OSS である OP-TEE は Arm コア向け TEE 実装の 1 つです。Arm TrustZone テクノロジーによって情報資産とその処理をセキュアワールドに隔離して攻撃から保護します。OP-TEE は GlobalPlatform によって定義される TEE Client API や TEE Core API といった GlobalPlatformAPI に準拠したライブラリを提供しています。これらの API を利用することでユーザーはカスタムアプリケーションを開発することが可能です。imx-optee-xxx は NXP による i.MX ボートのサポート対応が含まれる OP-TEE の派生プロジェクトです。

OP-TEE の詳しい情報は公式のドキュメントを参照してください。

OP-TEE documentation

https://optee.readthedocs.io/en/latest/

6.2. OP-TEE の構成

OP-TEE を構成する主要なリポジトリは以下のとおりです。

- imx-optee-os
  - セキュアワールドで動作する TEE。optee-os の派生
- imx-optee-client
  - TEE を呼び出すためのノンセキュア、セキュアワールド向けの API ライブラリ。optee-client の派生
- imx-optee-test
  - OP-TEE の基本動作のテスト、パフォーマンス測定を行う。optee-test の派生。NXP 独自のテストが追加される
- optee_examples
  - サンプルアプリケーション

このうち、imx- というプレフィックスが付加されるリポジトリは、アップストリームのリポジトリに対して NXP による i.MX シリーズ向けの対応が入ったリポジトリになります。OP-TEE を利用するためには imx-optee-os をブートローダーに配置するだけでなく、Linux 上で動作するコンパニオン環境 (imx-optee-client) が必要です。また、TEE を利用するために CA (Client Application), TA (Trusted Application) を、imx-optee-os と imx-optee-client を用いてビルドします。必要に応じてテスト環境 (imx-optee-test) も追加してください。

詳しい OP-TEE のシステム構成については 「6.7.1. ソフトウェア全体像」を参照してください。

OP-TEE git に関する詳しい情報は公式のドキュメントを参照してください。

OP-TEE gits

https://optee.readthedocs.io/en/latest/building/gits/

6.2.1. Armadillo Base OS への組み込み

前節で説明した OP-TEE の主要なリポジトリを実際に Armdillo BaseOS に適用する場合、Armadillo Base OS とどのように関係するのか全体像を説明します。

以下が Armadillo Base OS との関係を表した全体像。

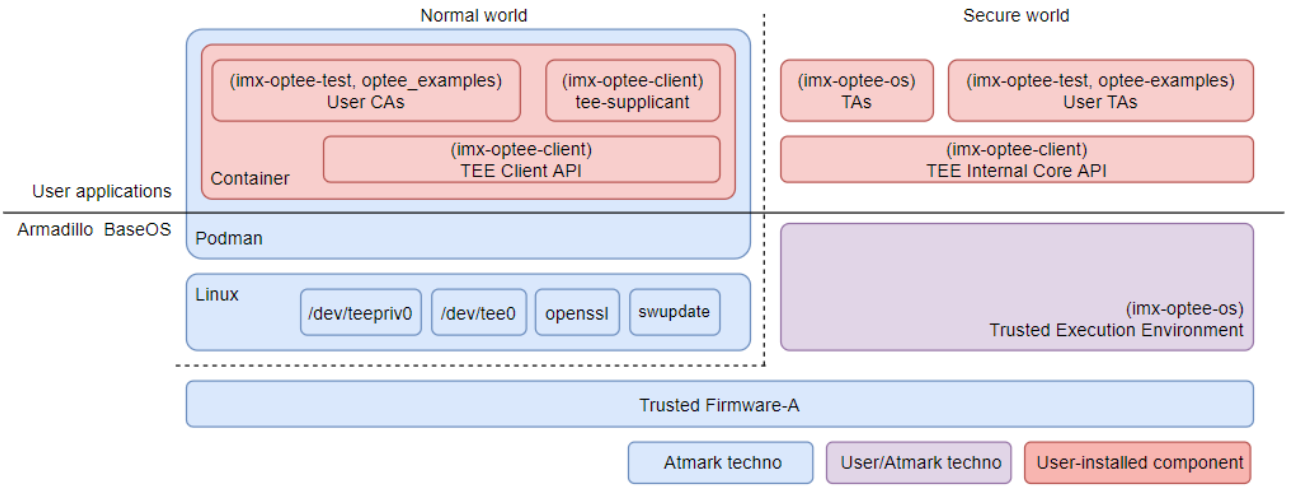

図 6.1 Armadillo Base OS と OP-TEE の担当範囲

- 青色の部分は Armadillo Base OS によってカバーされる範囲
- 赤色の部分は OP-TEE を組み込むために用意しなければならない部分
- 紫色の部分は デフォルトで Armadillo Base OS に組み込まれているが更新が必要な部分

imx-optee-os を組み込む作業は煩雑です。ユーザーの手間を省くために Armadillo Base OS には常に imx-optee-os バイナリがブートローダーに組み込むように実装されています。工場出荷状態イメージや製品アップデート向け Armadillo Base OS イメージにも含まれます。しかし、セキュリティ上の懸念から OP-TEE が利用できる状態で組み込まれていません。詳しくは 「6.3.1. 鍵の更新」 で説明します。

6.3. OP-TEE を利用する前に

OP-TEE を組み込むための準備を行います。

6.3.1. 鍵の更新

imx-optee-os リポジトリには TA を署名するためのデフォルトの秘密鍵が配置されています。その秘密鍵はあくまでテストや試行のためのものです。そのまま同じ秘密鍵を使い続けると、攻撃者がデフォルトの秘密鍵で署名した TA が動作する環境になってしまいます。各ユーザーが新しい鍵を用意する必要があります。

1. ソースコードを取得する Armadillo-IoT ゲートウェイ G4 製品マニュアルの「ブートローダーをビルドする [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.8.0/ch09.html#ch.yakushima-softwarebuild]」の章を参考にしてビルド環境の構築と、ブートローダーのソースコードの取得を行ってください。

2. RSA 鍵ペアを生成します。

   次のコマンドを実行します。RSA/ECC を選択できます。

```
[PC ~]$ cd imx-boot/imx-optee-os/keys
[PC ~/imx-boot/imx-optee-os/keys]$ openssl genrsa -out rsa4096.pem 4096
[PC ~/imx-boot/imx-optee-os/keys]$ openssl rsa -in rsa4096.pem -pubout -out rsa4096_pub.pem
```

生成した秘密鍵 (上記の rsa4096.pem) は TA を更新していくために、今後も利用するものです。壊れにくい、セキュアなストレージにコピーしておくことをお勧めします。

鍵の更新は計画性を持って行ってください。たとえば開発時のみ利用する鍵、運用時に利用する鍵を使い分ける。また、鍵は定期的に更新が必要です。以下を参考にしてください。

BlueKrypt Cryptographic Key Length Recommendation

https://www.keylength.com/

鍵の更新に関する詳しい情報は公式のドキュメントを参照してください。

Offline Signing of TAs

https://optee.readthedocs.io/en/latest/building/trusted_applications.html#offline-signing-of-tas

6.4. CAAM を活用した TEE を構築する

i.MX 8M Plus には CAAM (Cryptographic Acceleration and Assurance Module) と呼ばれる高機能暗号アクセラレータが搭載されています。CAAM は SoC 内部にあるためセキュアかつ高速に暗号処理を行うことが可能です。

6.4.1. ビルドの流れ

OP-TEE が含まれるブートローダーをビルドしていきます。Armadillo Base OS を利用した OP-TEE のビルドの流れは以下のとおりです。

1. ブートローダーをビルドする
2. imx-optee-client をビルドする
3. TA, CA をビルドする
4. ビルド結果を集める

本マニュアルで説明するビルド環境でのディレクトリ構成の概略は以下のとおりです。

optee_examples はユーザーアプリケーションを配置する場所です。本マニュアルでは Linaro が提供するアプリケーションのサンプルプログラムである optee_examples としました。本来は各ユーザーのアプリケーションを配置する場所になります。

```
├──── imx-boot
│     ├──── imx-atf
│     ├──── imx-mkimage
│     ├──── imx-optee-os
│     └──── uboot-imx
├──── imx-optee-client
├──── imx-optee-test
├──── optee_examples
└──── out
```

6.4.2. ビルド環境を構築する

Armadillo-IoT ゲートウェイ G4 製品マニュアルの「ブートローダーをビルドする [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.8.0/ch09.html#ch.yakushima-softwarebuild]」を参考にしてビルド環境の構築と、ブートローダーのソースコードの取得、ビルドを事前に行ってください。

imx-optee-test をビルドするために必要になります。

```
[PC ~]$ sudo apt install g++-aarch64-linux-gnu
```

6.4.3. ブートローダーを再ビルドする

新しい鍵を取り込むためにブートローダーを再ビルドする必要があります。

uboot-imx の localversion について

Armadillo-IoT ゲートウェイ G4 セキュリティガイド 1.1.0 以前では localversion を利用したバージョン更新を説明していましたが、SWUpdate の記述ファイル (desc) を利用した方法を推奨します。localversion との併用は可能です。すでに運用中の方はそのままご利用いただけます。

1. ブートローダーをビルドする

   次のコマンドを実行します。デフォルトの設定ではデフォルトの鍵を利用してしまうので make 引数で鍵のパスを渡す必要があります。「6.3.1. 鍵の更新」 で生成した鍵のパスを変数 TA_SIGN_KEY で渡します。ここでは鍵名を rsa4096 としましたが、各ユーザーの環境に合わせた鍵名と読み替えてください。

   ```
   [PC ~]$ make CFG_CC_OPT_LEVEL=2 ¥
           TA_SIGN_KEY="${PWD}/imx-boot/imx-optee-os/keys/rsa4096.pem" ¥
           -C imx-boot imx-boot_armadillo_x2
   ```

引数 TA_SIGN_KEY で秘密鍵をわたすことで、ビルド時に TA の署名に利用されます。また、実行時の署名確認のためには公開鍵を取り出して実行バイナリに取り込みます。

ビルド結果をコピーしておきます。

```
[PC ~]$ install -D -t ${PWD}/out/lib/optee_armtz ¥
        -m644 ${PWD}/imx-boot/imx-optee-os/out/export-ta_arm64/ta/*
```

6.4.4. imx-optee-client をビルドする

imx-optee-client は TA, CA が利用するライブラリ。アプリケーションをビルドする前にビルド必要がある。

imx-optee-client は imx-optee-os のビルド結果を参照しているので、事前にブートローダーをビルドする必要があります。

1. imx-optee-client をクローンする

   imx-boot のディレクトリと並列に配置されるように imx-optee-client をクローンして、必要に応じて適切なブランチなどをチェックアウトしてください。

   ```
   [PC ~]$ git clone https://source.codeaurora.org/external/imx/imx-optee-client.git ¥
           -b lf-5.10.72_2.2.0
   ```

2. imx-optee-client をビルドする

   基本的な情報を参照するために TA_DEV_KIT_DIR を渡します。フォルトのターゲットでインストールも合わせて行うので DESTDIR を渡します。アプリケーションをビルドする際に API ライブラリとして利用されます。

   ```
   [PC ~]$ make -C imx-optee-client ¥
           DESTDIR="${PWD}/out" ¥
           CROSS_COMPILE="aarch64-linux-gnu-" ¥
           TA_DEV_KIT_DIR="${PWD}/imx-boot/imx-optee-os/out/export-ta_arm64"
   ```

6.4.5. アプリケーションをビルドする

本来はユーザーが独自に作ったアプリケーションをビルドしますが、ここでは参考までにサンプルアプリケーション optee_examples をビルドします。アプリケーション開発の参考にしてください。

・アプリケーションをビルドするためには imx-optee-os, imx-optee-client のビルド結果を参照しているので、一度は imx-boot, imx-optee-client をビルドする必要があります

・optee_examples にはインストールする make ターゲットがないので、ビルド後に手動で集める作業が必要があります

1. optee_examples をクローンする

   imx-boot や imx-optee-client ディレクトリと並列に配置されるように optee_examples をクローンして、必要に応じて適切なブランチなどをチェックアウトしてください。

```
[PC ~]$ git clone https://github.com/linaro-swg/optee_examples.git ¥
        -b 3.15.0
```

2. optee_examples をビルドする

   基本的な情報を参照するために TA_DEV_KIT_DIR、TA を署名するために TA_SIGN_KEY、ライブラリ参照のために TEEC_EXPORT を渡します。インストールターゲットがないので後ほど手動でビルド結果を収集する必要があります。

```
[PC ~]$ make -C optee_examples ¥
        TA_CROSS_COMPILE="aarch64-linux-gnu-" ¥
        HOST_CROSS_COMPILE="aarch64-linux-gnu-" ¥
        TA_DEV_KIT_DIR="${PWD}/imx-boot/imx-optee-os/out/export-ta_arm64" ¥
        TEEC_EXPORT="${PWD}/out/usr" ¥
        TA_SIGN_KEY="$PWD/imx-boot/imx-optee-os/keys/rsa4096.pem"
```

   ビルド結果をコピーしておきます。

```
[PC ~]$ install -D -t ${PWD}/out/lib/optee_armtz -m644 ${PWD}/optee_examples/out/ta/*
[PC ~]$ install -D -t ${PWD}/out/usr/bin -m755 ${PWD}/optee_examples/out/ca/*
```

6.4.6. imx-optee-test をビルドする

imx-optee-test は必ずしも必要ではありません。利用機会は限られますが、OP-TEE の基本動作を確認するため、パフォーマンスを計測するために imx-optee-test を利用することができます。組み込むかどうかの判断はお任せします。

アプリケーションをビルドするためには imx-optee-os, imx-optee-client のビルド結果を参照しているので、一度は imx-boot, imx-optee-client をビルドする必要があります。

1. imx-optee-test をクローンする

imx-boot や imx-optee-client ディレクトリと並列に配置されるように imx-optee-client をクローンして、必要に応じて適切なブランチなどをチェックアウトしてください。

```
[PC ~]$ git clone https://source.codeaurora.org/external/imx/imx-optee-test.git ¥
        -b lf-5.10.72_2.2.0
```

2. imx-optee-test をビルドする

基本的な情報を参照するために TA_DEV_KIT_DIR、TA を署名するために TA_SIGN_KEY、ライブラリ参照のために TEEC_EXPORT を渡します。ビルドとインストールは別のターゲットなのでそれぞれ make を実行する必要があります。

```
[PC ~]$ CFLAGS=-O2 make -R -C imx-optee-test ¥
        CROSS_COMPILE="aarch64-linux-gnu-" ¥
        TA_DEV_KIT_DIR="${PWD}/imx-boot/imx-optee-os/out/export-ta_arm64" ¥
        OPTEE_CLIENT_EXPORT="${PWD}/out/usr" ¥
        TA_SIGN_KEY="${PWD}/imx-boot/imx-optee-os/keys/rsa4096.pem"
```

インストールします。

```
[PC ~]$ make -C imx-optee-test install ¥
        DESTDIR="${PWD}/out" ¥
        TA_DEV_KIT_DIR="${PWD}/imx-boot/imx-optee-os/out/export-ta_arm64" ¥
        OPTEE_CLIENT_EXPORT="${PWD}/out/usr"
```

本リリース時点では xtest 1014 にてエラーになる問題があります。そのため環境変数で CFLAGS=-O2 を渡しています。make 引数で渡すとヘッダファイルの include 設定がなくなります。ブートローダーのビルド時に O2 としているのも同様の理由です。

NXP LS Platforms: pkcs_1014 test is failing #4909https://github.com/OP-TEE/optee_os/issues/4909

6.4.7. ビルド結果の確認と結果の収集

imx-optee-os, imx-optee-client の最小構成では以下のとおりです。

```
out/
|-- lib
|   `-- optee_armtz ❶
|       |-- 023f8f1a-292a-432b-8fc4-de8471358067.ta
|       |-- f04a0fe7-1f5d-4b9b-abf7-619b85b4ce8c.ta
|       `-- fd02c9da-306c-48c7-a49c-bbd827ae86ee.ta
`-- usr
    |-- include
    |   |-- ck_debug.h
```

```
    |   |-- optee_client_config.mk
    |   |-- pkcs11.h
    |   |-- pkcs11_ta.h
    |   |-- tee_bench.h
    |   |-- tee_client_api.h
    |   |-- tee_client_api_extensions.h
    |   |-- tee_plugin_method.h
    |   `-- teec_trace.h
    |-- lib ❷
    |   |-- libckteec.a
    |   |-- libckteec.so -> libckteec.so.0
    |   |-- libckteec.so.0 -> libckteec.so.0.1
    |   |-- libckteec.so.0.1 -> libckteec.so.0.1.0
    |   |-- libckteec.so.0.1.0
    |   |-- libteec.a
    |   |-- libteec.so -> libteec.so.1
    |   |-- libteec.so.1 -> libteec.so.1.0.0
    |   |-- libteec.so.1.0 -> libteec.so.1.0.0
    |   `-- libteec.so.1.0.0
    `-- sbin ❸
        `-- tee-supplicant
```

❶　Dynamic TA。Linux のファイルシステムに保存される

❷　TEE client API, TEE, internal core API

❸　Linux 上で動作する OP-TEE の補助的な機能をもつ

imx-optee-test, optee_examples を含めたビルド結果は以下のとおりです。

```
out
|-- bin
|   `-- xtest
|-- lib
|   `-- optee_armtz
|       |-- 023f8f1a-292a-432b-8fc4-de8471358067.ta
|       |-- 2a287631-de1b-4fdd-a55c-b9312e40769a.ta
|       |-- 380231ac-fb99-47ad-a689-9e017eb6e78a.ta
|       |-- 484d4143-2d53-4841-3120-4a6f636b6542.ta
|       |-- 528938ce-fc59-11e8-8eb2-f2801f1b9fd1.ta
|       |-- 5b9e0e40-2636-11e1-ad9e-0002a5d5c51b.ta
|       |-- 5ce0c432-0ab0-40e5-a056-782ca0e6aba2.ta
|       |-- 5dbac793-f574-4871-8ad3-04331ec17f24.ta
|       |-- 614789f2-39c0-4ebf-b235-92b32ac107ed.ta
|       |-- 690d2100-dbe5-11e6-bf26-cec0c932ce01.ta
|       |-- 731e279e-aafb-4575-a771-38caa6f0cca6.ta
|       |-- 873bcd08-c2c3-11e6-a937-d0bf9c45c61c.ta
|       |-- 8aaaf200-2450-11e4-abe2-0002a5d5c51b.ta
|       |-- a4c04d50-f180-11e8-8eb2-f2801f1b9fd1.ta
|       |-- a734eed9-d6a1-4244-aa50-7c99719e7b7b.ta
|       |-- b3091a65-9751-4784-abf7-0298a7cc35ba.ta
|       |-- b689f2a7-8adf-477a-9f99-32e90c0ad0a2.ta
|       |-- b6c53aba-9669-4668-a7f2-205629d00f86.ta
|       |-- c3f6e2c0-3548-11e1-b86c-0800200c9a66.ta
|       |-- cb3e5ba0-adf1-11e0-998b-0002a5d5c51b.ta
|       |-- d17f73a0-36ef-11e1-984a-0002a5d5c51b.ta
|       |-- e13010e0-2ae1-11e5-896a-0002a5d5c51b.ta
```

```
|       |-- e626662e-c0e2-485c-b8c8-09fbce6edf3d.ta
|       |-- e6a33ed4-562b-463a-bb7e-ff5e15a493c8.ta
|       |-- f04a0fe7-1f5d-4b9b-abf7-619b85b4ce8c.ta
|       |-- f157cda0-550c-11e5-a6fa-0002a5d5c51b.ta
|       |-- f4e750bb-1437-4fbf-8785-8d3580c34994.ta
|       |-- fd02c9da-306c-48c7-a49c-bbd827ae86ee.ta
|       `-- ffd2bded-ab7d-4988-95ee-e4962fff7154.ta
`-- usr
    |-- bin
    |   |-- optee_example_acipher
    |   |-- optee_example_aes
    |   |-- optee_example_hello_world
    |   |-- optee_example_hotp
    |   |-- optee_example_plugins
    |   |-- optee_example_random
    |   `-- optee_example_secure_storage
    |-- include
    |   |-- ck_debug.h
    |   |-- optee_client_config.mk
    |   |-- pkcs11.h
    |   |-- pkcs11_ta.h
    |   |-- tee_bench.h
    |   |-- tee_client_api.h
    |   |-- tee_client_api_extensions.h
    |   |-- tee_plugin_method.h
    |   `-- teec_trace.h
    |-- lib
    |   |-- libckteec.a
    |   |-- libckteec.so -> libckteec.so.0
    |   |-- libckteec.so.0 -> libckteec.so.0.1
    |   |-- libckteec.so.0.1 -> libckteec.so.0.1.0
    |   |-- libckteec.so.0.1.0
    |   |-- libteec.a
    |   |-- libteec.so -> libteec.so.1
    |   |-- libteec.so.1 -> libteec.so.1.0.0
    |   |-- libteec.so.1.0 -> libteec.so.1.0.0
    |   |-- libteec.so.1.0.0
    |   `-- tee-supplicant
    |       `-- plugins
    |           `-- f07bfc66-958c-4a15-99c0-260e4e7375dd.plugin
    `-- sbin
        `-- tee-supplicant
```

ターゲット上のコンテナに展開するために、tarball で固めます。

```
[PC ~]$ tar -caf optee.tar.gz -C out .
```

Armadillo-IoT ゲートウェイ G4 製品マニュアルの「アプリケーションをコンテナで実行する [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.8.0/ch09.html#ch.yakushima-container]を参考にしてコンテナ作成の時に組み込むことをお勧めします。

6.4.8. OP-TEE を組み込む

SWUpdate 用のファイルをつくり、ターゲットデバイスのブートローダーを更新します。更新後に debian コンテナを起動して OP-TEE 関連ファイルを展開します。

セットアップの完了後はテストプログラム (xtest) とサンプルプログラム (optee_example_hello_world) を動作させます。

1. ATDE を立ち上げる

2. imx-boot_update.desc ファイルをつくる

   SWUpdate の記述ファイル (desc) をつくります。desc ファイルはアットマークテクノ独自の SWUpdate を制御するためのファイルです。

   ```
   [ATDE ~]$ vi imx-boot_update.desc
   ```

   以下の例を参考に、path などを変更してください。

   ```
   # version
   swdesc_option version=1

   # swdesc_boot <bootfile>
   swdesc_boot /path/imx-boot/imx-boot_armadillo_x2

   # swdesc_files [--basedir <basedir>] [--dest <dest>] <file> [<more files>]
   swdesc_files --dest optee /path/optee.tar.gz
   ```

swdesc_option version=<version>

新しいソフトウェアのバージョンを指定します。デバイス上のソフトウェアのバージョンより大きな値を設定するとアップデートが実行されます。OP-TEE を更新するたびにバージョンを上げてください。この option を使うと uboot-imx の localversion を利用する必要はありません。

<version>: バージョンには xx.yy.zz や yyyymmdd のフォーマットが利用できます。

swdesc_option <bootfile>

ブートローダーの更新のために利用します。

bootfile: 書き込むブートローダーを指定します。

swdesc_files [--basedir <basedir>] [--dest <dest>] <file> [<more files>]

Linux ファイルシステム上のファイルを更新するために利用します。

dest: ファイルの書込み先を指定します。書き込み先は、/var/app/rollback/volumes 以下に制限されます。

file: 書き込むファイルを指定します。

3. swu ファイルを作成する

   以下のコマンドで swu ファイルを作ります。

```
[ATDE ~]$ mkswu /path/imx-boot_update.desc -o ./update_imx-boot.swu
Enter pass phrase for /home/atmark/mkswu/swupdate.key: (password を入力)
Successfully generated update_imx-boot.swu
```

4. SWUpdate を実行する

   usb memory などに update_imx-boot.swu をコピーしてターゲットデバイス上で SWUpdate を実行します。

   以下のように armadillo 上で swupdate を実行してください。アップデートが完了すると、システムは再起動します。

```
[armadillo ~]# swupdate -i /path/update_imx-boot.swu
Licensed under GPLv2. See source distribution for detailed copyright notices.

[INFO ] : SWUPDATE running :  [main] : Running on iot-g4-es2 Revision at1
[INFO ] : SWUPDATE started :  Software Update started !
[  549.345329] exFAT-fs (mmcblk2p2): invalid boot record signature
[  549.351277] exFAT-fs (mmcblk2p2): failed to read boot sector
[  549.356956] exFAT-fs (mmcblk2p2): failed to recognize exfat type
[  549.384407] F2FS-fs (mmcblk2p2): Can't find valid F2FS filesystem in 1th superblock
[  549.392313] F2FS-fs (mmcblk2p2): Can't find valid F2FS filesystem in 2th superblock
[  549.470793] exFAT-fs (mmcblk2p2): invalid boot record signature
[  549.476739] exFAT-fs (mmcblk2p2): failed to read boot sector
[  549.482478] exFAT-fs (mmcblk2p2): failed to recognize exfat type
[  549.509020] F2FS-fs (mmcblk2p2): Can't find valid F2FS filesystem in 1th superblock
[  549.517000] F2FS-fs (mmcblk2p2): Can't find valid F2FS filesystem in 2th superblock
[INFO ] : SWUPDATE running :  [read_lines_notify] : No base os update: copying current os over
[INFO ] : SWUPDATE running :  [read_lines_notify] : Waiting for btrfs to flush deleted subvolumes
[INFO ] : SWUPDATE running :  [read_lines_notify] : Removing unused containers
[INFO ] : SWUPDATE running :  [read_lines_notify] : swupdate triggering reboot!
```

usb memory の root に swu ファイルを置いておくと、usb 接続時、もしくは、システム起動時に自動的に swupdate が走ります。アップデートが完了するとシステムは再起動します。

swupdate 中に swupdate を二重起動はできません。アップデート中はお待ちください。

5. ビルド結果をコンテナに展開する

/var/app/rollback/volumes/ 以下にまとめた tarball がコピーされます。上記の例では /var/app/rollback/volumes/optee に展開されています。

再起動後にコンテナを起動して tarball をコンテナに展開します。

コンテナを起動します。/dev/tee0 と /dev/teepriv0 を利用する以外は任意の設定で問題ありません。

```
[armadillo ~]# podman run -it --name=dev_optee --device=/dev/tee0 ¥
        --device=/dev/teepriv0 -v "$(pwd)":/mnt docker.io/debian /bin/bash
```

コンテナを起動したらファイルを展開します。

```
[container ~]# tar xavf /path/optee.tar.gz -C /
./
./lib/
./lib/optee_armtz/
./lib/optee_armtz/b3091a65-9751-4784-abf7-0298a7cc35ba.ta
./lib/optee_armtz/731e279e-aafb-4575-a771-38caa6f0cca6.ta
./lib/optee_armtz/873bcd08-c2c3-11e6-a937-d0bf9c45c61c.ta
./lib/optee_armtz/5b9e0e40-2636-11e1-ad9e-0002a5d5c51b.ta
./lib/optee_armtz/690d2100-dbe5-11e6-bf26-cec0c932ce01.ta
./lib/optee_armtz/528938ce-fc59-11e8-8eb2-f2801f1b9fd1.ta
./lib/optee_armtz/f157cda0-550c-11e5-a6fa-0002a5d5c51b.ta
./lib/optee_armtz/d17f73a0-36ef-11e1-984a-0002a5d5c51b.ta
./lib/optee_armtz/8aaaf200-2450-11e4-abe2-0002a5d5c51b.ta
./lib/optee_armtz/c3f6e2c0-3548-11e1-b86c-0800200c9a66.ta
./lib/optee_armtz/f04a0fe7-1f5d-4b9b-abf7-619b85b4ce8c.ta
./lib/optee_armtz/b6c53aba-9669-4668-a7f2-205629d00f86.ta
./lib/optee_armtz/a734eed9-d6a1-4244-aa50-7c99719e7b7b.ta
./lib/optee_armtz/380231ac-fb99-47ad-a689-9e017eb6e78a.ta
./lib/optee_armtz/a4c04d50-f180-11e8-8eb2-f2801f1b9fd1.ta
./lib/optee_armtz/5dbac793-f574-4871-8ad3-04331ec17f24.ta
./lib/optee_armtz/023f8f1a-292a-432b-8fc4-de8471358067.ta
./lib/optee_armtz/5ce0c432-0ab0-40e5-a056-782ca0e6aba2.ta
./lib/optee_armtz/cb3e5ba0-adf1-11e0-998b-0002a5d5c51b.ta
./lib/optee_armtz/fd02c9da-306c-48c7-a49c-bbd827ae86ee.ta
./lib/optee_armtz/484d4143-2d53-4841-3120-4a6f636b6542.ta
./lib/optee_armtz/f4e750bb-1437-4fbf-8785-8d3580c34994.ta
./lib/optee_armtz/e6a33ed4-562b-463a-bb7e-ff5e15a493c8.ta
./lib/optee_armtz/e13010e0-2ae1-11e5-896a-0002a5d5c51b.ta
./lib/optee_armtz/614789f2-39c0-4ebf-b235-92b32ac107ed.ta
./lib/optee_armtz/25497083-a58a-4fc5-8a72-1ad7b69b8562.ta
```

```
./lib/optee_armtz/b689f2a7-8adf-477a-9f99-32e90c0ad0a2.ta
./lib/optee_armtz/2a287631-de1b-4fdd-a55c-b9312e40769a.ta
./lib/optee_armtz/ffd2bded-ab7d-4988-95ee-e4962fff7154.ta
./lib/optee_armtz/e626662e-c0e2-485c-b8c8-09fbce6edf3d.ta
./bin/
./bin/xtest
./usr/
./usr/lib/
./usr/lib/libteec.so.1.0.0
./usr/lib/tee-supplicant/
./usr/lib/tee-supplicant/plugins/
./usr/lib/tee-supplicant/plugins/f07bfc66-958c-4a15-99c0-260e4e7375dd.plugin
./usr/lib/libteec.a
./usr/lib/libteec.so
./usr/lib/libteec.so.1
./usr/lib/libckteec.a
./usr/lib/libteec.so.1.0
./usr/lib/libckteec.so
./usr/lib/libckteec.so.0.1.0
./usr/lib/libckteec.so.0
./usr/lib/libckteec.so.0.1
./usr/bin/
./usr/bin/optee_example_hello_world
./usr/bin/optee_example_acipher
./usr/bin/optee_example_secure_storage
./usr/bin/optee_example_aes
./usr/bin/optee_example_hotp
./usr/bin/optee_example_plugins
./usr/bin/optee_example_random
./usr/include/
./usr/include/pkcs11_ta.h
./usr/include/tee_plugin_method.h
./usr/include/tee_client_api.h
./usr/include/pkcs11.h
./usr/include/tee_bench.h
./usr/include/tee_client_api_extensions.h
./usr/include/ck_debug.h
./usr/include/optee_client_config.mk
./usr/include/teec_trace.h
./usr/sbin/
./usr/sbin/tee-supplicant
```

6. tee-supplicant を起動する

```
[container ~]# tee-supplicant -d
```

7. xtest で動作を確認する

   xtest で OP-TEE の基本動作を確認します。以下のログは全テストをパスしたログです。

```
[container ~]# xtest
Run test suite with level=0

TEE test application started over default TEE instance
##########################################################
```

```
#
# regression+pkcs11+regression_nxp
#
########################################################

* regression_1001 Core self tests
  regression_1001 OK
: (省略)
+-------------------------------------------------------
33939 subtests of which 0 failed
114 test cases of which 0 failed
0 test cases were skipped
TEE test application done!
```

xtest の全テストをパスできない場合は環境構築から見直していただくことをお勧めします。問題が解決できないようであればサポートにご連絡ください。

1. アプリケーションを起動する

   ビルド結果を展開したことで CA も TA も配置されました。目的の CA を起動してください。ここでは optee_examples の optee_example_hello_world を実行します。

```
[container ~]# optee_example_hello_world
D/TA:  TA_CreateEntryPoint:39 has been called
D/TA:  TA_OpenSessionEntryPoint:68 has been called
I/TA: Hello World!
D/TA:  inc_value:105 has been called
I/TA: Got value: 42 from NW
I/TA: Increase value to: 43
I/TA: Goodbye!
Invoking TA to increment 42
TA incremented value to 43
D/TA:  TA_DestroyEntryPoint:50 has been called
```

D/TA: や I/TA: といった OP-TEE のログが出力されないケース

OP-TEE OS は uart を直接叩いてます。Linux の仕組みでは出力していないため、ssh や syslog などを利用してもログを閲覧できません。

tee-supplicant は OP-TEE の linux 環境のコンパニオンプロセスです。OP-TEE を利用するためにはなくてはならないものです。自動起動することをお勧めします。詳しくは、Armadillo-IoT ゲートウェイ G4 製品マニュアルの「アプリケーションをコンテナで実行する [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-

g4_product_manual_ja-1.8.0/ch09.html#ch.yakushima-container] 」を参考にしてください。

6.5. パフォーマンスを測定する

xtest を利用することで AES, SHA アルゴリズムの OP-TEE OS のパフォーマンスを測定することができます。

AES のパフォーマンスを計測するために次のコマンドを実行します。この結果は例になります。

```
[container ~]# xtest --aes-perf
min=113.753us max=191.881us mean=116.426us stddev=4.10202us (cv 3.5233%) (8.38786MiB/s)
```

SHA のパフォーマンスを計測するためのコマンドを実行します。この結果も例になります。

```
[container ~]# xtest --sha-perf
min=50.876us max=123.003us mean=52.8036us stddev=2.4365us (cv 4.61427%) (18.494
```

6.6. Edgelogck SE050 を活用した TEE を構築する

NXP Semiconductors の EdgeLock SE050 は IoT アプリケーション向けのセキュアエレメントです。様々なアルゴリズムに対応した暗号エンジン、セキュアストレージを搭載します。GlobalPlatform によって標準化されている Secure Channel Protocol 03 に準拠し、バスレベル暗号化 (AES)、ホストとカードの相互認証 (CMAC ベース) を行います。

OP-TEE で SE050 を用いるユースケースとしては、IoT アプリケーション向けに特化された豊富な機能を活用した上で、ホスト側の処理を守りたい場合に利用することが考えられます。OP-TEE を組み合わせることで SE050 へアクセスする部分、保存された情報資産を取り出して実際に処理する部分を守ることができます。

ユーザーが SE050 にアクセスする場合は、OP-TEE の TEE Client API や TEE Core API といった GlobalPlatformAPI を呼び出すことになります。デバイスの変更などの状況で比較的容易に移植が可能になります。

SE050 の詳細については以下の NXP Semiconductors のページから検索して、ご確認ください。

SE050　datasheethttps://www.nxp.com/docs/en/data-sheet/SE050-DATASHEET.pdf

6.6.1. OP-TEE 向け plug-and-trust ライブラリ

NXP Semiconductors が開発するライブラリ plug-and-trust を利用して SE050 にアクセスします。OP-TEE への移植は Foundries.io によって行われ、Github にて公開されています。現状、SE050 の全ての機能を使えるわけではありません。主に暗号強度が弱い鍵長が無効化されています。

現状で対応する処理:

- RSA 2048, 4096 encrypt/decrypt/sign/verify
- ECC sign/verify
- AES CTR
- RNG
- SCP03 (i2c communications between the processor and the device are encrypted)
- DieID generation
- cryptoki integration

OP-TEE 向け plug-and-trust の詳細の情報は以下を参照してください。

OP-TEE Enabled Plug and Trust Libraryhttps://github.com/foundriesio/plug-and-trust

本ガイドで利用したバージョンは以下のとおりです。

- plug-and-trust: 0.0.2
- imx-optee: lf-5.10.72_2.2.0 (3.15.0 ベース)
- trusted-fimware-a: lf_v2.4 (2.4 ベース)

6.6.2. ビルドの流れ

基本的には CAAM の場合と同様の流れになる。

1. ブートローダーをビルドする
2. imx-optee-client をビルドする
3. TA, CA をビルドする
4. ビルド結果を集める

ディレクトリ構成の概略は以下のとおりです。

```
├──── imx-boot
│     ├──── imx-atf
│     ├──── imx-mkimage
│     ├──── imx-optee-os
│     └──── uboot-imx
├──── imx-optee-client
├──── imx-optee-test
├──── optee_examples
├──── plug-and-trust
└──── out
```

6.6.3. ビルド環境を構築する

OP-TEE 向け plug-and-trust をビルドするために必要なパッケージをインストールします。

```
[PC ~]$ sudo apt install cmake
```

6.6.4. OP-TEE 向け plug-and-trust をビルドする

1. OP-TEE 向け plug-and-trust をクローンする

   imx-boot や imx-optee-client ディレクトリと並列に配置されるように OP-TEE 向け plug-and-trust をクローンして、必要に応じて適切なブランチなどをチェックアウトしてください。

   ```
   [PC ~]$ git clone https://github.com/foundriesio/plug-and-trust.git -b optee_lib
   ```

2. OP-TEE 向け plug-and-trust をビルドする

   ```
   [PC ~]$ mkdir -p plug-and-trust/optee_lib/build
   [PC ~]$ cd plug-and-trust/optee_lib/build
   [PC ~/plug-and-trust/optee_lib/build]$ cmake ¥
           -DCMAKE_C_FLAGS="-mstrict-align -mgeneral-regs-only" ¥
           -DCMAKE_C_COMPILER=aarch64-linux-gnu-gcc ¥
           -DOPTEE_TREE="${PWD}/../../../imx-boot/imx-optee-os" ..

   -- The C compiler identification is GNU 10.2.1
   -- The CXX compiler identification is GNU 10.2.1
   : (省略)
   -- Generating done
   -- Build files have been written to: /path/plug-and-trust/optee_lib/build
   ```

   ```
   [PC ~/plug-and-trust/optee_lib/build]$ make
   make
   Consolidate compiler generated dependencies of target se050
   [  4%] Building C object CMakeFiles/se050.dir/path/plug-and-trust/hostlib/hostLib/
   libCommon/infra/global_platf.c.o
   [  8%] Building C object CMakeFiles/se050.dir/path/plug-and-trust/hostlib/hostLib/
   libCommon/infra/sm_apdu.c.o
   : (省略)
   ```

```
[100%] Linking C static library libse050.a
[100%] Built target se050
```

6.6.5. imx-optee-os のコンフィグの修正

SE050 を crypto driver として利用するためにコンフィグを修正する。

SE050 向けにビルドするために imx-boot/Makefile を追加する。

```
$(OPTEE)/out/tee.bin: $(OPTEE)/.git FORCE
        $(MAKE) -C $(OPTEE) O=out ARCH=arm PLATFORM=imx CFG_WERROR=y ¥
        PLATFORM_FLAVOR=mx8mpevk ¥
        CFG_NXP_SE05X=y ¥ ❶
        CFG_IMX_I2C=y ¥ ❷
        CFG_CORE_SE05X_I2C_BUS=2 ¥
        CFG_CORE_SE05X_BAUDRATE=400000 ¥
        CFG_CORE_SE05X_OEFID=0xA200 ¥
        CFG_IMX_CAAM=n ¥ ❸
        CFG_NXP_CAAM=n ¥
        CFG_CRYPTO_WITH_CE=y ¥ ❹
        CFG_STACK_THREAD_EXTRA=8192 ¥ ❺
        CFG_STACK_TMP_EXTRA=8192 ¥
        CFG_NUM_THREADS=1 ¥ ❻
        CFG_WITH_SOFTWARE_PRNG=n ¥ ❼
        CFG_NXP_SE05X_PLUG_AND_TRUST_LIB=~/plug-and-trust/optee_lib/build/libse050.a ¥ ❽
        CFG_NXP_SE05X_PLUG_AND_TRUST=~/plug-and-trust/
```

コンフィグの修正に関する詳細

❶ SE050 を利用するために有効にする

❷ imx-i2c ドライバ を有効にする

❸ CAAM は無効化する

❹ AES や SHA は高速な Arm CE を利用する

❺ スタックを通常よりも多く消費するためにスタックを増量する

❻ スレッドによる複数のコンテキストに対応していないためスレッドを 1 つとする

❼ ハードウェア乱数発生器を利用するために無効にする

❽ SE050 のドライバの実装は OP-TEE 向け plug-and-trust ライブラリ内に存在する

- SE050 の host 接続用 I2C は最大 3.2 MHz (high speed)ですが、i.MX 8M Plus の i2c の最大周波数は 400kHz のため、遅い通信速度で実装されています
- CAAM と SE050 の共存は、SE050 を有効にすることによって CAAM の個別のドライバの依存関係が不正になるため、実行時にエラーになる問題があります

6.6.6. uboot-imx の修正

Armadillo は消費電力の削減のため SE050 を Deep Power-down モードに設定してパワーゲーティングしている。Deep Power-down モードを解除して SE050 を利用するためには、i.MX 8M Plus に接続されている SE050 の ENA ピンをアサートする必要があります。ENA ピンをアサートすると SE050 は一定時間の後に起動するので SE050 を利用すためには若干の待ち時間が必要となります。OP-TEE OS は起動時にドライバの初期化等を行う実装になっている。そのため、OP-TEE OS が起動する前に生存している SPL (Secondary Program Loader) で Deep Power-down を解除することで待ち時間を稼いでいる。

以下はシステムの起動と SE050 の関係を示したシーケンス図。

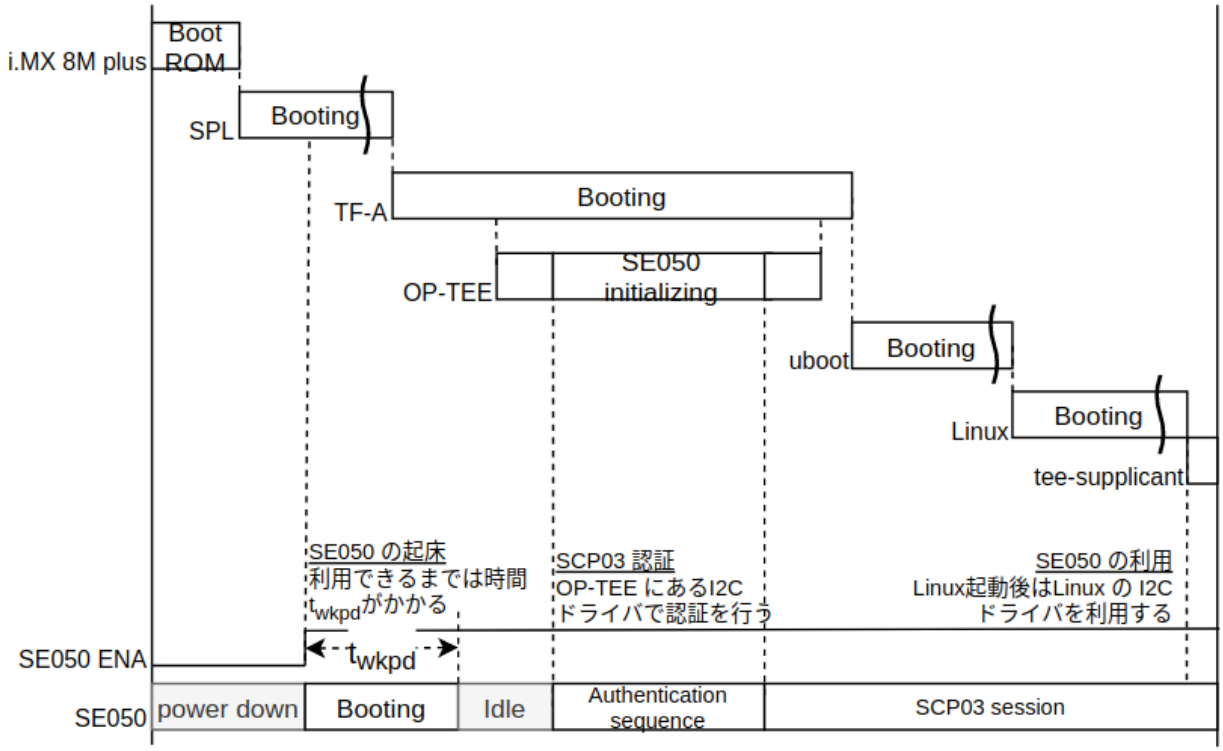

図 6.2 SE050 向け OP-TEE の起動シーケンス図

ENA をアサートするために以下のように変更する

```
diff --git a/board/atmark-techno/armadillo_x2/spl.c b/board/atmark-techno/armadillo_x2/spl.c
index a26bc85633..88f5b8560c 100644
--- a/board/atmark-techno/armadillo_x2/spl.c
+++ b/board/atmark-techno/armadillo_x2/spl.c
@@ -111,6 +111,11 @@ static struct fsl_esdhc_cfg usdhc_cfg[2] = {
        {USDHC3_BASE_ADDR, 0, 8},
 };

+#define SE_RST_N IMX_GPIO_NR(1, 12)
+static iomux_v3_cfg_t const se_rst_n_pads[] = {
+       MX8MP_PAD_GPIO1_IO12__GPIO1_IO12 | MUX_PAD_CTRL(NO_PAD_CTRL),
+};
+
```

```
 int board_mmc_init(bd_t *bis)
 {
        int i, ret;
@@ -228,6 +233,12 @@ void spl_board_init(void)
        clock_enable(CCGR_GIC, 1);
 #endif

+       imx_iomux_v3_setup_multiple_pads(se_rst_n_pads,
+                                        ARRAY_SIZE(se_rst_n_pads));
+
+       gpio_request(SE_RST_N, "se_rst_n");
+       gpio_direction_output(SE_RST_N, 1);
+
        puts("Normal Boot¥n");
 }
```

6.6.7. imx-optee-os の imx-i2c ドライバの修正

imx-optee-os の imx-i2c ドライバには i.MX 8M Plus の対応が入っていないため、レジスタ等の定義を追加する必要がある。imx-optee-os には lf-5.10.y_2.0.0 から SE050 ドライバが取り込まれている。

以下のように修正する。

```
diff --git a/core/arch/arm/plat-imx/conf.mk b/core/arch/arm/plat-imx/conf.mk
index b4fbfed5..3f6388f3 100644
--- a/core/arch/arm/plat-imx/conf.mk
+++ b/core/arch/arm/plat-imx/conf.mk
@@ -555,7 +555,7 @@ endif

 else

-$(call force,CFG_CRYPTO_DRIVER,n)
-$(call force,CFG_WITH_SOFTWARE_PRNG,y)
+#$(call force,CFG_CRYPTO_DRIVER,n) ❶
+#$(call force,CFG_WITH_SOFTWARE_PRNG,y) ❷

 ifneq (,$(filter y, $(CFG_MX6) $(CFG_MX7) $(CFG_MX7ULP)))
diff --git a/core/arch/arm/plat-imx/registers/imx8m.h b/core/arch/arm/plat-imx/registers/imx8m.h
index 9b6a50ee..59fcea88 100644
--- a/core/arch/arm/plat-imx/registers/imx8m.h
+++ b/core/arch/arm/plat-imx/registers/imx8m.h
@@ -42,6 +42,17 @@
 #define IOMUXC_I2C1_SDA_CFG_OFF        0x480
 #define IOMUXC_I2C1_SCL_MUX_OFF        0x214
 #define IOMUXC_I2C1_SDA_MUX_OFF        0x218
+#elif defined(CFG_MX8MP)
+#define I2C1_BASE              0x30a20000
+#define I2C2_BASE              0x30a30000
+#define I2C3_BASE              0x30a40000
+
+#define IOMUXC_I2C1_SCL_CFG_OFF        0x460
+#define IOMUXC_I2C1_SDA_CFG_OFF        0x464
+#define IOMUXC_I2C1_SCL_MUX_OFF        0x200
+#define IOMUXC_I2C1_SDA_MUX_OFF        0x204
+#define IOMUXC_I2C1_SCL_INP_OFF        0x5A4
+#define IOMUXC_I2C1_SDA_INP_OFF        0x5A8
```

```
 #endif

 #endif /* __IMX8M_H__ */
diff --git a/core/drivers/imx_i2c.c b/core/drivers/imx_i2c.c
index a318c32c..a9dab31c 100644
--- a/core/drivers/imx_i2c.c
+++ b/core/drivers/imx_i2c.c
@@ -34,6 +34,16 @@
 /* Clock */
 #define I2C_CLK_CGRBM(__x)     0 /* Not implemented */
 #define I2C_CLK_CGR(__x)       CCM_CCRG_I2C##__x
+#elif defined(CFG_MX8MP)
+/* IOMUX */
+#define I2C_INP_SCL(__x)       (IOMUXC_I2C1_SCL_INP_OFF + ((__x) - 1) * 0x8)
+#define I2C_INP_SDA(__x)       (IOMUXC_I2C1_SDA_INP_OFF + ((__x) - 1) * 0x8)
+#define I2C_INP_VAL(__x)       (((__x) == 1 || (__x) == 2) ? 0x2 : 0x4)
+#define I2C_MUX_VAL(__x)       0x010
+#define I2C_CFG_VAL(__x)       0x1c6
+/* Clock */
+#define I2C_CLK_CGRBM(__x)     0 /* Not implemented */
+#define I2C_CLK_CGR(__x)       CCM_CCRG_I2C##__x
 #elif defined(CFG_MX6ULL)
 /* IOMUX */
 #define I2C_INP_SCL(__x)       (IOMUXC_I2C1_SCL_INP_OFF + ((__x) - 1) * 0x8)
@@ -182,7 +192,7 @@ static void i2c_set_bus_speed(uint8_t bid, int bps)
        vaddr_t addr = i2c_clk.base.va;
        uint32_t val = 0;

-#if defined(CFG_MX8MM)
+#if defined(CFG_MX8MM) || defined(CFG_MX8MP)
        addr += CCM_CCGRx_SET(i2c_clk.i2c[bid]);
        val = CCM_CCGRx_ALWAYS_ON(0);
 #elif defined(CFG_MX6ULL)
```

以下は imx プラットフォームの makefile の問題です。回避するためにコメントアウトします。

❶ imx プラットフォームで CAAM 以外の crypto driver を利用することを想定していない

❷ CAAM の HWRNG を利用しないということは PRNG を使うことしか想定しない

6.6.8. ビルドとターゲットボードへの組み込み

修正した後は、ビルドからターゲットボードへの組み込みまで CAAM 向けの OP-TEE と同様の手順で作業することが可能です。 「6.4.3. ブートローダーを再ビルドする」 の作業から開始して組み込みしてください。

6.6.9. xtest の制限

「6.6.1. OP-TEE 向け plug-and-trust ライブラリ」 で説明したように一部のアルゴリズムに制限があります。そのため xtest の全てのテスト項目をパスするわけではありません。以下に失敗するテスト項目を列挙する。

仕様どおりのエラー:

- regression 1009 TEE Wait cancel

- キャンセル処理をするために OP-TEE はマルチスレッドが有効な構成でなくてはならない。SE050 へのアクセスをシリアライズする利用するために imx-optee-os の make 時にスレッドを 1 つにしているため

対応していない鍵長のためにエラー:

- regression 4006 Test TEE Internal API Asymmetric Cipher operations
- regression 4007 rsa.1 Generate RSA-256 key
- regression 4011 Test TEE Internal API Bleichenbacher attack
- pkcs11 1021.3 RSA-1024: Sign & verify - oneshot - CKM_MD5_RSA_PKCS
- pkcs11 1022.2 RSA-1024: Sign & verify - oneshot - RSA-PSS/SHA1
- pkcs11 1023.2 RSA OAEP key generation and crypto operations

optee os の不具合 (既知の問題):

- regression 4009 Test TEE Internal API Derive key ECDH
- regression 6018 Large object
- pkcs11 1019.3 P-256: Sign & verify - oneshot - CKM_ECDSA_SHA1

以下は特に問題はないが時間がかかるためにフリーズしているかのように見える。

- regression 1006 Secure time source
- regression nxp 0001, regression_nxp_0003

xtest を行う際にはデフォルトでテストに失敗しても先に進む設定となっています。ただ、時間のかかるテストは無効にすることも可能です。

```
[container ~]# xtest -x 1006 -x regression_nxp_0003
```

6.7. imx-optee-os 技術情報

6.7.1. ソフトウェア全体像

OP-TEE のアーキテクチャの概要を説明する。ここでは i.MX 8M Plus に搭載される Cortex-A53 コアのアーキテクチャである aarch64 を前提に話を進める。

以下にのシステム図を示す。

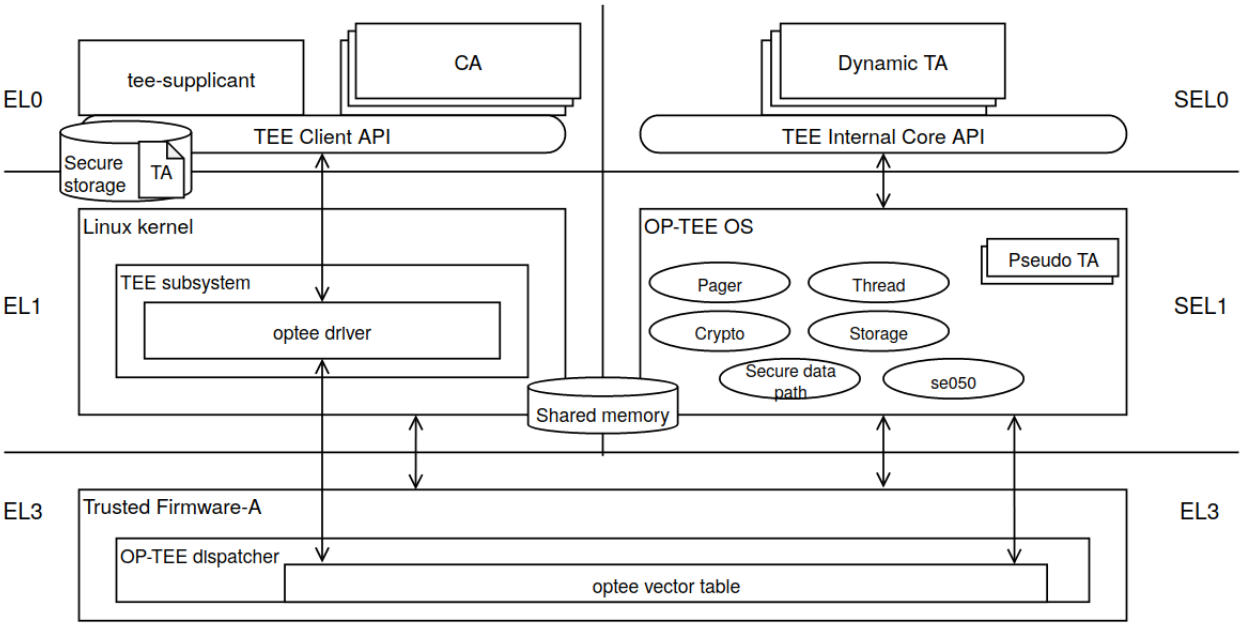

図 6.3 OPTEE のシステム図

以下のコンポーネントによってシステムが構成される。概要と主な責務について説明する。

- OP-TEE
  - GlobalPlatform の TEE 実装。secure EL1 に配置される
- Trusted Firmware-A
  - Linaro によって開発される Secure monitor の実装
  - secure state と non-secure state の遷移管理、PSCI に準拠した電源管理などを担当する
  - optee dispatcher と呼ばれる OP-TEE の呼び出しモジュールを内部に持つ
- Client Application (CA)
  - TA を呼び出すアプリケーション
- Trusted Application (TA)
  - TEE 上で CA からの呼び出しを処理
  - Dynamic TA と Pesudo TA がある
    - Pesudo は TA ではありません。通常は Dynamic TA を利用してください。Pesudo TA は OP-TEE OS に直接リンクされるため TEE Internal Core API は呼べません
  - Dynamic TA は Linux のファイルシステム上に配置される。tee-supplicant によって OP-TEE OS に引き渡される
- tee-supplicant

- Linux user 空間で動作する OP-TEE を補うプロセス。目的は Linux のリソースを OP-TEE OS が利用するため

6.7.2. フロー

TEE を呼び出すフローについて説明する。

CA が OP-TEE 上の TA とのセッションを確立する流れ

1. Linux 上の CA が、セッションを開くために uuid を指定して TEE Client API を呼び出す
   - システムコールで tee driver が呼ばれる
2. tee driver は セキュアモニタコールで Trusted Firmware-A (ATF) 上の OP-TEE dispacher を呼び出す
3. OP-TEE dispatcher は optee vector table に登録されている OP-TEE のハンドラを呼び出す
   - この段階ではまだ EL3 の状態
4. OP-TEE OS は自ら SEL1 に落ちて、内部処理をしてから、ここまでの逆順で tee-supplicant を呼び出す
5. tee-supplicant は uuid を基に TA をロードして共有メモリに配置して OP-TEE OS を呼び出す
6. OP-TEE は TA をロードする
7. セッションができる

CA が TEE Client API を通して TA 上である処理を実行する流れ

1. Linux 上の CA が TEE Client API を呼び出す
   - システムコールで tee driver が呼ばれる
2. tee driver は セキュアモニタコールで Trusted Firmware-A (ATF) 上の OP-TEE dispacher を呼び出す
3. OP-TEE dispatcher は optee vector table に登録されている OP-TEE のハンドラを呼び出す
4. ハンドラ (OP-TEE OS) は自ら SEL1 に落ちる。内部処理をしてから、SEL0 に落ちて TA を呼び出す
5. TA は API の引数を基にある処理を実行する
6. ここまでの逆順で CA まで戻る

より詳しい内容については公式ドキュメントをご覧ください。

Normal World invokes OP-TEE OS using SMChttps://optee.readthedocs.io/en/latest/architecture/core.html#normal-world-invokes-op-tee-os-using-smc

6.7.3. メモリマップ

セキュリティ関連の領域を含めた i.MX 8M Plus の物理メモリマップを次に示します。

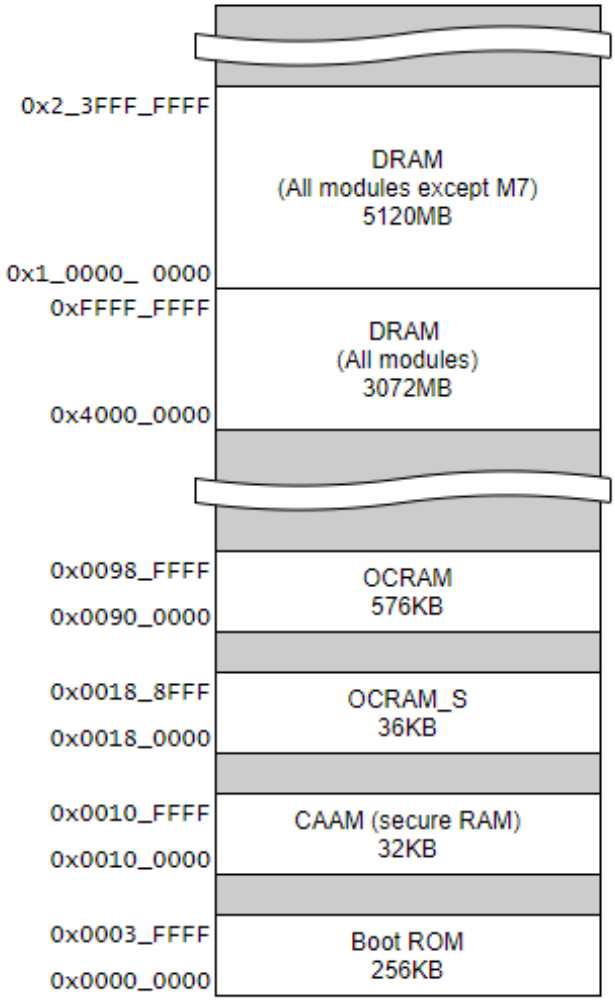

図 6.4 i.MX 8M Plus の物理メモリマップ

imx-optee-os のメモリマップを次に示します。デバッグ等にお役立てください。

表 6.1 OP-TEE メモリマップ

type	virtual address	physical address	size	description
TEE_RAM_RX/RW	0x5600_0000.. 0x561f_ffff	0x5600_0000.. 0x561f_ffff	0x0020_0000 (smallpg)	OP-TEE text + data セクション
IO_SEC	0x5620_0000.. 0x5620_ffff	0x32f8_0000.. 0x32f8_ffff	0x0001_0000 (smallpg)	TZASC
SHM_VASPACE	0x5640_0000.. 0x583f_ffff	0x0000_0000.. 0x01ff_ffff	0x0200_0000 (pgdir)	OP-TEE dynamic shared memory area, va の確保のみ
RES_VASPACE	0x5840_0000.. 0x58df_ffff	0x0000_0000.. 0x009f_ffff	0x00a0_0000 (pgdir)	OP-TEE 予約領域 (late mapping), va の確保のみ

type	virtual address	physical address	size	description
IO_SEC	0x58e0_0000.. 0x591f_ffff	0x3020_0000.. 0x305f_ffff	0x0040_0000 (pgdir)	AIPS1(GPIO1, GPT, IOMUXC など)
IO_NSEC	0x5920_0000.. 0x595f_ffff	0x3080_0000.. 0x30bf_ffff	0x0040_0000 (pgdir)	AIPS3(UART2, CAAM, I2C など)
IO_SEC	0x5960_0000.. 0x597f_ffff	0x3880_0000.. 0x389f_ffff	0x0020_0000 (pgdir)	GICv3
TA_RAM	0x5980_0000.. 0x5b1f_ffff	0x5620_0000.. 0x57bf_ffff	0x01a0_0000 (pgdir)	TA ロード、実行領域
NSEC_SHM	0x5b20_0000.. 0x5b5f_ffff	0x57c0_0000.. 0x57ff_ffff	0x0040_0000 (pgdir)	OP-TEE contignuous shared memory area

7. 市場出荷に向けてデバッグ機能を閉じる

ここでは、デバッグ機能を閉じる方法とその影響について説明します。

デバッグ機能は開発の効率を向上させるために開発フェーズではなくてはならないものです。製品が市場に出荷されると市場不良の解析にも効果を発揮します。しかし、開発者にとって便利であるということは、同時に攻撃者にとっても便利な解析手段となり得ます。想定される製品の運用形態によって、デバッグ機能が必須である運用形態もあります。セキュリティリスクとのトレードオフになる可能性があるので、有効にするかどうかを検討することをお勧めします。

デバッグ機能を閉じることは開発者と攻撃者だけでなく、**アットマークテクノによる解析についてもトレードオフになります。閉じられたインターフェースを利用した解析ができなくなることをご留意いただきますようお願いします。**

7.1. JTAG を無効化する

Armadillo-IoT ゲートウェイ G4 の出荷時は、JTAG ポートは有効なままで出荷されます。当たり前ですが JTAG が有効なままだと攻撃者のコードを走らせたり、メモリをダンプして鍵を抜いたり、平文データを抜いたりと何でも可能です。市場に出荷される最終段階では JTAG を無効化することをお勧めします。

i.MX 8M Plus では Secure JTAG を有効/無効にする設定と、Secure JTAG 有効時の３つのモードの設定があります。

Secure JTAG 無効

完全に JTAG が利用できない状態になります。

Secure JTAG 有効 (default)

1. JTAG enable mode (default)
   - 認証無しで JTAG を有効化できる
2. Secure JTAG mode
   - チャレンジ & レスポンス認証によって JTAG を有効化できる
3. No debug mode
   - バウンダリースキャンやパワーモードステータスビットの可視化などテストやボード接続のチェックを行うための機能を除き、すべての JTAG 機能が無効となる

Secure JTAG については詳しくは以下を参照してください。efuse の情報もあります。

> Secure Debug in i.MX 6/7/8M Family of Applications Processors
>
> https://www.nxp.com/search?keyword=AN4686

また、内部バスのトレースも無効にすることができます。

必要とするセキュリティレベルに合わせて設定してください。JTAG を無効化する設定は efuse なので一度書き込むと、そのデバイスの設定を戻すことはできません。

起動時に uboot のプロンプトを立ち上げて以下のコマンドを実行してください。

Secure JTAG を無効化

[21]:

- 0 (0x000000): Secure JTAG is enabled (default)
- 1 (0x200000): Secure JTAG is disabled

```
u-boot=> fuse prog -y 1 3 0x200000
```

Secure JTAG の設定

[23:22]:

- 00 (0x000000): JTAG enable mode (default)
- 01 (0x400000): Secure JTAG mode
- 11 (0xc00000): No debug mode

```
u-boot=> fuse prog -y 1 3 value
```

内部バスのトレースの無効化を行う

[20]:

- 0 (0x000000): bus tracing is enabled (default)
- 1 (0x100000): kill trace enable

```
u-boot=> fuse prog -y 1 3 0x100000
```

7.2. SD boot を無効化する

SD boot は SD メディアを挿すだけで起動する便利な機能です。その反面、SD メディアの盗難や流出によってシステムへの侵入、SD boot を利用したセキュアブート鍵に対する攻撃が考えられます。こ

れらのリスクは、SD boot を無効にすることで排除することが可能です。しかし、SD boot はシステムの復旧の役割も担っています。SD boot を無効化することによって、eMMC boot が起動できない 状態に陥った場合、合わせて JTAG の無効化が合わせて設定されていると、二度と復旧することができないデバイスになる (廃棄するしかない)可能性があることを考慮してください。SD boot を無効化する設定は efuse なので一度書き込むと、そのデバイスの設定を戻すことはできません。

ブートモードを eMMC ブートに固定することで SD boot の無効化を実現します。固定するためには i.MX 8M Plus のハードウェアピンへの参照を無効化して、efuse のみを起動時に参照するようにします。

以下が efuse (BOOT_MODE_FUSES = 2) のみを参照するための条件になります。BOOT_MODE_PINS を利用するとハードウェアの攻撃を受ける可能性があるので、FORCE_BT_FROM_FUSE を 1 にするべきです。

```
(BOOT_MODE_PINS=0 or FORCE_BT_FROM_FUSE = 1) and BT_FUSE_SEL=1
```

以下はブートモードに関係するポートと efuse です。

BOOT_MODE_PINS: hardware pins

BOOT_MODE_FUSES (0x470[15:12]):

- 2: USDHC3 (eMMC boot only, SD3 8-bit)

FORCE_BT_FROM_FUSE (0x480[20]):

- 0: Boot Mode pins
- 1: Boot from programmed fuses

BT_FUSE_SEL (0x470[28]):

- 0: Boot mode configuration is taken from GPIOs.
- 1: Boot mode configuration is taken from fuses.

efuse の詳しい仕様は以下を参照してください。

i.MX 8M Plus Applications Processor Reference Manual

6.2 Fusemap

https://www.nxp.com/search?keyword=IMX8MPRM)

以上のことを理解した上で SD boot を無効化する場合は、起動時に uboot のプロンプトを立ち上げて以下のコマンド起動してください。

BOOT_MODE_FUSES (0x470[15:12]):

- 2 (0x2000): USDHC3 (eMMC boot only, SD3 8-bit)

```
u-boot=> fuse prog -y 1 3 0x2000
```

FORCE_BT_FROM_FUSE (0x480[20]):

- 0 (0x000000): Boot Mode pins (default)
- 1 (0x100000): Boot from programmed fuses

```
u-boot=> fuse prog -y 2 0 0x100000
```

BT_FUSE_SEL (0x470[28]):

- 0 (0x00000000): Boot mode configuration is taken from GPIOs.(default)
- 1 (0x10000000): Boot mode configuration is taken from fuses.

```
u-boot=> fuse prog -y 1 3 0x10000000
```

7.3. BOOT_CFG_LOCK について

efuse はビットの状態によってはビットを書き換えてしまうことが可能です。また、i.MX 8M Plus は RAM にシャドウされた値を参照しています。

攻撃者によって efuse やシャドウの変更によって、デバイスの挙動を変えられてしまう可能性があります。i.MX 8M Plus には efuse をロックする機能があります。出荷前にロックすることをお勧めします。

BOOT_CFG_LOCK は 0x470-4B0 の範囲の efuse をロックすることができます。本手順書で言及している efuse 以外も範囲に含まれます。影響範囲を確認してからの作業をお勧めします。

ロック範囲については以下を参照してください。

efuse の詳しい仕様は以下を参照してください。

i.MX 8M Plus Applications Processor Reference Manual

6.2 Fusemap

https://www.nxp.com/search?keyword=IMX8MPRM)

efuse のロック を行う場合は 起動時に uboot のプロンプトを立ち上げて、以下のコマンドを実行してください。

- 00 (0x0): no protect (default)
- 01 (0x4): WP (write protect)
- 10 (0x8): OP (overridden protect)

・ 11 (0xc): OP (overridden protect) and WP (write protect)

```
u-boot=> fuse prog -y 0 0 0xc
```

7.4. u-boot プロンプトを無効にする

Armadillo Base OS の u-boot はデフォルトで autoboot が有効になっています。autoboot が有効な状態では決まったディレイ (bootdelay) の間キー入力を待ち、入力がなければ自動的に bootcmd を実行して Linux を起動します。一方、決まった時間にキー入力があるとプロンプトが表示されて、様々なコマンドを実行することができます。これらのコマンドはとても便利なものですが、攻撃者にとっても攻撃を仕掛けるために有効な手段となり得ます。ここでは、決まった時間待つ処理を無効化する方法を説明します。

bootdelay
- 2 : デフォルト。2 秒待つ
- 0 : 待ち時間なし。ただしキー入力でプロンプトが表示される
- -1 : autoboot が無効
- -2 : 待ち時間なし。キー入力も無効

Armadillo Base OS の /boot/uboot_env.d/no_prompt の様なファイルを作って、そちらに変数を設定すれば今後のアップデートにも適用されます。

詳細は、Armadillo-IoT ゲートウェイ G4 製品マニュアルの「u-boot の環境変数の設定 [https://manual.atmark-techno.com/armadillo-iot-g4/armadillo-iotg-g4_product_manual_ja-1.8.0/ch09.html#sct.uboot-env]」を参考にしてください。

```
[armadillo ~]# vi /boot/uboot_env.d/no_prompt ❶
# bootdelay を -2 に設定することで u-boot のプロンプトを無効化します
bootdelay=-2
[armadillo ~]# persist_file -v /boot/uboot_env.d/no_prompt ❷
'/boot/uboot_env.d/no_prompt' -> '/mnt/boot/uboot_env.d/no_prompt'
[armadillo ~]# fw_setenv -s /boot/uboot_env.d/no_prompt ❸
Environment OK, copy 0
[armadillo ~]# fw_printenv | grep bootdelay ❹
bootdelay=-2
```

❶ コンフィグファイルを生成します。

❷ ファイルを永続化します。

❸ 変数を書き込みます。swupdate で書き込む場合は自動的に行われています。

❹ 書き込んだ変数を確認します。

CONFIG_BOOTDELAY は環境変数で管理されてますので、ビルドした際に設定しても Armadillo Base OS に含まれているデフォルトの変数が適用されて CONFIG_BOOTDELAY が無効になります。

必ず /boot/uboot_env.d で設定してください。

ここで説明した以外にもいくつかの方法があります。ソースコードにドキュメントがあります。以下を参照してください。

imx-boot-[VERSION]/uboot-imx/doc/README.autoboot

8. 悪意のある攻撃者への対策

この章では悪意のある攻撃者から Armadillo-IoT ゲートウェイ G4 を守る方法を説明します。

8.1. KASLR

KASLR(Kernel Address Space Location Randomization)は、Linux カーネルの実行時の仮想アドレス空間を起動のたびにランダム化するセキュリティ機能です。

KASLR を有効化すると、攻撃者が Linux カーネル内の特定の機能やデータのアドレスを予測することが困難になります。それによって、アドレスが判明している場合に可能な攻撃に対する保護レベルを向上させることができます。

工場出荷状態の Armadillo-IoT ゲートウェイ G4 では、KASLR は無効化されています。

ユーザランドでも同様の仕組みとして ASLR という機能があります。工場出荷状態の Armadillo-IoT ゲートウェイ G4 では、ASLR は有効化されています。

ASLR が有効化されていることを確認するには、randomize_va_space の値が 0 以外になっていることを確認してください。

```
[armadillo ~]# cat /proc/sys/kernel/randomize_va_space
2
```

ここからは KASLR を有効化し、有効化されていることを確認する方法を説明します。

8.1.1. KASLR の有効化

電源を投入するとすぐに以下のように uboot からデバッグ出力されます。**その間にシリアル通信ソフトウェア上で何らかのキーを押して、uboot の プロンプトに入ってください。**

```
Hit any key to stop autoboot:  2
u-boot=>
```

環境変数 optargs の値を確認します。nokaslr が指定されている場合は、KASLR は無効化されています。

```
u-boot=> env print optargs
optargs=quiet nokaslr
```

環境変数 optargs から nokaslr を削除します。上記で、quiet と nokaslr が指定されていることを確認したので、quiet のみを指定します。

```
u-boot=> env set optargs quiet
```

設定が反映されていることを確認します。optargs に quiet のみが指定されていることが確認できます。

```
u-boot=> env print optargs
optargs=quiet
```

設定を保存します。

```
u-boot=> env save
Saving Environment to MMC... Writing to MMC(2)... OK
```

以上で KASLR の有効化を終わります。

8.1.2. KASLR の有効化確認

Linux を起動して、シンボルのアドレスを確認します。ここでは例として vprintk_deferred のアドレスを確認します。

```
[armadillo ~]# cat /proc/kallsyms | grep vprintk_deferred
ffffdaffcac98530 T vprintk_deferred
```

ffffdaffcac98530 であることが確認できます。再起動後、再度 vprintk_deferred のアドレスを確認します。

```
[armadillo ~]# reboot
:（省略）
[armadillo ~]# cat /proc/kallsyms | grep vprintk_deferred
ffffa5c2de898530 T vprintk_deferred
```

ffffa5c2de898530 であることが確認でき、アドレスが変化していることが確認できました。

上位 16bit と下位 20bit はランダム化されません。

Armadillo-IoT ゲートウェイ G4 セキュリティガイド

改訂履歴

バージョン	年月日	改訂内容
1.0.0	2022/06/28	・初版発行
1.1.0	2022/10/26	・「6.4.4. imx-optee-client をビルドする」 の optee のバージョンを lf-5.10.72_2.2.0 に更新 ・「7.4. u-boot プロンプトを無効にする」 で例示している CONFIG_BOOTDELAY の使用例を uboot_env.d を使用したものに変更
1.2.0	2022/11/28	・「6.4.3. ブートローダーを再ビルドする」バージョン更新方法の推奨を変更 ・swu を利用したアップデート方法を追加
1.3.0	2023/03/28	・「8. 悪意のある攻撃者への対策」 を追加
1.3.1	2023/06/29	・「5.3.3. 署名ツール (CST) を準備する」 内に署名ツールのバージョンに関する補足説明を追記
1.3.2	2023/10/30	・署名検証コマンドの例で sig.txt としていた箇所を sig.bin に修正

Armadillo-IoT ゲートウェイ G4 セキュリティガイド
Version 1.3.2
2023/10/27