関係各部署　御中

To departments concerned

Confidentiality classification	PROTECTED 関係者外秘	原紙保管 Storage of original	M/Y /
		コピー保管 Storage of copy	M/Y /

セキュアブート要求仕様書
Requirements Specification of Secure Boot

制御電子プラットフォーム開発部
制御ネットワーク・アーキ開発室 4G
System Network & Architecture Development Dept. 4G
E/E Architecture Development Div.

No. SEC-ePF-IDS-SBT-REQ-SPEC-a01-02-a

承認 Approved by	調査 Checked by	作成 Created by	
平林	平井 宮内	古川	2023/03/31

適用先 Target	リプロ/OTA 機能を有する ECU の内、以下のいずれかの機能を持つ ECU ・ メッセージフィルタリング機能 ・ 多層分離 ・ Cellular/Wi-Fi/Bluetooth 経由で通信を終端する機能 Allocated to ECUs that have a reprogramming/OTA function and one of those functions below. ・ Message filtering function ・ Multilayered separation function ・ Function that terminates communication through Cellular/Wi-Fi/Bluetooth
特記 Special note	【展開規則　Distribution rule】 必要に応じて、関係会社・関係部署（海外事業体、ボデーメーカ、ECU サプライヤ）への展開をお願いします。 Please distribute this document to affiliated companies, or departments (e.g. overseas business entities, car body manufacturers, or ECU suppliers) if necessary. 【問合せ先　Contact information】 制御電子プラットフォーム開発部　制御ネットワーク・アーキ開発室 セキュリティ仕様問合せ窓口 System Network & Architecture Development Dept. E/E Architecture Development Div. Contact for Security Inquiries email: epf-sec-sp@mega.tec.toyota.co.jp

変更履歴

Version	変更内容	日付	変更者
a01-00-a	SEC-19PF-SBT-REQ-SPEC-a00-04-a をベースに Post19 向けに新規作成 4.1. ソフトウェアの完全性検証に関する要求 【SBTREQ_00012】 ・RSA の暗号アルゴリズムを追加、鍵長を変更 ・公開鍵暗号方式に ECDSA を追加 ・指定外の暗号アルゴリズム使用について追記 【SBTREQ_00013、SBTREQ_00014】 ・+B ECU 時の対応を追加(WakeUp 等でも完全性検証必要) 5.2. 鍵運用に関する要求 【SBTREQ_01104】 ・統合 ECU の場合は、TMC で生成してもよい旨を追記	2021/04/05	46F 4G 松本
a01-00-b	英訳を追加	2021/05/14	46F 4G 松本
a01-01-a	・用語定義を追加 ・多段で完全性検証する際の補足を追加(SBTREQ_00005) ・バックグラウンドでの完全性検証の時間要求（SBTREQ_01001）を削除し、代わりに補足を SBTREQ_00007 に追加 ・揮発性メモリの改ざんは対象外であることを追記（SBTREQ_00013） ・表現の改善（SBTREQ_01003、SBTREQ_01104、2.3.3 の序文） ・ECU の解析者をサプライヤに限定しない表現に修正（SBTREQ_00101） ・適用範囲（表紙および 1.2）を変更	2021/12/03	46F 4G 竹山
a01-01-b	・SBTREQ_00013 補足の一部を要求として記載	2022/06/09	46F 4G 菅原
a01-02-a	・SBTREQ_01003 乱数要件の明確化 ・SBTREQ_01104 鍵生成要件修正 ・SBTREQ_01003, SBTREQ_01101, SBTREQ_01102, SBTREQ_01103, SBTREQ_01104 - ECU 外への要求であることを明記	2023/03/31	46F4G 古川

目次

1. はじめに

1.1. 本書の目的

本書の目的は、米国国立標準研究所（NIST）が作成したサイバーセキュリティ対策に関するフレームワークにおける「検知」機能を車載で実現するためのシステムの一部であるセキュアブートの要求を定義することである。

ソフトウェアの脆弱性を悪用され、不揮発性メモリに保管されたソフトウェアを改ざんされる攻撃に対して、セキュアブート技術を対策として導入する。

本書では、セキュアブートを実現するための手法を定義する。

1.2. 適用範囲

本書の適用範囲はリプロ/OTA 機能を有する ECU の内、以下のいずれかの機能を持つ ECU とする

- メッセージフィルタリング機能
- 多層分離
- Cellular/Wi-Fi/Bluetooth 経由で通信を終端する機能

1.3. 前提条件

なし

1.4. 要求事項の記載

【要求事項 : **】と記載されている部分が本書で要求する仕様とする。ただし、<補足>と記載されているものは補足事項のため要求する仕様ではない。

1.5. 関連文書

[1] 共通脆弱性対策要求仕様書, SEC-ePF-VUL-CMN-REQ-SPEC-###-##-#, 46F

1.6. 用語定義

本書で用いる用語を以下に示す。

表 1-1 用語定義

用語	解説
RoT	Root of Trust の略称。信頼の基点となるポイントを意味する。 セキュアブートにおいては、改ざんが行われていないか検証する処理を行わずに信頼を置くことができるデータおよびプログラムを指す。

2. 要求概要

2.1. システム構成

セキュアブートは単一 ECU に閉じたシステム構成をとる。

2.2. 技術解説

2.2.1. セキュアブート概要

セキュアブートとは、改ざんされたソフトウェアの実行を防止するためのセキュリティ対策である。事前に承認された（完全性が保証された）ソフトウェアのみ起動を許可することで、改ざんされたソフトウェアの実行を防止する。

セキュアブートの完全性検証は、はじめに RoT に保管された鍵・完全性検証プログラムを利用して、最初に起動するソフトウェアの起動前に完全性を検証する。

次に起動するソフトウェアも RoT もしくは、完全性が保証されたソフトウェア内にある鍵・完全性検証プログラムを利用して、完全性を検証実施後に起動する。

上記のように、ソフトウェアの起動前に常に RoT もしくは、完全性が保証されたソフトウェア内にある鍵・完全性検証プログラムを利用して完全性検証を実施する。

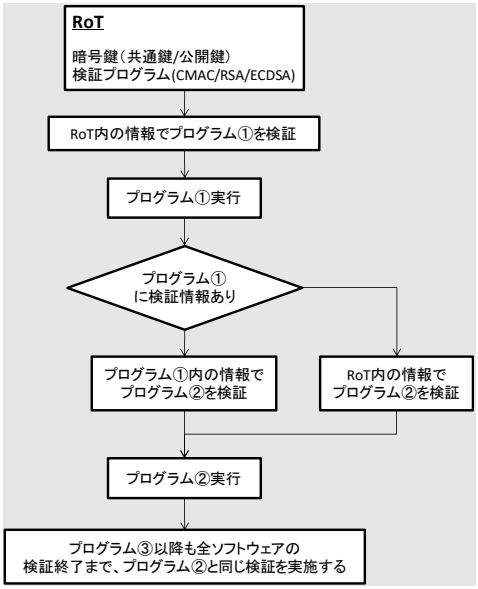

図 2-1 セキュアブートの完全性検証フロー

ECU 起動時 or リセット時に使用している全ソフトウェア領域に対してセキュアブートを実施すること。

しかし、セキュアブートは暗号処理等、高負荷処理が必要なため、ECU の起動時間が増加し、起動条件が満たせない可能性がある。ECU の起動条件を満たせない場合に限り、以下のようにセキュアブートを実施することを許容する。

(1)ECU 起動時 or リセット時は少なくとも(※)重要な領域に対してソフトウェアの完全性検証を実

施

⑵⑴で検証未実施の領域に対してバックグラウンドで完全性検証を実施

（※）重要な領域とは「リプログラミングソフトウェア」、「BootLoader」及び、各システム部署・ECU 設計部署で重要と判断したソフトウェア領域を指す。

セキュアブート失敗時の動作は、**表 2-1** を参照すること。

表 2-1　セキュアブートエラー時の動作

	ECU 起動時 or リセット時	バックグラウンド
重要な領域	セキュアブートに失敗した制御ソフトウェアは起動しないこと(※)。制御ソフトウェア以外の領域はお客様の安全に影響がでない動作とすること。	ー
重要な領域以外		お客様の安全に影響がないよう、機能縮退やフェールセーフを考慮した動作をする

(※) ソフトウェアを起動しないことにより安全に影響がある ECU は、セキュリティ主管部署と調整すること

＋B ECU は、ECU 起動時 or リセット時に加え、WakeUp 時にも全ソフトウェア領域に対してセキュアブートを実施すること。ただし、起動制約等により WakeUp 時に全ソフトウェア領域ののセキュアブート実施が難しい場合は、バックグランド検証を許容する。

セキュアブートエラー時の動作は、**表 2-1** を参照。WakeUp 時は“ECU 起動時 or リセット時”に、WakeUp 以外は“バックグラウンド”に該当する。

2.3. 動作シーケンス

ECU 起動時 or リセット時 or WakeUp 時の動作概要およびバックグラウンド時の動作概要を「2.3.1ECU 起動時 or リセット時 or WakeUp 時の動作概要」、「2.3.2 バックグラウンド時 or Sleep 時 or IG-OFF 時の動作概要」 にフローにて示す。

2.3.1. ECU 起動時 or リセット時 or WakeUp 時の動作概要

ECU 起動時 or リセット時 or WakeUp 時の動作概要をフローにて示す。

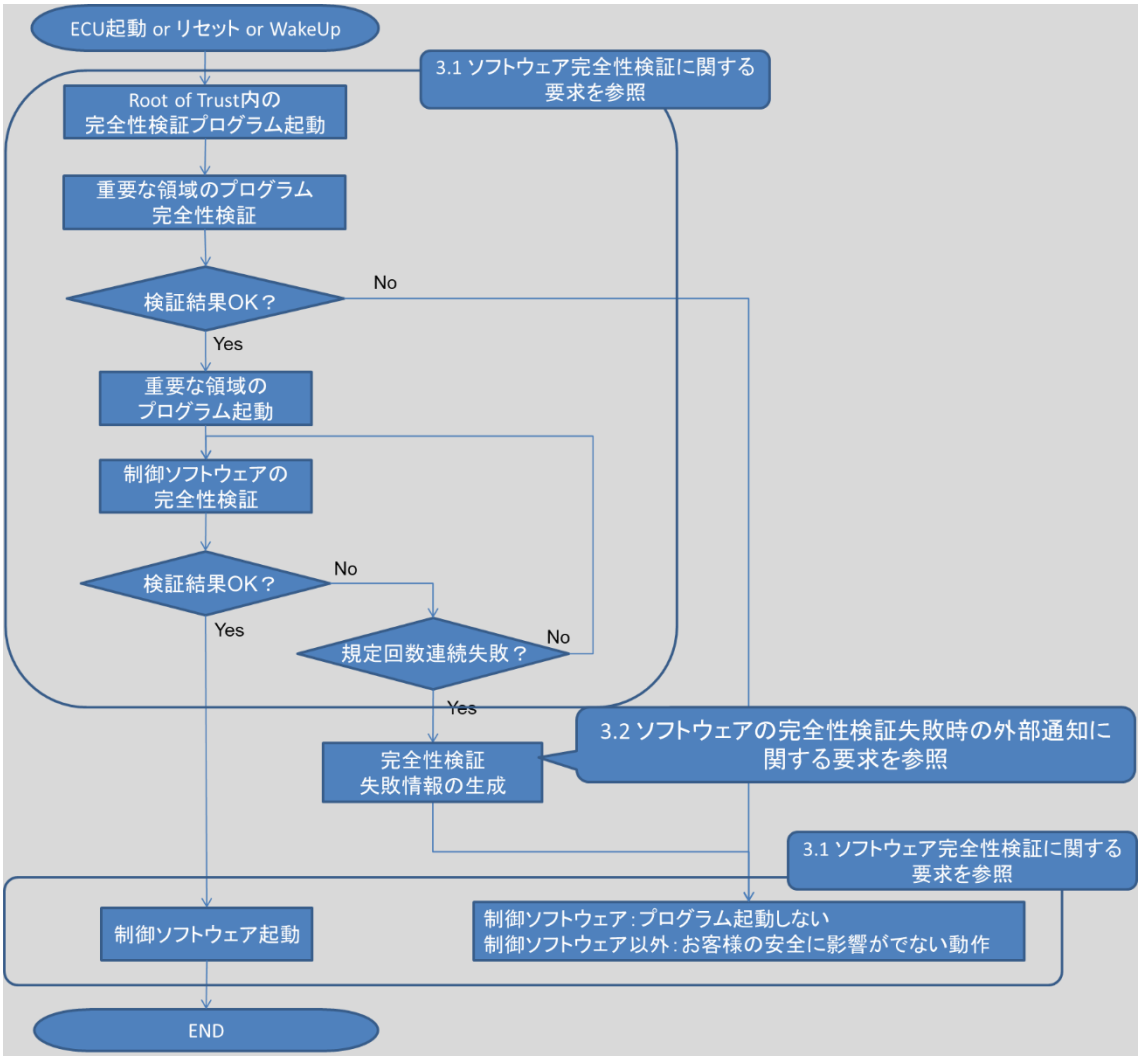

図 2-2 ECU 起動時 or リセット時 or WakeUp 時の動作概要フロー

2.3.2. バックグラウンド時 or Sleep 時 or IG-OFF 時の動作概要

バックグラウンド時 or Sleep 時 or IG-OFF 時の動作概要フローを以下に示す。

※ECU 起動時 or リセット or WakeUp 時に使用している全ソフトウェア領域の完全性検証を実施する場合、本章の動作は不要。

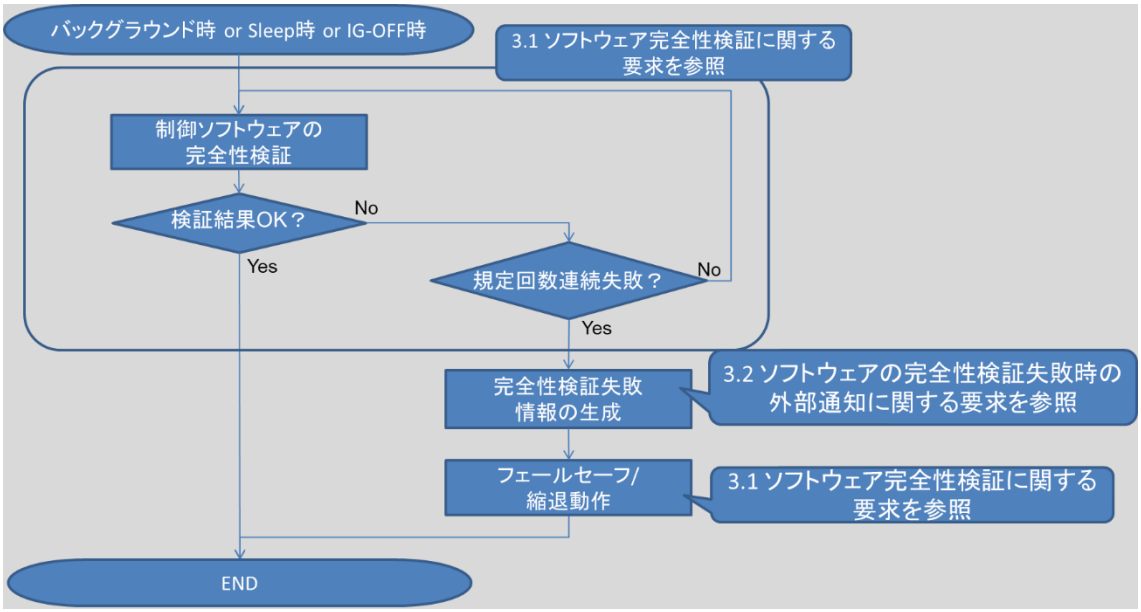

図 2-3 バックグラウンド時 or Sleep 時 or IG-OFF 時の動作概要フロー

2.3.3. ソフトウェアの完全性検証の実現方法

ソフトウェアの完全性検証を実現する暗号アルゴリズムは、「3.1 ソフトウェアの完全性検証に関する要求」を参照すること。

また、鍵漏洩による影響を最小限にするため、以下のように鍵を分けること。

表 2-2 セキュアブートの鍵

	ソフトウェアを検証する鍵
共通鍵暗号方式	ECU 個別 もしくはソフト品番別とすること。 ただし、リプログラミングでは変わらないようにすること。
公開鍵暗号方式	ECU ノード名＋サプライヤ＋マイコン型式(※1)

(※1) 例：ECU_A のサプライヤが 2 社(サプライヤ A,サプライヤ B)でマイコン型式がサプライヤごとに違う場合、ECU_A のサプライヤ A で１つ、ECU_A のサプライヤ B で１つ

2.3.3.1. セキュアブートの実現方法例

本章では、以下の場合の 2 例を示す。

＜実現方法例 1＞

RoT に保存された情報を利用して、ソフトウェアの完全性を検証する実装例を以下に記載する。

<a>ECU 起動時 or リセット時 or WakeUp 時、完全性検証プログラムで重要な領域のソフトウェアを、最初に起動するソフトウェアから順々に完全性を検証する。

【共通鍵方式】

①重要な領域の MAC を生成

②フラッシュ ROM 内の重要な領域の MAC と①を比較

③比較した結果一致していた場合、OK と判断しソフトウェアを起動する

【公開鍵方式】

①重要な領域のソフトウェアからハッシュ値を演算

②フラッシュ ROM 内の署名からハッシュ値を復号し、①のハッシュ値を比較

③比較した結果一致していた場合、OK と判断しソフトウェアを起動する

<b><a>の検証結果が OK だった場合、重要な領域外のソフトウェアを順々に完全性検証する。

【共通鍵方式】

①重要な領域外の MAC を生成

②フラッシュ ROM 内の MAC と①を比較

③比較した結果一致していた場合、OK と判断しソフトウェアを起動する

【公開鍵方式】

①重要な領域外のソフトウェアからハッシュ値を演算

②フラッシュ ROM 内の署名からハッシュ値を復号し、①のハッシュ値を比較

③比較した結果一致していた場合、OK と判断しソフトウェアを起動する

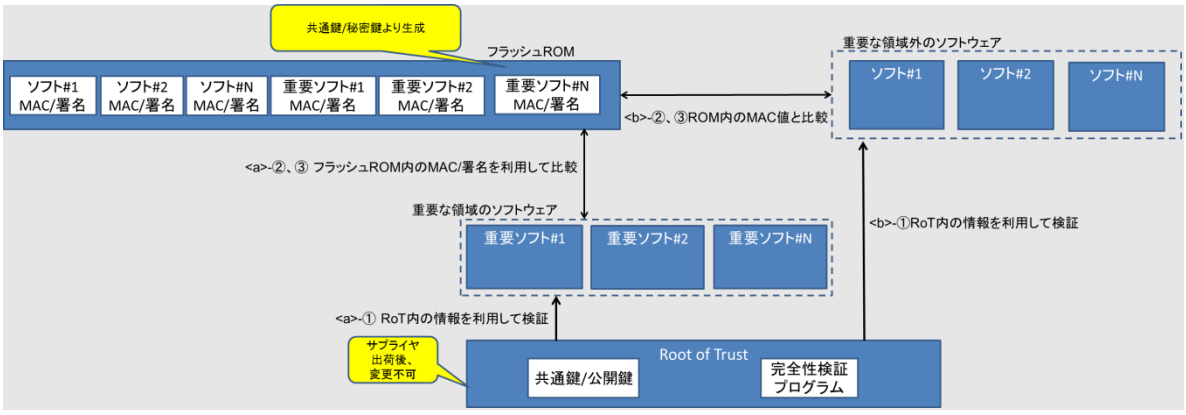

図 2-4 セキュアブートの実現方法例 1

＜実現方法例 2＞

RoT に保存された情報を利用して、最初に起動するソフトウェアの完全性を検証する。
次に完全性検証が正常に実行されたソフトウェア内にある情報と RoT 内の鍵を利用して、2 番目以降に起動するソフトウェアの完全性を順々に検証する実装例を以下に記載する。

<a>ECU 起動時 or リセット時 or WakeUp 時、完全性検証プログラムで重要な領域のソフトウェアの最初に起動するソフトウェアの完全性を検証する。

【共通鍵方式】

①RoT 内の情報を利用して重要な領域の MAC を生成
②フラッシュ ROM 内の重要な領域の MAC と①を比較
③比較した結果一致していた場合、OK と判断しソフトウェアを起動する

【公開鍵方式】

①RoT 内の情報を利用して重要な領域のソフトウェアからハッシュ値を演算
②フラッシュ ROM 内の署名からハッシュ値を復号し、①のハッシュ値を比較
③比較した結果一致していた場合、OK と判断しソフトウェアを起動する

<b><a>の検証結果が OK だった場合、重要な領域の 2 番目に起動するソフトウェアの完全性を検証する。

【共通鍵方式】

①<a>で検証したソフトウェア内の完全性検証プログラムを利用して重要な領域の MAC を生成
②<a>で検証したソフトウェア内の重要な領域の MAC と①を比較
③比較した結果一致していた場合、OK と判断しソフトウェアを起動する

【公開鍵方式】

①<a>で検証したソフトウェア内の完全性検証プログラムを利用して重要な領域のハッシュ値を演算
②RoT 内の鍵と<a>で検証したソフトウェア内の署名からハッシュ値を復号し、①のハッシュ値を比較
③比較した結果一致していた場合、OK と判断しソフトウェアを起動する

<c><b>の検証結果が OK だった場合、<b>の最後に検証したソフトウェア内にある情報を利用して、重要な領域外のソフトウェアの最初に起動するソフトウェアから順々に完全性を検証する。

【共通鍵方式】

①RoT 内の鍵と<b>で最後に検証したソフトウェア内の完全性検証プログラムを利用して、重要な領域の MAC を生成

②<b>で最後に検証したソフトウェア内の重要な領域外の MAC と①を比較

③比較した結果一致していた場合、OK と判断しソフトウェアを起動する

【公開鍵方式】

①<b>で最後に検証したソフトウェア内の完全性検証プログラムを利用して重要な領域外のハッシュ値を演算

②RoT 内の鍵とフラッシュ ROM 内の署名からハッシュ値を復号し、①のハッシュ値を比較

③比較した結果、一致していた場合、OK と判断しソフトウェアを起動する

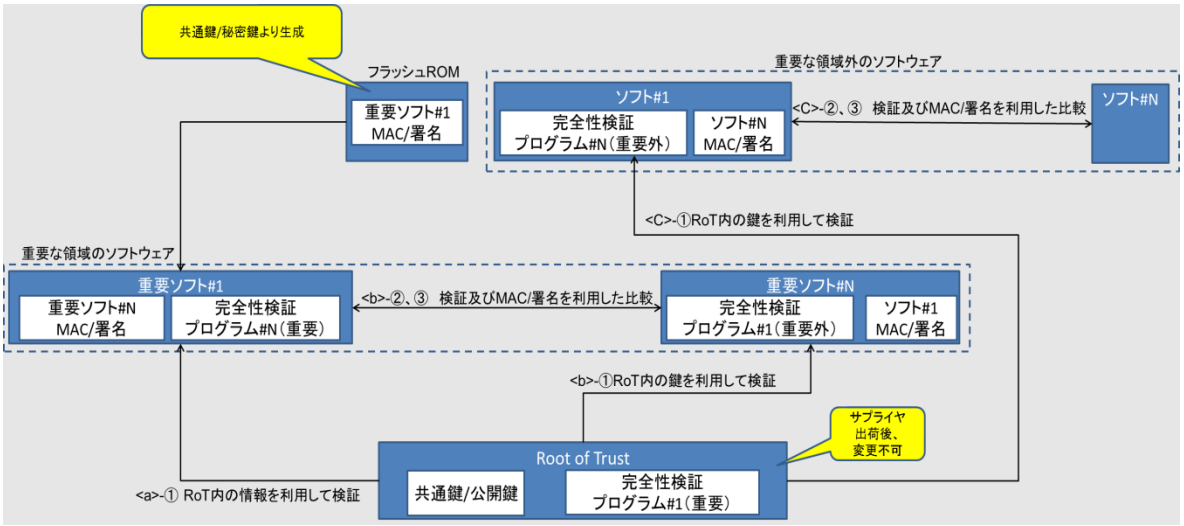

図 2-5 セキュアブートの実現方法例 2

2.3.4. ソフトウェア更新時の処理概要

リプログラミングにてソフトウェアを更新した場合、ソフトウェアの内容が変更になるため、MAC/署名の更新が必要となる。「2.3.4.1MAC/署名更新の処理概要」に更新処理概要を記載する。

2.3.4.1. MAC/署名更新の処理概要

以下にデジタル署名更新の処理概要を記載する。

・更新するソフトウェア生成後、該当ソフトウェア領域のデジタル署名を生成

・更新するソフトウェアおよびデジタル署名を一緒に更新

本章では、公開鍵暗号方式採用時の署名更新を例に記載しているが、共通鍵暗号方式の MAC 更新も同様の処理を行うこと。

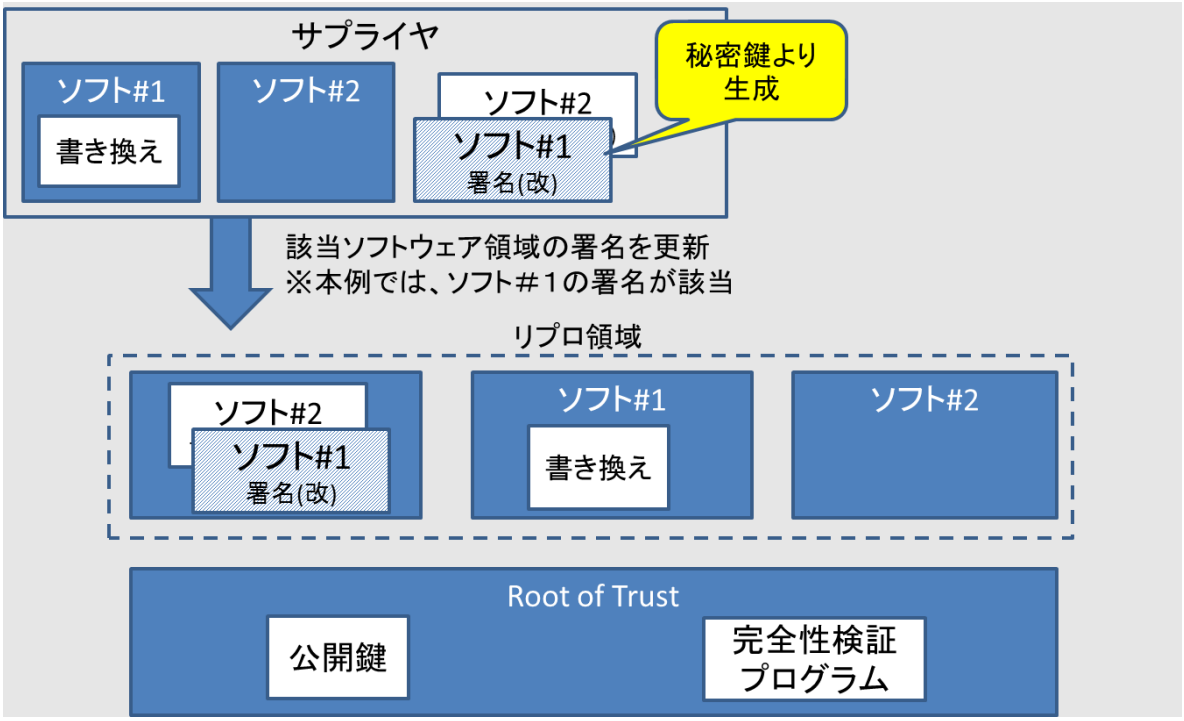

図 2-6 デジタル署名更新の処理概要

2.4. 鍵運用概要

セキュアブートは、鍵が他 ECU と同じ値である。万が一秘密鍵・共通鍵が漏洩した場合、他 ECU のセキュアブートも突破される可能性があるため、鍵漏洩判明後の新規生産の ECU の鍵は、漏洩した鍵と異なる値とすること。

2.5. 要求一覧

また、各エンティティが対応すべき要求事項の一覧を“表 2-3 要求事項対応表”に示す。要求事項の詳細については、3 章以降を参照。

表 2-3 要求事項対応表

要求事項	ハードウェア関連要件
SBTREQ_00001	○
SBTREQ_00002	-
SBTREQ_00003	-
SBTREQ_00004	-
SBTREQ_00005	-
SBTREQ_00006	-
SBTREQ_00007	-
SBTREQ_00008	-
SBTREQ_00009	-
SBTREQ_00010	-
SBTREQ_00011	-
SBTREQ_00012	-
SBTREQ_00013	-
SBTREQ_00014	-
SBTREQ_00101	-
SBTREQ_01002	-
SBTREQ_01003	○
SBTREQ_01101	-
SBTREQ_01102	-
SBTREQ_01103	-
SBTREQ_01104	-

3. 機能要求詳細

3.1. ソフトウェアの完全性検証に関する要求

【要求事項：SBTREQ_00001】

RoT には少なくとも以下の内容を格納すること。

＜保管情報＞

・最初に起動するソフトウェアの完全性検証プログラム

・共通鍵（共通鍵暗号方式採用時）

・公開鍵（公開鍵暗号方式採用時）

・公開鍵は鍵の代わりにハッシュ値を保管しても良い

<保管要件>

・完全性検証プログラム

改ざんを防ぐため、HSM 内に保管するか、もしくは書き換えのできないメモリ(e.g. OTP メモリ)に保管すること

・共通鍵

改ざん、漏洩を防ぐため、HSM 内に保管すること

・公開鍵/ハッシュ値

改ざんを防ぐため、HSM 内に保管するか、もしくは書き換えのできないメモリ(e.g. OTP メモリ)に保管すること

【要求事項：SBTREQ_00002】

MAC/署名を格納する領域は ECU 毎に決定すること。

【要求事項：SBTREQ_00003】

ECU 起動またはリセット後、ソフトウェア起動前に完全性検証プログラムを起動すること。

【要求事項：SBTREQ_00004】

RoT 内の完全性検証プログラムで、少なくとも最初に起動するソフトウェアの完全性を検証すること。

【要求事項：SBTREQ_00005】

ソフトウェアの完全性検証は、RoT に保管された完全性検証プログラムを利用した検証から順々に検証のチェーンを組む事。（図 2-1 参照）

<補足> RoT 以外に保管した公開鍵や完全性検証プログラムを使用する場合、前段階の検証により、その公開鍵や完全性検証プログラムの完全性が保証されていること。

【要求事項：SBTREQ_00006】
重要な領域の完全性検証に成功した場合、重要な領域外の完全性検証を行うこと。

【要求事項：SBTREQ_00007】
ソフトウェアの完全性検証は、使用している全ソフトウェア領域に対して実施すること。
ただし、ソフトウェアの書き換えが不可能な領域(例：マスク ROM 領域)と未使用領域は除外してもよい。
未使用領域とは、実行しない領域、Flash 書込みしない領域を指す。

ECU の時間制約等により、起動 or リセット時に使用している全ソフトウェア領域の完全性検証が難しい場合は、BootLoader、リプログラミングソフトウェア、システム・ECU 設計部署毎で判断した重要機能の完全性検証を起動時に行うこと。

ECU 起動時 or リセット時に完全性検証が出来なかったソフトウェア領域は、アプリケーション起動後、バックグラウンドでの完全性検証を許容する。
※マイコン制約等で上記動作の実現が困難な ECU は、セキュリティ主管部署と調整すること。
<補足>バックグラウンドでの完全性検証は可能な限り速やかに行うこと

【要求事項：SBTREQ_00008】
(ECU 起動時 or リセット時の完全性検証)
該当ソフトウェアを実行するまでに、完全性検証すること。

【要求事項：SBTREQ_00009】
ソフトウェアの完全性検証に失敗した場合、一時的な故障を考慮し、ソフトウェアの完全性検証をリトライ or ECU リセットすること。
リトライ or ECU リセットの回数は、少なくとも 1 回とする。

(例外 1)ソフトウェアの完全性検証前に一時的な故障を別の方法で検知し、リトライ出来る場合は、完全性検証のリトライ動作は必須ではない。
(例外 2)BootLoader、リプログラミングソフトウェアの完全性検証に失敗した場合はリトライ、ECU リセットは必須としない。

【要求事項：SBTREQ_00010】
＜ECU 起動時 or リセット時の完全性検証＞
リトライ or ECU リセットを含めソフトウェアの完全性検証が成功しなかった場合、**表 2-1** の振舞いを参照すること。(※1)

(※1)ソフトウェアを起動しないことにより、安全に影響がある ECU は、セキュリティ主管部署と調整すること。

＜バックグラウンドの完全性検証＞
リトライ or ECU リセットを含め、ソフトウェアの完全性検証が成功しなかった場合、お客様への安全に影響がないよう、機能縮退やフェールセーフを考慮した動作をすること。

【要求事項：SBTREQ_00011】
リプログラミングに伴う MAC/デジタル署名の更新は以下に従うこと。

＜共通鍵暗号方式採用時＞
・リプログラミング対象ソフトウェアと合わせて、該当のソフトウェア領域に対する MAC を書き込むこと。(※1)
・リプログラミング後は、書き込まれた MAC を用いてソフトウェアの完全性を検証すること。

＜公開鍵暗号方式採用時＞
・リプログラミング対象のソフトウェアと合わせて、該当のソフトウェア領域に対するデジタル署名を書き込むこと。(※1)
・リプログラミング後は、書き込まれたデジタル署名を用いてソフトウェアの完全性を検証すること。

(※1)MAC またはデジタル署名の書き込みは、リプログラミングによってソフトウェアと合わせて書き込むため、セキュアブート機能による MAC またはデジタル署名の生成処理は不要である。

【要求事項：SBTREQ_00012】
暗号アルゴリズムは、関連文書[1]の暗号アルゴリズムに関する対策要件に従うこと。
以下にアルゴリズムの例を示す。
＜共通鍵暗号方式採用時の例＞
・暗号アルゴリズム：AES
・鍵長：128bit
・ブロック長：128bit
・MAC 生成アルゴリズム：CMAC 方式

＜公開鍵暗号方式(RSA)採用時の例＞
・暗号アルゴリズム：RSASSA-PKCS1_v1_5 or RSASSA-PSS
・鍵長：3072bit 以上
・ハッシュ関数：SHA-256 以上
・鍵生成パラメータ：e=65537

＜公開鍵暗号方式(ECDSA)採用時の例＞
・暗号アルゴリズム：ECDSA
・鍵長：256bit 以上
・ハッシュ関数：SHA-256 以上
・楕円曲線：P-256

【要求事項：SBTREQ_00013】
+B ECU は、ECU 起動時やリセット時に加え、WakeUp 時にもソフトウェアの完全性検証を実施すること。ただし、起動制約等により WakeUp 時に全ソフトウェア領域の完全性検証が難しい場合は、バックグランドでの検証を許容する。また、Sleep 時に完全性検証済のソフトウェアが保持している揮発性メモリのソフトウェア領域は WakeUp 時の完全性検証の対象外である(※)。
<補足>
※セキュアブートの目的は不揮発性メモリに保管されたソフトウェアの改ざん検知のため

【要求事項：SBTREQ_00014】
WakeUp 時のソフトウェアの完全性検証の要求は、ECU 起動時 or リセット時と同じとする。

3.2. ソフトウェアの完全性検証失敗時の外部通知に関する要求

【要求事項：SBTREQ_00101】
ECU を解析することにより、ソフトウェア完全性検証が失敗しているかどうかが判別できること。

4. 非機能要求詳細

4.1. ソフトウェアの完全性検証に関する性能要求

【要求事項：SBTREQ_01002】
ソフトウェアの完全性検証を実施(リトライ or ECU リセット含む)
しても ECU 起動時間等の性能要求には影響を与えないこと。
※実現が困難な ECU は、セキュリティ主管部署と調整すること。

4.2. 鍵運用に関する要求

【要求事項：SBTREQ_01101】
本要求は ECU 外で生成する鍵への要求である。
共通鍵/秘密鍵が漏洩した場合、共通鍵/公開鍵の値を更新した ECU のソフト品番を更新して出荷すること。

【要求事項：SBTREQ_01102】
本要求は ECU 外で生成する鍵への要求である。
ソフトウェアのリプログラミングが共通鍵暗号を用いて暗号化している場合、リプログラミングの共通鍵とソフトウェアの完全性検証の共通鍵は、異なる鍵を使うこと。

【要求事項：SBTREQ_01103】
本要求は ECU 外で生成する鍵への要求である。
＜共通鍵暗号方式採用時＞
・共通鍵は、ソフト品番ごとにユニークにすること。ただし、リプログラミングでは変わらないようにすること。
＜公開鍵暗号方式採用時＞
公開鍵、秘密鍵は、ECU×サプライヤ×マイコン型式で１つにすること。

【要求事項：SBTREQ_01104】
本要求は ECU 外で生成する鍵への要求である。
ECU 設計部署は、セキュアブートで使用される以下鍵の生成者を決めること。

＜共通鍵暗号方式採用時＞
共通鍵
＜公開鍵暗号方式採用時＞
公開鍵、秘密鍵

【要求事項：SBTREQ_01003】
本要求は ECU 外で生成する鍵への要求である。
鍵の生成者は、乱数を使用して【要求事項：SBTREQ_01104】の鍵を生成すること。
関連文書[1]の基準を満たした真正乱数生成器もしくは疑似乱数生成器で鍵生成に用いる乱数を生成すること。

5. 設計値

本書で定義すべき設計値はなし。

Revision history

Version	Change	Date	Reviser
a01-00-a	First version based on No. SEC-19PF-SBT-REQ-SPEC-a00-04-a issued for Post19ePF 4.1. Software integrity validation requirements 【SBTREQ_00012】 ・RSA cryptographic algorithm added, and key length changed ・ECDSA added in Public Key Encryption ・Usage of unspecified cryptographic algorithms added 【SBTREQ_00013、SBTREQ_00014】 ・Support during +B ECU added (integrity verification also required for WakeUp, etc.) 5.2. Key operation requirements 【SBTREQ_01104】 ・Added that keys for an integrated ECU may be generated by TMC	2021/04/05	46F 4G Matsumoto
a01-00-b	English translation added	2021/05/14	46F 4G Matsumoto
a01-01-a	- Glossary added - Supplementary information on integrity verification added (SBTREQ_00005) - Requirement of time limit of background integrity verification deleted (SBTREQ01001) and supplementary note added (SBTREQ_00007) - Added that tampering non-volatile is out of target (SBTREQ_00013) - Readability improved (SBTREQ_01003, SBTREQ_01104, the preface of 2.3.3) - Analyzer of ECU changed to one not restricted to ECU supplier (SBTREQ_00101) - Target of this document changed (the cover and 2.1)	2021/12/03	46F 4G Matsumoto
a01-01-b	- SBTREQ_00013 The part of the note moved to requirement	2022/06/09	46F 4G Takeyama
a01-02-a	- SBTREQ_01003 Random number requirement clarified - SBTREQ_01104 Key generation requirement modified - SBTREQ_01003, SBTREQ_01101, SBTREQ_01102, SBTREQ_01103, SBTREQ_01104 - Clarified that the requirements are allocated to the outside of car	2023/03/31	46F 4G Furukawa

Table of Content

1. Introduction

1.1. Purpose of this Document

The purpose of this document is to define the requirements for secure boot, a part of the system for in-vehicle implementation of the "detection" function in the framework for cyber security measures developed by the U.S. National Institute of Standards and Technology (NIST).

This document introduces secure boot technology as a countermeasure against attacks where software vulnerabilities are exploited and software stored in non-volatile memory is tampered.

In this document, we define a methodology to achieve secure boot.

1.2. Scope of this Document

Allocated to ECUs that have a reprogramming/OTA function and one of those functions below.

- Message filtering function
- Multilayered separation function
- Function that terminates communication through Cellular/Wi-Fi/Bluetooth

1.3. Prerequisites

None

1.4. Description of requirements

We describe requirements as [Requirement: **] in this document where <Note> means just a supplementary note.

1.5. Related Documents

[1] Requirements Specification of Common Vulnerability Countermeasure, SEC-ePF-VUL-CMN-REQ-SPEC-###-##-#, 46F

1.6. Glossary

Table 1-1 List of Terms

Term	Meaning
RoT	The abbreviation for Root of Trust. It means point that serves as starting point of trust. In the term *secure boot*, it refers to data or program that can be trusted without process of verifying whether it has been tampered, or to place where the program or/and program are stored.

2. Summary of requirements

2.1. System configuration

Secure boot comprises one ECU.

2.2. Technology description

2.2.1. Outline of secure boot

Secure boot is security to prevent tampered software from being executed. It allows only software which has been approved beforehand (or whose integrity has been assured) to be launched and thereby it can prevent tampered software from being executed.

Verifying integrity of the first software to be launched by secure boot is conducted with the key and integrity verification program stored in RoT before the software is launched.

Software to be launched after the first some is also launched after the integrity verification of the software with RoT or the key and integrity verification program stored in the software whose integrity has been verified.

As described above, integrity verification is always conducted with RoT or key and integrity verification program stored in software whose integrity has been verified.

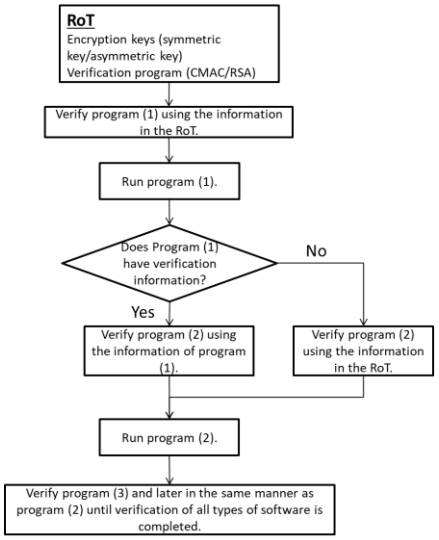

Figure 2-1 Flow of integrity verification by secure boot

During ECU StartUp or Reset, secure boot is performed to all software to be used. However, there is the possibility that requirements to ECU startup are not satisfied since secure boot need much machine power for encryption procedure and so on and therefore that makes the ECU startup time longer. Only if requirements to ECU startup cannot be satisfied, such secure boot as some shown below is accorded.

(1) During ECU StartUp and Reset, verifying integrity of critical software (*1) is performed at least.

(2) Verifying integrity of software that is not verified in (1) is performed in the background.

(*1) Critical software refers software including "reprogramming software", "BootLoader" and other software considered critical by ECU designers.

Refer to **Table 2-1** for the operations when secure boot fails.

Table 2-1 Operations when secure boot errors occur

	During ECU StartUp or Reset	In the background
Critical software	Control software that failed with secure boot shall not be started (*1). Software other than control some shall operate so that it does not affect the safety of passengers.	-
Other		Non-critical software shall perform such an operation as fallback or fail-safe not to endanger the safety of passengers.

(*1) If not starting up software in ECU affects the safety, discuss operations not to affect the safety with security department.

+B ECUs shall perform secure boot to all software during WakeUp as well as ECU StartUp and Reset. However, if it is difficult to perform secure boot to all software during WakeUp due to constraints on startup, +B ECUs may perform secure boot to the rest of software in the background. Moreover, ECUs in the second or further layer may perform secure boot after starting up software such as During ECU Sleep or IG-OFF. See Table 2-1 for operations when secure boot errors occur. *During Wakeup* falls into *During ECU Startup or Reset* and other than *at Wakeup* fall into *In the background*.

2.3. Operation sequences

We show operation sequences of *During ECU StartUp, Reset* and *Wakeup* and *In the background* in 2.3.1 and 2.3.2.

2.3.1. Overview of Operation duirng ECU StartUp, Reset and WakeUp

We show overview of operation at ECU Startup, Reset and WakeUp by flowchart (Figure 2-2).

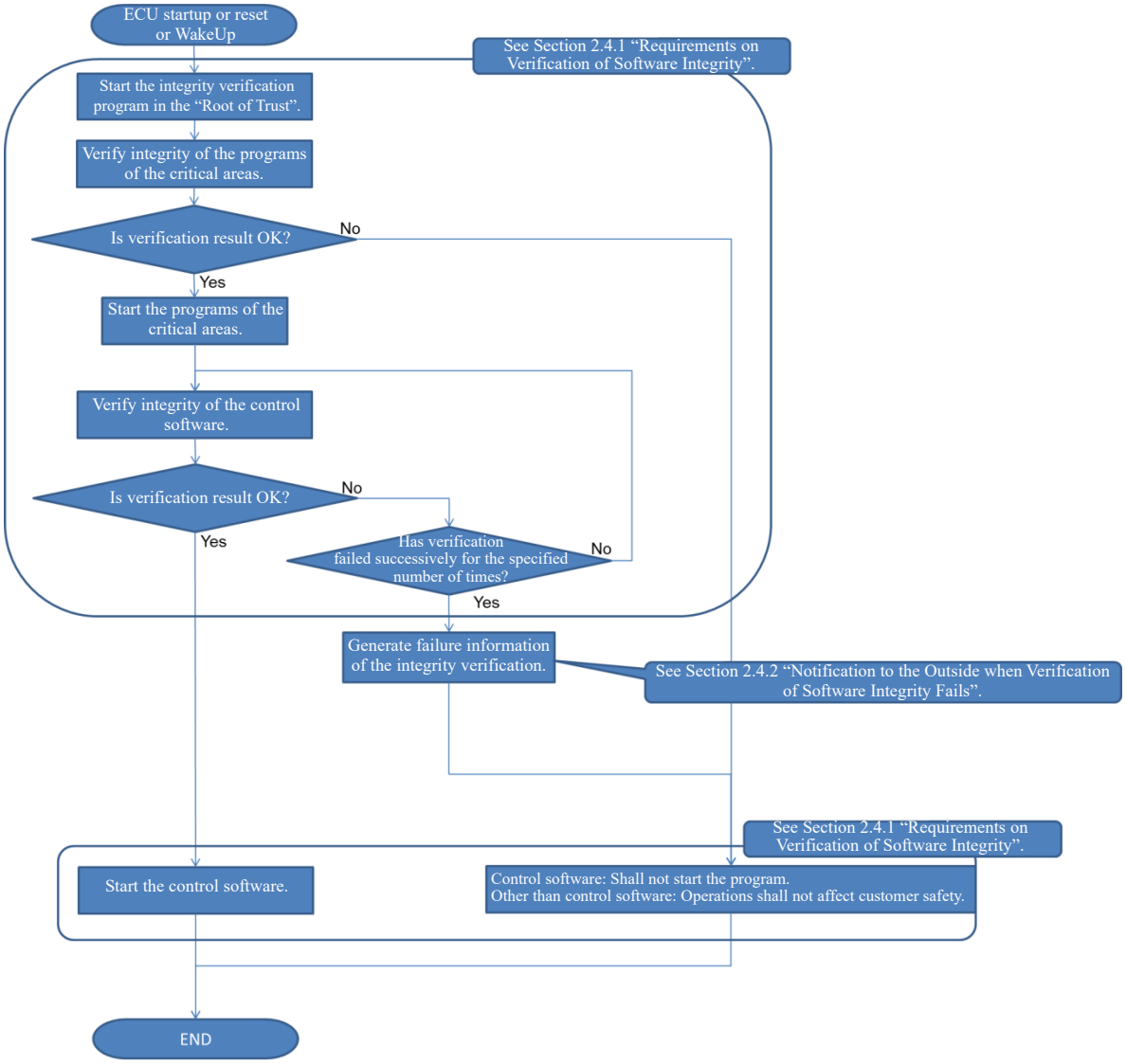

Figure 2-2: Overview of operation during ECU Startup, Reset and WakeUp

2.3.2. Overview of operation in the background and during Sleep and IG-OFF

We show overview of operation in the background and during Sleep and IG-OFF. Note that the operation in this section is not necessary if integrity of all software is verified during ECU StartUp, Reset or WakeUp.

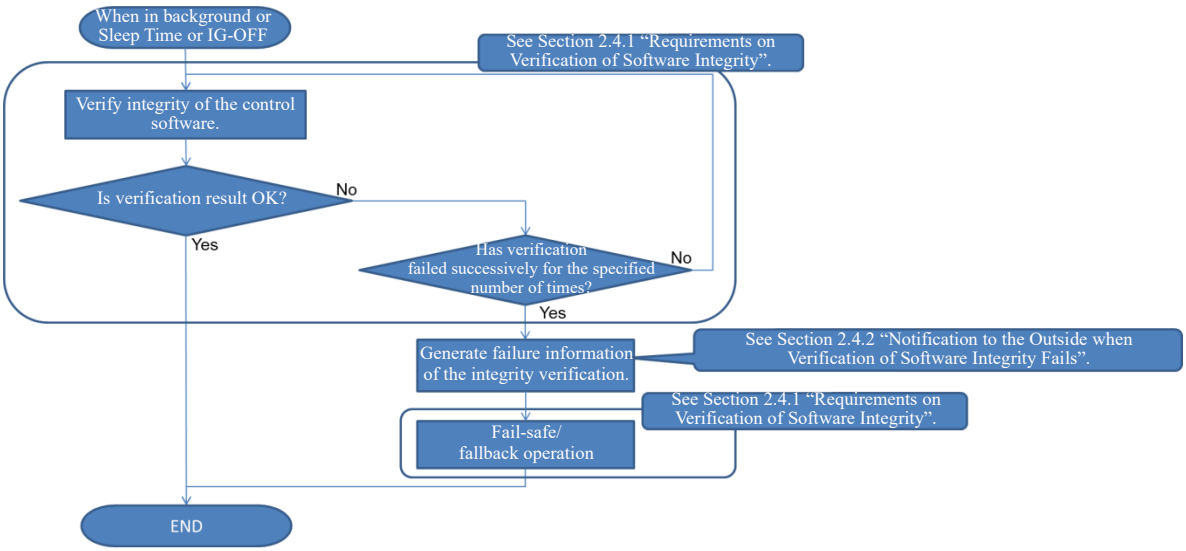

Figure 2-3 Overview of operation in the background and during Sleep and IG-OFF

2.3.3. How to implementate verification of software integrity

See 3.1 for cryptographic algorithm to realize verification of software integrity.

Keys shall be unique as shown in Table 2-2 to minimize impact of leakage of keys.

Table 2-2 Encryption keys for secure boot

	Key uniqueness
Symmetric key cryptography	Keys shall be unique for each ECU or software part number. However, keys shall not be changed by reprogramming.
Asymmetric key cryptography	Keys shall be unique for each ECU node name, supplier and microcontroller (*1).

(*1) Example:
If there are two suppliers of ECU_A and their microcontroller models are different, keys are unique to ECU_A and ECU_B.

2.3.3.1. Implementation exmaples of secure boot

We show two implementation examples of secure boot in this section.

<Implementation example 1>

We show below the implementation example 1 to verify integrity of all software with information stored in RoT.

<a> During ECU StartUp, Reset, and WakeUp, integrity of critical software is verified in order from the first software to invoke.

[Symmetric key cryptography]

(1) Generate a MAC for critical software

(2) Compare the MAC generated at (1) with the corresponding MAC stored in flash memory.

(3) Invoke the software if the MACs are the same.

[Asymmetric key cryptography]

(1) Calculate hash of critical software.

(2) Decrypt the corresponding signature stored in flash memory to get the corresponding hash and compare it with the hash obtained at (1).

(3) Invoke the software if the hashes are the same.

<b> If verification of integrity of all critical software has been succeeded at <a>, integrity of software other than critical some is verified in order.

[Symmetric key cryptography]

(1) Generate a MAC for non-critical software

(2) Compare the MAC generated at (1) with the corresponding MAC stored in flash memory.

(3) Invoke the software if the MACs are the same.

[Asymmetric key cryptography]

(1) Calculate hash of non-critical software.

(2) Decrypt the corresponding signature stored in flash memory to get the corresponding hash and compare it with the hash obtained at (1).

(3) Invoke the software if the hashes are the same.

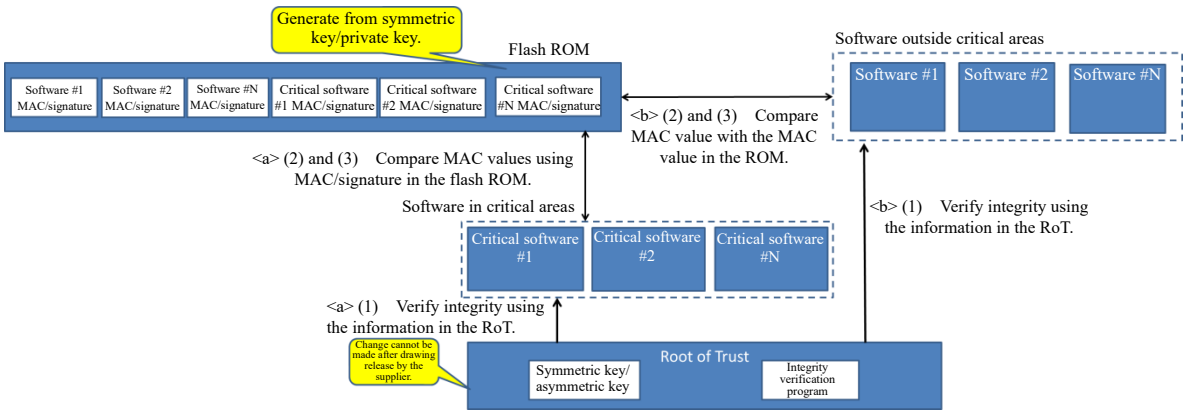

Figure 2-4 Secure Boot Realization Method Example 1

<Implementation example 2>

We show below the implementation example 2. Integrity of the first software to invoke is verified with information stored in RoT. Then integrity of the rest of software is verified in order with information in software verified successfully and key in RoT.

<a> During ECU StartUp, Reset, and WakeUp, integrity of the first critical software to invoked is verified.

[Symmetric key cryptography]

(1) Generate a MAC for critical software

(2) Compare the MAC generated at (1) with the corresponding MAC stored in flash memory.

(3) Invoke the software if the MACs are the same.

[Asymmetric key cryptography]

(1) Calculate hash of critical software.

(2) Decrypt the corresponding signature stored in flash memory to get the corresponding hash and compare it with the hash obtained at (1).

(3) Invoke the software if the hashes are the same.

<b> If verification of integrity of the first software to invoke has been succeeded at <a>, integrity of the rest of critical software is verified in order.

[Symmetric key cryptography]

(1) Generate a MAC for critical software with integrity verification program in the software verified at <a>.

(2) Compare the MAC generated at (1) with the MAC in the software verified at <a>.

(3) Invoke the software if the MACs are the same.

[Asymmetric key cryptography]

(1) Calculate hash of critical software with integrity verification program in the software verified at <a>.

(2) Decrypt the corresponding signature in the software verified at <a> to get the corresponding hash and compare it with the hash obtained at (1).

(3) Invoke the software if the hashes are the same.

<c> If verification of integrity has been succeeded at <b>, integrity of non-critical software is verified in order with information stored in software verified at the last of <b>

[Symmetric key cryptography]

(1) Generate a MAC for non-critical software with the key stored in RoT and integrity verification program in the software verified at the last of <b>.

(2) Compare the MAC generated at (1) with the MAC in the software verified at the last of <b>.

(3) Invoke the software if the hashes are the same.

[Asymmetric key cryptography]

(1) Calculate hash of critical software with integrity verification program in the software verified at the last of <b>

(2) Decrypt the corresponding signature stored in ROM with the key stored in RoT to get the corresponding hash and compare it with the hash obtained at (1).

(3) Invoke the software if the hashes are the same.

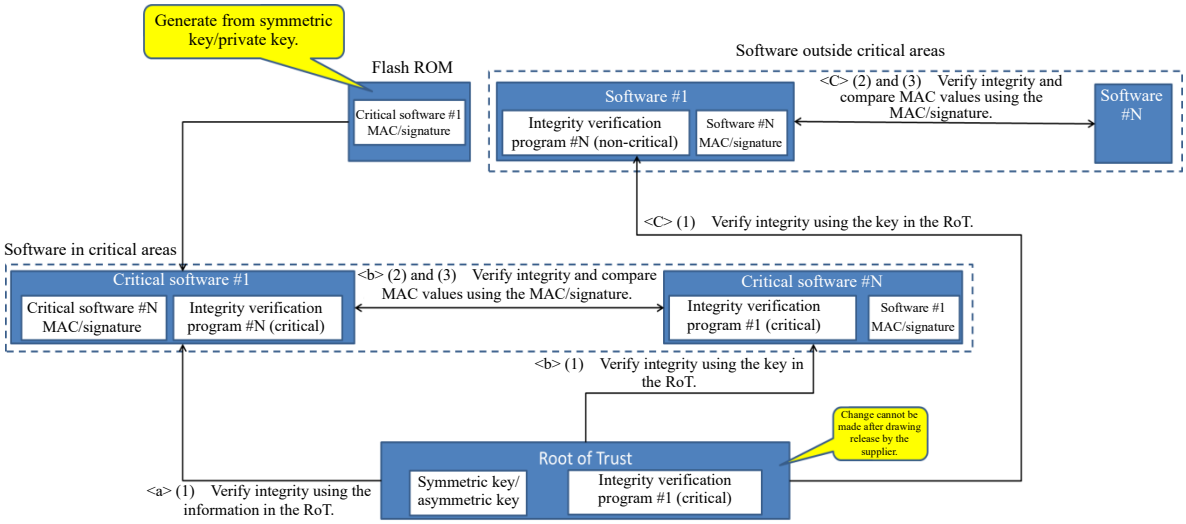

Figure 2-5 Example 2 for implementing secure boot

2.3.4. Overview of MAC/signature update process

When software is updated due to reprogramming, the corresponding MAC/signature shall be updated since the content of the software changes. We describe the overview of the update process in 2.3.4.1.

2.3.4.1. Overview of MAC/signature update process

We describe below overview of MAC/signature update process.

- Generate digital signature of software having been newly generated
- Update software and the digital signature together

In this section, we show an example of signature update process only since MAC update process is the same.

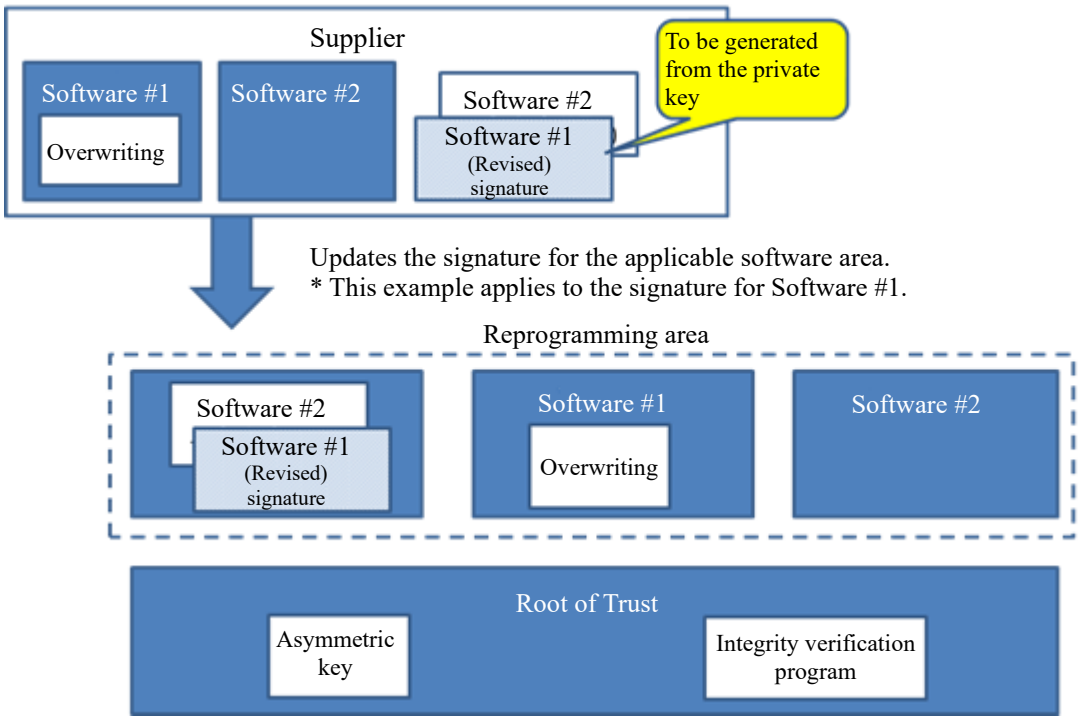

Figure 2-6 Outline of Update Processing of Digital Signature

2.4. Overview of key operation

Each ECU has one key for secure boot with other ECUs. Therefore, if a pre-shared or private key has been leaked, the different key shall be used in ECU produced after the leakage turned out since the leakage can breach secure boot in other ECUs.

2.5. List of requirements

We show the list of all requirements that to be allocated to each ECU in Table 2-3.

Table 2-3: List of requirements

Requirement ID	Hardware-Related Requirement
SBTREQ_00001	○
SBTREQ_00002	-
SBTREQ_00003	-
SBTREQ_00004	-
SBTREQ_00005	-
SBTREQ_00006	-
SBTREQ_00007	-
SBTREQ_00008	-
SBTREQ_00009	-
SBTREQ_00010	-
SBTREQ_00011	-
SBTREQ_00012	-
SBTREQ_00013	-
SBTREQ_00014	-
SBTREQ_00101	-
SBTREQ_01002	-
SBTREQ_01003	○
SBTREQ_01101	-
SBTREQ_01102	-
SBTREQ_01103	-
SBTREQ_01104	-

3. Functional requirement

3.1. Software integrity verification requirements

[Requirement: SBTREQ_00001]

At least the following information shall be stored in RoT.

<Stored information>

・Integrity verification program to verify the first software to invoke

・Pre-shared key (if symmetric key cryptography is adopted)

・Public key (if asymmetric key cryptography is adopted) or hash

<Storage Requirements>

・Integrity verification program shall be stored in HSM or non-rewritable memory (e.g. OTP memory) not to be tampered.

・Pre-shared key shall be stored in HSM not to be tampered nor leaked.

・Public key and hash shall be stored in HSM or non-rewritable memory (e.g. OTP memory) not to be tampered.

[Requirement: SBTREQ_00002]

Area to store MAC/signature shall be chosen for each ECU.

[Requirement: SBTREQ_00003]

Integrity verification program shall be invoked after ECU starts up or resets and before software to be verified starts up.

[Requirement: SBTREQ_00004]

Integrity verification program shall verify at least integrity of the first software to invoke.

[Requirement: SBTREQ_00005]

Chain of Trust shall be established anchored by RoT (see Figure 2-1).

<Note>

If a public key or integrity verification program not stored in RoT is used, the public key or integrity verification program shall have been verified before it is used.

[Requirement: SBTREQ_00006]
Integrity of non-critical software shall be verified after integrity of all critical software has been verified.

[Requirement: SBTREQ_00007]
Integrity of all software to be used shall be verified. However, you may verify only software other than some non-rewritable (e.g. software in mask ROM) or unused. Unused software means software that is not used.

If it is difficult to verify integrity of all software by secure boot during ECU WakeUp or Reset due to the time limit, integrity of bootloader, reprogramming software, and functions system/ECU designer consider as critical shall be verified during ECU WakeUp or Reset.

If integrity of some software has not been verified at the end of ECU StartUp or Reset, integrity of the software shall be verified after the software is invoked. However, if it is difficult for ECU to perform the operation due to constraints on microcontroller, discuss the operation with security department.
<Note>
Integrity verification in the background shall be completed as soon as possible.

[Requirement: SBTREQ_00008]
<Integrity verification during ECU StartUp or Reset>
Verification of software integrity shall be completed before execution of the software.

[Requirement: SBTREQ_00009]
If verification of software integrity fails, the verification shall be retried or the ECU shall be reset with possible temporary breakdown considered.
The number of retrying or resetting ECU is one or more.

Exception 1:
If temporal breakdown can be detected somehow before verification of software integrity, it is not mandatory to retry integrity verification.
Exception 2:
If verification of integrity of BootLoader or/and reprogramming software fails, it is not mandatory to retry verification nor reset ECU.

[Requirement: SBTREQ_00010]
If verification of software integrity including the case of the retry or resetting of the ECU does not succeed, refer to behavior shown in Table 2-1 (*1).

(*1) If a failure to start up the software would affect the safety, the measures shall be discussed with the department responsible for security.

<Integrity verification in the background>
If verification of software integrity including the case of the retry or resetting of the ECU does not succeed, the functions shall fall back or fail safe so as not to affect the safety of passengers.

[Requirement: SBTREQ_00011]
MAC/digital update by reprogramming shall be performed as below.

<Symmetric key cryptography adopted>
MAC of software to update shall be updated with the software to update (*1).
Integrity of the software shall be verified with the MAC updated after reprogramming.

<Asymmetric key cryptography adopted>
Digital signature of software to update shall be updated with the software to update (*1).
Integrity of the software shall be verified with the digital signature updated after reprogramming.

*1: Secure boot function does not have to generate MAC or digital signature since it is written with the software to update by reprogramming function.

[Requirement: SBTREQ_00012]
Encryption algorithm shall be in accordance with the countermeasure requirements for cryptographic algorithms in [1].
We show examples of algorithms.
<Example of symmetric key cryptography>
・Encryption algorithm: AES
・Key length: 128 bits
・Block length: 128 bits
・MAC generation algorithm: CMAC

<Example of asymmetric key cryptography (RSA)>
・Encryption algorithm: RSASSA-PKCS1_v1_5 or RSASSA-PSS
・Key length: 3072 bits or more
・Hashing function: SHA-256 or more
・Encryption key generation: e=65537

<Example of asymmetric cryptography (ECDSA)>

・Encryption algorithms: ECDSA

・Key length: 256 bits or more

・Hashing function: SHA-256 or more

・Elliptic Curve: P-256

[Requirement: SBTREQ_00013]

+ B ECU shall verify integrity of software during ECU WakeUp as well as during ECU StartUp and Reset. However, if it is difficult to verify integrity of all software during WakeUp due to constraints on ECU StartUp and so on, verification in the background shall be performed, and software, whose integrity has been verified, in volatile memory during Sleep is out of scope of verification target.

<Note>

*1: The purpose of secure boot is to detect tampered software in non-volatile memory.

[Requirement: SBTREQ_00014]

Verification of software integrity during ECU WakeUp shall be performed the same way with during ECU StartUp or Reset.

3.2. Software integrity verification error notification requirements

[Requirement: SBTREQ_00101]

It shall be capable to identify whether verification of software integrity succeeds or not by analyzing ECU.

4. Non-functional requirements

4.1. Quality requirements on verification of software integrity

[Requirement: SBTREQ_01002]

Execution of verification of software integrity (including retry or resetting of the ECU) shall not affect the performance requirements on ECU startup time and so on (*1).

*1: If it is difficult to satisfy this requirement, discuss the way.

4.2. Requirements of key operation

[Requirement: SBTREQ_01101]

This requirement is allocated to the keys generated outside the ECU.

If a pre-shared or private key in ECU is leaked, only ECU which has the updated key and software part number shall be shipped out.

[Requirement: SBTREQ_01102]

This requirement is allocated to the keys generated outside the ECU.

If reprogramming is performed with symmetric cryptography, the key for reprogramming shall be different from one for verification of software integrity.

[Requirement: SBTREQ_01103]

This requirement is allocated to the keys generated outside the ECU.

<Symmetric key cryptography adopted>

・Pre-shared key shall be unique for each software part number. However, symmetric key shall not change by reprogramming.

<Asymmetric key cryptography adopted>

Public and private keys shall be unique for each ECU, supplier and microcontroller model.

[Requirement: SBTREQ_01104]

This requirement is allocated to the keys generated outside the ECU.

The ECU designer shall determine who generates the following keys to use in the Secure Boot.

<Symmetric key cryptography adopted>

Pre-shared key

<Asymmetric key cryptography adopted>

Public key, private key

[Requirement: SBTREQ_01003]

This requirement is allocated to the keys generated outside the ECU.

The key required by [Requirement: SBTREQ_01104] shall be generated using random number.

The random number used for the key generation shall be generated by the true random number generator or pseudo random number generator that the criteria shall be in accordance with [1].

5. Design value

There are no design values to define in this document.