User's Manual

セキュアブートマネージャアーキテクチャ

ユーザーズマニュアル

Renesas Synergy™ プラットフォーム

ルネサス エレクトロニクス
www.renesas.com

Rev.1.02 2019.10

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1　2017.11)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。

Renesas Synergy™ プラットフォーム

セキュアブートマネージャアーキテクチャ

目次

(注1) 本資料は英語版を翻訳した参考資料です。内容に相違がある場合には英語版を優先します。資料によっては英語版のバージョンが更新され、内容が変わっている場合があります。日本語版は、参考用としてご使用のうえ、最新および正式な内容については英語版のドキュメントを参照ください。

(注2) 英語版ドキュメントはプロジェクト（サンプルソフトウェア）と共に ZIP ファイル化されています（下記）。 プロジェクトを動作させながら本ドキュメントを参照することを推奨します。ダウンロードには My Renesas への登録が必要です。 r12an0093eu0102-synergy-secure-boot-manager.zip

(注3) 本資料と関連する資料「セキュアブートマネージャ（参考資料）」（下記）も合わせて参照ください。
r12an0093ju0102-synergy-secure-boot-manager.pdf

1. セキュアブートマネージャの概要（Secure Boot Manager Overview）

セキュアブートマネージャ (SBM) は、迅速に製品を製造する環境において信頼されるデバイス（Trusted Device）を作るためのコンポーネントを提供します。ユーザは製品のライフサイクルを通して、ファームウェアの保護と管理がおこなえます。高度なセキュアソリューションを実現するため、セキュアブートマネージャは PKI（Public Key Infrastructure、公開鍵暗号基盤）とハードウェアアクセラレーション暗号化アルゴリズム（hardware accelerated cryptographic algorithm）を利用しています。SBM は、Synergy S5 デバイスファミリで動作する IoT デバイス管理向けのリファレンスとして使用できるアプリケーションプロジェクト（application project）として公開されています。このアーキテクチャはモジュール形式であり、必要なコンポーネントを追加するだけでユーザは広い範囲の要求に対応することができます。その結果、アプリケーションに合わせて最適化したブートマネージャが実現できるようになります。必要なセキュリティレベルに応じて、ユーザは公開済みのオープンインタフェースを使用してインフラストラクチャを構築するか、以下の機能やサービスの提供をルネサスのパートナーに依頼できます。

- ファームウェアのマスタリング（mastering、デジタル署名）ツール
- 高速プログラミング (書き込み) 機器
- セキュアプログラミングセンター (安全性の高い書き込み拠点) における高速プログラミング
- 証明書プロビジョニングサービスと製品ライフサイクルを通した管理
- リモート更新サービス
- TLS とクラウドのプロビジョニングおよび接続サービス

1.1 特長（Features）

- 以下の方法で開発するファームウェア（firmware）の整合性（integrity、改ざんされていないこと）を保証し、顧客とユーザブランドを保護します。
  - デジタル署名を使用してデバイスのファームウェアが正規の出所から供給されたものであり、改ざんされていないことを検証します。
  - フィールド (現場) でファームウェアを更新して、整合性を維持することを可能にします。
- 以下の方法で、信頼された特定可能（identifiable）なデバイスを実現します。
  - 各デバイスは、ハードウェアベースの一意（unique）でセキュアな **デバイス ID**（Device Identity）が割り当てられています。
  - **デバイス ID** を使用して、サードパーティ（クラウドなど）に対して、そのデバイスが信頼されている、すなわち SBM を通じてファームウェア整合性が保証されていることを示します。
- 以下の方法で、IP (知的財産) の盗用を防止します。
  - チップへの保存 (書き込み) 時とダウンロードプロセスの実施時に更新されたアプリケーションを暗号化します
  - 暗号鍵（encryption key）は、チップごと、またセッションごとに一意です

これら 3 つの主なセキュリティ機能を階層型/モジュール形式で実現しているので、実現に関連するオーバーヘッドとコストは必要な機能を追加するときにだけかかるだけです。各層に関連する機能と特徴を以下の表に示します。ブートマネージャは、以下の表に示す Synergy MCU のハードウェアセキュリティ機能を活用するように設計してあります。

ハードウェア機能	SBM の機能
鍵のラッピング（Key Wrapping）、暗号化コアの分離（Cryptographic core isolation）を実現するための SCE（Secure Crypto Engine、セキュア暗号化エンジン）	鍵ペアのうち秘密鍵（private key）の部分は暗号化済みで、セキュア化してあります。秘密鍵の盗用や MCU 外部での使用は不可能です
Renesas セキュア MPU – 機密のコードとデータを他から分離します	ファームウェアの読み取りは防止されています。 機密データの、未承認組織や個人による読み取りは防止されています。鍵をラッピングすると、悪用からさらに保護することができます。
FAW (フラッシュアクセスウィンドウ) – 製造時にフラッシュメモリの領域の一部をロックし、消去や再プログラミング (上書き) を防止します	SBM は不変であり、強力な信頼の基点（root of trust）となります。
暗号化アクセラレーション（Cryptographic Acceleration）と低消費電力を、SCE により実現しています	すべての暗号化操作は、非常に効率的です。
真性乱数生成（True Random Number Generation、TRNG）	乱数を必要とする暗号化操作は、TRNG 機能を使用して真性乱数を生成し、非常に効果的なセキュリティを実現します。

1.2　インフラストラクチャ（Infrastructure）

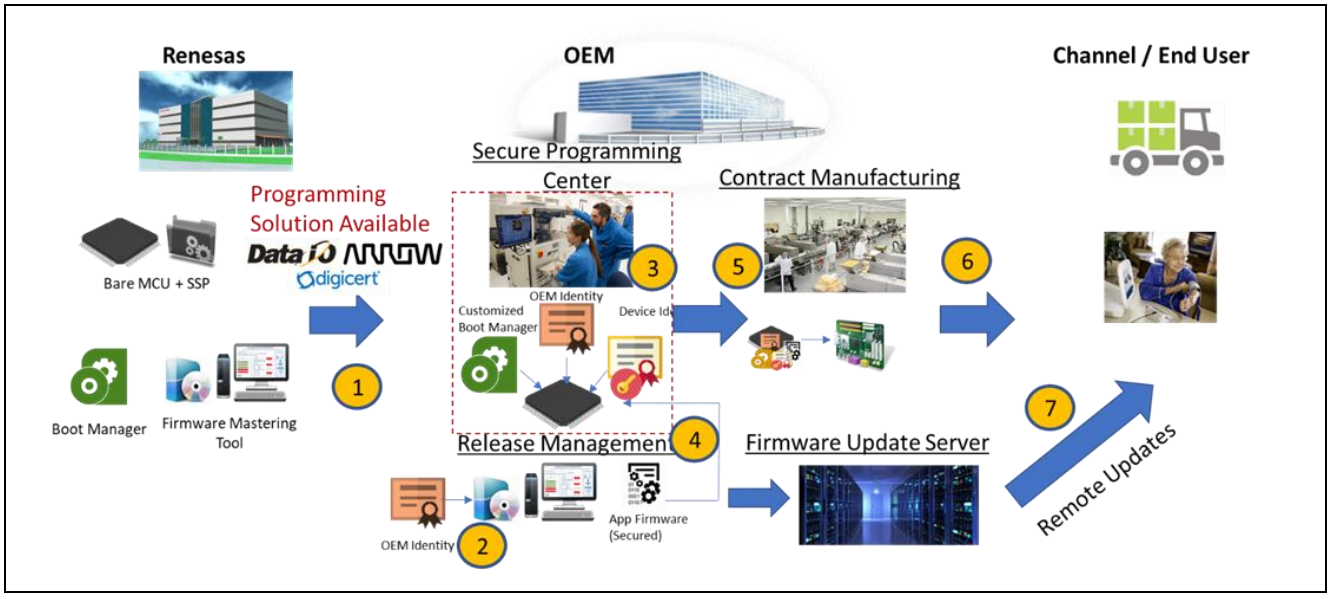


図 1. 展開用インフラストラクチャ

セキュアブートマネージャは製造時に、上の図に示すインフラストラクチャにより展開されます。以下にそれらインフラストラクチャを説明します。

1. ルネサスは、MCU、SSP（Synergy Software Package）ソフトウェアライブラリ、およびセキュアブートマネージャ (セキュアブートローダ、1st Receiver、ファームウェアのマスタリング (セキュア化) と展開を行うための PC ツールで構成) を提供します。
2. ユーザ (すなわち、OEM 各社) は、マスタリングツール（Mastering Tool）に関連付けた OEM ID（OEM Identity）を確立します。この OEM ID により、このユーザが展開したファームウェアで改ざんされていないものだけが、これらのデバイス上で動作することになります。
3. セキュアプログラミングセンターで、MCU にはセキュアブートローダ（Secure Bootloader）、マスタリングツールのインストールによる OEM ID 設定、および一意のデバイス ID（Unique Device Identity）として使用するオプションの証明書（ceretificate）が準備されます。
4. このアプリケーションファームウェア（application firmware）は、マスタリングツールで OEM ID を使用したデジタル署名（digitally signed）と暗号化 (オプション) を行い、MCU にインストールされます。また、プログラミング (書き込み) 時と、毎回のブート起動前にセキュアブートローダを使用して検証をおこないます。
5. MCU は、製品の製造のため契約先の製造業者（Contract Manufacturer）に配送されます。ファームウェアのデジタル署名と暗号化を行っているため、契約先の製造業者側では特別なセキュリティは必要ありません。ファームウェアのフラッシュ書き込みをプログラミングセンターで実施する代わりに、契約先の製造業者で実施すること

が可能な点に注意してください。契約先の製造業者がデバイスを過剰に製造する (すなわち、未承認の複製品を製造する) 可能性を減らすためには、プログラミングセンターでファームウェアのフラッシュ書き込みを行う方がより安全です。

6. 製造されたデバイスは、販売チャネルを経由して顧客に配送されます。
7. ファームウェアのリモート更新は、インストールをセキュアに実施するインフラストラクチャを通して提供され、インストールを確実にセキュアにするためにセキュアブートローダとのインタフェースを確立します。

注記:

- セキュアブートマネージャは、セキュアブートローダ、1st Receiver(「1 次受信部」アプリケーション)と複数のサポートモジュール、および PC ツール(PC Tools)で構成されています。
- コードやドキュメントで簡潔な表現が必要な場合、セキュアブートローダのことを代わりに「ブートローダ」と呼ぶことがあります。
- 従来、セキュアブートローダは「カーネル(kernel)」と呼ばれていました。この表現をほぼ更新しましたが、一部のコードやコメントで「カーネル」という表現が残っている可能性があります。将来、この表現は削除される予定です。

1.3　機能の概要（Functionality Overview）

セキュアブートマネージャは、Windows ベースのツールを含んでいます。このツールは Synergy MCU にインストールするブートローダの提供と管理を行う際にリファレンスとして使用されます。デバイス認証の目的でセキュアブートマネージャがクラウドとの間でどのようなやり取りを行うかを示すために、以下の図では複数のクラウドサービスを図示しています。ただし、これらのサービスはパートナーとサードパーティが提供するもので、セキュアブートマネージャソリューションの一部ではありません。

以下の図は、複数の層とそれらの機能や運用モデル（operational model）を示しています。

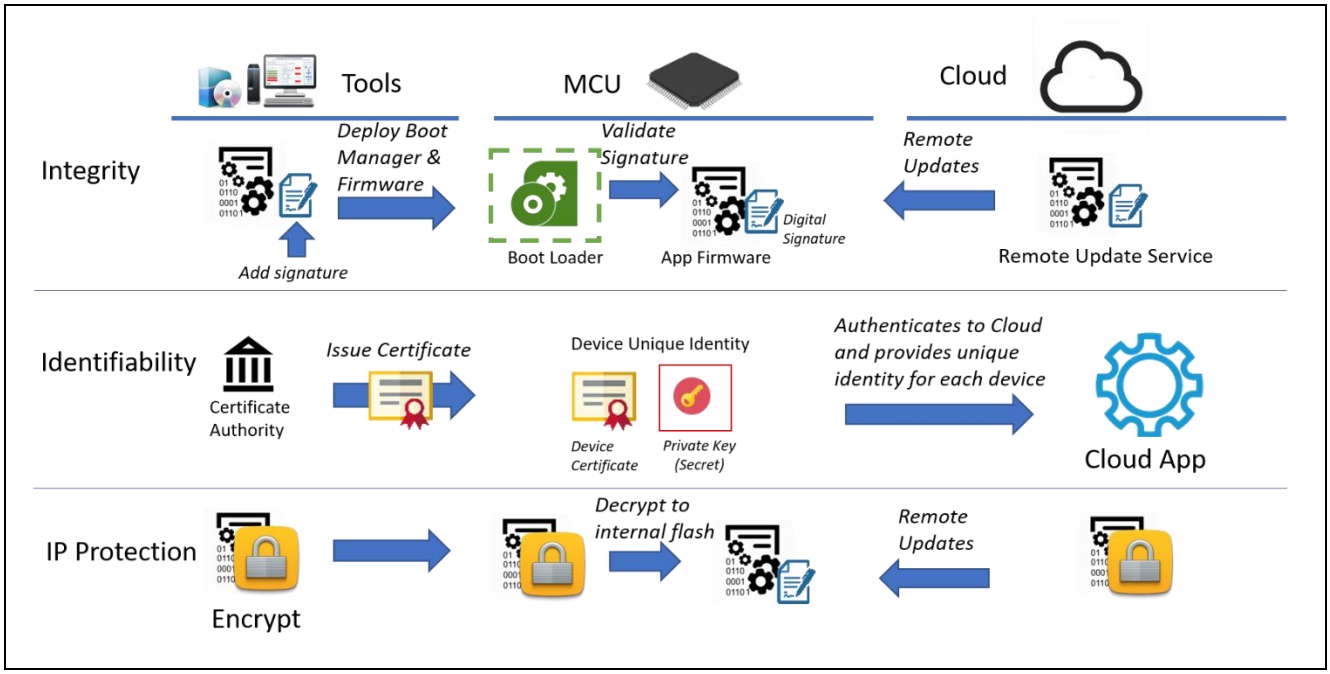


図 2. セキュアブートマネージャの使用事例に関係する複数の層

機能の概要：

ファームウェア署名の検証

- **MCU**：ブートローダはファームウェアに対するデジタル署名を検証し、改ざんされていないファームウェアのみがデバイス上で実行されることを確認します。
- **ツール**：ファームウェアに署名を追加し、セキュアブートマネージャやユーザアプリケーションのファームウェアを展開する目的で、複数のツールが提供されます。
- **クラウド**：製造時に、OEM またはサードパーティはセキュアブートマネージャにリモート更新をさせるためのインフラストラクチャを提供します。セキュアブートマネージャは、デバイスを更新する前に、ファームウェアに書き込まれている署名を検証します。セキュアブートマネージャには、このクラウド機能を実現するために提供されるセキュリティ機能をシミュレートし、説明するツールが含まれています。

デバイス ID の提供

- **MCU**：各 MCU にはデバイス ID が割り当てられます。この ID は、デバイスごとに一意の値にすることが可能です。この ID は、「Synergy ハードウェア内でセキュア化した秘密鍵（private key）」と、それに関連付けられた「公開鍵（public key）を保持している X.509 証明書（certificate）」で構成されています。この証明書はデバイスを特定する目的で使用し、秘密鍵はデバイスがこの証明書を「所有している」ことを証明するために使用します。秘密鍵に対するアクセス権を持つデバイスのみが、ID 確認に応答することができます。
- **クラウド**：この証明書 + 秘密鍵の組み合わせを使用し、TLS 等の通信プロトコルを介して、クラウドに対しデバイスを特定します。SBM のデバイス ID を使用したデバイスの直接的な特定や、デバイスを特定する目的で使用する別の ID (ここでは図示していない) とのチェーンを確立することができます。
- **ツール**：デバイスのプログラミングの際に、認証局（Certificate Authority（CA））は証明書を発行します。この証明書は、MCU 上で生成された鍵ペア (秘密鍵 + 公開鍵) に関連付けられています。

IP (知的財産) 盗用の防止

- **クラウド**:更新パッケージは、そのパッケージ向けに生成された AES 鍵を使用してクラウド上で暗号化されます。次に、各デバイスの公開 ID 鍵を使用してこの AES 鍵を暗号化します。更新の際に各デバイスに提供するパッケージ全体も暗号化します。
- **MCU**:MCU は自らの秘密 ID 鍵(private identity key)を使用してパッケージの暗号を解除し、AES 鍵を取得します。次に、この AES 鍵を使用して更新パッケージ全体の暗号を解除します。このパッケージを MCU フラッシュのプログラム領域(execution area)にプログラムする前に、パッケージの整合性(integrity)と正当性(authenticity)を検証します。
- **ツール**:更新パッケージの暗号化、デバイスからのデバイス ID 鍵の再取得、暗号鍵の暗号化を含むパッケージの作成を行う目的でツールが提供されています。

注記: いくつかの Windows ツールは製造時に使用できますが、ユーザ環境で使用するには修正が必要となる可能性があります。

1.4 セキュアブートマネージャの動作概要(Secure Boot Manager Operational Overview)

セキュアブートマネージャ (SBM) は集合的な用語であり、セキュアブートローダおよび **1st Receiver**(セキュアブートローダとともにフラッシュ書き込まれる) を含む複数のサポートモジュールで構成されています。セキュアブートローダにはダウンロード機能や通信機能はなく、最初のアプリケーションプログラミング (書き込み) は 1st Receiver に依存します。現在のところ、1st Receiver は以下に示す方法によって、UART を経由する通信を実装しています。

1. **ファームウェアの初期ロード** (例:製造時)
   A. セキュアブートローダと 1st Receiver を書き込みされていない MCU (bare MCU)に初めてフラッシュ書き込みしてリセットした後に、セキュアブートローダと 1st Receiver の検証が行われます。これは改ざんされていないことを確認するために行われます。1st Receiver が起動された後、この Receiver は初期のアプリケーションがダウンロードされるのを待ちます。
   B. ファームウェアは MCU フラッシュの更新領域(update area)にダウンロードされ、更新保留フラグ(pending update flag)がセット (1 に設定) され、システムはリブートされます。
   C. リブート時に、セキュアブートローダは以下のことを実行します。
      a. 更新保留フラグをチェックし、そのフラグがセットされている場合は更新に進みます。
      b. セキュアブートローダが利用する管理テーブルを検証し、改ざんされていないことを確認します。
      c. 暗号化が有効な場合、セキュアブートローダが利用する管理テーブルの暗号化を解除します。
      d. セキュアブートローダが利用する管理テーブルを実行領域にプログラムします。
      e. 保留フラグをクリア (0 に設定) します。
      f. ソフトウェアを使用してリブートをトリガします。
   D. リブート時に、セキュアブートローダは実行領域内のアプリケーションを検証し、そのアプリケーションが有効な場合は実行します。リブートのたびに、この検証を行います。
2. フィールドで実施する**ファームウェア更新(Firmware Updates)**は、初回のファームウェアロードに似ています。ただし、以下の点に注意してください。
   A. ユーザアプリケーションは、セキュアブートローダが利用する管理テーブルを更新領域にダウンロードする作業に対する責任を負います。各ユーザの実装は、使用するサービス、インフラストラクチャ、通信メカニズム (例:Wi-Fi、イーサネット、シリアル) などによって異なります。
   B. ブートローダは、アプリケーションを実行領域に書き込みする前にセキュアブートローダ更新領域内で利用する管理テーブルを検証しますが、その際は従来のバージョンがインストールされている必要があります。さらに古いファームウェアバージョンに対するフラッシュの上書き込みのみに制限されています。

セキュアブートマネージャには、以下の各ステージを示すサンプルが付属しています。

- **ファームウェア初回ロード(Firmware Initial Load)**:シリアルまたは USB インタフェースを使用する場合、**1st Receiver** をそのまま使用することができます。または、製造時に他の通信方式を使用するようにカスタマイズすることも可能です。
- **ファームウェア更新(Firmware Updates):サンプルアプリケーション**が付属しており、このアプリケーションはシリアルインタフェースまたは USB インタフェース(アクティブ化可能)を使用して、ローカルデスクトップモードでファームウェアを更新するデモを行います。製造時において、ユーザの既存インフラストラクチャを使用して、クラ

ウド等の適切な環境で動作するリモート更新インフラストラクチャとこのアプリケーションとの統合が必要となる可能性があります。

1.5　開発フローとツールの使用法（Development Flow and Tools Usage）

この章では、提供されているファームウェアモジュールとツールを使用する場合の開発フローについて説明します。

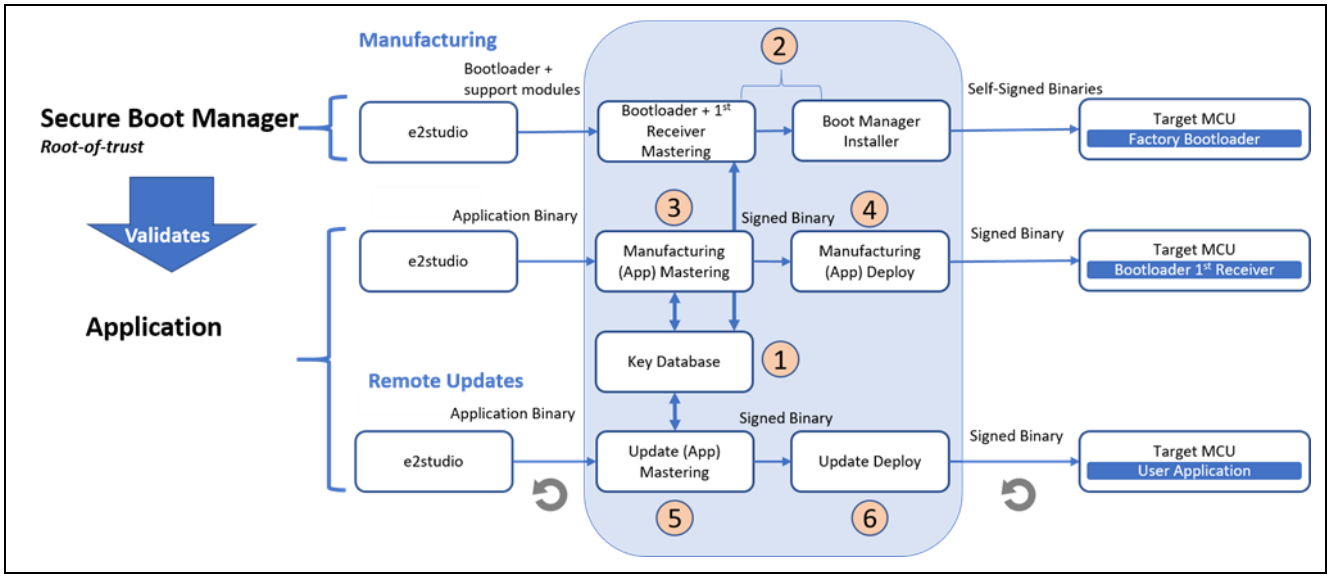


図 3. 開発ステージとツール

このソリューションは、以下の Windows PC ツールで構成されています。上の図に示す各ステージについて、以下に説明します。

1. **鍵データベース（Key Database）**：各ツールが使用する秘密鍵（private key）を保護するためのセキュアストレージ機能です。
2. **ブートローダマスタリングツールとブートマネージャインストールツール（Bootloader Mastering and Boot Manager Installation tools）**：ブートマネージャ (セキュアブートローダと 1st Receiver) を、ファクトリブートローダを使用して未書き込み MCU（bare MCU）にインストールします。ファクトリブートローダは出荷の時点で MCU 上に存在しており、USB-CDC と UART をサポートします。セキュアブートマネージャは「信頼の基点」（Root-of-Trust）であり、アプリケーションをインストールする際、およびデバイスをブートするたびにアプリケーションを検証します。
3. **製造アプリケーションマスタリングツール（Manufacturing Application Mastering tool）**：アプリケーションバイナリ (srec ファイル) を、製造時にセキュアに展開するために必要とされる形式に変換します。
4. **製造アプリケーション展開ツール（Manufacturing Application Deploy Tool）**：セキュア化されたアプリケーション (製造マスタリングツールを使用してマスタリング済み) を MCU にインストールします。1st Receiver は、ステップ 2 でインストールしたブートマネージャモジュールの一部であり、ファクトリブートローダに替わって、バイナリをフラッシュに書き込みする目的で使用します。
5. **更新アプリケーションマスタリングツール（Update Application Mastering tool）**：アプリケーションバイナリ (srec ファイル) を、デバイスをフィールドでセキュアに更新するために必要とされる形式に変換します。
6. **更新アプリケーション展開ツール（Update Application Deployment tool）**：1 つの例は、更新マスタリングツール（Updating Mastering Tool）で定義したルールに従い、アプリケーションの更新を MCU にプログラムするツールです。
   ステップ 4 でインストールしたアプリケーションを使用して、更新をダウンロードします。各ユーザがフィールドでデバイスに更新を渡すために異なるメカニズムを使用するため、このツールはサンプルとして提示されています。更新展開ツール（Update Deployment tool）サンプルは、この機能をデモンストレーションするために、シリアルインタフェースを使用してデバイスとの通信を行います。ただしフィールドでは、ユーザはフラッシュ更新の目的でバイナリを MCU に渡すために、ユーザ独自の通信メカニズムをユーザのアプリケーションに統合することになります。反復アイコン（repeat icon）は、デバイスのライフサイクルでリモート更新プロセスを複数回実行することを表しています。フラッシュ書き込みされる新しいアプリケーションはいずれも、次回の更新をフラッシュ書き込みする機能に関する責任を負うことになります。

注記：　提供されているこれらのツールは、リファレンスとして提供されている Windows アプリケーションであり、USB とシリアルの各インタフェースを使用して Synergy デバイス (リファレンスボード) との通信を行います。これらのツールは、試作のまたは製造のツールに関するリファレンスとして採用/使用することができます。

2. セキュアブートマネージャの実装（Secure Boot Manager Implementation）

この章では、アーキテクチャ要素、メモリレイアウト、インタフェース、コードのフローを含む、セキュアブートマネージャの実装について説明します。

2.1 モジュールとモジュールテーブル（Modules and Module Tables）

セキュアブートマネージャアーキテクチャにより、MCU のフラッシュ上に複数の独立したプログラムが存在させることが可能です。これらのプログラムを「モジュール」と呼びます。各モジュールは、メモリの特定の領域に割り当てられ、1 つのモジュールテーブル（module table）を保持しています。各モジュールテーブルは、現在インストールされているモジュールの詳細を定義します。

モジュールテーブルは、以下の役割を果たします。

- モジュールの整合性 (デジタル署名) を検証します。
- モジュールの位置とそのエントリポイント（entry point）を識別します。
- モジュールに対応する更新を認証（authenticate）します。

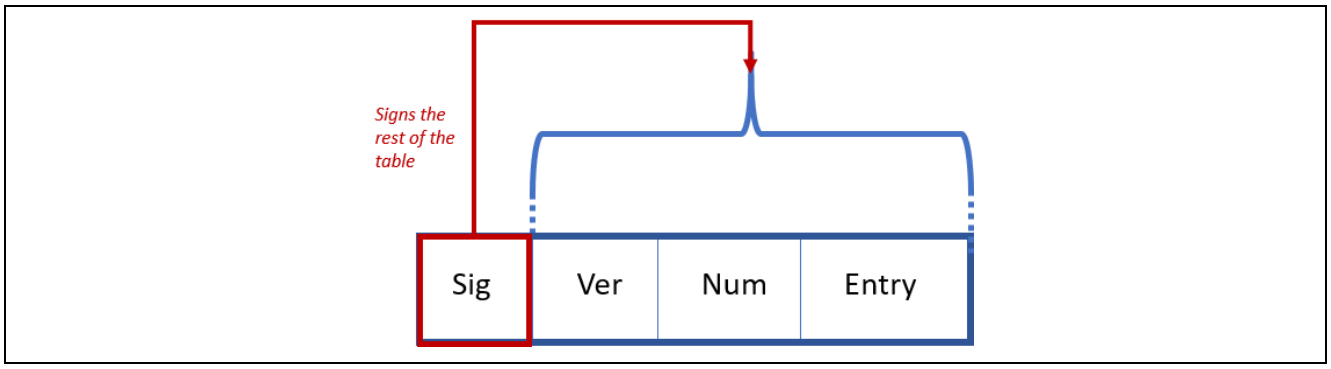


図 4.モジュールテーブルの形式

Sig (署名) ：モジュールテーブル（module table）の残り部分に対する ECDSA P256 形式の署名です。これらモジュールテーブルの残り部分には、セキュアブートローダに組み込まれているモジュールテーブル署名鍵（Module Table Signing Key）を使用します。これはブートローダマスタリングツールを使用して作成します。モジュールテーブルのメンテナンスは、セキュアブートローダが実施します。テーブルを更新するときは常に、セキュアブートローダが署名も更新します。

Ver (バージョン) ：テーブルのバージョンを表す、モジュールテーブルのバージョン番号です。現在は 1 に設定されています。

Num (数) ：モジュールテーブル内にあるエントリ数です。この値によって、システム内に存在することができるアプリケーションモジュールの数が決まります。現在は、任意に決定した 25 という値に限られています。

Entry (エントリ) ：個別モジュールに関するエントリ (項目) です。以下の図は、エントリの構造を示しており、その定義については以下で説明します。

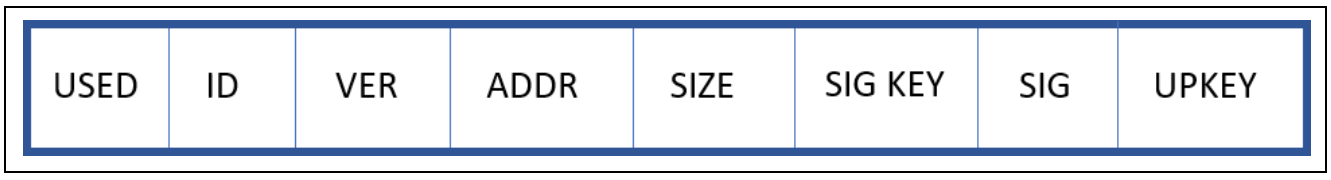


図 5.モジュールテーブルのエントリ

USED (使用中) ：そのエントリが使用中であることを表します。

ID ：モジュールの種類を表すモジュール ID です。以下の ID リストを参照してください。

VER (バージョン) ：インストール済みモジュールの現在のバージョンです。モジュールの新バージョンは、通常はこの値を大きくします (例：1、2、3)。この値を通じて、現在のバージョンより古いバージョンのインストールができなくなります。将来のバージョンは、特定のバージョンのインストールを制御するなど、古いバージョンのインストールを許可する可能性もあります。

ADDR (アドレス) :モジュールの開始場所であるフラッシュアドレスです。このアドレスは、実行時のエントリポイントでもあります。何もモジュールが存在していない場合、0x00000000 に設定されます。

SIZE (サイズ) :モジュールのバイト単位のサイズです。

SIGKEY (署名鍵) :モジュールを起動する前に、ブートローダがモジュールの署名を検証するために使用する ECDSA P256 形式の公開鍵(public key)です。

SIG (署名) :モジュールコードに対する ECDSA P256 形式の署名です (SIGKEY の値を使用してこの署名を検証します)。

UPKEY (更新鍵) :モジュールの次回更新署名の検証に使用する、ECDSA P256 形式の公開モジュール更新鍵(public Module Update Key)です。メモリ内モジュールに対してコードの上書き/コピーをする前に(例:付属のアプリケーション)、ブートローダはモジュール更新の署名を検証します。

以上から、単一のエントリを保持しているモジュールテーブルは、以下の図の構造になります。

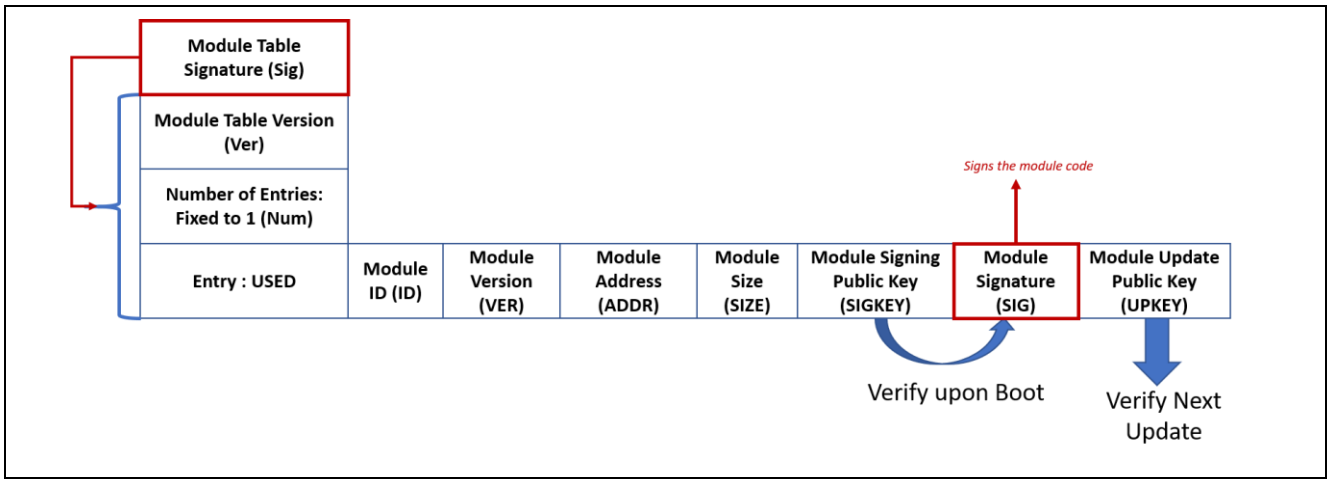


図 6. 単一のエントリを保持しているモジュールテーブル

以下の表に、現在使用されているモジュール ID を示します。モジュール ID は、モジュールに関連付けられている鍵や、モジュールに関連付けられていない他の鍵を識別する目的でも使用します。

表 1. モジュール ID と使用法

モジュール ID	使用法
1	メインのモジュールテーブルで使用できます。
192	OEM ルート鍵(OEM Root key) – 未使用です - 将来の使用状況向けです
193	OEM ファブ鍵(OEM Fab key) – 未使用です - 将来の使用状況向けです
194	予約
195	予約
202	ブートローダモジュール
203	1st Receiver モジュール

2.2　メモリマップ（Memory Map）

今回の S5D9 の実装の場合、プログラムフラッシュのアドレスマップは以下のように割り当てられています。

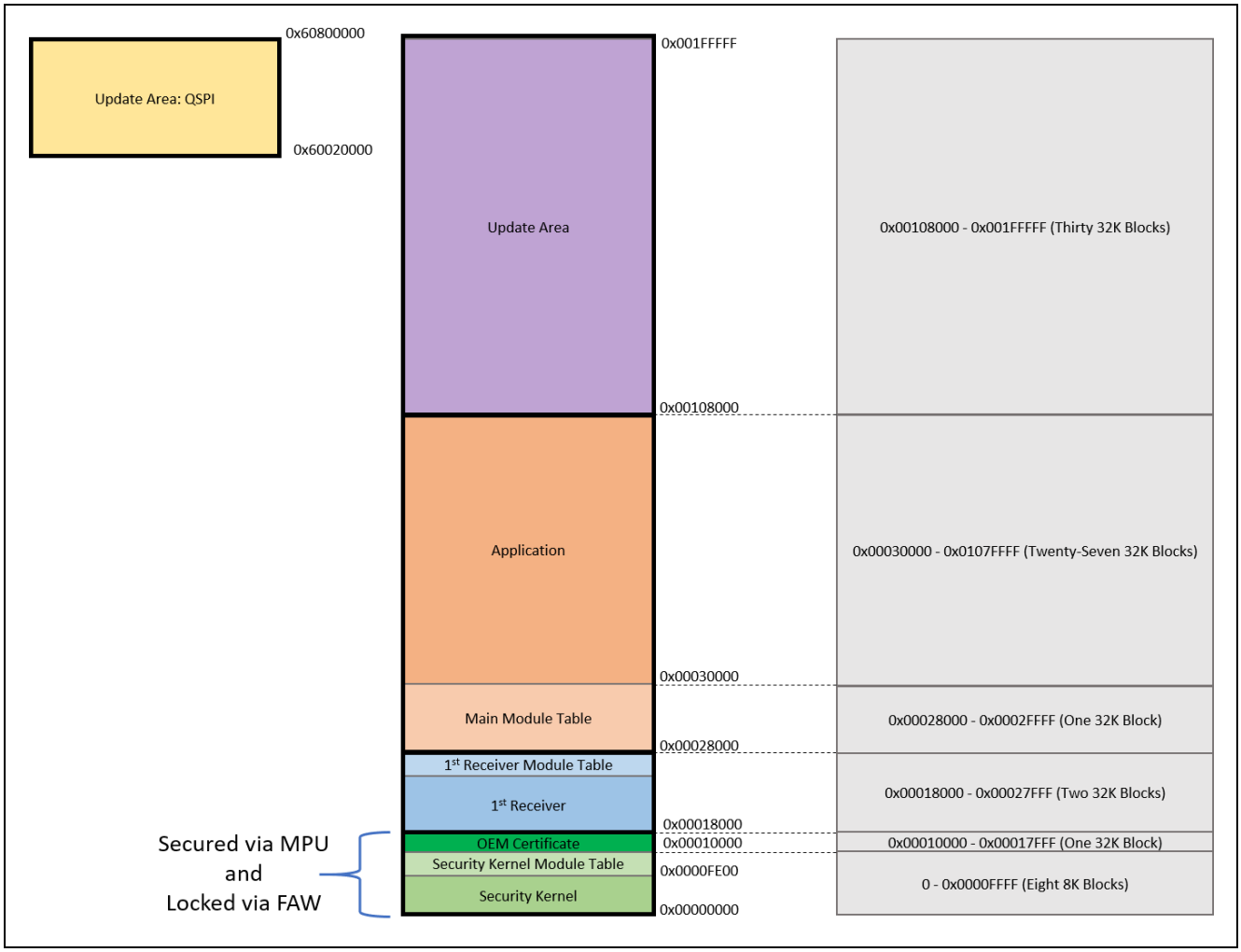


図 7. S5D9 2MB ROM デバイスのプログラムアドレスマップと物理メモリのブロックレイアウト

メモリマップ内の主要な要素は以下のとおりです (下側から上側の順)。

1. セキュアブートローダ（Secure Boot Loader）
   A. セキュアブートローダモジュールとそのモジュールテーブル。
   B. OEM 証明書（OEM Certificate）– 第 2 層 (ティア 2) 固有のデバイス ID。
2. 1st Receiver モジュールとそのモジュールテーブル。
3. アプリケーションモジュールとそのモジュールテーブル (注記：この場合、モジュールテーブルが最初に登場し、モジュールの名前は「Main Module」(メインモジュール) になります)。
4. 更新領域：いずれかのモジュールに対して更新を適用する前に、それらの更新を保存するために使用するスクラッチ領域（scratch area、一時領域）。
5. MCU のメモリ内にある更新領域の**代わりに**、更新領域として使用可能な QSPI 。アプリケーションも、この領域を使用できます。

すべてのモジュールとそれらに対応するモジュールテーブルのアドレスレイアウトは、コンパイルされてブートローダのコードの一部になります。したがって、ブートローダをデバイスにプログラムした後は、アドレスレイアウトを変更することができません。アプリケーションの要求に合わせるため、デバイスにプログラムする前であれば上記のメモリマップに変更を加えることができます。（例：1st Receiver のサイズを変更する、または製造時に 1st Receiver を使用した後で 1st Receiver のメモリを再利用する、など）

セキュアブートローダモジュール、および証明書は、特別なメモリセグメント内で以下のようにセキュア化されます。

- FAW (フラッシュアクセスウィンドウ) により、セキュアブートローダをロックしての改ざんを防止します。

注記: 強力な信頼の基点（strong root-of-trust）を保証するため、フィールドに出荷した後ではこのコンフィグレーションでのセキュアブートローダを変更することはできません。ただし、ブートローダの一部に変更を加えることで、ビルド時にブートローダをカスタマイズすることは可能です。（例：2 ステージブートローダの場合、不変である最初のステージが、2 番目のステージに対する更新を検証する、など）デバイス ID 証明書（device identify certificate）も、FAW を通じて変更を防止します。

- これらは、セキュア MPU によって保護されたメモリセグメント内に分離されます。このセグメントはファームウェアとデータの読み取りを防止し、このセグメント内にあるデータはファームウェアの他のセグメントによるアクセスから分離されます（例：アプリケーションをハッキングしても、セキュアブートローダ SBL のデータにアクセスできません。SBL のみが自らのデータにアクセスできます）。

2.3 セキュアブートローダ（Secure Boot Loader(SBM)）

セキュアブートローダは、リセットベクタのメモリアドレスである 0x00000000 に配置されており、デバイスのブート時に実行されます。更新がある場合、それらを検証した後、MCU のフラッシュを更新するのはセキュアブートローダの役割です。また、自らを検証し、MCU に存在するあらゆるモジュールを実行前に検証するのも、セキュアブートローダの役割です。製造時に、既に説明した FAW を使用して MCU に SBM をプログラムした後は、SBM メモリに対する読み取り/消去/書き込みは防止されます。デフォルトでは、ここで説明しているような評価目的のために、FAW によるロックをリセットし、テスト目的でボードへの再プログラムを有効にすることができます。ただし、製造時は、FAW を永続的にロックすることになります。その場合リセットすることはできません。

以下の図は、ブートシーケンス（boot sequence）の定義を示し、続いてそのステップについて説明します。赤い文字で記載したステップは、暗号化機能を使用する場合にのみ適用されます。

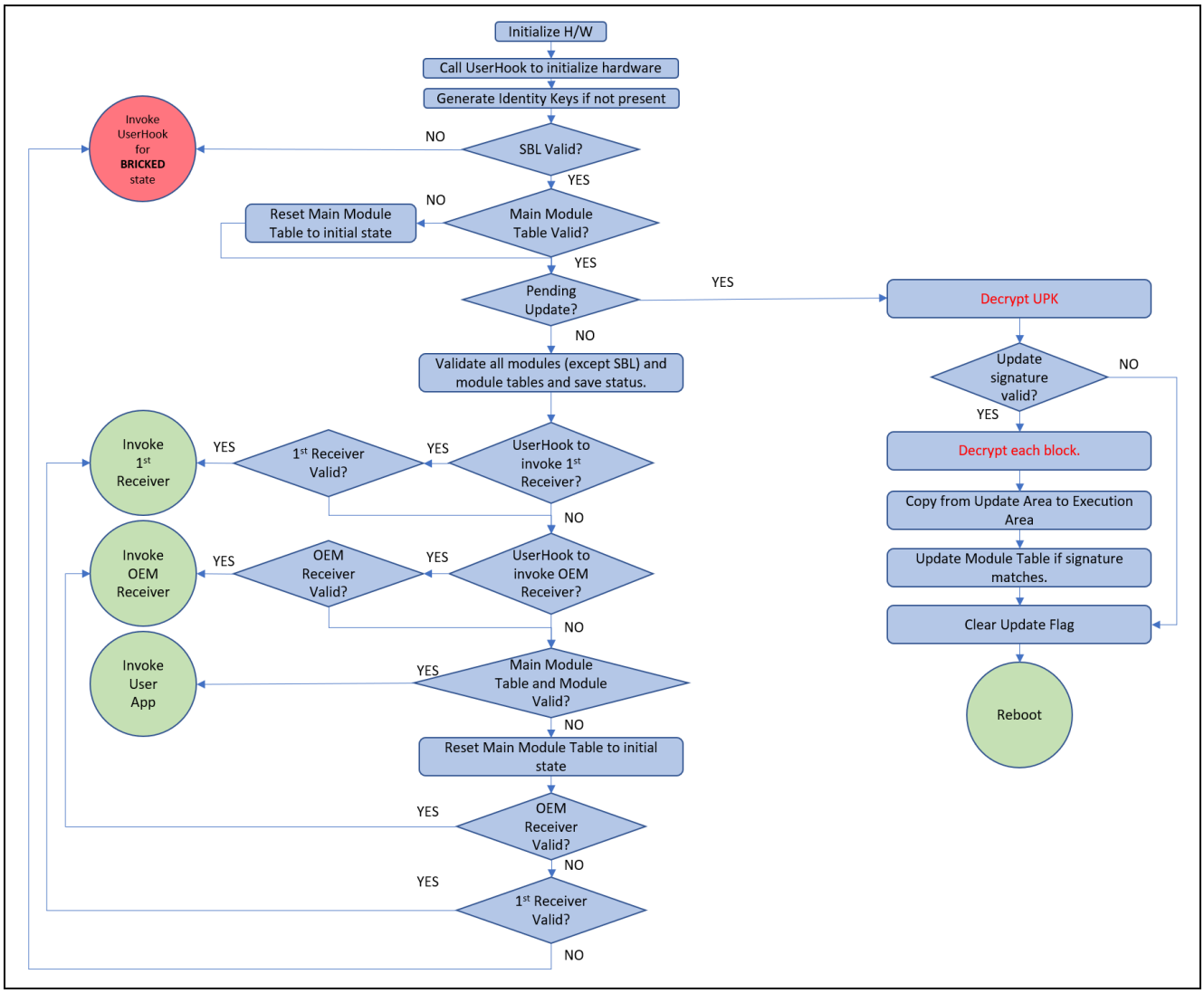


図 8. ブートシーケンス

1. デバイスのブート時に、ブートローダは自らの自己署名デジタル証明書（self-signed digital signature）(モジュールの署名) を検証し、ブートローダが改ざんされていないことを確認します。この検証に失敗した場合、デバイスの機能が阻止されます。(永続的に機能を停止したように見えます)
2. 次に、ブートローダはメインのモジュールテーブルを検証します。モジュールテーブルが無効な場合 (この状態になるのは通常、1 回目のブート時のみ)、メインのモジュールテーブルを初期状態に設定し、メインモジュールに関係のあるすべての鍵を消去します。
3. その後、ブートローダは保留されている更新の有無を確認します。更新が存在している場合、その更新の検証とインストール (モジュールに対する更新をモジュールのメモリに上書きコピー) を行った後、デバイスをリブートします。
4. 更新の必要がない場合、ブートローダは 1st Receiver の「ユーザフック」(User Hook) を参照し、1st Receiver を起動する必要があるかどうか確認します。「ユーザフック」は(ブートローダのビルド時に定義した) ユーザ定義関数であり、他のモジュールの状態に関係なく 1st Receiver の起動を可能にします。起動時に Button 1 (ボタン 1) が押下されているかどうかを検出するように、デフォルトでこの関数が設定されています。1st Receiver が起動されるのは、モジュールテーブルが 1st Receiver のコードが改ざんされていないことも検証した場合のみです。
5. 次に、ブートローダはメインのモジュールテーブルの有効性を確認するために、テーブルの署名を参照します。その後、モジュールテーブルのエントリの有効性を確認するために、モジュールのコード領域全体に対する署名を参照します。成功した場合、アプリケーションを実行します。
6. この確認に失敗した、またはアプリケーションがインストールされていない場合、1st Receiver がインストール済みか、その署名が有効かを確認し、インストール済みで署名が有効な場合は 1st Receiver を実行します。1st Receiver は常にインストールされている必要があります。例外は、製造時に最初のアプリケーションフラッシュ書き込みを行った後に 1st Receiver が削除された場合です。SBM を最初にインストールした後は、1st Receiver が唯一のインストール済みコンポーネントであり、このコンポーネントが起動されることに注意してください。

7. すべての署名確認に失敗した場合や、何もインストールされていない場合は、デバイスは機能を停止します。(永続的に機能を停止したように見え、回復する唯一の方法は製造施設でチップ全体を消去することです。) 評価目的のため、ブートローダにはデバイス機能阻止を防止するメカニズムが搭載されています。 (既に説明したように、デバイスのリセットを許可するように FAW が設定されています。)

2.3.1 ブートローダ API（Boot Loader API）

ブートローダ API は、他のモジュール (1st Receiver または付属のアプリケーション) から起動することができます。

戻り値 (リターンコード) **true** (真)成功を表し、それ以外の場合は何もデータが返されません。

2.3.1.1 更新の検証と有効な場合の適用（Validate and (if valid) apply an update）

この関数を呼び出す前に、アプリケーションをシャットダウンし、関連するすべてのコンテキストを不揮発メモリに格納して、すべての周辺回路を安全な状態に移行させる必要があります。

この関数は、制御を戻しません。更新が有効な場合、その更新が適用されます。デバイスは常にリブートします。

関数プロトタイプ:

```
void secureBootloaderApiApplyUpdate(uint8_t *pBuffer, uint32_t length);
```

名前	方向	説明
pBuffer	入力	更新パッケージを保持しているバッファへのポインタ。「3.3 章 セキュア製造 - アプリケーションのマスタリング (3)（Secure Manufacturing - Application Mastering (#3)）」で説明しています。
length	入力	更新されたユーザアプリケーションの長さ（Length of the update image）

戻り値:

なし

2.3.1.2 インストール済み全モジュールのバージョン番号の取得（Obtain the version numbers of all the installed modules）

この関数は、バージョン番号やモジュール ID を含め、システム内にあるすべてのモジュールに関する情報を提供します。通常、問い合わせ側（querying party）はこの情報を使用して、MCU 上に存在しているソフトウェアのバージョン番号を判定します。

関数プロトタイプ:

```
bool secureBootloaderApiGetModuleVersionList(uint8_t *pBuffer,
uint32_t *pLength);
```

名前	方向	説明
pBuffer	出力	MCU 内に存在するすべてのモジュールのバージョンに関するデータを格納している、長さ *pLength バイトのバッファへのポインタを返します (*ModuleVersionInfo 配列を通じて)。
pLength	入出力	pBuffer の長さを保持している整数へのポインタ。長さが不十分でデータ全体（whole data）を保持できない場合、エラーが返されます。 参照先の値が 0 に設定されている場合、データ全体を保持するために必要な長さが返されます。0 以外の長さを指定してこの関数を再度呼び出すまでは、データ全体を保持している pBuffer が返されることはありません。

戻り値:

True、False

2.3.1.3 デバイス証明書の内容の取得（Obtain the contents of the device certificate）

この関数は、デバイス証明書を取得します。現在、デバイス証明書は使用されていません (将来使用する目的で予約)。

関数プロトタイプ:

```
bool secureBootloaderApiGetDevCert(uint8_t  *pBuffer, uint32_t *pLength );
```

名前	方向	説明
pBuffer	出力	証明書を格納している、長さ *pLength バイトのバッファへのポインタを返します。
pLength	入出力	pBuffer の長さを保持している整数へのポインタ。長さが不十分で証明書を保持できない場合、エラーが返されます。参照先の値が 0 に設定されている場合、証明書を保持するために必要な長さが返されます。0 以外の長さを指定してこの関数を再度呼び出すまでは、証明書を保持している pBuffer が返されることはありません。

戻り値:

True、False

2.3.1.4 OEM 証明書の内容の取得（Obtain the contents of the OEM certificate）

この関数は、OEM がインストールした証明書を取得します。

関数プロトタイプ:

```
bool secureBootloaderApiOemCertGet(uint8_t  *pBuffer, uint32_t *pLength );
```

名前	方向	説明
pBuffer	出力	証明書を格納している、長さ *pLength バイトのバッファへのポインタを返します。
pLength	入出力	pBuffer の長さを保持している整数へのポインタ。長さが不十分で証明書を保持できない場合、エラーが返されます。参照先の値が 0 に設定されている場合、証明書を保持するために必要な長さが返されます。0 以外の長さを指定してこの関数を再度呼び出すまでは、証明書を保持している pBuffer が返されることはありません。

戻り値:

True、False

2.3.2 OEM 証明書の形式（OEM Certificate Format）

OEM 証明書は、デバイスに供給される一意の ID (識別子) です (第 2 層のみ)。

以下の図に、MCU のメモリ内に保存されている OEM 証明書の形式を示します。

図 9. OEM 証明書の形式

certMagic (証明書のマジック番号) :メモリ上の位置がガーベッジ (断片化し、現在は使用されていない解放済み領域) ではないことを表す目的で使用するマジック番号（magic number）です (4 バイト)。

privKeyHandle (秘密鍵ハンドル) :デバイス ID を確立する目的で使用したラッピング済みの MCU 固有の秘密鍵です。関連付けられている公開鍵は、「certPEM」にある証明書に組み込まれています。このセクションを「ハンドル」と呼んでいますが、実態はラッピング済みの鍵そのものです。鍵を保持している別の場所へのポインタではありません。(52 バイト)

certPEMLen (PEM 証明書の長さ) :certPEM の長さ を表しています (2 バイト)。

certPEM (PEM 証明書) :PEM 形式の X.509 OEM 証明書です。(「certPEMLen」が表しているバイト数)。

2.3.3 デバイス証明書の形式（Device Certificate Format）

デバイス証明書は、暗号化をサポートする SBM の将来のバージョン がサポートする機能で使用する予定です。この証明書の構造や関係するコードの確認は今のところ必要ありませんが、この証明書はコードベースで記述されているので、以下にその構造を示します。実装では変更される可能性があることに注意してください。

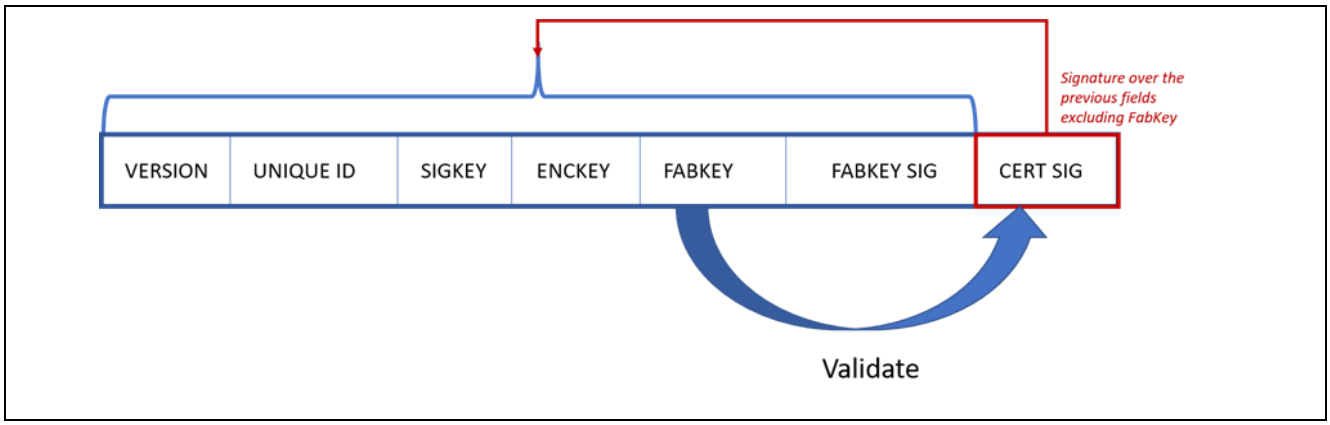


図 10. デバイス証明書の形式

2.4　1st Receiver

1st Receiver モジュールは、セキュア製造の際に、署名済みのアプリケーションバイナリ（application binary）を更新領域にロードするためのメカニズムです。付属の 1st Receiver のリファレンスでは、SCI-2 シリアルポートを使用してセキュア製造プロセスとのインタフェースを確立します。USB も実装され、有効にすることも可能です。ユーザは必要に応じてこの仕様を変更し、他の通信チャネルを使用することもできます。

署名済みのアプリケーションバイナリと鍵情報（key material）は、更新領域のアドレス空間にロードされ、次にリブートした後に secureBootloaderApiApplyUpdate() 関数を呼び出す形でインストールされます。この関数は、更新が保留されていることを表すフラグをセットします。

デフォルトでは、1st Receiver のメモリはセキュアブートローダとともにロックされており、偶発的な削除や破損を防止しています。

2.5　通信プロトコル（Communication Protocol）

1st Receiver と PC ツールは、UART (または USB-CDC) を経由する Renesas フレームプロトコル（Frame Protocol）の実装を使用して PC ツールとの通信を行います。このプロトコルは、Renesas ファクトリブートローダ（factory bootloader）が使用しているのと同じものです。詳細については、『System Specifications for Standard Boot Firmware - Application Note』(英語) を参照してください。付属のアプリケーションも、この実装を使用しています。ただし、ユーザが開発するアプリケーションでプロトコルを定義し、そのプロトコルで置き換えることを想定しています。

3.　PC ツール（PC Tools）

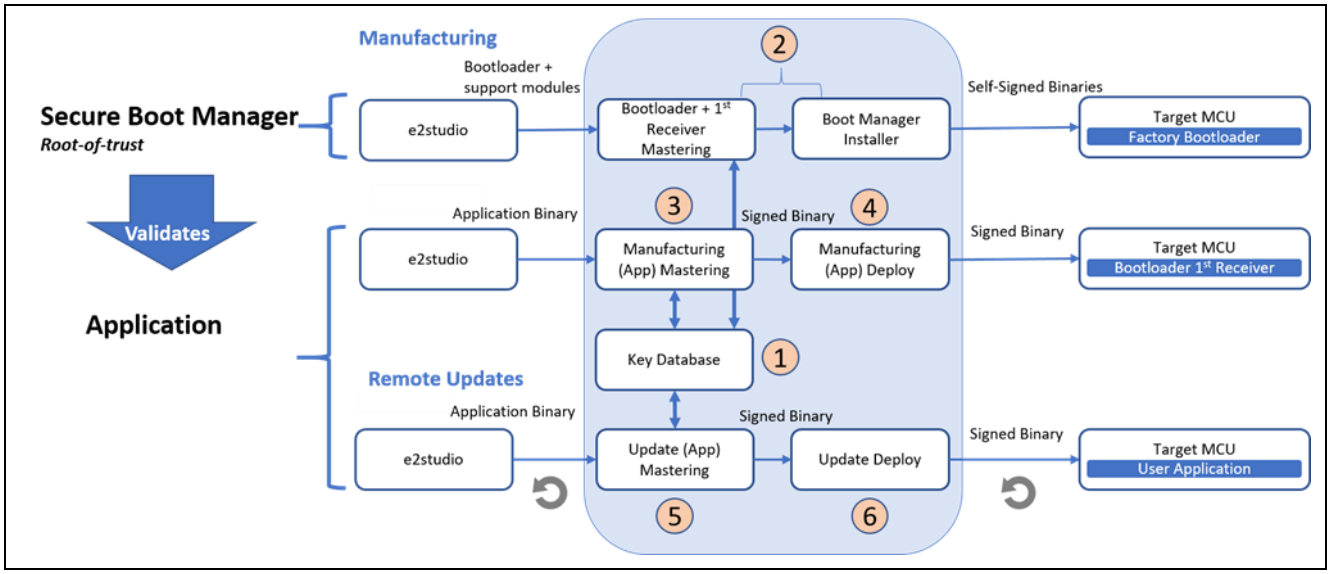


図 11. 開発ステージとツール

以下の章で、この図と各ツールについて詳細に説明します。

3.1 鍵データベース (1)（Key Database (#1)）

SBM PC ツールはさまざまなステージで暗号化鍵（cryptographic key）を活用し、検証（validation）と認証（authentication）を実行します。すべての暗号化署名アルゴリズム（cryptographic signing algorithms）で、**秘密鍵（private key）を使用して署名生成（signature generation）を実行し、公開鍵（public key）を使用して署名検証（signaure verification）を実行します**。このため、署名を生成する特権を失う可能性について心配せずに、公開鍵を自由に配布することができます。他方、秘密鍵は厳重に防護しておく必要があります。鍵データベースは、ブートマネージャのコンポーネントとアプリケーションバイナリをマスタリング（mastering）する目的で使用するすべての鍵情報の格納を目的とする、保護された保存領域（protected store）です。鍵ストア（key store）の目的は、秘密鍵を平文（plaintext）の形でアプリケーションまたはユーザに提示することなく、署名、署名検証、暗号化などの鍵を必要とするすべての操作を実行できるようにすることです。

鍵データベースと暗号化アルゴリズムは、Microsoft® Windows® CNG を使用して実装されています。この API は抽象化レイヤ（abstraction layer）として、セキュアな鍵保存や、それらの鍵に対する暗号化操作を Microsoft® Windows® システムで実行できるようにします。これらの API を使用すると、デフォルトのソフトウェアソリューションを実装しておき、その後でよりセキュアなソリューション（例：HSM、スマートカード、USB トークン）に容易に置き換えることができます。この際、マスタリングツールから API を呼び出すための設定コード以外は何も変更する必要がありません。

デフォルトのソリューションは、Microsoft® ソフトウェアキーストレージ・プロバイダ (KSP) です。これは Windows® システムで使用できることが保証されており、特別なハードウェアを必要としません。デフォルトは、初期テスト向けです。必要な場合、HSM のようなよりセキュアなソリューションを使用するために、これらのツールを簡単に変更できます。

X.509 OEM 証明書を生成する第 2 層の要求を満たすために、ローカル CA (認証局) で openSSL を使用しています。製造時は、ローカル CA を製造 CA で置き換えるか、よりセキュアにするために、提供されている CA に変更を加える必要があります。

鍵データベースは、以下のツールをサポートしています。鍵を生成するときに、秘密鍵は鍵データベース内で維持され、公開鍵は SBM ファームウェアの中にインストールされます。

3.2 セキュアブートマネージャのマスタリングとインストール（信頼の基点）(2)（Secure Boot Manager Mastering and Installation (Root of Trust) (#2)）

ブートローダをマスタリングするには、付属の**ブートローダマスタリングツール (KernelPackager)** を使用します。このツールはブートローダのバイナリと 1st Receiver のバイナリを取得して処理し、それらの更新されたアプリケーションに対して適切な鍵と署名を追加します。その結果、その更新されたアプリケーションを未プログラムの MCU にプログラムし、セキュアブートマネージャとして機能させることができます。

以下に、**ブートローダ**へのマスタリングに関係する複数のステップを示します。

1. 鍵データベースの初期化 – データベースをクリアします。
2. ブートローダの srec ファイルを読み取ります。
3. SBL モジュールテーブルにインストールするため、ブートローダ署名鍵（signing key）と更新鍵（update key）のペアを生成します。各モジュールには、署名鍵と更新鍵の両方が割り当てられています。これらの鍵は、以下の 2 つの目的で使用します。
   a. **署名鍵**：SBL の場合、起動時に自らが改ざんされていないことを確認できるようにします。これは自己署名済み (すなわち、公開鍵を使用して、更新されたアプリケーションの中で署名がなされていることを確認する) であり、CRC に似たメカニズムです。これは更新されたアプリケーションが改ざんされていないことを確認するものであり、更新されたアプリケーションの正当性を証明するわけではありません。FAW と FSPR の設定を使用して SBL の改ざんをを防止します。このチェックは SBL の改ざんまたは破損が生じていないことを二重に確認する目的で実施します。
   b. **更新鍵**：製造時、デバイスへの初回ファームウェアフラッシュ書き込みに署名します。(そのパッケージを 1st Receiver に渡します。)

   注記：

- このツールは署名を作成するために秘密鍵を使用し、このツールが作成した署名を検証するために SBL は公開鍵を使用します。
- OEM のルート鍵（Root Key）とファブ鍵（Fab key）のペアも生成されます。**これらはデバイス証明書の一部であり、現在は使用されていません。**SBM の将来のバージョンをサポートする目的で、これらを使用する予定です。

4. モジュールテーブルの署名鍵ペアを生成し、秘密鍵をブートローダの srec の決められた場所に挿入します。
   この鍵ペアを使用して、モジュールテーブルおよび関連付けられている鍵と署名が改ざんされていないことを確認します。秘密鍵は、データへの署名と検証の両方に使用します。この鍵は、他の鍵が使用する鍵管理方式 (例：秘密鍵は鍵データベース内で維持) の例外です。すなわち、秘密鍵をデータベースから削除し、SBL の中に挿入します。
5. 上記 (3) で作成した署名公開鍵と更新公開鍵を、ブートローダモジュールテーブルに入れます。
6. マスタリング済みの ssrec ファイルを PC に書き込みます。(次のステップで、ブートマネージャインストーラツールを使用して展開します。)

1st Receiver も同様の方法で処理します。

1. 1st Receiver の srec ファイルを読み取ります。
2. 1st Receiver の署名鍵と更新鍵のペアを生成します。これらは、1st Receiver のモジュールテーブルにインストールされます。
   SBL はこの公開鍵を使用して、1st Receiver を検証します。1st Receiver は更新されないため更新鍵は破棄されます。
3. 上記 (2) で作成した署名鍵を、1st Receiver のモジュールテーブルに入れます。
4. マスタリング済みの ssrec ファイルを PC に書き込みます。(次のステップで、ブートマネージャインストーラツールを使用して展開します)

セキュアブートローダと 1st Receiver は、**カーネルインストーラ（Kernel Installer）**を使用してプログラムされていない MCU にインストールします。この作業を行うには、フレームプロトコルを使用してファクトリブートローダと通信を行う USB 接続を経由します。UART も使用できます。製造時に使用できるインフラストラクチャに応じて、インタフェースを選択してください。

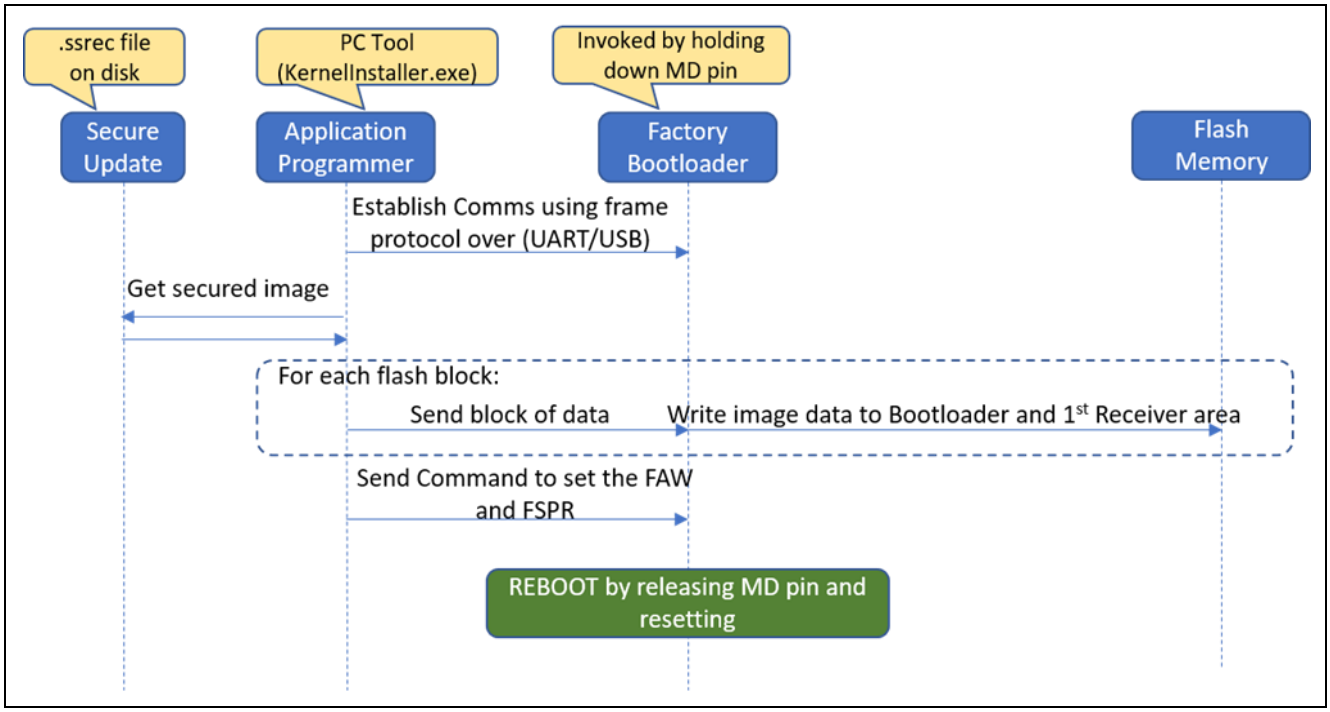


図 12. セキュアブートローダと 1st Receiver のプログラミング

セキュアブートローダのインストールは、セキュアデバイスのライフサイクルで最初のステージであり、サードパーティのセキュアプログラミングセンターや、ユーザがセットアップしたセキュア施設など、物理的にセキュリティを実現する製造施設で実施される必要があります。セキュアブートローダは信頼の基点（the root-of -trust）です。これは、将来

のセキュア動作のすべての妥当性（validity）は、セキュアブートローダの正当性（authenticity）と不可侵性（invaiolability）に基づいているためです。すべてのセキュアデバイスは、自らの信頼の起点をセキュアな施設でインストールする必要があります。すべての暗号化鍵（cryptographic key）はルート鍵（root key）に結びついている（chain to）必要があります。ルート鍵は、物理的なセキュリティと運用上のセキュリティなど、暗号化されていないプロセスで確立されたものです。たとえば、ある鍵は別の鍵を使用して暗号化することができます。別の鍵は、さらに異なる鍵を使用して暗号化することができます。ただし、チェーンのルート (基点)/最上位にある鍵は、暗号化することができません。

鍵生成フローの概要については、「3.4.1 章 OEM 証明書の生成とプログラミング（OEM Certificate Generation and Programming）」を参照してください。

3.3 セキュア製造 - アプリケーションのマスタリング (3)（Secure Manufacturing - Application Mastering (#3)）

セキュアブートローダをデバイスにプログラムした後のステップは、アプリケーションをフラッシュに書き込みすることです。アプリケーションの開発は通常、e² studio と Synergy SSP を使用しておこなわれます。この方法によって、セキュア製造マスタリングツールを使用して署名したアプリケーションコードのみをデバイスにインストールして実行することが確実にできるようになります。

セキュア製造アプリケーションマスタリングツール（Secure Manufacturing Application Mastering Tool）は、このファームウェアをリリースする目的で、新しい署名鍵（signing key）と更新鍵（update key）のペアを生成します。このツールは次に、e² studio 開発プロセスの出力からアプリケーションバイナリ (S-record 形式) を取得し、新しく生成した署名鍵を使用してそのバイナリに署名します。この署名鍵を、署名済みのアプリケーションバイナリとともにパッケージに格納し、以前のバージョンの更新鍵を使用してそのパッケージに署名して、セキュアな方法でセキュアブートローダに渡します。詳細については、本章で後ほど説明します。

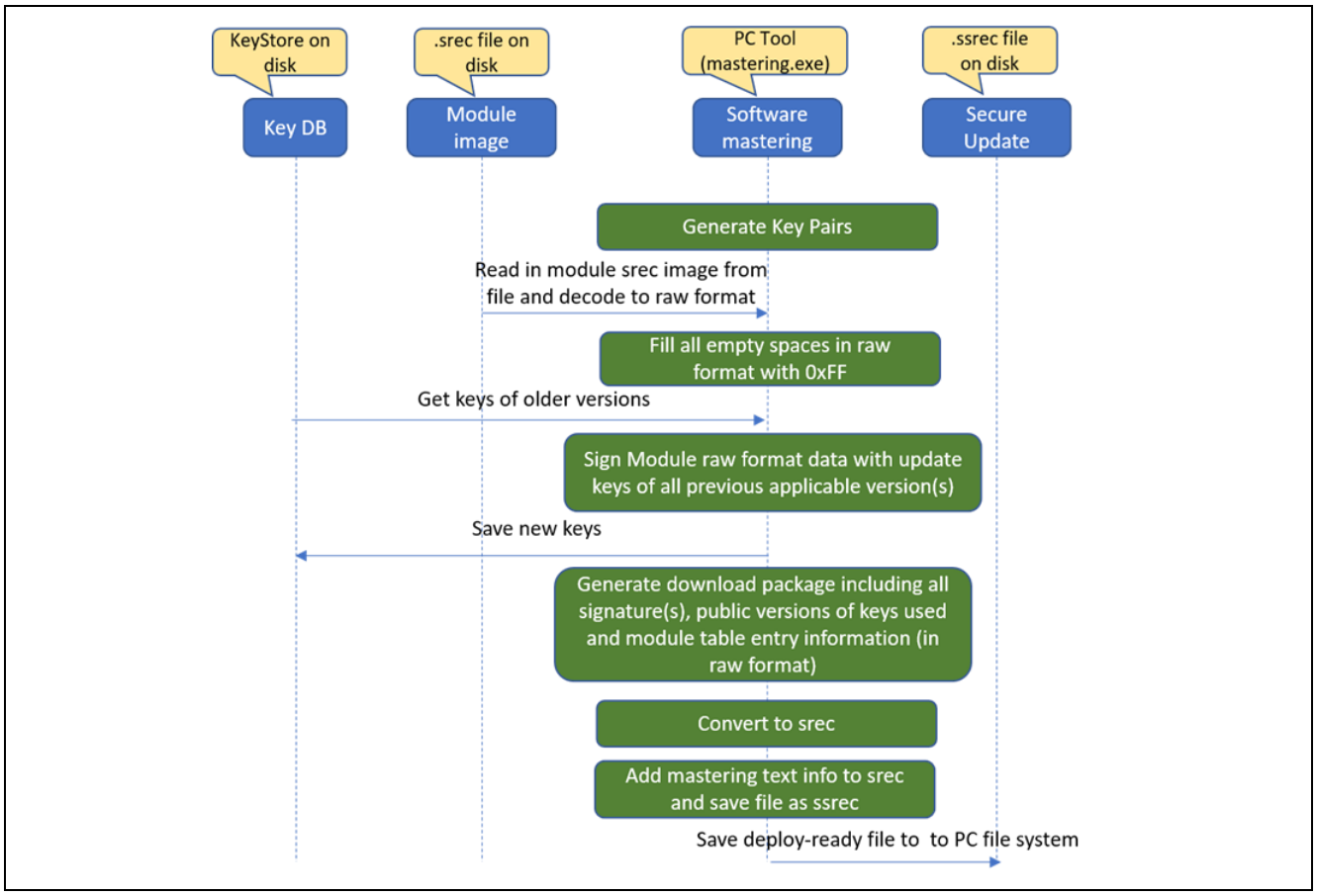


図 13. アプリケーションイメージのマスタリング

デバイスに渡すメッセージを使用して、更新を実施します。3 個のメッセージが生成され、独自形式 (ssrec) の srec ファイルに保存された後、次のステップで展開することになります。これらのメッセージは、以下のとおりです。

1. **Update Header Operation** (更新ヘッダ操作) :ブートローダのモジュール更新鍵を使用して署名されます。
2. **Install Module Operation** (モジュールのインストール操作) :ブートローダのモジュール更新鍵を使用して署名されます。
3. **Update Trailer Operation** (更新トレーラー操作) :ブートローダのモジュール更新鍵を使用して署名されます。

Install Module message (#2) (モジュールのインストールメッセージ (#2)) は、以下の内容を保持しています。

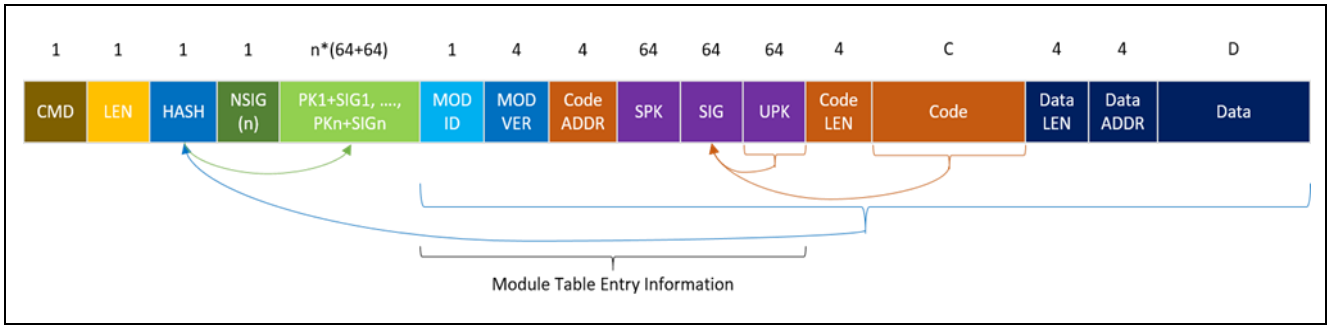


図 14. マスタリングされたモジュールパッケージの形式 (上記の各フィールドのサイズをバイト単位で表記)

CMD (コマンド) :このパッケージに対して実行するアクションの種類を指定するコマンドです。Add/Modify/Delete (追加/変更/削除) のいずれかです。Add (追加)は、これがマスタリング対象モジュールの最初のバージョンである場合です。Modify (変更)はマスタリングを更新しようとする場合です。Delete (削除)はモジュールを削除しようとする場合です。

LEN (長さ) :このフィールド (すなわち、CMD と LEN のフィールド) より後に続くメッセージの長さです。

HASH (ハッシュ) :MOD ID から DATA までの項目に対応する SHA256 のハッシュ値です。

NSIG (署名の数):以下のように規定される署名の数です。

署名は、アプリケーションが前のバージョンから更新されるのを承認する目的で使用します。通常、前にリリースした各バージョンに対して 1 つの署名が存在します。製造時の最初のアプリケーションロードは特別なケースで、前のバージョンはセキュアブートローダとみなします。
詳細については、「4 章 モジュール鍵を通じた信頼できる更新（Trusted Updates via Module Keys）」を参照してください。

PKn+SIGn:**フラッシュへの書き込み（flash update）の更新**を承認するために使用する、鍵と署名のセット (例:KEY 1 | SIG 1 | KEY 2 | SIG 2) です。更新を更新領域（update area）から実行領域（execution area）にコピーする前に、これらの署名がチェックされます。提供されるそれぞれの公開鍵の後に、対応する秘密鍵を使用する HASH フィールドの署名が続きます。更新を検証する際に、SBL はこれらの鍵のいずれかが、現在のモジュールテーブルの中にある更新鍵と一致するかどうかを確認する方法でチェックを行います。その鍵が見つかった場合は、パッケージ (MOD ID から DATA までの各フィールドが対象) に対する自ら独自のハッシュを基準として、その後に続く署名を検証します。最初にモジュールをマスタリングする場合、PK はブートローダの署名鍵（Boot Loader Signing Key）であり、SIG はその鍵を使用して生成されます。

MOD ID (モジュール ID) :モジュールの ID です (例:1 はメインアプリケーションに対応)。

MOD VER (モジュールバージョン) :更新のインストール先であるモジュールのバージョンです (1 は 1 回目に対応し、その後、2、3…)。

Code ADDR (コードアドレス) :モジュールのターゲットフラッシュのスタートアドレスです。

SPK (署名公開鍵):ECDSA P256 形式の署名公開鍵であり、**ブート時に** モジュールの署名 (例:このリリースに対して生成したばかりの署名鍵) を検証する目的で使用します (フラッシュ書き込み時に署名を検証する目的で使用する PKn+SIGn と対照をなします)。

SIG (署名) :SPK (署名公開鍵) の秘密バージョン (秘密鍵) を使用する、UPK (更新公開鍵) + コード領域に対する ECDSA P256 形式の署名です。**ブート時に**、この署名を使用してコードを検証します。データが製品の通常動作中に変更される可能性があるため、データ (データフラッシュ領域にある要素) はこのコードには含まれません。

UPK (更新公開鍵) :ECDSA P256 形式のモジュール更新公開鍵であり、モジュールのこのバージョンに対応する将来の更新の署名を検証する目的で使用することを予定しています。すなわち、将来の更新で、この鍵は PKn+SIGn (公開鍵 n + 署名 n) フィールドの中に登場します。

Code LEN (コードの長さ) :モジュールが使用するコードフラッシュ領域(code flash area)の長さを表します。

Code (コード) :コードフラッシュにプログラムする**バイナリ**イメージ(binary image)です。署名済みアプリケーションの解析 (16 進の srecord 形式ファイル) を行い、バイナリに変換します。コード内に何らかのギャップ (連続していない複数のパーティションなど) が存在する場合、それらのギャップを 0xFF で埋めます。

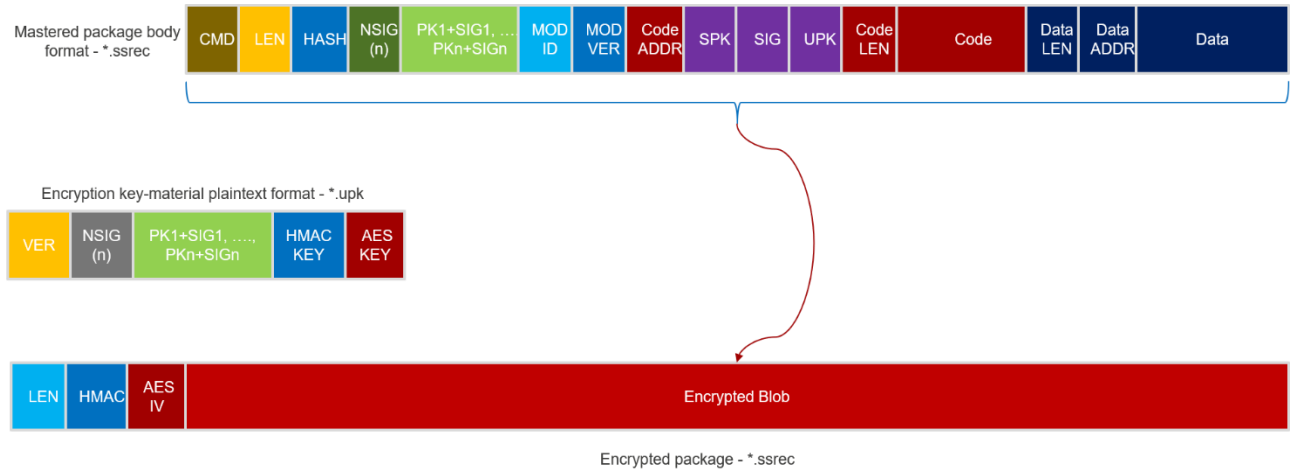
Data LEN (データの長さ) :モジュールが使用するデータフラッシュ領域(data flash area)の長さです。

Data ADDR (データアドレス) :モジュールが使用するデータフラッシュセクションのスタートアドレスです。

Data (データ) :データフラッシュにプログラムする更新データの一部です (初期化済みデータなど)。

暗号化が有効な場合、1 回限り使用する目的で PC 上で生成した AES 128 鍵を使用し、上記で作成したパッケージを暗号化します。

以下の図は、暗号化済みのパッケージと UPK (更新公開鍵) を示します。この UPK は、AES 暗号化鍵とともにその HMAC 鍵を保持しています。



展開時に、デバイス証明書を使用して、デバイスごとに UPK のみが暗号化されます。

3.4　セキュア製造 - アプリケーションの展開 (4)(Secure Manufacturing - Application Deploy (#4))

1st Receiver と通信するセキュア製造展開ツール(Secure Manufacturing Deploy Tool)を使用して、署名済みのアプリケーションバイナリをデバイスにダウンロードします。この作業は、ブートローダをインストールするときに使用したのと同じセキュア製造施設で実施することも、他の施設で実施することもできます。後者の場合、それほどセキュアではない契約先の製造施設などです。これは、アプリケーションは既に署名済みなので、このステップは高いレベルのセキュリティを必要しないためです。セキュア製造展開ツールプログラムは、製造プログラミング (書き込み) 環境をエミュレートする PC アプリケーションです。USB - シリアルへのコンバータを経由して PC に接続されている評価システムに、署名済みのアプリケーションバイナリをインストールします。デフォルトではデバイス上の UART を使用しますが、USB を使用するようにツールを構成することもできます。

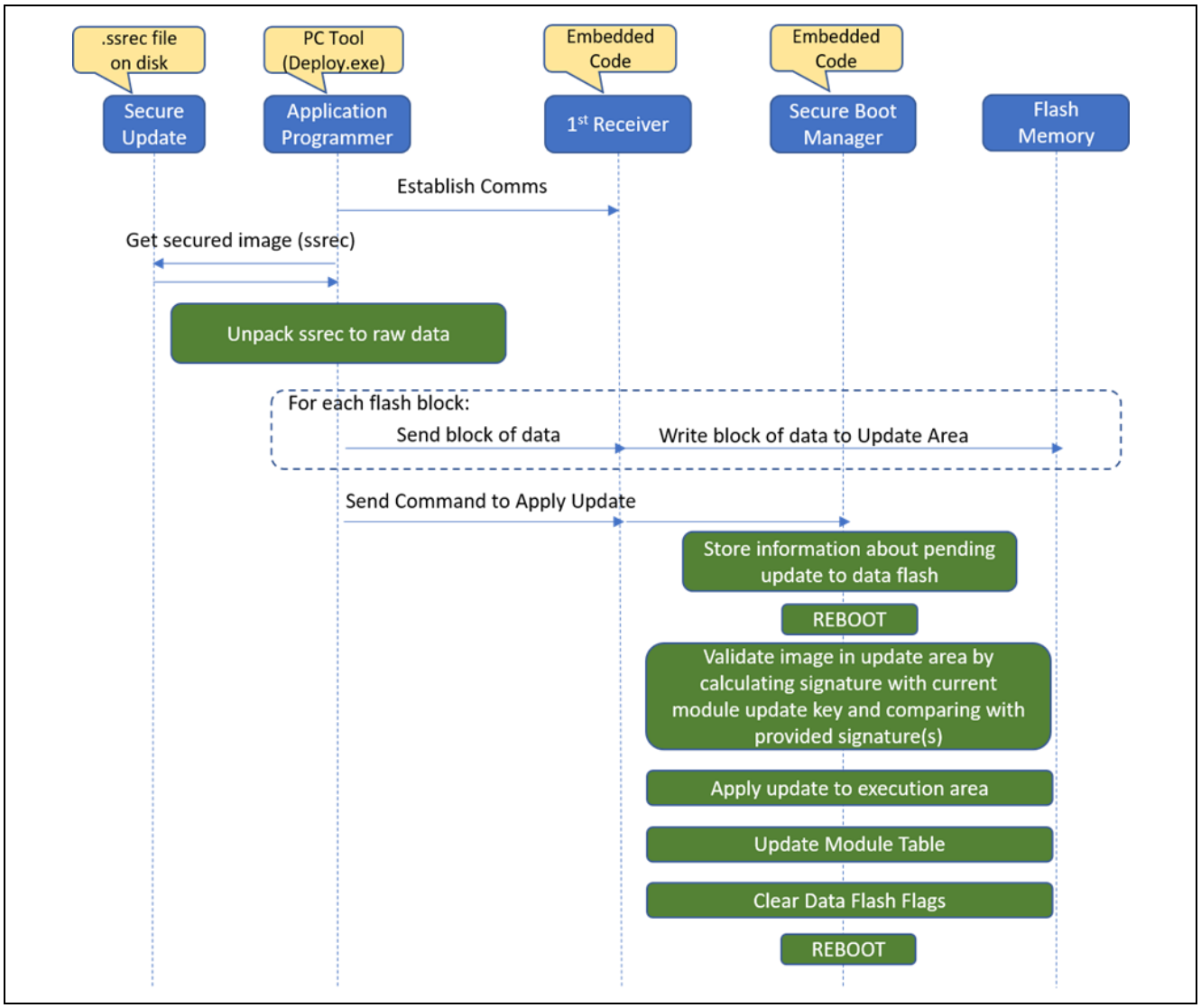


図 15. 最初のアプリケーションイメージのインストール

アプリケーションのセキュア製造の際に、以下のステップを実行します。

1. デバイスとの通信を確立します。
   A. PC のシリアルポート/USB ポートをスキャンし、有効な Synergy デバイスを見つけます。
   B. プログラム先として使用するデバイスを選択します。
2. インストールしようとするマスタリング済みアプリケーションバイナリ (.ssrec ファイル) を選択します。
3. (1st Receiver を経由して) 署名済みのアプリケーションパッケージを未加工（raw）形式でデバイスに渡し、更新領域にプログラムします。
4. API 経由でブートローダを呼び出し、更新をインストールし、システムをリブートします。
5. ブートローダは、PK + SIG フィールドを使用して、以下のように更新されたアプリケーションを検証します。
   A. 各 PK (公開鍵) を、モジュールテーブル内に既に存在している更新鍵に対して照合しようとします。
   B. 該当する鍵が見つかった場合、それに続く署名の検証を行います。更新領域内にあるパッケージが検証の基準になります。
6. パッケージの署名が有効な場合、更新を適切なモジュールに適用し、モジュールテーブルを更新します。署名が無効な場合、アプリケーションの更新領域にあるコードに何も変更を加えず、更新プロセスを中止します。
7. 更新領域と更新フラッシュをクリアし、システムをリブートします。

この時点で、製品をフィールドに展開する準備ができました。

暗号化が有効な場合、以下に示すように、フローはわずかに異なります。

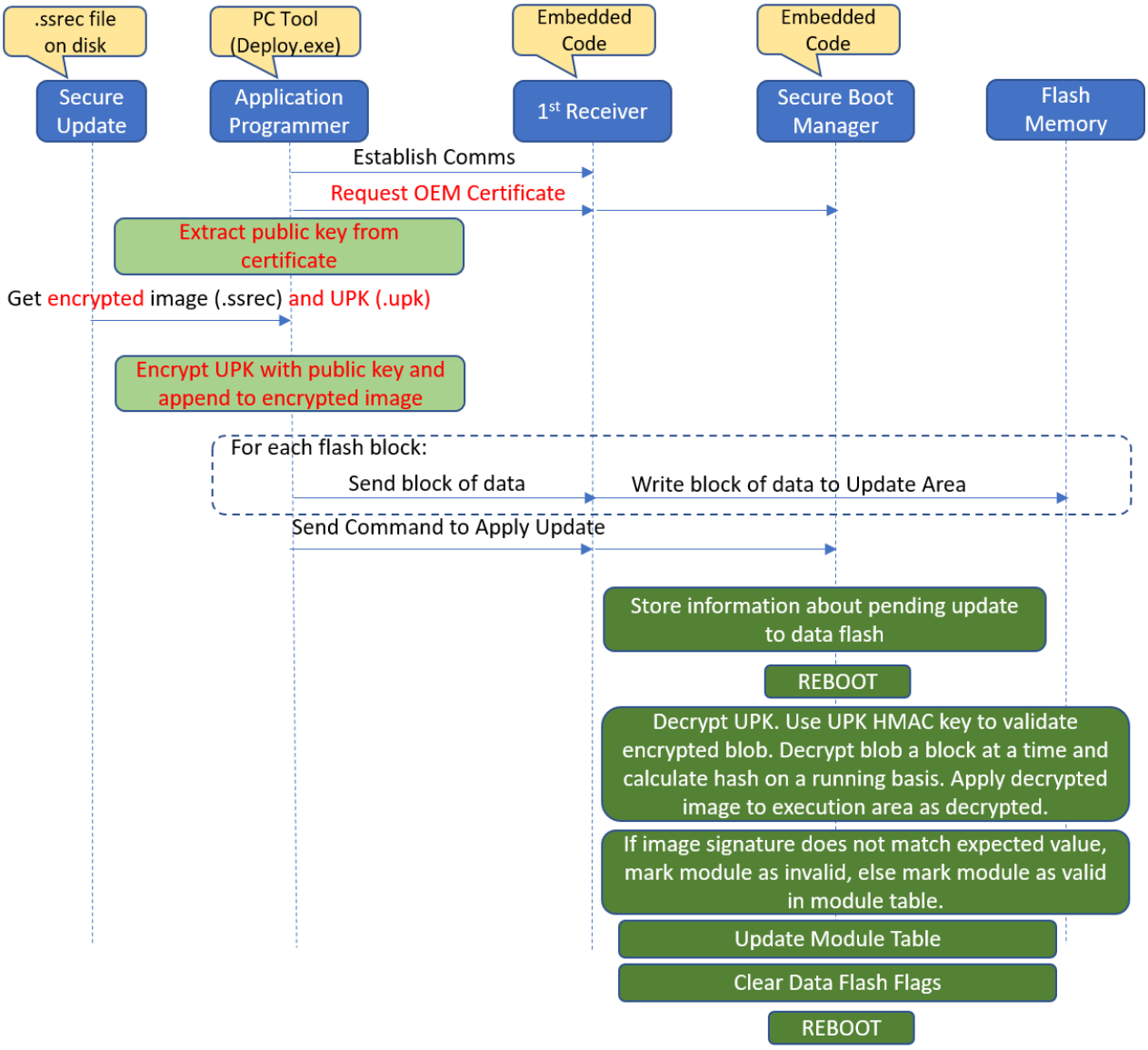


図 16. 暗号化を有効にしたアプリケーションのインストール

1. デバイスとの通信を確立します。
   A. PC のシリアルポート/USB ポートをスキャンし、有効な Synergy デバイスを見つけます。
   B. プログラム先として使用するデバイスを選択します。
2. インストールしようとするマスタリング済みおよび暗号化済みアプリケーションバイナリ (.ssrec ファイル) と、それに関連する UPK (更新公開鍵) ファイルを選択します。
3. デバイスから OEM 証明書を取得し、公開鍵を抽出します。
4. 公開鍵を使用して UPK を暗号化します。
5. 暗号化済みの UPK を、署名済みで暗号化済みの ssrec ファイルに追加した後、1st Receiver を経由してパッケージを未加工形式でデバイスに渡し、更新領域にプログラムします。
6. API 経由でブートローダを呼び出し、更新をインストールし、システムをリブートします。
7. ブートローダは秘密 ID 鍵を使用して UPK を暗号化解除し、AES 鍵と HMAC 鍵を取得します。
8. その後、HMAC 鍵を使用し、最初に暗号化済みイメージの HMAC を検証し、その整合性を確認します。
9. 次に、AES 鍵を使用して一度に 1 つのブロックを暗号化解除します。
10. ヘッダを暗号化解除し、予期される署名を抽出した後、イメージデータを実行メモリにプログラムすると同時に、署名検証に使用する HASH 値のランニングタブ（running tab、連続的に更新される一連の数値を指しており、「tab」は「tabulation」の省略形）を維持します。

11. すべてのデータを暗号化解除してメモリにプログラムした後、計算したハッシュと署名を、予期される値と比較します。
12. 値が一致する場合、そのモジュールに「valid」(有効) マークを付け、その後、モジュールテーブルを更新します。一致しない場合、そのモジュールに「invalid」(無効) マークを付け、その後、モジュールテーブルを更新します。モジュールに「invalid」(無効) マークを付けた場合、そのモジュールからブートすることはできません。
13. システムをリブートします。

3.4.1　OEM 証明書の生成とプログラミング（OEM Certificate Generation and Programming）

ユーザは、OEM 証明書生成とインストール手順を通じて、デバイス固有のセキュア ID をインストールすることができます（オプション）。Synergy MCU の SCE (Secure Crypto Engine: セキュア暗号化エンジン) が、ラッピングされた秘密鍵を生成する機能を使用して、ID のセキュリティを確保します。この ID は、アプリケーションではなく、セキュアブートマネージャ自体に関連付けられています。ただし、この ID の生成は、最初のアプリケーションを展開する際に並行して実施されます。これは、このプロセスでブートマネージャをリブートする必要があるため、このプロセスの一部として実施する方が効率的だからです。暗号化機能を使用する場合、このステップは必須です。

注記: ファームウェアのプロビジョニングが、ブートローダのプロビジョニングに使用されるのと異なる拠点、すなわち、セキュリティがより低い施設で実施される場合は、このプロセスはより早いステージで実施する必要があります。それは、ブートマネージャのインストール時です。ブートマネージャの将来のバージョンは、この作業を単一ステップで実施するモデルをサポートする予定です。

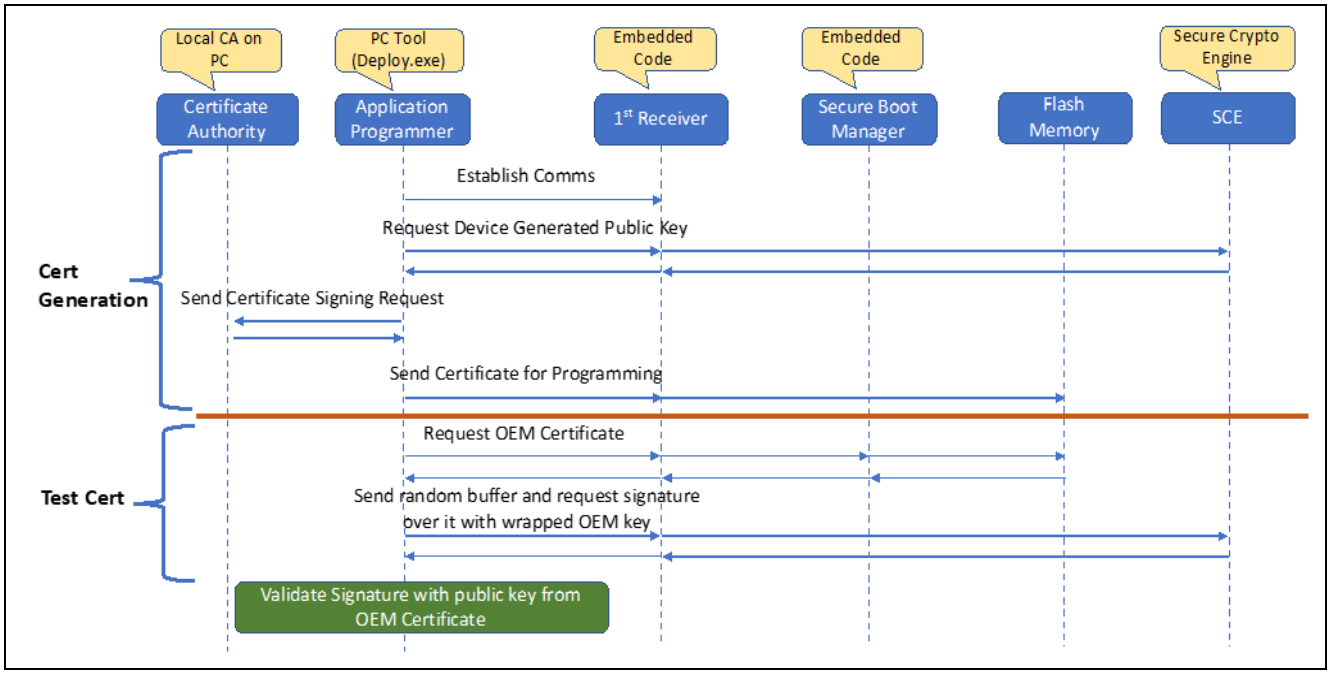


図 17. OEM 証明書の生成とプログラミング

OEM 証明書の生成とプログラミングは、製造展開 GUI（Manufacturing Deploy GUI）で実行します。この GUI は、以下のステップを使用して CertificateInstaller.exe ツールを実行します。

証明書（Certificate）は以下のように生成します。

1. この展開ツールから MCU に、鍵ペアを生成して公開鍵を返すことを求める要求を送信します。
2. その公開鍵を認証局 (CA) に送信し、X.509 PEM 形式の証明書を生成して、提供された公開鍵を使用して証明書に署名します。この場合のローカル CA は、シンプルなプロトコルを使用する、シンプルなデモ CA です。この仕様を拡張し、製造用のローカル委任 CA、またはクラウド内の CA とのインタフェースを確立することもできます。
3. 生成された証明書をデバイスに送信し、デバイスのメモリにプログラムします。
4. デバイスは OEM 証明書を作成し、フラッシュメモリに書き込みます。

証明書は以下のようにテストします。

1. 展開ツールはデバイスに対し、デバイスの証明書を送信するように要求します。API 関数 secureBootloaderApiOemCertGet() を通じてデバイスから証明書を取得し、PC で公開鍵を抽出します。

注記: この証明書は現時点では、PC 上で CA のルート鍵や信頼のチェーン（trust chain）を基準とした検証を行っていません。ユーザがカスタマイズを行う際に、この機能を追加する必要があります。

2. チャレンジ (challenge：ランダムデータ) を作成し、デバイスに送信します。
3. デバイスは、OEM 証明書に記録されているラッピング済み秘密鍵を使用し、暗号化 API を通じてチャレンジにデジタル署名した後、その署名を PC ツールに送信します。
4. 証明書から抽出した公開鍵を基準として、このデジタル署名を検証します。

注記: API を通じて、デジタルチャレンジに対応する応答機能を搭載するように、SBM を拡張する予定です。

3.5 リモート更新 - アプリケーションのマスタリング (5)（Remote Update – Application Mastering Tool (#5)）

セキュアリモート更新は、前述のセキュア製造プロセスを通過したデバイスに対して適用できます。最初に、**セキュア更新アプリケーションマスタリングツール（Secure Update Application Mastering Tool）**を使用して、e² studio の開発プロセスの出力からアプリケーションバイナリ (srecord 形式) を取得し、そのバイナリに署名します。暗号化が有効な場合、このステージでパッケージの暗号化も行います。このマスタリングツールを使用すると、モジュール ID が 1 であるアプリケーション (メインアプリケーション)のみ をデバイス上で更新することができます。

このプロセスは、以下の点を除き、「3.3 章 セキュア製造 - アプリケーションのマスタリング (3)（Secure Manufacturing - Application Mastering (#3)）」で使用したプロセスに似ています。

1. CMD (コマンド) は、「Modify」 (変更) です。
2. NSIG (署名の数) と PK+SIG (公開鍵 + 署名) の各フィールドは、更新の対象として使用できる以前のバージョンすべてを反映している必要があります。
   初期のバージョンに限り、ブートローダ署名鍵を使用して更新 (初期フラッシュ) に署名します。新しいバージョンに移行するための更新に対し、このバージョンに更新することができる**以前の各バージョン**のモジュール更新鍵を使用して署名します。 通常、以前のバージョンすべてが、このバージョンに該当します。この GUI を使用して、鍵データベースに格納されている以前のバージョンを複数選択することができます。現在、10 バージョンがサポートされていますが、この数を拡張することもできます。

3.6 リモート更新 - アプリケーション展開ツール (6)（Remote Update – Application Deploy Tool (#6)）

今度は、更新インフラストラクチャをエミュレートする**セキュア更新アプリケーション展開ツール（Secure Update Application Deployment Tool）**を使用して、署名済みで暗号化済みのバイナリを、展開済みのデバイスに渡します。このツールは、USB または UART (デフォルトで USB) を使用する点で、「3.4 章 セキュア製造 - アプリケーションの展開 (4)（Secure Manufacturing - Application Deploy (#4)）」に似ています。展開更新（Deploy Update）プログラムは、SCI-2 シリアルポートを経由してデバイスに接続しますが、この場合は、このツールはデバイス上で動作しているアプリケーションと通信を行います。付属の「Appl1」サンプルが取り扱っているのは、更新の転送、更新アドレス空間へのインストール、更新のインストールを処理するためのブートローダ API の呼び出しです。

ただし、製造時はサーバを使用する形でこれらの機能を実装することになります。その場合、署名済みのバイナリをサーバにロードし、ブートマネージャに更新を渡すアプリケーションとの間に通信経路（communication conduit）を確立します。以下の図と、その後のステップで、これらの動作方法の概要を示します。

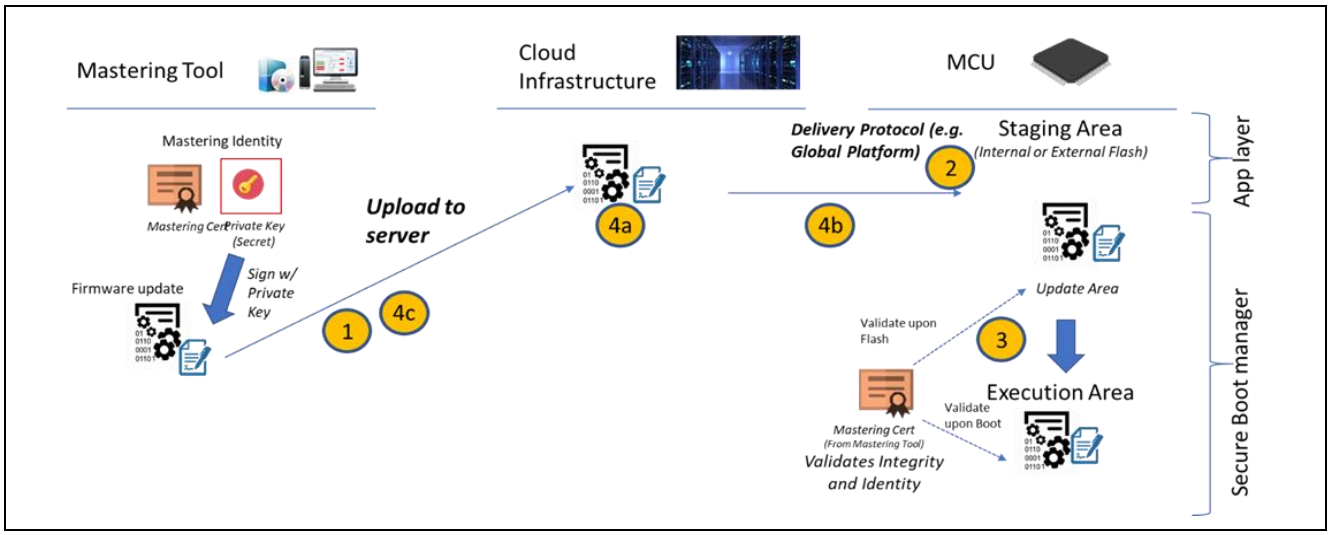


図 18. リモート更新インフラストラクチャの概要

1. 更新マスタリングツールから入手したファームウェア更新を、サーバにロードします。
2. 特定のアプリケーションレイヤを使用し、選択したプロトコルを使用して、更新を MCU に渡します。ルネサスは現在、クラウド環境でこの動作を行うサンプルを提供していませんが、この機能を実現できるようにパートナーに依頼済みです。このレイヤは、更新領域(update area)に更新内容を渡します。グローバルプラットフォーム(Global Platform) は、BLOB (バイナリラージオブジェクト)、すなわち内容を知る必要のないデータを転送する目的で使用するプロトコルの一例です。
3. セキュアブートマネージャは、サンプルと同じ方法で動作し、更新領域内にあるパッケージを処理します。
4. ユーザまたはサードパーティは、以下のように、アプリケーションレイヤにセキュリティを追加することができます。
   A. クラウドインフラストラクチャで、更新をセキュア化
      デジタル署名のハッキングは事実上不可能なので、この対策が絶対に必要というわけではありません。ただし、ユーザはファームウェアの IP に対するアクセスを制限する方法を選択することもできます。
   B. クラウドと MCU の間の通信をセキュア化
      配信プロトコルはセキュリティプロトコルを使用できない可能性があり、そして、IP へのアクセスを制限するため、TLS のような別プロトコルでラッピングする必要性が生じることがあります。TLS は、デバイスに ID の提供を要求します。このために、デバイスに接続するゲートウェイ、または MCU 上の Synergy SSP TLS スタックを使用することができます。また、セキュアブートマネージャが提供している第 2 層 ID を活用することもできます。
   C. ファームウェア更新をクラウドにアップロードするために使用する通信をセキュア化
      この作業は、セキュア FTP や PGP のようなシンプルなプロセスを使用して実施することができます。または、クラウドサービスが、この作業を実行するためのポータルを提供していることもあります。

注記: 付属している複数のサンプルは、デバイス証明書とモジュールテーブルの情報をサーバに引き渡す機能を有しています。現在はこの情報を使用していませんが、将来のバージョンで使用することを計画しています。

4. モジュール鍵を通じた信頼できる更新（Trusted Updates via Module Keys）

4.1 実装（Implementation）

モデルのバージョンごとにデュアル鍵ペア (署名鍵と更新鍵のペア) を使用するモデルは、ブートローダを開始点とし、初期のコード展開から将来のフィールドでの更新までのファームウェアの各バージョンを含めて、各更新を確実に信頼できるようにすることを意図しています。各バージョンには、以下のような独自の署名鍵（signing key）と更新鍵（update key）のセットを所有しています。

- **署名鍵**：ブート時にコードを検証する目的で、コードに署名します。
- **更新鍵**：更新をフラッシュ書き込みする前にコードを検証する目的で、更新パッケージの次期バージョンに署名します。

この仕組み（design）によって、モジュールに対して任意の更新が行える組織（party）を、モジュールの初期バージョンおよびセキュアブートローダに署名したときに使用したのと同一の鍵データベース（key database）にアクセスできる組織として、確実に限定することができます。その理由は、**この新しいバージョンに更新することができる以前のバージョンすべての更新鍵を使用して**更新パッケージ**に署名する必要がある**からです。

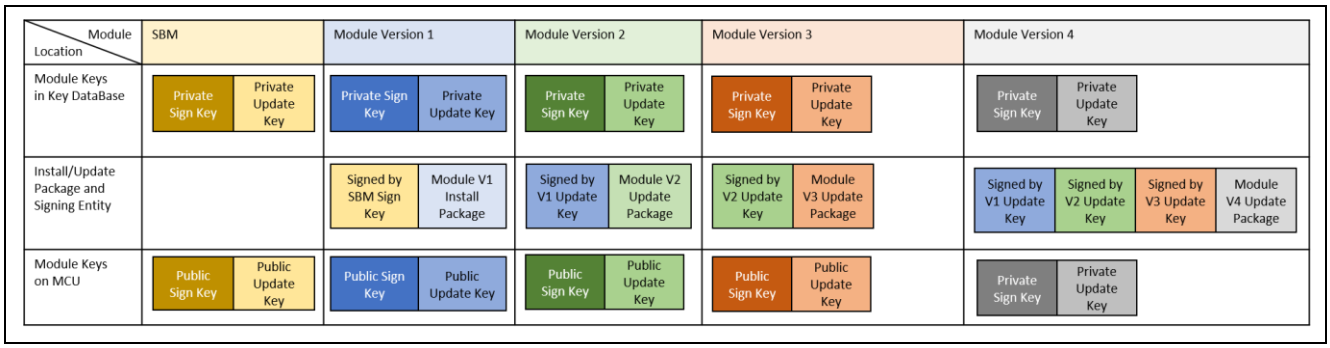


図 19. 各バージョンは、関連する以前のバージョンすべての更新鍵を使用した署名が必要

上記の例では、V2 更新パッケージを使用できるのは、V1 バージョンを実行している任意のデバイスのみです。これは V2 更新パッケージが、V1 の更新鍵のみを使用して署名してあるからです。一方、V3 更新パッケージは、V1 と V2 両方の更新鍵を使用して署名してあります。このため V3 パッケージは、V1 と V2 どちらかを実行しているデバイスにインストールすることができます。ただし V2 の更新鍵のみを使用して V3 更新パッケージに署名した場合、V1 バージョンを実行しているデバイスは、その V3 更新パッケージをインストールできなくなります。V4 更新パッケージは、それ以前のバージョンすべての更新鍵を使用して署名してあります。

すべての鍵ペアの秘密鍵部分は鍵データベースのみに保存されているため、このデータベースにアクセスできる組織のみが更新モジュールパッケージに署名し、現在展開済みのバージョンに対する有効な更新として、ブートローダに受け入れさせることができます。各鍵ペアの秘密鍵部分は鍵データベースに保存されており、公開鍵はコード/モジュール内に保存されています。

次ページの表に、出所を含め、使用されている鍵と、各鍵の関係を示します。

鍵の生成と使用法の要約

ライフサイクル ステージ	鍵データベースの初期化 [1]	SBM マスタリング	SBM マスタリング (1st Receiver)	SBM + 1st Receiver のインストール	OEM 証明書の生成とプログラミング (第 2 層のみ)	Appv1 のマスタリング	Appv1 の展開	Appv2 のマスタリング	Appv2 の展開
データベース内で生成される鍵 [2]	すべての鍵がクリアされる	SBM の署名鍵 SBM の更新鍵 モジュールテーブルの署名鍵 [3]	1st Receiver の署名鍵			Appv1 の署名鍵 Appv1 の更新鍵		App2 の署名鍵 App2 の更新鍵	
MCU で生成された鍵					ラッピング済みの鍵ペア				
MCU にプログラムされる鍵/署名				SBM モジュールテーブル: • SBM の公開署名鍵 • SBM の公開更新鍵 1st Receiver のモジュールテーブル: • 1st Receiver の公開署名鍵 モジュールテーブルの秘密署名鍵	ラッピング済み秘密鍵 OEM 証明書		アプリケーションのモジュールテーブル: • App v1 の公開署名鍵 • App v1 の公開更新鍵 署名: Appv1 パッケージ: SBM の署名鍵を使用して署名		アプリケーションのモジュールテーブル: • Appv2 の公開署名鍵 • Appv2 の公開更新鍵 署名: Appv2 パッケージ: App v1 の更新鍵を使用して署名済み

図 20. 鍵の生成と使用法のシーケンス

[1] 鍵データベースの初期化、および関連する鍵の生成は、製品ラインごとに 1 回だけ実施します。このステージで 2 つの鍵 (マスタ鍵とファブ鍵) が生成されますが、使用されないことに注意してください。これらは、将来の機能で使用する予定です。

[2] 秘密鍵のみがデータベースに保存されます。

[3] SBM マスタリングが完了した後、モジュールテーブル署名鍵はデータベースから削除されます。PC ツール側では、もうこの鍵を使用しないからです。

以下のステップは、上述の表に掲載した情報の要約を説明します。

ライフサイクルは以下のように機能します。

1. 製品ライフサイクル中に、PC 上の鍵データベース（Key Database）は 1 回だけ初期化されます。
2. SBM マスタリングツールは、以下の鍵を生成します。
   A. SBM の署名鍵：マスタリング時にその秘密バージョン (Private version) を使用して SBM のコードに署名し、ブート時に SBM の整合性（integrity）を自己検証（self-validation）する目的で SBM が公開バージョン（public version）を使用します。
   B. SBM の更新鍵：任意のモジュールの最初のバージョンにマスタリングする際にその秘密バージョンを使用し、モジュールをインストールする前にパッケージを検証する目的で公開バージョンを使用します。
   C. モジュールテーブル：署名鍵は、すべてのモジュールテーブルに署名するときに使用します。
3. SBM の 1st Receiver のマスタリング (同じツールを使用して実行します) により、以下の鍵が生成されます。
   A. 1st Receiver の署名鍵は、マスタリング時にその秘密バージョン (秘密鍵) を使用して 1st Receiver のコードに署名し、ブート時にモジュールの整合性を検証する目的で SBM が公開バージョンを使用します。
   B. 1st Receiver の更新鍵は使用されません。
4. 直前の 2 つのステップで生成された鍵は、メモリの適切な位置に、直接フラッシュ書き込みされます。
5. OEM 証明書の生成とプログラミング — MCU 上で 1 組の鍵ペア（a key pair）が生成され、公開鍵（public key）のみに対応する 1 つの証明書も生成され、その後、その証明書は MCU にインストールされます。
6. このツールは、FAW と MPU を使用してメモリをロックおよび保護するためのコマンドを送信します。
7. アプリケーションのバージョン 1 (v1) のマスタリング – 以下のように、このバージョン向けの新しい 1 セットの鍵ペアを生成します。
   A. 署名鍵（Signing key）- マスタリング時にその秘密バージョンを使用してコードに署名し、**ブート時に**モジュールの整合性（validity）を検証する目的で SBM が公開バージョンを使用します。
   B. 更新鍵（Update key）- その秘密バージョンを使用して、アプリケーションに対応する更新パッケージ (たとえば、バージョン 2) に署名します。**フラッシュ更新をインストールする前に**、更新パッケージを検証する目的で SBM が公開鍵を使用します。
8. アプリケーションのバージョン 1 の展開：
   A. 以前のバージョン、すなわちこの場合は「ルート」の更新鍵を使用して、コード、データ、生成したばかりの鍵のうち、公開バージョンを保持しているパッケージに署名します。「ルート」とは、セキュアブートローダのことです。
   B. **展開時に**、モジュールの更新の**フラッシュ書き込み**を実行する前に、この署名をチェックします。**ブート時**に、以前のステップで使用した**署名鍵**を基準として、コードをチェックします。
9. アプリケーションのバージョン 2 (v2) のマスタリング – アプリケーションのバージョン 1 (v1) のマスタリングに似ています。
10. アプリケーションのバージョン 2 の展開 – アプリケーションのバージョン 1 の展開に似ていますが、今回は SBM の鍵ではなく、アプリケーションのバージョン 1 の更新鍵を使用してパッケージに署名します。これが特別な状況であることに注意してください。アプリケーションのバージョン 2 より後の各バージョンでは、それ以前のすべてのバージョンの更新鍵を使用してパッケージに署名することになります。通常は、それより前のすべてのバージョンを更新することができますが、必ずしもすべてのバージョンが更新対象になるとは限りません。この場合 (v2) は、以前のバージョンが 1 つだけ存在しています。

5. Synergy ハードウェアの機能（Synergy Hardware Features）

この章では、SBM の設計と実装にとって必須の Synergy ハードウェの機能について説明します。ハードウェアの機能の詳細は、ハードウェアマニュアルを参照してください。

5.1 セキュア暗号化エンジン（Secure Crypto Engine）

SCE (セキュア暗号化エンジン) は、SBM が以下の作業を実行するために使用する Synergy ハードウェア周辺回路です。

1. SHA256 HASH 計算の実行。
2. ECC 鍵の生成の実行。
3. ECC 署名の生成の実行。
4. ECC 署名の検証の実行。

5.1.1 Synergy SCE のラッピング済みの鍵ペア（Synergy SCE Wrapped Key Pair）

Synergy SCE は、鍵長が可変の鍵ペアを生成することができ、非対称鍵が必要とされるさまざまなアルゴリズムでそれらの鍵ペアを使用することできます。通常動作モードで鍵生成要求を送信した場合、周辺回路は公開鍵と秘密鍵を生成し、それらを平文（plaintext）で渡します。ただし、ラッピングモードで鍵生成要求を送信した場合、周辺回路は平文の公開鍵（plaintext public key）を生成しますが、秘密鍵（private key）は「ラッピング済み」の形式（wrapped form）で渡されます。このラッピング済みの状態は、SCE 内部で実現されます。SCE は一意のデバイス鍵（device key）を使用して平文の秘密鍵を暗号化し、暗号化済みの秘密鍵を要求側に渡します。この結果、その鍵を生成したデバイスで使用するときのみ、そのラッピング済みの鍵の暗号化解除が確実にできることになります。したがって、その鍵ペアを使用するどのコードも、他の MCU では動作しなくなります (クローン不可能：unclonable)。ラッピング済みの秘密鍵は、あらゆる操作で平文の秘密鍵の代わりに使用することができますが、ラッピング済みの秘密鍵を使用するには特別な API call が必要になります。また、鍵をセキュアに保存する、より柔軟性の高いメカニズムを実現するために、ラッピング済みの鍵を保護されていないメモリ領域に保存することも可能です。ただし、以下で説明するように、ラッピング済みの鍵は MPU で保護されているメモリ領域に保存することを推奨します。セキュリティレベルを 1 つ追加し、そのデバイス上で鍵が誤用（misused）されないようにするためです。たとえば、ハッキングされたチャネルを経由して取得した偽造データに署名する目的で使用されることを防ぎます。

5.2 セキュア MPU（Secure MPU）

Synergy デバイスが搭載しているセキュア MPU は、特権コード（privileged code）のみが ROM/RAM の特定の領域にアクセスできるように構成することが可能です。さらに、特権のないコード（non-privileged code）に対して、特権コードによる読み取りを出来ないようにすることもできます。すなわち、これはファームウェアの一部を盗難から保護します。

セキュア MPU のレジスタは、以下の作業を実行します。

1. 最大 2 つのセキュアデータ領域を設定できます。— 1 つはコードフラッシュ内、もう 1 つは SRAM 内です。
2. 最大 2 つのセキュアコード領域を設定できます。— 1 つはコードフラッシュ内、もう 1 つは SRAM 内です。

   これらの領域内にあるコードのみが、セキュアデータ領域にアクセスできます。他のコードがセキュアデータ領域へのアクセスを試みると、アクセスは失敗します。

セキュアコード実行領域（Secure Code Execution region）は、セキュアではないコードも使用出来る（exposed）コードを持っています。たとえば、SBM API です。この使用は、関数の開始アドレスへのポインタを介して実施されます。セキュアではないコードがセキュアコード領域を読み取ることはできませんが、実行アドレスが既知である場合、セキュアではないコードが、セキュアコード領域内にある関数を呼び出すことは可能です。SBM はこの機能を使用して、特権のないコード（unprivileged code）がセキュアデータ領域（Secure Data area）にアクセスすることを防止します。

セキュア MPU のレジスタは、ユーザがセキュア領域の消去または再プログラム (再書き込み) することを妨げませんが、以下に説明するフラッシュアクセスウィンドウ (FAW) を使用して、これらの作業を実行することができます。

5.3 フラッシュアクセスウィンドウ（Flash Access Window）

フラッシュアクセスウィンドウ (FAW) レジスタは、コードフラッシュのアドレス範囲（消去とプログラム (書き込み) が可能な範囲）を設定する目的で使用します。この範囲の外部にあるアドレスは、フラッシュアクセスウィンドウの外部領域（outside the Flash Access Window）と呼ばれ、一度プログラムした後は変更できません。

この機能を使用して、SBM が消去または再プログラムされることを防止します。

注記：FAW は書き込み可能なフラッシュ領域に対して設定するものです。そのため、ロックされている (書き込み禁止) メモリ領域は、FAW アドレス範囲とは逆の意味です。

5.4 FSPR (ワンタイムプログラマブル設定)（FSPR (One Time Programmable Setting)）

FSPR (ワンタイムプログラマブル、1 回限り書き込み可能) ビットを使用して、FAW レジスタと MPU レジスタを恒久的に設定することもできます。したがってこの設定は、すべての設定が確認され、デバイスが量産工程から離れる準備ができたときにのみ、このビットをセットする (1 に設定する) 必要があります。

FAW レジスタと MPU レジスタがフィールド (現場) で変更 (改ざん) される事態を防止するために、このビットをセットすることが重要です。たとえば、MPU を使用する場合は、FAW の設定と FSPR ビットのセットを行い、MPU の設定をロックします。

6. システムの制限事項（System Limitations）

6.1 制限事項（Limitations）

6.1.1 機密性（Confidentiality）

ブートローダの機密性を保護するために、セキュリティ MPU は、ブートローダと証明書の領域（Boot Loader and the Certificate area）が占有しているアドレス空間全体を網羅する形で設定する必要があります。その目的は、未承認の組織/個人（parties）に対して IP (知的財産) を開示しないこと、および存在する可能性のあるセキュリティ脆弱性 security vulnerability）を容易に検討される事態を防止することです。

6.1.2 整合性（Integrity）

ブートマネージャをインストールするときに、デバイスでフラッシュアクセスウィンドウを設定することができます。これにより、ブートローダや OEM 証明書を誤って消去してしまう事態や、ブートローダ機能をバイパスする目的で意図的にハッキングされる事態を防止するためです。

評価セットでは、FSPR ビットはセット (1 に設定) されていません。このため、フラッシュアクセスウィンドウを再設定することができます。

実際のシステムでは、FSPR ビットをセットします。その結果、ブートマネージャの消去を防止できます。

6.1.3 OFS レジスタ（OFS registers）

OFS レジスタは、ブートローダによって網羅されていますが、フラッシュアクセスウィンドウの外部（outside the Flash Access Window）のアドレス空間に存在しています。このため一度書き込むと、変更はできません。

これらの項目すべてに関する設定は、「7.1 章 セキュリティ設定（Security Settings）」で説明します。

6.2 サンプルアプリケーション（Application Example）

App1 は SSP サンプルアプリケーションであり、ブートマネージャ（Boot Manager）を包含するデバイスにインストールできます。App1 は、デバイスの新しい更新情報（a new update）を受信するための SCI-2 シリアル通信コード（SCI-2 serial code）が含まれており、その更新を更新用アドレス（Pending Update Address）空間にロードし、そしてその更新をインストールするためにブートローダ（Boot Loader）を起動（invoke）することができます。また、第 2 層の使用事例に合わせて、チャレンジ/応答メカニズムを実装しています。

同様に、App2 は USB-CDC を経由する点を除き、まったく同じ機能を有しています。

App1 のコードを基礎として、評価システムへのインストールや実行が可能な他のアプリケーションをビルドすることができます。

App1 プロジェクトは、PK-S5D9 向けの「Blinky with ThreadX」サンプルプロジェクトをベースにしています。

このプロジェクトは、以下の場所に配置されます。**\SecureBootloader\renesas\src\embedded\programs\app1 (app2 は USB バージョン)。**

Blinky スレッドに加えて、更新のための UART/USB を取り扱う PMOD スレッドや、更新の際にボタン押し下げてチェックする button (ボタン) スレッドも存在しています。

SecureBootloader (セキュアブートローダ) 対応のベクタテーブルは、0x0000 0000 アドレスに位置しています。

ユーザアプリケーションのため、このベクタテーブルは、アプリケーションフラッシュ領域のベースアドレス（base address）に配置されています。このベースアドレスは 0x0002 8000 に設定されており、以下の 2 つ 1 組のリンカスクリプト（linker script）である S5D9.ld と memoryMap.ld を使用して、アプリケーションを適切な位置にリンクすることができます。

- **\SecureBootloader\renesas\src\embedded\programs\app1\script\S5D9.ld**
- **\SecureBootloader\renesas\src\embedded\common\loaderScript\memoryMap.ld**

7. プロジェクトの設定（Project Configurations）

7.1 セキュリティ設定（Security Settings）

この章では、セキュリティ戦略の一環として、データアクセス制御を目的として使用される MCU 内のさまざまなレジスタについて説明します。

7.1.1 セキュリティ MPU 領域の設定（Configuring the Security MPU area）

セキュアブートマネージャのファームウェアとインストーラ PC ツールは、セキュリティ MPU の各レジスタが、セキュアブートマネージャ、およびそのモジュールテーブルのコード領域（area of code）をカバーするように設定してあります。この方法により、セキュリティブートマネージャのコードベースが読み取られることを防止しています。

これらのレジスタは、SecureBootloader\sbmCfg\mcu_rom_cfg.h 内で構成します。これらのレジスタを変更する方法は、ハードウェアマニュアルを参照してください。これらの設定は、SBM と一緒にプログラムします。

現在の設定は、以下のとおりです。

```
#define SECMPUAC_DISPC0_VAL (0U) //SECMPU Program Counter region 0 Enabled (SECMPU プログラムカウンタ領域 0 の有効化)

#define MCU_CFG_SECMPU_PC0_START 0x00000000U // Security MPU Program counter region 0 Start address (セキュリティ MPU プログラムカウンタ領域 0 の開始アドレス)

#define MCU_CFG_SECMPU_PC0_END ((ADDR_BOOTLOADER_MT -1) | 0x3) // * Security MPU Program counter region 0 End address (セキュリティ MPU プログラムカウンタ領域 0 の終了アドレス)


#define SECMPUAC_DISPC1_VAL (1U) // * SECMPU Program Counter region 1 Disabled (SECMPU プログラムカウンタ領域 1 の無効化)

#define MCU_CFG_SECMPU_PC1_START 0x00000000U // Security MPU Program counter region 1 Start address (セキュリティ MPU プログラムカウンタ領域 1 の開始アドレス)

#define MCU_CFG_SECMPU_PC1_END 0xFFFFFFFFU // Security MPU Program counter region 1 End address (セキュリティ MPU プログラムカウンタ領域 1 の終了アドレス)


#define SECMPUAC_DIS0_VAL (0U) // SECMPU Region 0 Enabled (SECMPU 領域 0 の有効化)

#define MCU_CFG_SECMPU_S0_START 0x00000000U // Security MPU Region 0 (code flash) Start address (セキュリティ MPU 領域 0 (コードフラッシュ) の開始アドレス)

#define MCU_CFG_SECMPU_S0_END ((ADDR_1ST_RX -1) | 0x3) // Security MPU Region 0 (code flash) End address (セキュリティ MPU 領域 0 (コードフラッシュ) の終了アドレス)


#define SECMPUAC_DIS1_VAL (1U) // SECMPU Region 1 Disabled (SECMPU 領域 1 の無効化)

#define MCU_CFG_SECMPU_S1_START 0x1FF00000U // Security MPU Region 1 (SRAM) start address (セキュリティ MPU 領域 1 (SRAM) の開始アドレス)

#define MCU_CFG_SECMPU_S1_END 0x200FFFFFU // Security MPU Region 1 (SRAM) end address (セキュリティ MPU 領域 1 (SRAM) の終了アドレス)


#define SECMPUAC_DIS2_VAL (1U) // SECMPU Region 2 Disabled (SECMPU 領域 2 の無効化)

#define MCU_CFG_SECMPU_S2_START 0x400C0000U // Security MPU Region 2 (security function 1) start address (セキュリティ MPU 領域 2 (セキュリティ機能 1) の開始アドレス)

#define MCU_CFG_SECMPU_S2_END 0x400DFFFFU //Security MPU Region 2 (security function 1) end address (セキュリティ MPU 領域 2 (セキュリティ機能 1) の終了アドレス)


#define SECMPUAC_DIS3_VAL (1U) // SECMPU Region 3 Disabled (SECMPU 領域 3 の無効化)

#define MCU_CFG_SECMPU_S3_START 0x400C0000U // Security MPU Region 3 (security function 2) start address (セキュリティ MPU 領域 3 (セキュリティ機能 2) の開始アドレス)
```

```
#define MCU_CFG_SECMPU_S3_END 0x400DFFFFU //Security MPU Region 3 (security function 1) end address
(セキュリティ MPU 領域 3 (セキュリティ機能 1) の終了アドレス)
```

セキュリティ MPU を有効にした状態でセキュアブートローダのインストールが成功した後、プログラミングインタフェース J5 はアクセス不可になります。(すなわち、通常の JTAG インタフェースを経由してアプリケーションのフラッシュ書き込みを再度実施することは許可されません)プログラミングインタフェース J5 へのアクセスを再度可能にするには、ブロック 0 全体を消去して、セキュリティ MPU 設定を無効にする必要があります (ユーザーズマニュアルを参照)。この結果、ブートローダは消去されますので、全てのコードの再プログラムが必要になります。

7.1.2　フラッシュアクセスウィンドウ(FAW)の設定(Configuring the Flash Access Window)

フラッシュアクセスウィンドウ (FAW) は、セキュアブートマネージャと 1st Receiver に 1 回目の書き込みを行った後に、これらが上書きまたは変更されることを防止します。FAW は、ユーザが消去と再プログラムを実行できるアドレス範囲を設定します。フラッシュアクセスウィンドウは、Boot Manager Installer (ブートマネージャインストーラ) アプリケーション内で、**OEM 証明書領域(OEM Certificate area)の最後から、プログラムフラッシュの先頭までのアドレスをカバーするように設定してあります**。セキュアブートローダと OEM 証明書は変更 (改ざん) から保護されていますが、フラッシュの残りの部分は変更可能です。

開発中は、セキュアブートマネージャと 1st Receiver を再プログラムできるように、すなわち、恒久的にボードがロックされることがないように FAW はデフォルトで無効(disabled)になっています。

FAW を有効にするには、Visual Studio の Kernel Installer プロジェクト内で「ENABLE_FLASH_ACCESS_WINDOW」を定義します。この作業を行うには、そのプロジェクトのプロパティを開き、[C/C++>>Preprocessor] (C/C++ プリプロセッサ) の下で、「ENABLE_FLASH_ACCESS_WINDOW」という [Preprocessor Definition] (プリプロセッサ定義) を追加します。

プロジェクト内では、この定義はデフォルトで無効になっています。

FAW の範囲は、memoryMap.h 内で定義されています。

```
#define ADDR_FAW_START          ADDR_OEM_CERT
#define ADDR_FAW_END            CODE_FLASH_END_ADDR
```

7.1.3　FSPR の有効化(Enabling the FSPR)

FSPR ビットは、セキュア MPU とフラッシュアクセスウィンドウの各レジスタの変更を防止します。製造時にセキュリティブートマネージャをデバイスにプログラムする時にのみ、このビットをセットする必要があります。

FSPR をプログラムするには、Visual Studio の Kernel Installer プロジェクトで「ACTIVATE_THE_FSPR」を定義します。この作業を行うには、そのプロジェクトのプロパティを開き、[C/C++>>Preprocessor] (C/C++ プリプロセッサ) の下で、「ACTIVATE_THE_FSPR」という [Preprocessor Definition] (プリプロセッサ定義) を追加します。

プロジェクト内では、この定義はデフォルトで無効になっています。

FSPR を有効化した場合、誤った内容をプログラミングするとデバイスを回復できなくなるので、ブートローダとアプリケーションが正しく構築されているか、充分注意を払ってください。

7.1.4　JTAG アクセスの設定(Configuring JTAG access)

OSIS レジスタ内の ID Code (ID コード) はまだ設定されていません。このことにより、JTAG がデバイスにアクセスできます。証明書インストーラツール(Certificate Installer tool)は、ID Code レジスタを設定することと、セキュアブートローダでプロビジョニングしたデバイスに対する JTAG アクセスを制限することもできます。

開発中は、セキュアブートマネージャと 1st Receiver を再プログラムできるように、すなわち、永続的にボードをロックすることがないように JTAG ID はデフォルトで無効になっています。

JTAG ID を有効にするには、Visual Studio の Certificate Installer プロジェクト内で「ENABLE_JTAG_LOCKING」を定義します。この作業を行うには、そのプロジェクトのプロパティを開き、[C/C++>>Preprocessor] (C/C++ プリプロセッサ) の下で、「ENABLE_JTAG_LOCKING」という [Preprocessor Definition] (プリプロセッサ定義) を追加します。

次にこのプロジェクトを再コンパイルし、希望の JTAG ロック ID を使用して実行可能ファイルを起動します。

7.2　全般的な設定(General Settings)

7.2.1　OEM 証明書の使用の有効化(Enabling usage of OEM Certificate)

システムで OEM 証明書を有効にするには、「ENABLE_SBM_OEM_CERTIFICATE」シンボルをコンパイラのプリプロセッサに追加します。そのために、e² studio で [SecureBootloader] プロジェクトを右クリックし、**[Settings]>[Cross ARM C Compiler]** の下でシンボルを追加します。この作業を 1st Receiver プロジェクトとユーザアプリケーションでも同様に実行します (新しい更新をダウンロードするために、ユーザアプリケーションで frameProtocolTarget.c ファイルを使用している場合)。

7.2.2　暗号化の有効化（Enabling Encryption）

システムで暗号化を有効にするには、「ENABLE_ENCRYPTION」シンボルをコンパイラのプリプロセッサに追加します。そのため、e² studio で [SecureBootloader] プロジェクトを右クリックし、**[Settings]>[Cross ARM C Compiler]** の下にシンボルを追加します。PC Mastering (PC マスタリングプロジェクト) でも、同様にこの操作を実施します。

7.2.3　ハードウェア暗号化エンジンの使用の有効化（Enabling use of Hardware Crypto Engine）

ユーザはこのソフトウェアを使用して、暗号化操作にハードウェア要素またはソフトウェア要素を使用するかどうかを決定することができます。S5D9 は必要なハードウェア暗号化要素すべてをサポートしているので、最高の性能を達成するために、すべての要素を有効なままにすることを推奨します。

crypto_cfg.h ファイルは、SHA256 と ECC (署名と検証) に対するハードウェアサポートを有効にするために定義が必要なマクロを記述しています。

7.2.4　デバッグ出力の有効化（Enabling Debug Prints）

SCI8 シリアルポートでデバッグ出力を無効にするには、「DISABLE_PRINTF」マクロをコンパイラのプリプロセッサに追加します。そのために、e² studio で [SecureBootloader] プロジェクトを右クリックし、**[Settings]>[Cross ARM C Compiler]** の下でマクロを追加します。この作業により、コードサイズが非常に小さくなります。

7.2.5　OFS レジスタの設定（Configuring the OFS registers）

OFS register (OFS レジスタ) ビットは、mcu_rom_cfg.h ファイル内で設定できます。これらのレジスタの詳細は、MCU のハードウェアマニュアルを参照してください。

7.3　設定の概要（Configuration Summary）

マクロ	説明	設定方法	PC 向けプロジェクト	e²studio プロジェクト
RX_1ST_BUILD (1st Receiver のビルド)	現在動作しているコードが 1stRX (1st Receiver) であるかどうかを判定するために、フレームプロトコルの組み込み実装の中で使用します。変更しないでください。	プロジェクト側で定義	N/A	1ST_RX
ENABLE_WINDOWS_KEYSTORE (Windows のキーストローク有効化)	Microsoft CAPI (暗号化 API) を使用し、鍵の管理に基づいて Excel のシートを置き換えます。変更しないでください。	プロジェクト側で定義	KernelPackager、KernelInstaller、Mastering、Deploy (カーネルパッケージ化、カーネルインストーラ、マスタリング、展開)	N/A
ENABLE_FLASH-ACCESS_WINDOW (フラッシュアクセスウィンドウ FAW の有効化)	フラッシュアクセスウィンドウ (FAW) を設定する目的で使用します。	プロジェクト側で定義	KernelInstaller (カーネルインストーラ) または CertificateInstaller (証明書インストーラ)	N/A
ADDR_FAW_START (FAW 開始アドレス) **ADDR_FAW_END (FAW 終了アドレス)**	FAW のアドレス範囲を設定します	memoryMap.h	N/A	SecureBootloader (セキュアブートローダ)
ACTIVATE_THE_FSPR (FSPR の有効化)	FSPR ビットをロックします	プロジェクト側で定義	KernelInstaller (カーネルインストーラ) または CertificateInstaller (証明書インストーラ)	N/A
ENABLE_QSPI_FLASH_DOWNLOAD_SUPPORT (QSPI フラッシュダウンロードのサポートの有効化)	QSPI を Update Area (更新領域) として使用することを許可します	プロジェクト側で定義	Mastering、Deploy (マスタリング、展開)	すべて

ENABLE_QSPI_FLASH_USERDATA_SUPPORT (QSPI フラッシュユーザデータのサポートの有効化)	アプリケーション自体の一部として QSPI を使用するアプリケーションのプログラミングをサポートします。	プロジェクト側で定義	N/A	SecureBootloader (セキュアブートローダ)
ENABLE_SBM_OEM_CERTIFICATE (SBM OEM 証明書の有効化)	デバイス ID の作成とプログラミング (書き込み) をサポートします	プロジェクト側で定義	N/A	SecureBootloader、1ST_RX (セキュアブートローダ 1st Receiver)
ENABLE_ENCRYPTION (暗号化の有効化)	更新アプリケーションの暗号化と暗号化解除をサポートします	プロジェクト側で定義	Deploy (展開)	SecureBootloader (セキュアブートローダ)
ENABLE_JTAG_LOCKING (JTAG ロックの有効化)	JTAG ロック ID を設定します	プロジェクト側で定義	CertificateInstaller (証明書インストーラ)	N/A
ADDR_BOOTLOADER (ブートローダのアドレス) **BOOTLOADER_FLASH (ブートローダのフラッシュ)**	ブートローダの開始アドレス	memoryMap.h memoryMap.ld	すべて	すべて
ADDR_BOOTLOADER_MT (ブートローダ MT のアドレス) **MT_BOOTLOADER (ブートローダの MT)**	ブートローダモジュールテーブルの開始アドレス	memoryMap.h memoryMap.ld	すべて	すべて
ADDR_1ST_RX (1st Receiver のアドレス) **FIRST_RX (1st Receiver)**	1st Receiver のコードの開始アドレス	memoryMap.h memoryMap.ld	すべて	すべて
ADDR_1ST_RX_MT (1st Receiver MT のアドレス) **FIRST_RX_MT (1st Receiver の MT)**	1st Receiver モジュールテーブルの開始アドレス	memoryMap.h memoryMap.ld	すべて	すべて
ADDR_OEM_CERT (OEM 証明書のアドレス) **OEM_CERT (OEM 証明書)**	OEM 証明書を保持する領域の開始アドレス	memoryMap.h memoryMap.ld	すべて	すべて
SIZE_OEM_CERT (OEM 証明書のサイズ)	OEM 証明書領域のサイズ。フラッシュのブロックサイズ 1 つ分にする必要があります。	memoryMap.h	すべて	すべて
FLASH (フラッシュ)	アプリケーションの開始位置	memoryMap.ld	すべて	すべて
ADDR_MAIN_MT (メイン MT のアドレス) **MT_MAIN (メインの MT)**	アプリケーションのモジュールテーブルの開始位置	memoryMap.h memoryMap.ld	すべて	すべて
ADDR_PENDING_UPDATE (保留中の更新のアドレス) **PENDING_UPDATE (保留中の更新)**	更新のダウンロード先となる領域の開始位置	memoryMap.h memoryMap.ld	すべて	すべて
BOOTLOADER_RAM_START_ADDR (ブートローダ RAM の開始アドレス) **BOOTLOADER_RAM (ブートローダの RAM)**	ブートローダが使用する RAM の開始位置	memoryMap.h memoryMap.ld	すべて	すべて
APP_RAM_START_ADDR (アプリケーションの RAM の開始アドレス) **RAM**	ユーザアプリケーションが使用できる RAM の開始位置	memoryMap.h memoryMap.ld	すべて	すべて
BOOTLOADER_DF_START_ADDR (ブートローダのデータフラッシュの開始アドレス) **BOOTLOADER_DF_END_ADDR (ブートローダのデータフラッシュの終了アドレス)** **BOOTLOADER_DATA_FLASH (ブートローダのデータフラッシュ)**	ブートローダが使用するデータフラッシュ	memoryMap.h memoryMap.ld	すべて	すべて

7.4　カスタマイズ（Customization）

この章では、付属のソリューションに対してユーザが加えることのできるカスタマイズについて説明します。

7.4.1　更新（Update）

最終製品にサポートされるデータインタフェースは、デバイスにブートマネージャをインストールする時点では不明です。製品ごとに、異なるトランスポートインタフェース（transport interface）が搭載されます。独自のトランスポートプロ

トコルを実行している独自インタフェースを使用する可能性もあります。このためブートマネージャ内に、それら予測不能なインタフェース全般に対応するトランスポートすべてを記述するのは不可能です。

セキュア化されていない（unsecured）Synergy チップは、初期のアプリケーションプログラムをチップにインストールするために OEM が使用できる 以下の 3 種類の方法をサポートしています。

- JTAG
- Serial over SCI2 (SCI2 を経由するシリアル)
- USB over USB0 (USB0 を経由する USB)

JTAG によって、プロセッサ制御の要らない（uncontrolled）アクセスを実施できます。この結果、ユーザはすべてのメモリを表示し、ほとんどのメモリに対して変更を加えることができます。セキュアブートローダをインストールするプロセスでは、JTAG を無効にする必要があります。

セキュア化されていない Synergy MCU にある程度のセキュア機能を実現するため、ブートマネージャ（Boot Manager）インストールの一部の代替として、1st Receiver ダウンローダ（downloader）をセキュア化された Synergy MCU にインストールします。このアプリケーションは、標準的な Synergy ブートローダと同じプロトコル (Frame protocol) を使用し、「Serial over SCI2」(SCI2 を経由するシリアル) と USB-CDC をサポートします。アプリケーションをデバイスにインストールした後は、OEM は以下の選択肢のいずれかを選択できます。

1. 更新をインストール (および更新の失敗から回復) するためのメカニズムとして、1st Receiver ダウンローダを引き続き使用する。
2. 更新領域として QSPI を使用し、MCU メモリ領域はアプリケーション使用のために解放（free up）する。

OEM の場合は、OEM 側が開発するアプリケーションの一部として、更新トランスポートメカニズムを実装する必要があります。インストールを開始したアプリケーションとアップローダが、更新の転送を実施します。

7.4.2　更新戦略（Update Strategies）

更新戦略は、チップがサポートできる最大のアプリケーションサイズに影響を及ぼします。ブートマネージャは、更新戦略を以下のようにサポートしています。

Dual-application (デュアルアプリケーション) (デフォルト、上記の選択肢の上側に相当) アプリケーションフラッシュのうち最大 50% を、特定のアプリケーションに使用します。このアプリケーションに対応する、セキュリティ保護された新しいバージョンを更新領域 (「メモリマップ（Memory Map）」を参照) に渡した後、ブートローダを起動して新しいバージョンをインストールして、更新を実行します。最大アプリケーションサイズは、S5D9 デバイス上で約 900 K バイトです。この中には、フラッシュファイルシステム（flash file system）やキャリブレーションテーブル（calibration table）などに使用するコードフラッシュ全般も含まれます。

Direct Download (直接ダウンロード) (ベータ版 – 詳細はルネサスにお問い合わせください) 更新のダウンロード中に、通常の機能は行わなくてもよい製品の場合は、アプリケーション領域も更新領域として使用することができます。この結果、より広いアプリケーション領域が実現可能です。この方法を実現するには、ダウンローダをアプリケーションから分離し (例：1st Receiver を通常のダウンローダとして使用) 、更新のダウンロードプロセスを実行します。アプリケーション領域（Application Area）が更新領域（Update Area）と一部重複する（overlap）ように、メモリマップを変更する必要があります (memoryMap.h 内の ADDR_PENDING_UPDATE と、memoryMap.ld 内の PENDING_UPDATE を変更します)。現状と同様、更新領域は、アプリケーション領域が使用する 128 KB を残しておく必要があります。使用可能なメモリマップを以下に示します。

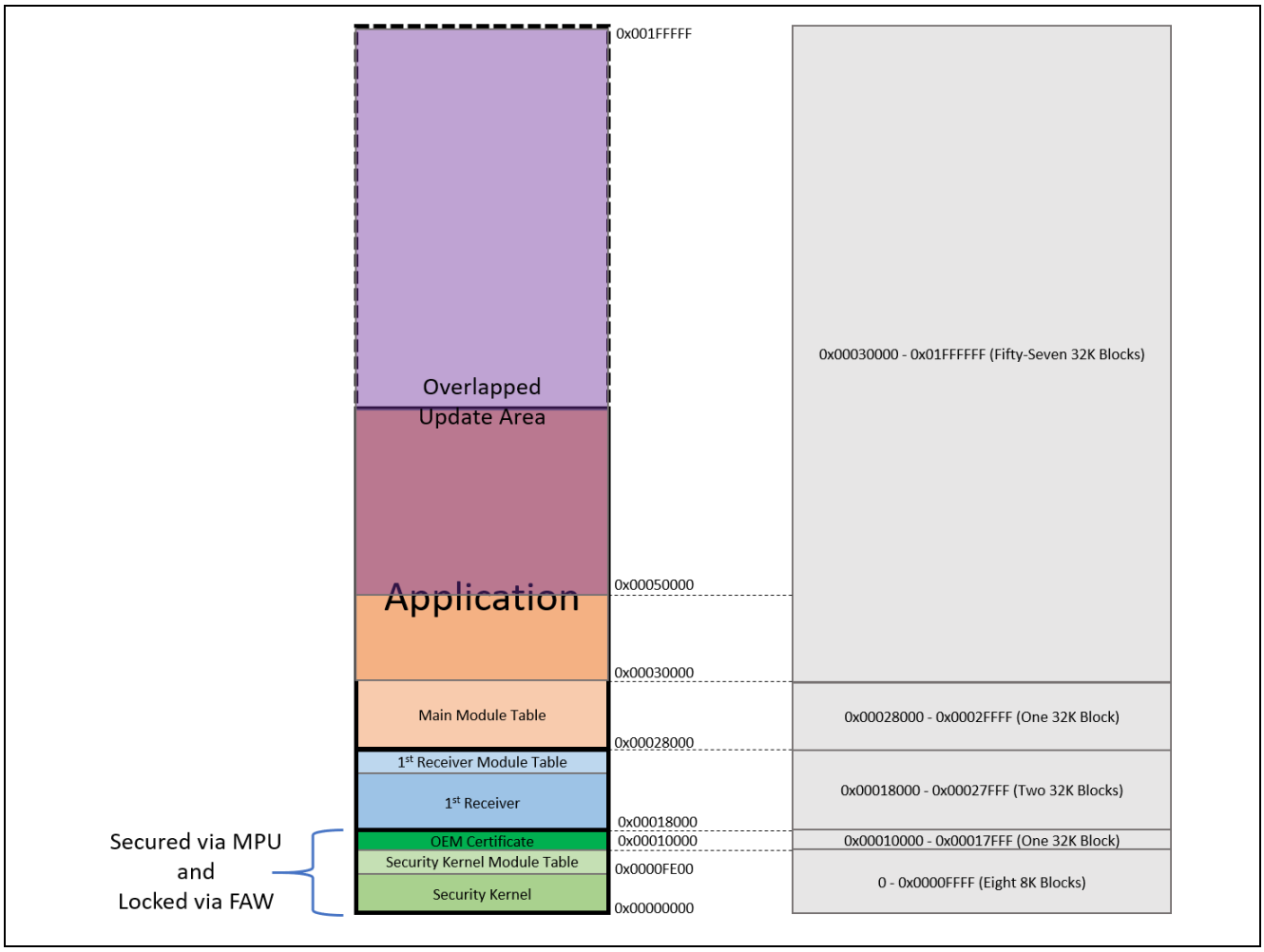


図 21. Direct Download (直接ダウンロード) モデルのメモリマップ

このアプローチの場合、メインアプリケーションは、ブートローダが更新をダウンロードする目的で 1st Receiver を強制的に実行する方法を準備する必要があります。これはユーザ定義関数を使用することで実現可能です。特定のデータフラッシュメモリが既知の値に設定された時点で 1st Receiver が実行されるようにユーザが SecurityUserForceExecution を定義すると、メインアプリケーションはそのデータフラッシュメモリをその既知の値に設定するだけで、NVIC 呼び出しを使用して、ソフトウェアリセットを強制的に実行します。リセット時には、SBM は SecurityUserForceExecution を実行します。この関数は 1st Receiver に対して true (論理的な「真」) を返した後、そのモジュールを実行します。更新が完了した後、メインアプリケーションはデータフラッシュのフラグをクリアすることができます。

7.4.3　ユーザ定義関数（User-defined Functions）

SecureBootloader (セキュアブートローダ) プロジェクトでは、セキュアブートローダのフローを制御する目的で、ユーザが定義できるフック（hook）を記述しています。アプリケーションが必要とする場合は、ユーザはこれらの関数を置き換えることができます。このようなフックの使用法については、「図 11」で説明しています。

これらの関数は、securityDebug.h と securityUser.h にリスト化されています。またサンプルの実装は securityDebug.c と securityUser.c で記述されています。アプリケーションに合わせて、ユーザがこれらをカスタマイズすることもできます。

7.4.4　メモリマップの設定（Configuring the memory map）

現在のメモリマップは、デバッグ領域を含めたすべての機能に十分な空間を割り当てるように定義されています。このメモリマップは、以下に示す、各組み込みプロジェクトのリンカファイルと共通のリンカファイルを通じて定義します。

- **\SecureBootloader\renesas\src\embedded\programs\<プロジェクト>\script\S5D9.ld**
- **\SecureBootloader\renesas\src\embedded\common\loaderScript\memoryMap.ld**

PC ツールは以下のファイルを通じて、同じメモリレイアウト情報を共有しています。

- **\SecureBootloader\renesas\src\common\api\memoryMap.h**

その結果得られたマップを図 7 に示しています。

これら 3 個のファイルすべてが常に同期した状態にあることが不可欠です。

システム要件に合わせるためにプロジェクトに対して何らかの設定変更を加えた後は、上記の各ファイルで定義しているメモリレイアウトにも変更を加えて、メモリ使用法を最適化してください。

注記: すべてのモジュールは、MCU のフラッシュブロック境界（flash block boundary）に合わせる（align）必要があります。(図 7 に図示) これは、全ての Synergy MCU で、消去動作はブロック単位で実行されることに合わせるための仕様です。

8. セキュアブートローダを伴った SSP アプリケーションの使用（Using an SSP application with the Secure Boot Loader）

Synergy Security Solution (セキュリティソリューション) は、ユーザアプリケーション機能には直接関与しません。このためユーザアプリケーションは、通常のアプリケーションと同様に開発することができます。ただし、いくつかの注意点を明確にするため、セキュアブートローダと組み合わせて使用する SSP アプリケーションの開発プロセスを、以下で説明します。セキュアブートローダアプリケーションで SSP アプリケーションを使用できるようにするには、共有システム要素（shared system element）について明確に理解する必要があり、下記事項を検討する必要があります。

1. SSP ユーザアプリケーションのリンカ（linker）セクションを更新し、コードの他の部分と重複しないようにする。
2. SSP configurator は、アプリケーション内で ROM レジスタ (FAW、OFS、S-MPU) を構成するオプションを表示しますが、実際は、これらレジスタはブートローダ内のみで構成されます。このためこれらのレジスタは、SBM を使用してプログラムする**必要があります**。
3. アプリケーションを更新するためには、アプリケーション側に、更新をダウンロードするため外部と通信を実行できる能力、およびフラッシュ API を使用して更新を内部の更新領域にプログラムできる能力が必要です。

以下に、新しい SSP アプリケーションが必要とするリンカ変更の実行ステップと、更新をダウンロードするために、SSP アプリケーションに対してフレームプロトコル実装を追加するのに必要なステップを示します。

8.1 リンカスクリプトの更新（Updating the linker script）

これらステップに関連するメモリレイアウトを理解するには、図 7 を参照してください。

1. 参考資料として、付属の app1 プロジェクトに対応するリンカスクリプトファイルを使用してください。このファイルは、以下のステップの説明に似た構造になっている、**script** フォルダ内にある S5D9.ld ファイルです。e² studio 内で開く場合、このファイルを開いた後、下部にある **[s5d9.ld]** タブをクリックすると、このファイルがテキストモードで表示されます。
2. 新しい SSP プロジェクトを作成し、必要とする任意の機能を追加します（特定のシーケンスを指定した Blinky など)。
3. SSP プロジェクトで、そのプロジェクトの scripts フォルダ内にある **s5d9.ld** リンカスクリプトファイルを開きます。e² studio 内で開く場合、このファイルを開いた後、下部にある [s5d9.ld] タブをクリックすると、このファイルがテキストモードで表示されます。このファイルの先頭で、MEMORY 領域は SSP プロジェクト向けのレイアウトになっています。
4. Bootloader（ブートローダ）プロジェクトで使用するメモリレイアウトは、\SecureBootloader \evaluation\ src\embedded\common\loaderScript\memoryMap.ld 内で定義されています。
5. アプリケーションのリンカスクリプトファイル内の MEMORY 定義を**置き換える**ために、S5D9.ld ファイルの先頭で **memoryMap.ld** ファイルをインクルードします。ただし、ID Code (ID コード) セクションは除きます。include で、memoryMap.ld ファイルのパス全体を指定する必要が生じることがあります。

```
INCLUDE memoryMap.ld

MEMORY

{

  ID_CODES (rx)      :ORIGIN = 0x0100A150, LENGTH = 0x10    /* 16 bytes (16 バイト)
*/

}
```

- メモリレイアウトを理解するために、memoryMap.ld ファイル全体を参照します。特に、ユーザアプリケーション向けに予約されている領域と、更新のための一時的な保持領域 (Update Area、更新領域) に注意してください。図 7 も参照してください。
- 「KEEP(*(.vectors))」という行の前に、「__Vectors_Start = .;」という行を追加します。SSP が使用する「__Vectors_Start」というシンボルに対応する定義を作成するうえで、この行が必須です。
- DTC ベクタテーブルのエントリをコメント化（Comment out）します。この結果、以下に示すように、DTC ベクタテーブルは強制的に固定の RAM 位置に配置されます。DTC ベクタテーブルを SSP プロジェクト内で使用する場合、RAM の中にこのテーブルが自動的に配置されます。

```
/*
.ssp_dtc_vector_table :
{
       .= ORIGIN(RAM);
       *(.ssp_dtc_vector_table)
} > RAM
*/
```

- プロジェクトをビルドします。ここで、セキュア製造プロセスまたはセキュア更新プロセスに対応する app1.srec の代わりに、この段階で生成された srec ファイルを使用します。

注記: ID Code (ID コード) と ROM register (ROM レジスタ) の各セクションは SSP アプリケーションの中で維持されますが、これらはメモリには書き込まれません。これらのメモリ位置が、ブートローダコードの中に設定されるだけです。SSP は、これらセクションが存在することを予期しているために、これらは維持されています。この点は将来修正される予定です。

9. 付録（Appendix）

9.1 用語集（Glossary）

用語	意味
非対称型暗号化アルゴリズム（Asymmetrical Encryption Algorithm）	鍵ペア (以下の項目を参照) を使用する暗号化アルゴリズムであり、暗号化と暗号化解除の両方で同じ鍵を使用する対称型暗号化と対をなします。
AES	Advanced Encryption System（高度暗号化システム）の略称です。対称型暗号化アルゴリズムの 1 つです。[2]
認証（Authentication）	デバイスを明確に識別することを意味します。
認証局（Certificate Authority、CA）	ポリシーベースのルールに従ってデジタル証明書 (「PKI」を参照) を発行する組織です。CA として使用できるのは、クラウド内に位置するパブリック CA またはプライベート CA、あるいは社内（オンプレミス、on-premises）CA (ローカル CA) です。社内 CA は通常、1 台のセキュア機器上でホストされています。
CNG	Microsoft 暗号化 API 新世代（Microsoft CryptoAPI New Generation）の略称です。この API は、マスタリングツールと展開ツールで、あらゆる暗号化操作に使用します。
暗号化（Cryptography）	サードパーティ (直接関係のない第三者) が存在する環境で、安全な通信を実現するための手法に関する実践と研究です。
暗号化強度（Cryptographic Strength）	攻撃に対する暗号化アルゴリズムの抵抗強度の測定値です。
デバイス（Device）	Renesas Synergy シリーズのシリコンチップです。

用語	意味
デバイス証明書 (ID)（Device Certificate (Identity)）	個別の Synergy デバイスを一意に識別する証明書です。この証明書はデジタル署名されており、証明書が既知の出所から到着したものであることや、変更 (改ざん) されていないこと、そのデバイスが信頼されている (信頼されたアプリケーションファームウェアとともに、Secure Boot Manager (セキュアブートマネージャ) がインストールされている) ことを示します。
デジタル署名（Digital Signature）	デジタルデータが既知の出所から供給されたものであり、改ざんされていないことを示すためのメカニズムです。通常は小規模なデータ (たとえば、64 バイト) であり、「既知の存在」である組織や個人がドキュメントの末尾に追加します。その後、受信者などの他者が、その「既知の存在」を識別する公開情報 (たとえば、証明書) を使用して、そのデータを検証します。
DSA	Digital Signature Algorithm（デジタル署名アルゴリズム）の略称です。
ECC	Elliptic Curve Cryptography（楕円曲線暗号化）の略称です。最新の非対称型暗号化アルゴリズムです。
ファクトリブートローダ（Factory Bootloader）	MCU 製造業者 (たとえばルネサス) の施設ですべての MCU にプログラムする (書き込む) 基本的なブートローダであり、消去や変更は不可能です。主に、開発段階や製造ラインでプログラミング (書き込み) をする際に使用します。
(暗号化) ハッシュ（(Cryptographic) Hash）	ハッシュ機能のうち、反転することが不可能、すなわち、ハッシュ値単独から入力データを再生成することが不可能だと考えられているものを指します。
整合性（Integrity）	デジタル項目 (たとえば、ファイル、ファームウェア、バイナリデータ) が既知の出所から供給されたものであり、意図的な改ざんやエラーによる変化のどちらも発生していないことを確実にします。
JTAG	IC のデバッグポートとして広く使用されている通信規格です。
鍵ペア (Key Pair)	非対称鍵（asymmetric key）は、公開鍵と秘密鍵のペアという形で生成されます。秘密鍵は特定の組織または個人のみが秘密のまま保持するもので、その組織または個人の身元を示す目的で使用します。公開鍵は自由に配布することができ、特定の秘密鍵と一意に関連付けられています。
モジュール（Module）	一連のコードで構成されたブロックのことで、デバイス上で実行できるほか、インストールと管理を他のモジュールから独立した形で実行できます。
モジュールテーブル（Module Table）	モジュールに関連付けられているさまざまなパラメータ (場所、サイズ、デジタル署名) を保持しています。
MPU	Memory Protection Unit（メモリ保護ユニット）の略称です。フラッシュと RAM を保護します。
PKI	Public Key Infrastructure（公開鍵基盤）の略称です。デジタル証明書の作成、管理、配布、使用、保存、取り消しを行うのに必要とされる、一連の役割、ポリシー、手続きです。公開鍵暗号化を通じてセキュア ID を管理する目的で、通常は PKI を使用します。
信頼の基点（Root of Trust）	システムが、整合性が確保されていると想定する必要のある特定のコンポーネントのことです。
RSA	Rivest, Shamir, Adelman（リベスト、シャミア、エーデルマン）の略称です。以前の非対称型暗号化アルゴリズムです。
RTOS	Real Time Operating System（リアルタイムオペレーティングシステム）の略称です。デバイス上で、スーパーバイザモードで動作する可能性のあるオペレーティングシステムのことです。
SCE	Secure Crypto Engine（セキュア暗号化エンジン）の略称です。MCU 内にあるモジュールの 1 つであり、効率的な低消費電力の暗号化アクセラレーションである TRNG (True Random Number Generation 真性乱数生成) を実現するほか、分離用暗号化鍵（isolating cryptographic key）の分離（isolation）を実施します。
セキュアブート（Secure Boot）	デバイス上で動作するすべてのコードが正確かどうか、また危殆化されていないかどうかをチェックするためのメカニズムです。
SHA-1	160 ビットの暗号化ハッシュアルゴリズムです [3]。
SHA-256	256 ビットの暗号化ハッシュアルゴリズムです [3]。
スーパーバイザモード（Supervisor Mode）	RTOS が通常動作している特権モード（Privileged mode）のことです。
ユーザモード（User Mode）	全体的に特権がないモードのことで、デバイスでアプリケーションを実行するときはこのモードを使用します。
SBM	Secure Boot Manager（セキュアブートマネージャ）の略称です。Synergy マイクロコントローラ上でアプリケーションバイナリのダウンロード、ブート、更新をセキュアに実施できるようにする一連のソフトウェアを指します。

9.2 参考資料（References）

1. NIST SP 800-57 Recommendation for Key Management Part 1 (sp800-57_part1_rev3_general.pdf)
2. AES: FIPS-197 (fips-197.pdf)
3. SHA-256: FIPS 180-4 (fips-180-4.pdf)
4. NIST Recommended Curves for Government use (NISTReCur.pdf)
5. RFC 7525 - Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) (rfc7525.txt)

Web サイトおよびサポート

以下のさまざまな URL にアクセスし、Synergy プラットフォームの主な要素に関する詳細を確認し、それらに関連するドキュメントをダウンロードし、サポートを活用してください。

Synergy ソフトウェア	www.renesas.com/synergy/software
Synergy ソフトウェアパッケージ	www.renesas.com/synergy/ssp
ソフトウェアアドオン	www.renesas.com/synergy/addons
ソフトウェア用語集	www.renesas.com/synergy/softwareglossary
開発ツール	www.renesas.com/synergy/tools
Synergy ハードウェア	www.renesas.com/synergy/hardware
マイクロコントローラ	www.renesas.com/synergy/mcus
MCU 用語集	www.renesas.com/synergy/mcuglossary
パラメトリック検索	www.renesas.com/synergy/parametric
キット	www.renesas.com/synergy/kits
Synergy ソリューション Gallery	www.renesas.com/synergy/solutionsgallery
パートナープロジェクト	www.renesas.com/synergy/partnerprojects
アプリケーションプロジェクト	www.renesas.com/synergy/applicationprojects
セルフサービスサポートリソース:	
ドキュメント	www.renesas.com/synergy/docs
ナレッジベース	www.renesas.com/synergy/knowledgebase
フォーラム	www.renesas.com/synergy/forum
トレーニング	www.renesas.com/synergy/training
ビデオ	www.renesas.com/synergy/videos
Web チケット	www.renesas.com/synergy/resourcelibrary

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.02	Oct 11.19	—	初版 英語版 R12UM0026EU0102 Rev.1.02 を翻訳

セキュアブートマネージャアーキテクチャユーザーズマニュアル

発行日: 2019.10.11

発行元: ルネサス エレクトロニクス株式会社

Renesas Synergy™ プラットフォーム
セキュアブートマネージャアーキテクチャ

Renesas Electronics Corporation

R12UM0026JU0102