2．Uptane基礎

経済産業省
Ministry of Economy, Trade and Industry

国土交通省

本章の目的

- Uptaneで想定されている脅威（攻撃）を理解する
- Updateで想定されていない脅威を理解する
- Uptaneが行う防御の基本的な仕組みを理解する
- Uptaneが行うソフトウェア更新の手順について理解する

アジェンダ

1. Uptaneとは
    概要，ゴール，想定システムの前提条件
    The Update Framework(TUF)
2. 脅威モデル
    攻撃者のゴール，能力（機能），想定される攻撃
3. Uptaneの設計概要
    3.1 TUFの設計原理，Uptaneの構成，TUFとの違い
    3.2 リポジトリの役割
    3.3 メタデータ構造
    3.4 サーバー/リポジトリの実装条件
    3.5 車載ECUの実装要件

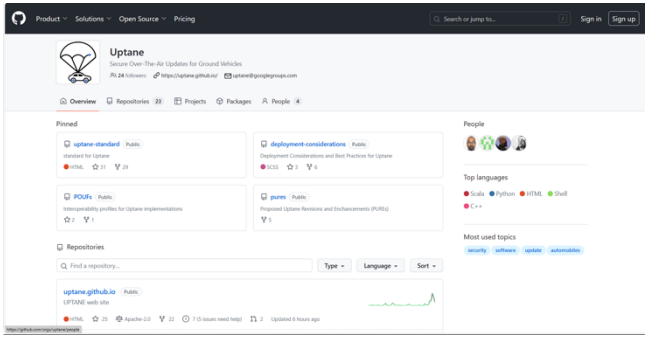
アジェンダ

1. Uptaneとは
   概要，ゴール，想定システムの前提条件
   The Update Framework(TUF)
2. 脅威モデル
   攻撃者のゴール，能力（機能），想定される攻撃
3. Uptaneの設計概要
   3.1 TUFの設計原理，Uptaneの構成，TUFとの違い
   3.2 リポジトリの役割
   3.3 メタデータ構造
   3.4 サーバー/リポジトリの実装条件
   3.5 車載ECUの実装要件

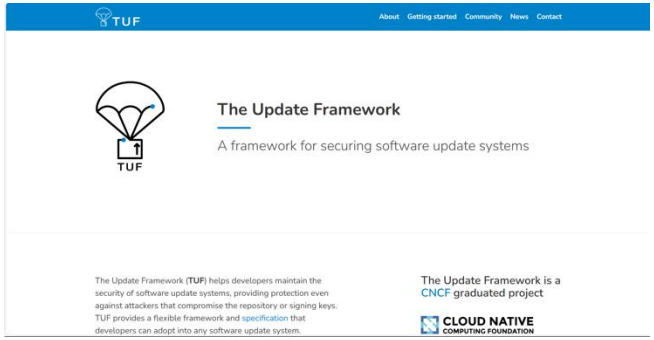
Uptaneの概要

- ECUに対してOTAを実現するためのオープンなフレームワーク
- 米国の公的機関(NHTSA：米国運輸省道路交通安全局)が助成
  ミシガン大学，ニューヨーク大学などが参加
- 自動車のOTAに対するセキュリティ上の脅威を指摘
- 一部のソフトウェア実装がGitHub上に公開
- TUFの基本設計をベースに開発をスタート(2016-)

https://uptane.github.io/



https://github.com/uptane



https://theupdateframework.io/

The update framework(TUF)とは

マルウェアを拡散したり，リポジトリを侵害したりする可能性のある攻撃に対して，システムの回復力を構築するフレームワークとして2010年より開始.

仕様，標準化プロセス，参照実装の３つから構成.

TUFの主な目標

- 新規及び既存のソフトウェア更新システムをセキュリティで保護するために使用するフレームワーク(ライブラリ，ファイル形式，ユーティリティ)を提供
- 主要な侵害の影響を最小限に抑える手段を提供
- さまざまさソフトウェア更新システムのニーズを満たすための十分な柔軟性を備えていること
- 既存のソフトウェア更新システムとの統合を容易にすること

なぜUptaneが必要なのか

- TUFは車載向けではない（PCやスマートデバイスと比べて）
- 単体で動作するわけではない（車内には複数のECUが存在）
- 特に賢いわけではない（組込み向けマイコンを使用）
- 個々の構成要素はお互いを信頼していない

Uptane Design goals

- 可能な限り高いレベルのセキュリティを保持する自動車用のソフトウェア更新メカニズムを構築する
- ダウンロード元に関係無く，ソフトウェア更新の真正性と完全性を可能な限り確保する
- OEMが実際に導入できるソフトウェア更新メカニズムを提供する

Uptane Non goals(範ちゅう外)

- **物理的な攻撃**
  (自動車外のECUを手動で改ざんするなど)

- **パッケージソフトの侵害**
  (信頼できるパッケージにマルウェアが埋め込まれる等)

- **サプライチェーンの侵害**（ビルドシステム、バージョン管理システム、パッケージングプロセスなど)
  (in-totoとUptaneを組み合わせたScudoを将来的にUptaneの拡張機能として採用する可能性を示唆)

- **ソフトウェア自体の脆弱性**
  (ECU間通信の認証など，OBDやUDSに起因する問題)

- **ランダム故障**
  (劣化による部品故障，単純作業におけるヒューマンエラーなど)

想定システムの前提条件

- 自動車には必要なバックエンドサービスへの接続を確立する機能があること
- ECUは通信チャネルに直接接続しているか，何らかのネットワークゲートウェイを介して間接的に接続していること
- ECUはプログラム可能で，更新するための十分な性能を持っていること
- ECUは公開鍵暗号処理を実行し，更新ファイルやメタデータファイルのハッシュ計算を行えること
- Director, Imageリポジトリなどのセキュアサーバが設置されていること

アジェンダ

1. Uptaneとは
   - 概要，ゴール，想定システムの前提条件
   - The Update Framework(TUF)
2. 脅威モデル
   - 攻撃者のゴール，能力（機能），想定される攻撃
3. Uptaneの設計概要
   - 3.1 TUFの設計原理，Uptaneの構成，TUFとの違い
   - 3.2 リポジトリの役割
   - 3.3 メタデータ構造
   - 3.4 サーバー/リポジトリの実装条件
   - 3.5 車載ECUの実装要件

脅威モデル

攻撃者のゴール

攻撃者は重大度の高い順に次のゴールを達成したいと考えている可能性がある.

- **Read Updates**：
  - ソフトウェア更新の内容を読んで機密情報を発見
  - ファームウェアのリバースエンジニアリング
  - イメージファイルを比較してセキュリティ修正箇所の特定
- **Deny Updates**：
  ソフトウェア更新を拒否して，ソフトウェアの問題を解決できないようにする.
- **Deny functionality**：
  ECUの正常機能を停止するように試みる.
- **Control**：
  自動車内の制御を乗っ取る.

脅威モデル

攻撃者の能力

攻撃者は次の１つ以上の機能を実行する能力を持つと想定.

- **Intercept**
  ネットワークを傍受して改ざんする(中間者攻撃).
  ・車外：自動車とリポジトリ間の伝送を傍受および変更
  ・車内：バス上の伝送を傍受および変更

- **Compromise ECUs**
  ECUの侵害(プライマリまたはセカンダリECUのどちらか一方).

- **Compromise keys**
  サーバーに格納されている鍵を侵害(入手).

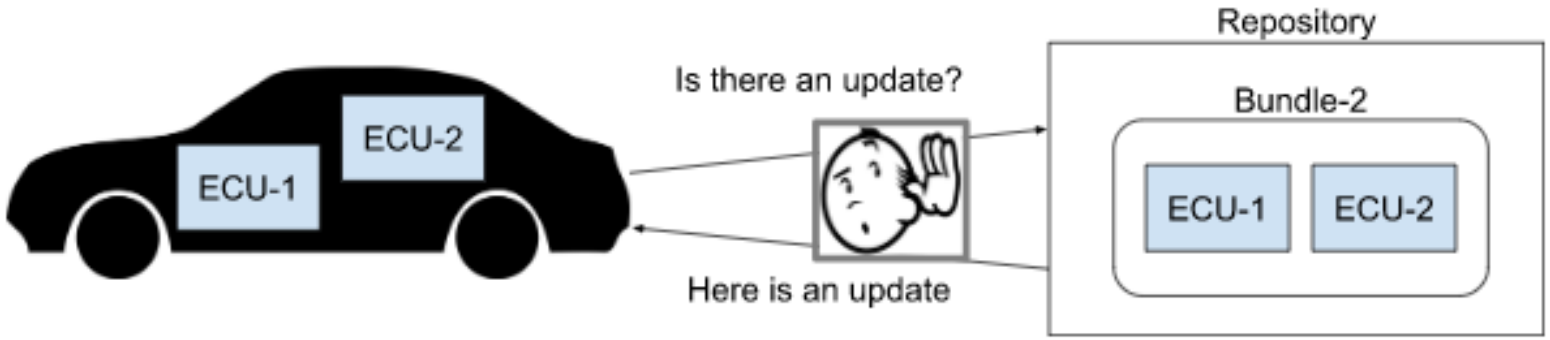
想定される攻撃

攻撃名	概要
Eavesdrop attack （更新内容の盗聴）	更新データの内容を読み取る.
Drop-request attack （ドロップリクエスト攻撃）	車内，車外のネットワーク通信をブロックしてECUが更新出来ないようにする.
Slow retrieval attack （遅延攻撃）	ECUからの要求に対してゆっくり応答などしてタイムアウトを誘発して更新を遅らせる攻撃.
Freeze attack （フリーズ攻撃）	古いソフトウェア更新情報を送り付けることにより，新しい情報を固定化してしまう.
Partial bundle installation attack (部分的なバンドルインストール攻撃)	幾つかのECUが最新のソフトウェア更新をインストールさせないようにする攻撃.
Rollback attack （ロールバック攻撃）	現在インストールされているバージョンよりも古いバージョンの更新データをインストールさせる.
Endless data attack （エンドレスデータ攻撃）	大量のデータを送り付けてECUをクラッシュさせる攻撃.
Mix-and-match attack （ミックスアンドマッチ攻撃）	あり得ないソフトの組み合わせ(バンドル)を作成して送り付ける攻撃.
Arbitrary software attack （任意のソフトウェアへの攻撃）	ECUのソフトウェアを改ざんされたソフトウェアで上書きさせる攻撃.

Read updates

Eavesdrop attack

特定のECUに対して暗号化を目的としたソフトウェア更新から機密情報を読み取る.



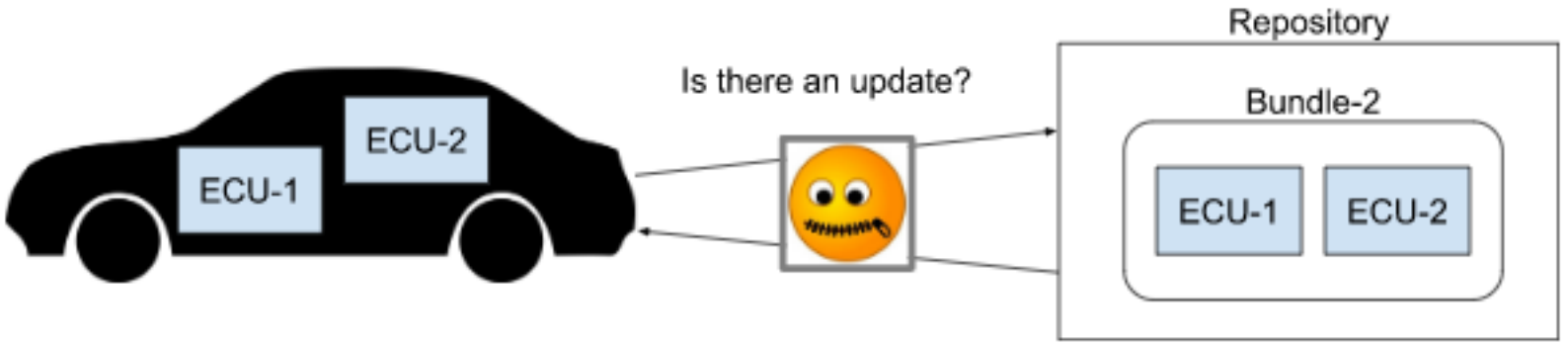
Uptane Design Overview (v2019.02.21)より図を引用

https://docs.google.com/document/d/1pBK--40BCg_ofww4GES0weYFB6tZRedAjUy6PJ4Rgzk/

Deny updates

Drop-request attack

自動車内外のネットワークトラフィックを遮断して，ECUが任意のソフトウェア更新を受信しないようにする攻撃.



Uptane Design Overview (v2019.02.21)より図を引用

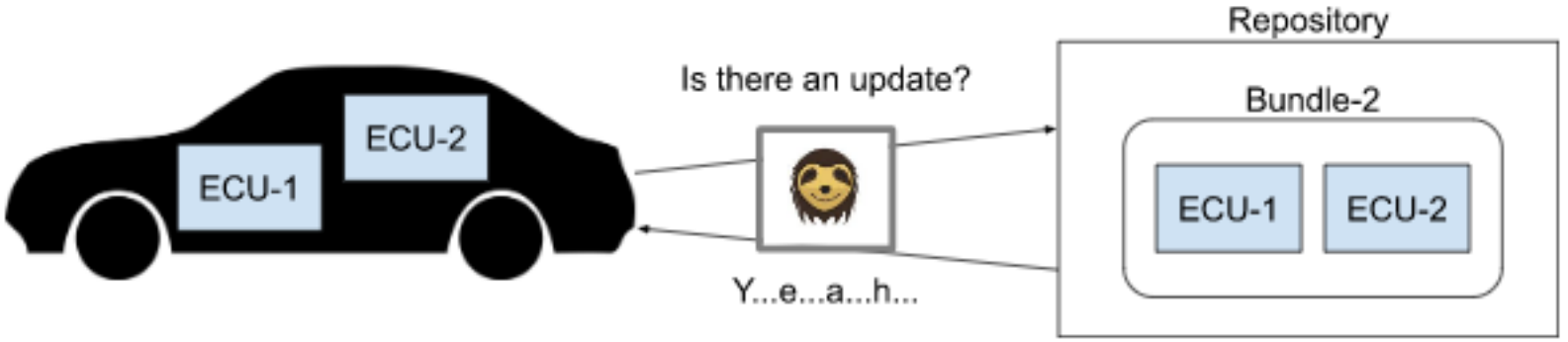
https://docs.google.com/document/d/1pBK--40BCg_ofww4GES0weYFB6tZRedAjUy6PJ4Rgzk/

Deny updates

Slow retrieval attack

ネットワークトラフィックを遅くして，ECUがソフトウェア更新を受け取るのを遅くする攻撃.

Drop-request attackと似ているが，トラフィックが妨げられていないと考えている点が異なる．（タイムアウトを回避するために程度のトラフィック速度がある）



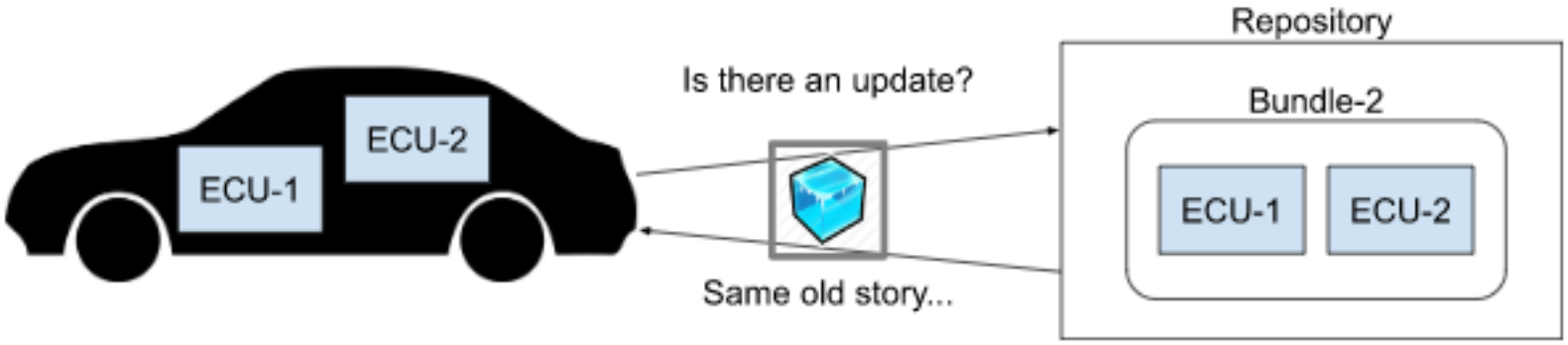
Uptane Design Overview (v2019.02.21)より図を引用

https://docs.google.com/document/d/1pBK--40BCg_ofww4GES0weYFB6tZRedAjUy6PJ4Rgzk/

Deny updates

Freeze attack

新しいソフトウェア更新が存在する場合でも，正しく署名された古いソフトウェア更新をECUに送信し続ける攻撃.



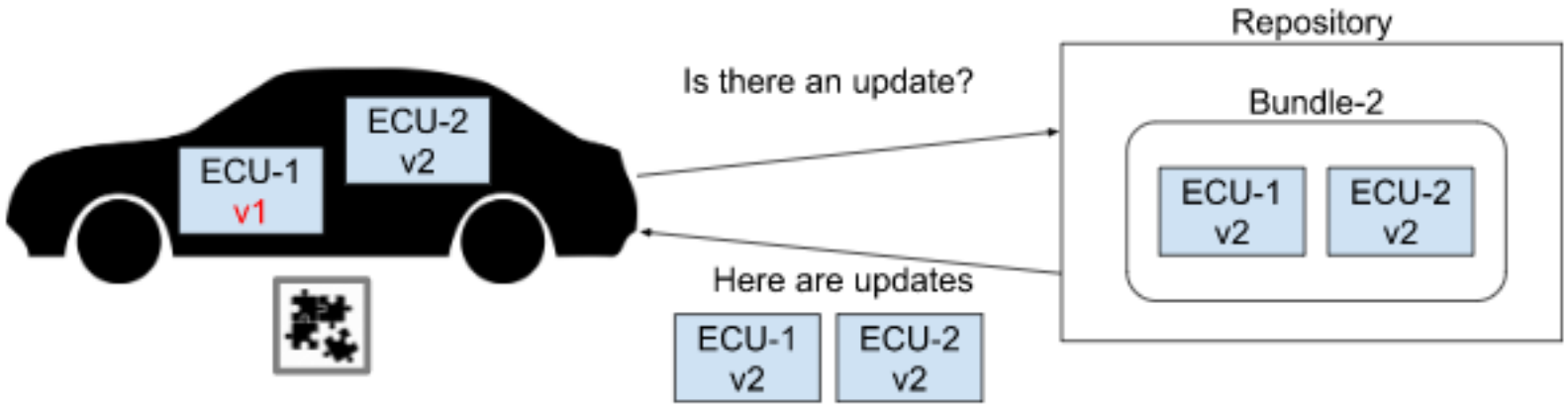
Uptane Design Overview (v2019.02.21)より図を引用

https://docs.google.com/document/d/1pBK--40BCg_ofww4GES0weYFB6tZRedAjUy6PJ4Rgzk/

Deny updates

Partial bundle installation attack

ECUにソフトウェア更新の一部をインストールさせない攻撃.



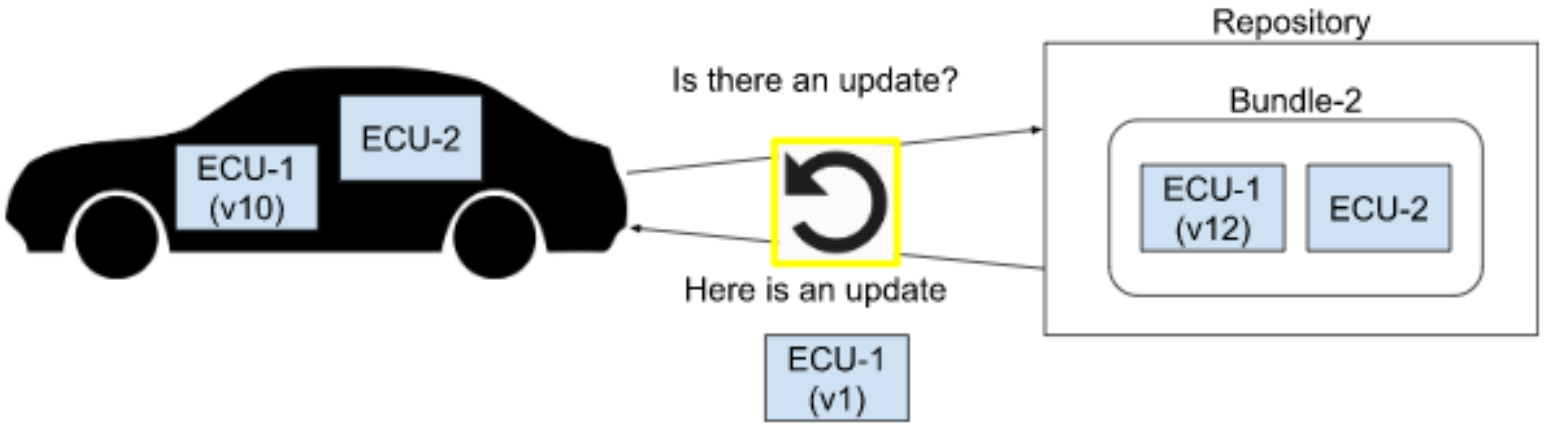
Uptane Design Overview (v2019.02.21)より図を引用

https://docs.google.com/document/d/1pBK--40BCg_ofww4GES0weYFB6tZRedAjUy6PJ4Rgzk/

Deny functionality

Rollback attack

現在インストールされているバージョンよりも古い，以前に有効だったバージョンのソフトウェア(脆弱性のあるソフトウェア)をインストールさせる攻撃.



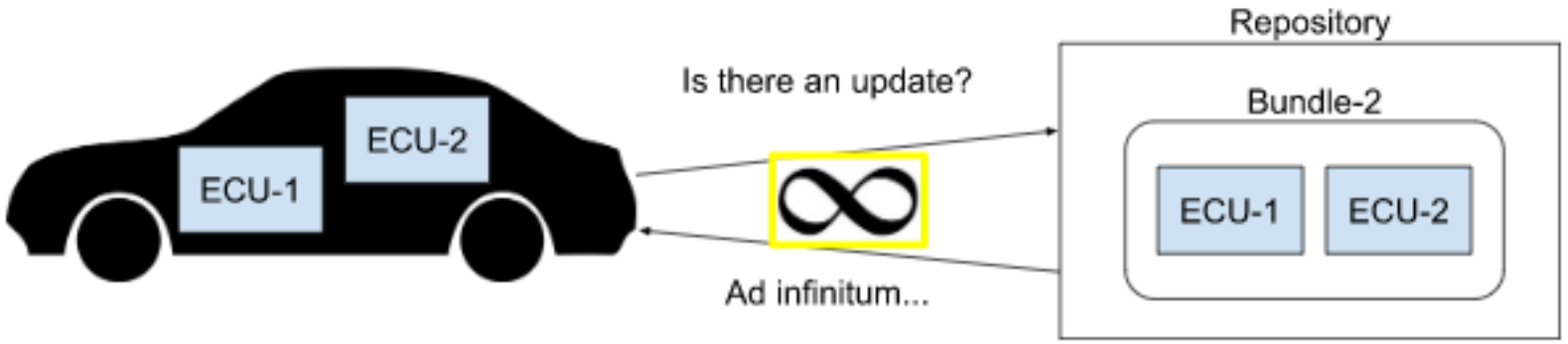
Uptane Design Overview (v2019.02.21)より図を引用

https://docs.google.com/document/d/1pBK--40BCg_ofww4GES0weYFB6tZRedAjUy6PJ4Rgzk/

Deny functionality

Endless data attack

ストレージが不足するまで大量のデータを送信し，ECUが動作しなくなる攻撃.



Uptane Design Overview (v2019.02.21)より図を引用

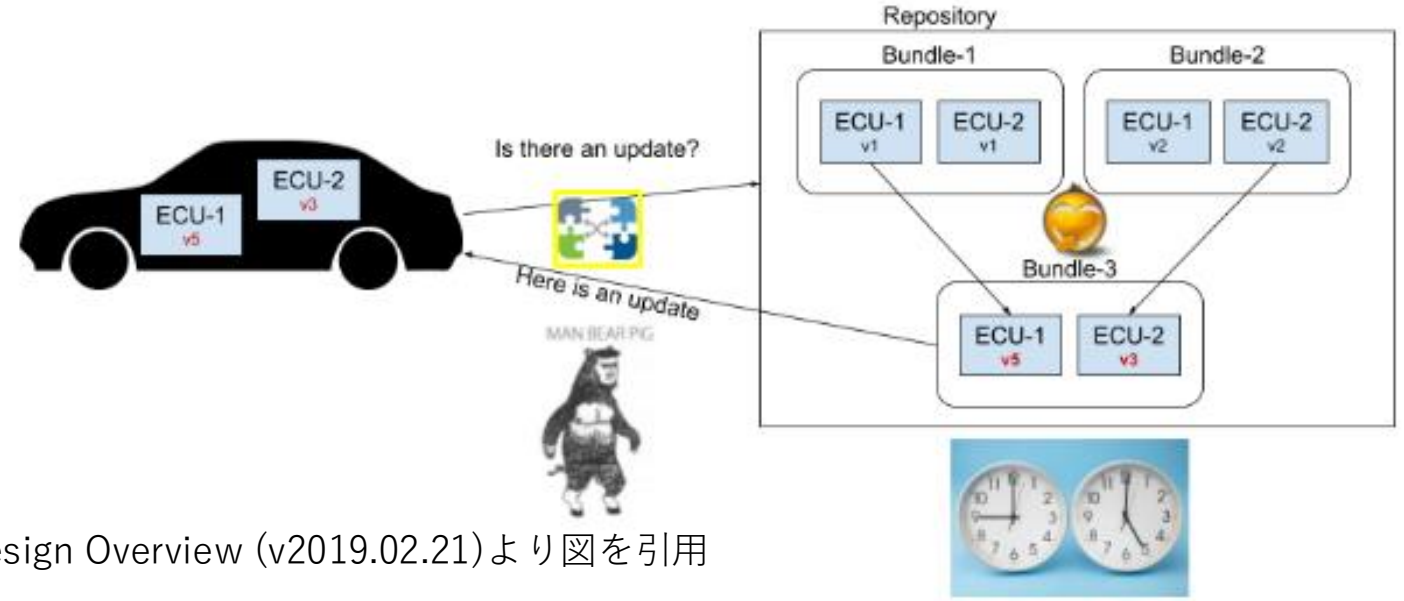
https://docs.google.com/document/d/1pBK--40BCg_ofww4GES0weYFB6tZRedAjUy6PJ4Rgzk/

Deny functionality

Mix-and-match attack

一部のソフトウェア更新が適切に相互運用されない悪意のあるソフトウェアバンドルをインストールさせる攻撃.

個々のソフトウェア更新が全て有効である場合でも，相互の整合性が取れないバンドルの組み合わせが存在する限り実現が可能.



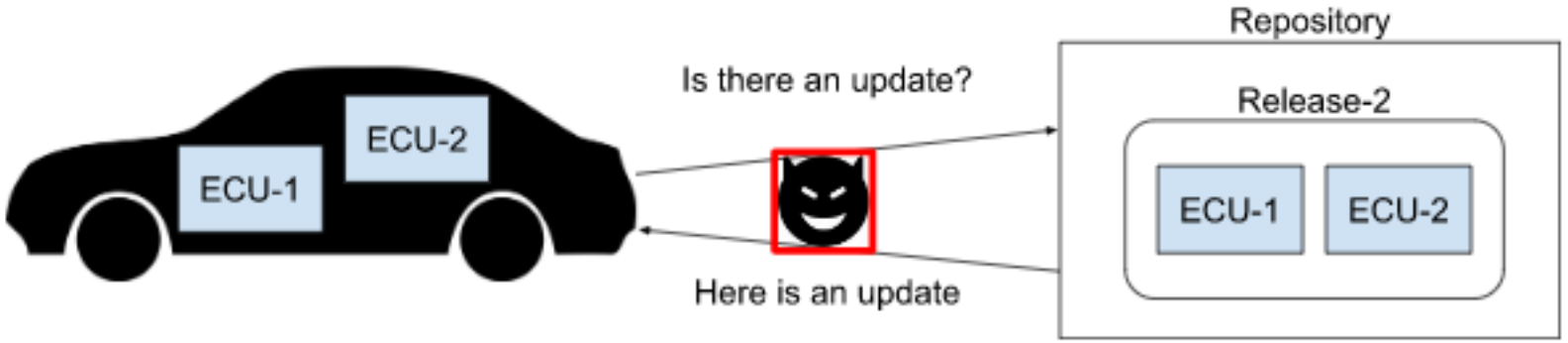
Uptane Design Overview (v2019.02.21)より図を引用

https://docs.google.com/document/d/1pBK--40BCg_ofww4GES0weYFB6tZRedAjUy6PJ4Rgzk/

Control

Arbitrary software attack

攻撃者が悪意のあるソフトウェアでECUを上書きする攻撃.



Uptane Design Overview (v2019.02.21)より図を引用

https://docs.google.com/document/d/1pBK--40BCg_ofww4GES0weYFB6tZRedAjUy6PJ4Rgzk/

Uptaneでは防げない脅威とその理由

攻撃名	対抗策	Uptaneの対象範囲	理由
Eavesdrop attack	暗号化	×	ネットワーク上のデータが盗聴されているか監視する事は，Uptane仕様から外れているため，防ぐ事が出来ない.
Drop-request attack	再送	×	ネットワーク上のデータが遮断されているか監視する事は，Uptane仕様から外れているため，防ぐ事が出来ない.
Slow retrieval attack	デッドライン監視	×	ネットワーク上のデータ速度，遅延を監視する事は，Uptane仕様から外れているため，防ぐ事が出来ない.
Freeze attack	バージョン監視	×	ソフトそのものの脆弱性，バージョンをチェックする事は，Uptane仕様から外れているため，防ぐ事が出来ない.
Partial bundle installation attack	レポート監視	○	
Rollback attack	バージョンダウン禁止	○	
Endless data attack	デッドライン監視 ２面持ち	×	大量に送信されるネットワーク上のデータをブロックする事は，Uptane仕様から外れているため，防ぐ事が出来ない.
Mix-and-match attack	デジタル署名	○	
Arbitrary software attack	デジタル署名	○	

アジェンダ

1. Uptaneとは
   - 概要，ゴール，想定システムの前提条件
   - The Update Framework(TUF)
2. 脅威モデル
   - 攻撃者のゴール，能力（機能），想定される攻撃
3. Uptaneの設計概要
   - 3.1 TUFの設計原理，Uptaneの構成，TUFとの違い
   - 3.2 リポジトリの役割
   - 3.3 メタデータ構造
   - 3.4 サーバー/リポジトリの実装要件
   - 3.5 車載ECUの実装要件

Uptaneの設計概要

TUFの４つの設計原理をUptaneでも踏襲

TUFの中心的な考え方は侵害の回復力(compromise-resilience).
特定の攻撃によってもたらされる脅威の範囲を最小限に抑える能力.

1. **Separation of trust**
   メタデータの署名に対する責任を分散する.
2. **Threshold signatures**
   ソフトウェア更新をダウンロードする前に，一定数の署名を収集する.
3. **Explicit and implicit revocation of keys**
   侵害された鍵を置き換えるメカニズムを提供する.
4. **Keeping the most vulnerable keys offline**
   特定の署名鍵をオフラインにしておくことを義務付ける.

Uptaneの設計概要

TUF設計からの主な違い

1. 検証プロセスの**処理**と**責任**を分割する２つ目のリポジトリの追加

   Imageリポジトリ：

   オフライン鍵でメタデータに署名することで更新ファイルの侵害が困難

   Directorリポジトリ：

   オンライン鍵でメタデータに署名することで処理が迅速

2. Uptaneが更新ファイルを検証する方法

   ECUの性能に応じて２つの異なる検証戦略を用意

   （Full verification, Partial verification）.

Uptaneの設計概要

標準仕様と実装，POUF

UptaneやTUFは実装仕様の詳細は指定していない．（参考実装はある）

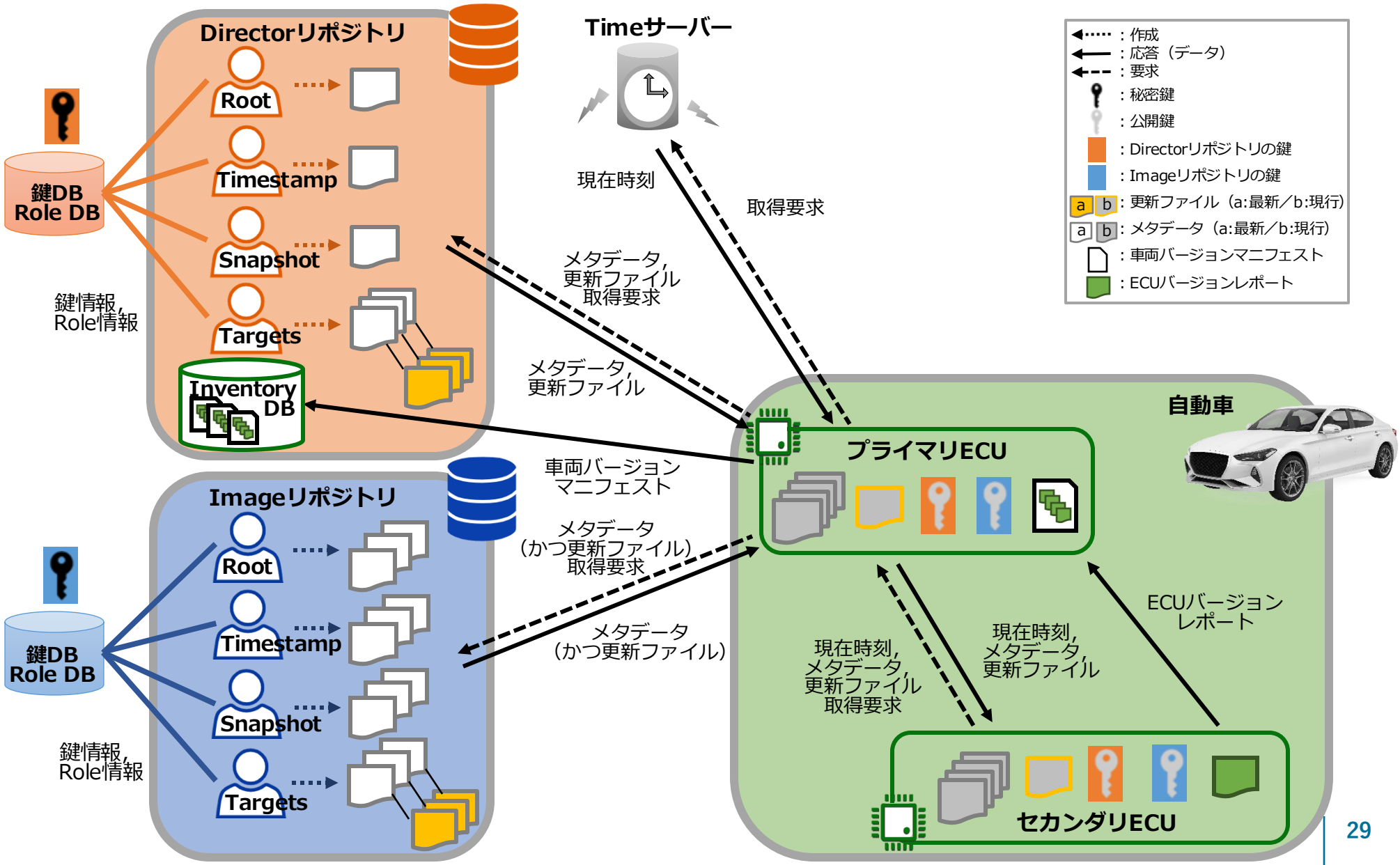
標準仕様とベストプラクティス，POUFガイドラインが規定されている．

POUF：**p**rotocol, **o**perations, **u**sage, and **f**ormats

Uptaneの構成

- 2つのリポジトリ（Image, Director）
- 更新ファイルに関するUptane固有のメタデータを作成するリポジトリツール
- 各リポジトリで必要なメタデータの作成と署名をサポートする公開鍵基盤
- Inventoryデータベース
- ECUが時間を知るための安全な方法
- サーバから更新ファイルやメタデータをダウンロードするECU（プライマリECU，セカンダリECU）

Uptaneの構成図


Directorリポジトリ
Root
Timestamp
Snapshot
Targets
Inventory DB
鍵DB
Role DB
鍵情報,
Role情報
Timeサーバー
現在時刻
取得要求
メタデータ,
更新ファイル
取得要求
メタデータ,
更新ファイル
車両バージョン
マニフェスト
Imageリポジトリ
Root
Timestamp
Snapshot
Targets
鍵DB
Role DB
鍵情報,
Role情報
メタデータ
(かつ更新ファイル)
取得要求
メタデータ
(かつ更新ファイル)
自動車
プライマリECU
セカンダリECU
現在時刻,
メタデータ,
更新ファイル
取得要求
現在時刻,
メタデータ,
更新ファイル
ECUバージョン
レポート
：作成
：応答（データ）
：要求
：秘密鍵
：公開鍵
：Directorリポジトリの鍵
：Imageリポジトリの鍵
a b：更新ファイル（a:最新／b:現行）
a b：メタデータ（a:最新／b:現行）
：車両バージョンマニフェスト
：ECUバージョンレポート

アジェンダ

1. Uptaneとは
   - 概要，ゴール，想定システムの前提条件
   - The Update Framework(TUF)
2. 脅威モデル
   - 攻撃者のゴール，能力（機能），想定される攻撃
3. Uptaneの設計概要
   - 3.1 TUFの設計原理，Uptaneの構成，TUFとの違い
   - 3.2 リポジトリの役割
   - 3.3 メタデータ構造
   - 3.4 サーバー/リポジトリの実装要件
   - 3.5 車載ECUの実装要件

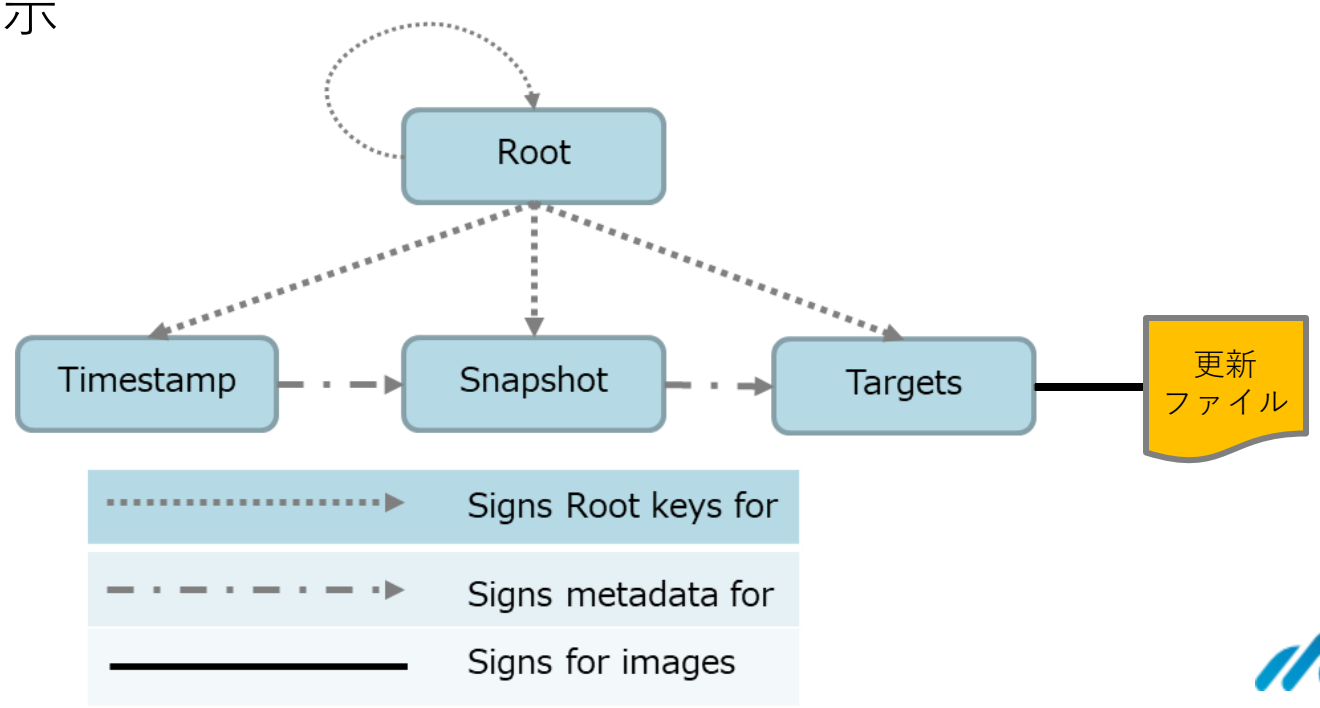
リポジトリの概要

- リポジトリ，またはソフトウェア更新の状態に関する検証可能なレコードを追加することでセキュリティを強化
- ソフトウェア更新の暗号化ハッシュ，ファイルサイズなど，署名されたメタデータをリポジトリに追加
- メタデータを検証することにより，インストール前にセキュリティ攻撃を検出して防止する仕組みを提供
- 4つのRole(Root, Timestamp, Snapshot, Targets)で構成

リポジトリのRole（役割）

Root, Timestamp, Snapshot, Targetsで構成.

- Root：認証局の役割，公開鍵の配布，失効
- Timestamp：新しいメタデータや更新ファイルが存在するかを提示
- Snapshot：リポジトリにリリースされた更新ファイルを提示
- Targets： 更新ファイルのサイズやハッシュ値などのメタデータを提示



4つのRole

1. Root Role

- 「RFC3647」で定義されている認証局に相当
- Rootメタデータを作成して署名
- Timestamp, Snapshot, Targetsによって作成されたメタデータを検証するための公開鍵に署名
- 他のRoleが侵害された場合，その公開鍵を取り消す必要がある

4つのRole

2. Timestamp Role

- リポジトリに新しいメタデータ，または更新ファイルがあるかどうかを示すメタデータを作成して署名

3. Snapshot Role

- リポジトリがリリースするすべてのTargetsメタデータに関するメタデータを作成して署名

4. Targets Role

- 更新ファイルと委任(Delegation)に関するメタデータを作成して署名

アジェンダ

1. Uptaneとは
   - 概要，ゴール，想定システムの前提条件
   - The Update Framework(TUF)
2. 脅威モデル
   - 攻撃者のゴール，能力（機能），想定される攻撃
3. Uptaneの設計概要
   - 3.1 TUFの設計原理，Uptaneの構成，TUFとの違い
   - 3.2 リポジトリの役割
   - 3.3 メタデータ構造
   - 3.4 サーバ/リポジトリの実装要件
   - 3.5 車載ECUの実装要件

メタデータ構造

概要

- Uptaneのセキュリティ保証は，すべて指定された構造に従って適切に作成されたメタデータに依存

- Uptane Standardでは，特定のフォーマットやエンコーディングの指定無し(ASN.1, JSON, XMLなどの使用が可能)
  ※メタデータの検証の一部として文字列比較は必要

全ての公開鍵は公開鍵識別子を使用して表す.

- 公開鍵自体の値（例えばPEM文字列）
- 鍵によって使用される公開鍵アルゴリズム（RSA，ECDSAなど）
- 署名の検証に使用される特定のスキーム

メタデータ構造

共通するメタデータのペイロード

- Roleタイプのインジケーター
- 有効期限
- 整数のバージョン番号（更新される毎にインクリメントされる）
- Role固有(Root, Timestamp, Snapshot, Targets) のメタデータ

メタデータ構造

Rootメタデータ

- 4つのRoleすべての公開鍵
- 4つのRoleに必要な署名のthreshold

```
{'_type': 'Root',          インジケーター
 'version': 0,             バージョン
 'expires': '2023-01-06T15:34:25Z',   有効期限
 'keys': {'94c836f0c45168f0a437eef0e487b910f58db4d462ae457b5730a4487130f290':
            {'keytype': 'ed25519', 'keyid_hash_algorithms': ['sha256', 'sha512'], 'keyval': {'public': 'f4ac8d95cfdf65a4ccaee072ba5a48e8ad6a0c30be6ffd525aec6bc078211033'}},
          'c24b457b2ca4b3c2f415efdbbebb914a0d05c5345b9889bda044362589d6f596':
            {'keytype': 'ed25519', 'keyid_hash_algorithms': ['sha256', 'sha512'], 'keyval': {'public': '729d9cb5f74688ef8e9a22fae1516f33ff98c7910b64bf3b66e6cfc51559840e'}},
          'aaf05f8d054f8068bf6cb46beed7c824e2560802df462fc8681677586582ca99':
            {'keytype': 'ed25519', 'keyid_hash_algorithms': ['sha256', 'sha512'], 'keyval': {'public': '497f62d80e5b892718da8788bb549bcf8369a1460ec23d6d67d0ca099a8e8f83'}},
          '6fcd9a928358ad8ca7e946325f57ec71d50cb5977a8d02c5ab0de6765fef040a':
            {'keytype': 'ed25519', 'keyid_hash_algorithms': ['sha256', 'sha512'], 'keyval': {'public': '97c1112bbd9047b1fdb50dd638bfed6d0639e0dff2c1443f5593fea40e30f654'}}
 },
 'roles': {'root': {'keyids': ['94c836f0c45168f0a437eef0e487b910f58db4d462ae457b5730a4487130f290'], 'threshold': 1},
           'targets': {'keyids': ['c24b457b2ca4b3c2f415efdbbebb914a0d05c5345b9889bda044362589d6f596'], 'threshold': 1},
           'snapshot': {'keyids': ['aaf05f8d054f8068bf6cb46beed7c824e2560802df462fc8681677586582ca99'], 'threshold': 1},
           'timestamp': {'keyids': ['6fcd9a928358ad8ca7e946325f57ec71d50cb5977a8d02c5ab0de6765fef040a'], 'threshold': 1}
 },
 'consistent_snapshot': False,
 'compression_algorithms': ['gz']
}
```

4つのRoleの公開鍵

4つのRoleのThreshold

Rootメタデータの例

メタデータ構造

Timestampメタデータ

- リポジトリ上の最新のSnapshotメタデータのファイル名とバージョン番号

- Snapshotメタデータの1つ以上のハッシュ値と使用されるハッシュ関数


```
{"_type": "Timestamp",
  "expires": "2022-04-27T05:13:45Z",
  "version": 1,
  "meta": {"snapshot.der": {"hashes": {"sha256": "1039cff7c021b02a5d4a90596df836b129bc00b20060cfd120aaf18257be9a1b"}, "length": 217, "version": 1}}
}
```


Timestampメタデータの例

Snapshotメタデータの
ファイル名，バージョン番号，
ハッシュ値，ハッシュ関数

メタデータ構造

Snapshotメタデータ

- Targetsメタデータのバージョン番号とファイル名
- Rootメタデータのファイル名とバージョン番号

```
{'_type': 'Snapshot',
  'version': 0,
  'expires': '2022-01-13T09:46:15Z',
 'meta': {'root.der': {'length': 631, 'hashes': {'sha256': '4c5302ee31cba0ca6e536ef1733ba491f42d093bb169f313fb975af022941656'}, 'version': 1},
          'targets.der': {'version': 1}
          }
}
```

Rootメタデータの
ファイル名，バージョン番号

Targetsメタデータの
ファイル名，バージョン番号

Snapshotメタデータの例

メタデータ構造

Targetsメタデータ

- 更新ファイルのファイル名
- 更新ファイルのサイズ(バイト単位)
- 更新ファイルのハッシュ値(使用しているハッシュ関数含む)


```
{'_type': 'Targets',
  'version': 0,
  'expires': '2022-04-07T17:13:25Z',
  'targets': {'/firmware.img': {'length': 20, 'hashes': {'sha256': 'daeec2555599b8e7a82b6f1339d5f419346b57a0eb7a39ee6334b8f205595752',
                'sha512':
'1570937a84e9e74e35f5e56a8f8518c91e18258cffc8ace249d9acf173d1845d82002583b1b53373b79edf52494d524f2619f5f1896f3085038deca92c950486'},
                'custom': {'ecu_serial': 'TCUdemocar'}}},
  'delegations': {'keys': {}, 'roles': []}
}
```


更新ファイルのファイル名，
サイズ，ハッシュ値

Targetsメタデータの例

アジェンダ

1. Uptaneとは
   - 概要，ゴール，想定システムの前提条件
   - The Update Framework(TUF)
2. 脅威モデル
   - 攻撃者のゴール，能力（機能），想定される攻撃
3. Uptaneの設計概要
   - 3.1 TUFの設計原理，Uptaneの構成，TUFとの違い
   - 3.2 リポジトリの役割
   - 3.3 メタデータ構造
   - 3.4 サーバー/リポジトリの実装要件
   - 3.5 車載ECUの実装要件

概要

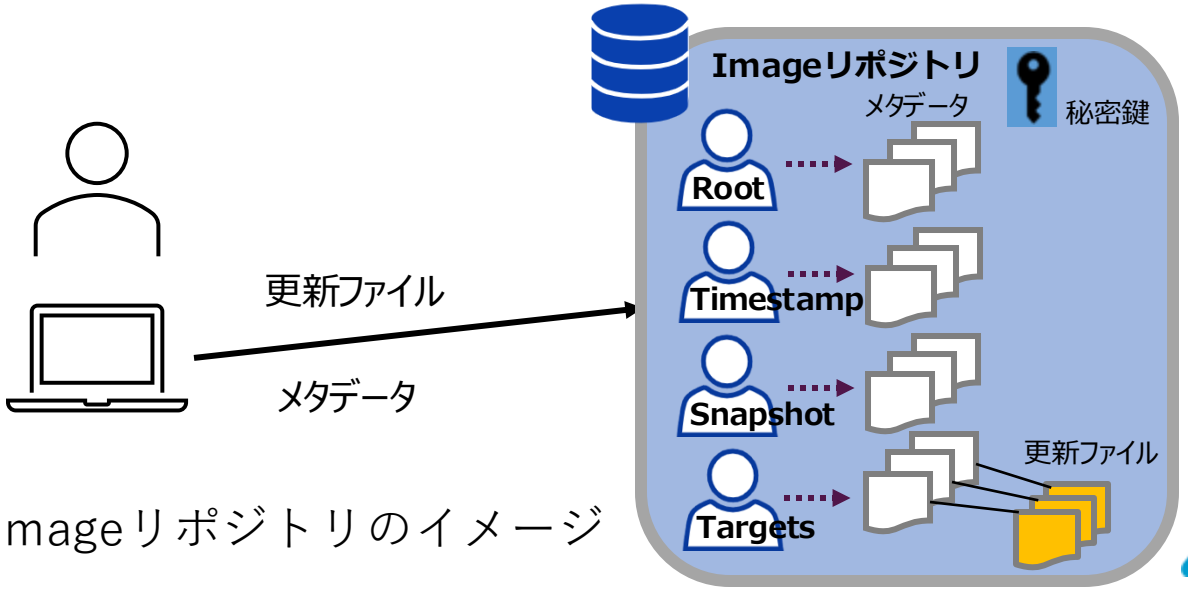
- Uptaneでは，自動車が利用可能な以下のサーバーを作る必要がある
  1. Imageリポジトリ
  2. Directorリポジトリ

- さらにECUが現在の時刻を知るための安全な方法を持つ必要がある

Imageリポジトリ

OEMやサプライヤが更新ファイルとそれに関連するメタデータをアップロード.
ユーザの振る舞いによって制御されるため，比較的頻繁に更新されない.

- 更新ファイルとメタデータのダウンロードを許可するインターフェースを公開
- アップロードする際に認証が必要
- 認証されたユーザがアップロードするための方法を提供



Imageリポジトリのイメージ

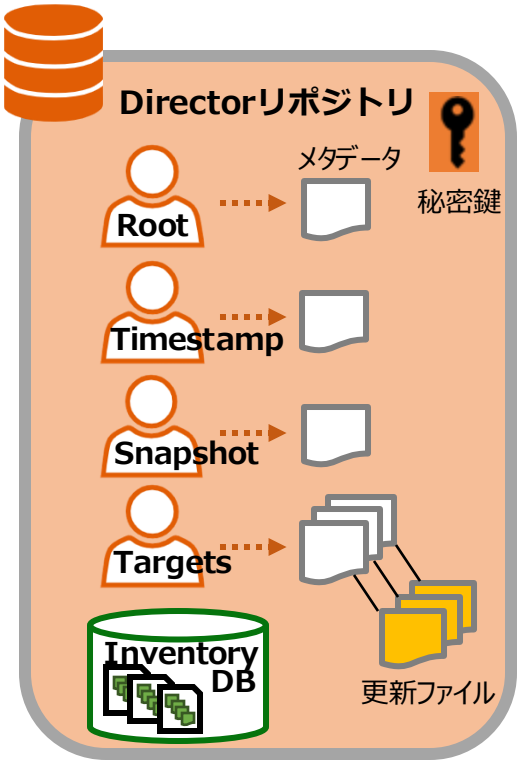
Directorリポジトリ

署名付きメタデータを作成することで，どの更新ファイルをインストールするかをECUに指示.

ほとんど自動化されたオンラインプロセスによって制御.

自動車，ECU，ソフトウェアリビジョンに関する情報を含むInventoryデータベースも参照.

- プライマリECUが車両バージョンマニフェストをアップロード，メタデータをダウンロードするためのインターフェースを公開
- リアルタイムで暗号化するか，暗号化したデータをリポジトリに格納することよりECUの更新ファイルを暗号化
- 自動化されたサービスが作成されたメタデータを格納するストレージを用意



Directorリポジトリのイメージ

Directorリポジトリ

Directorリポジトリは，自動車への更新ファイルのインストールを指示するために6つのステップに準拠する必要がある．

1. 車両の固有識別子により車両を識別
2. 車両の固有識別子を使用して，自動車内のECUに関する情報をInventoryデータベースに照会
3. Inventoryデータベース内の情報と比較して，車両バージョンマニフェストが正しいことを確認
4. ECUバージョンの偽装を防ぐためにカウンター，またはnonce(※)を確認
5. 車両の固有識別子から現在インストールされている更新ファイルの情報を抽出
6. 自動車にインストールする必要がある更新ファイルのセットを表すメタデータを作成

※nonce：”number used once(一度だけ使用される数)”の略で，
主に暗号通信等で使用される使い捨てのランダムな数字

Inventoryデータベース（Inventory DB）

Inventory DBには以下の情報を記憶する必要がある.

- 車両情報
  - 車両の固有識別子 (VIN番号など)
- ECU情報
  - ECUの固有識別子 (シリアル番号など)
  - ECUが関連付けられている車両ID
  - ECU署名検証鍵(共通鍵，または公開鍵)
  - ECU鍵識別子(鍵IDなど)
  - ECU種別（プライマリかセカンダリか）

Inventory DBと車両バージョンマニフェストの関係性

Inventory DBは，プライマリECUから車両バージョンマニフェストを取得し，Inventory DB内の情報と比較，差分があればECUの更新を指示する．ECU更新後に再度プライマリECUから車両バージョンマニフェストを取得し，Inventory DB内の情報を更新する.

車両バージョンマニフェスト

- 署名情報
  - 署名に使用される鍵の公開鍵識別子
  - 署名方式
  - 署名するペイロードのハッシュ値
  - 使用したハッシュ関数
  - ハッシュの署名

- その他情報
  - 車両の固有識別子（VIN番号等）
  - プライマリECUの固有識別子
  - ECUバージョンレポート

ECUバージョンレポート

- 署名情報
  - 署名に使用される鍵の公開鍵識別子
  - 署名方式
  - 署名するペイロードのハッシュ値
  - 使用したハッシュ関数
  - ハッシュの署名

- その他情報
  - ECUの固有識別子（シリアル番号等）
  - 現在インストールされている更新ファイルのファイル名、長さ、ハッシュ値（＝Targetsメタデータ）
  - 検出されたセキュリティインジケータ（指標となるもの）
  - バージョンレポートが作成された時刻
  - nonceまたはカウンター（ECUバージョンレポートの複製を防ぐため、更新ごとにインクリメントが必要）

アジェンダ

1. Uptaneとは
    - 概要，ゴール，想定システムの前提条件
    - The Update Framework(TUF)
2. 脅威モデル
    - 攻撃者のゴール，能力（機能），想定される攻撃
3. Uptaneの設計概要
    - 3.1 TUFの設計原理，Uptaneの構成，TUFとの違い
    - 3.2 リポジトリの役割
    - 3.3 メタデータ構造
    - 3.4 サーバー/リポジトリの実装要件
    - 3.5 車載ECUの実装要件

車載ECUの実装要件

前提条件

ECUがUptaneで保護された更新ファイルを受信できるようにするため次のデータが運用可能な状態にされている必要がある.

- 製造時，またはインストール時に必要なUptaneの最新版メタデータ
- 現在の時刻
- ECU署名生成鍵
  - ECUに固有の秘密鍵（共通鍵でも可）
    （ECUバージョンレポートへの署名と更新ファイルの復号にも使用）

プライマリECUの振舞い

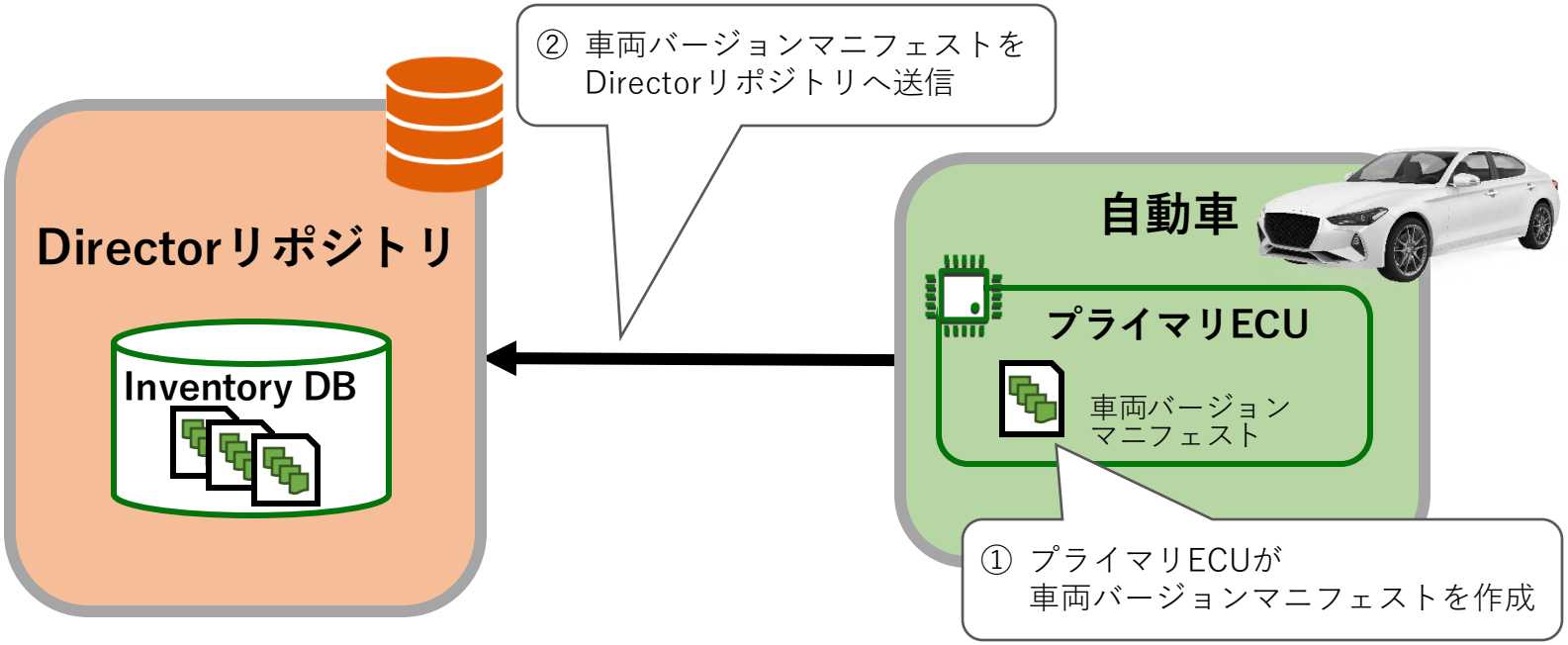
最新の時刻，メタデータ，及び更新ファイルをダウンロード，検証，配布するために次の7つのステップを実行.

1. 車両バージョンマニフェストの作成と送信
2. 現在時刻をダウンロードして確認
3. メタデータのダウンロードと検証
4. 更新ファイルのダウンロードと検証
5. 最新時刻をセカンダリECUに送信
6. セカンダリECUへメタデータを送信
7. セカンダリECUへ更新ファイルを送信

プライマリECUの振舞い

1. 車両バージョンマニフェストの作成と送信

① プライマリECUは，車両バージョンマニフェストを作成
② Directorリポジトリへ送信

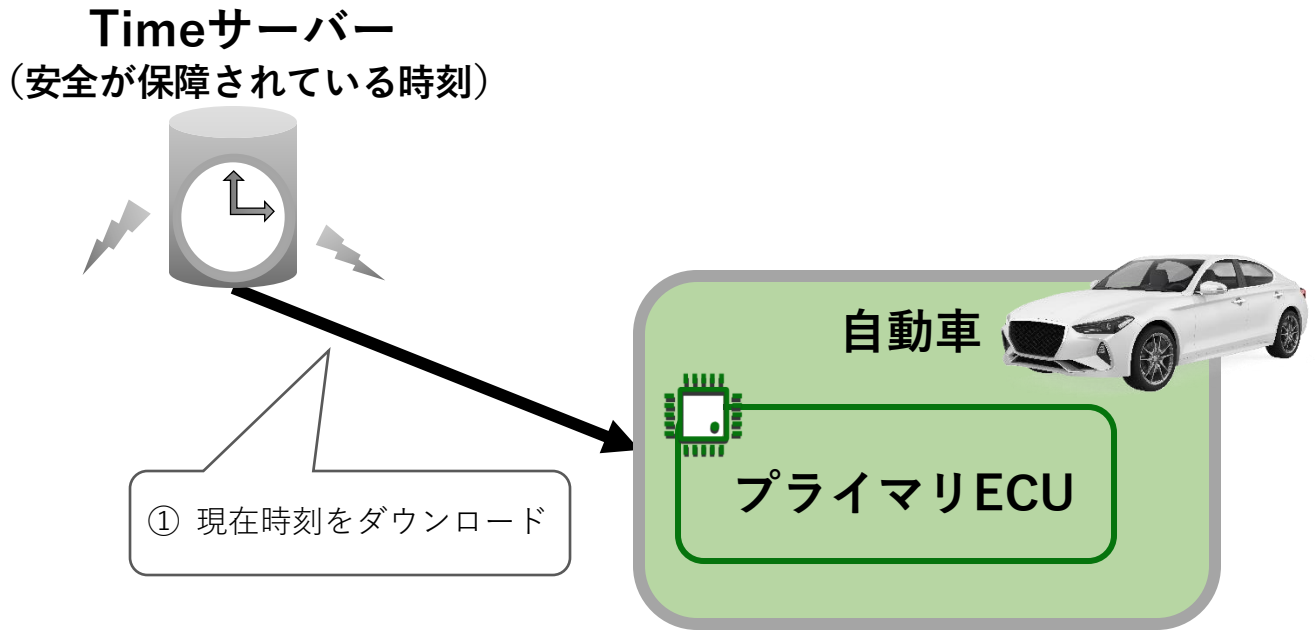


車両バージョンマニフェストの作成と送信

プライマリECUの振舞い

2. 現在時刻をダウンロードして確認

① プライマリECUは，セキュリティで保護されたソースから現在時刻をダウンロード

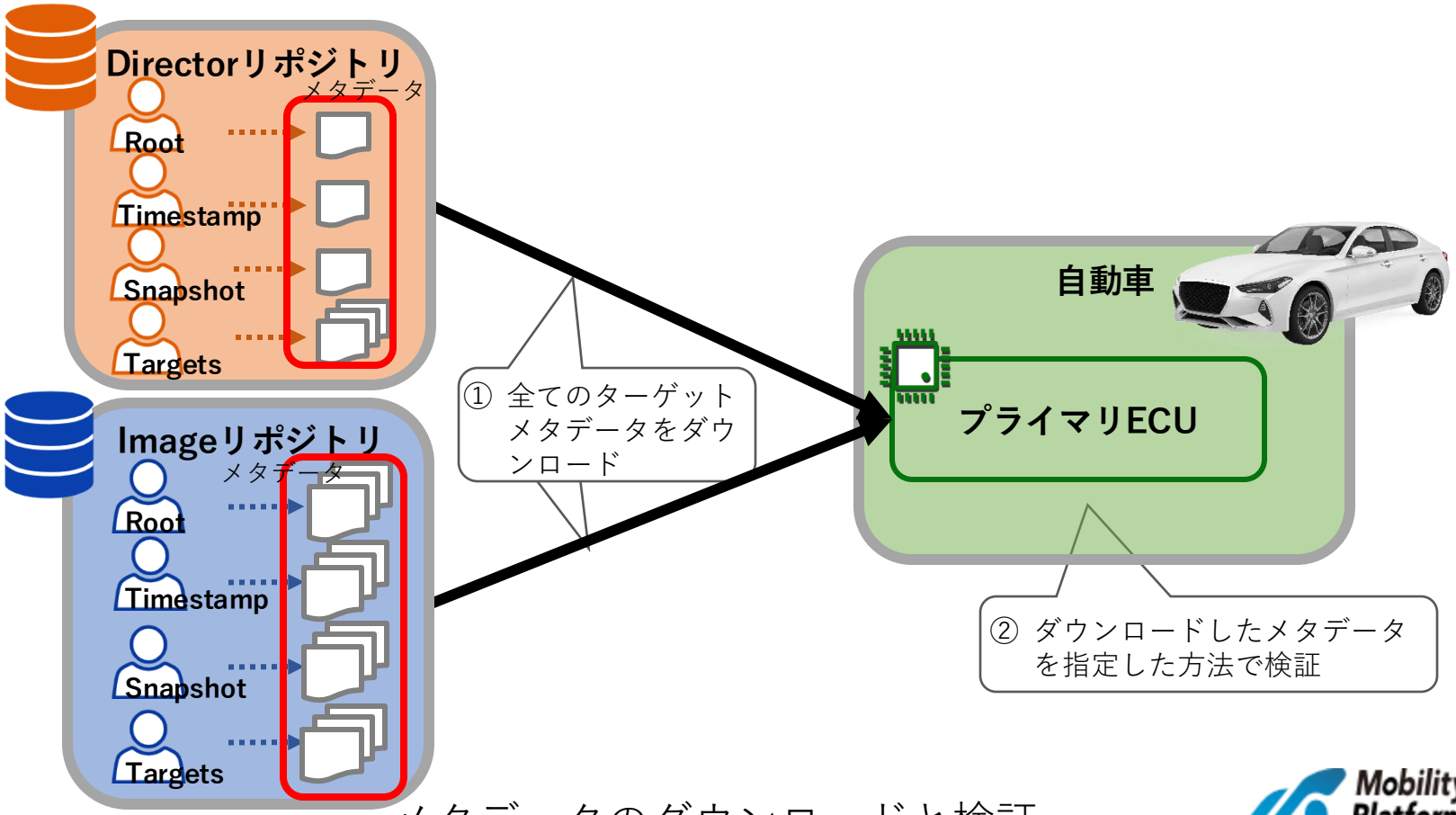


現在時刻をダウンロードして確認

プライマリECUの振舞い

3. メタデータのダウンロードと検証

① すべてのターゲットのメタデータをダウンロード
② 指定されている方法で検証実施



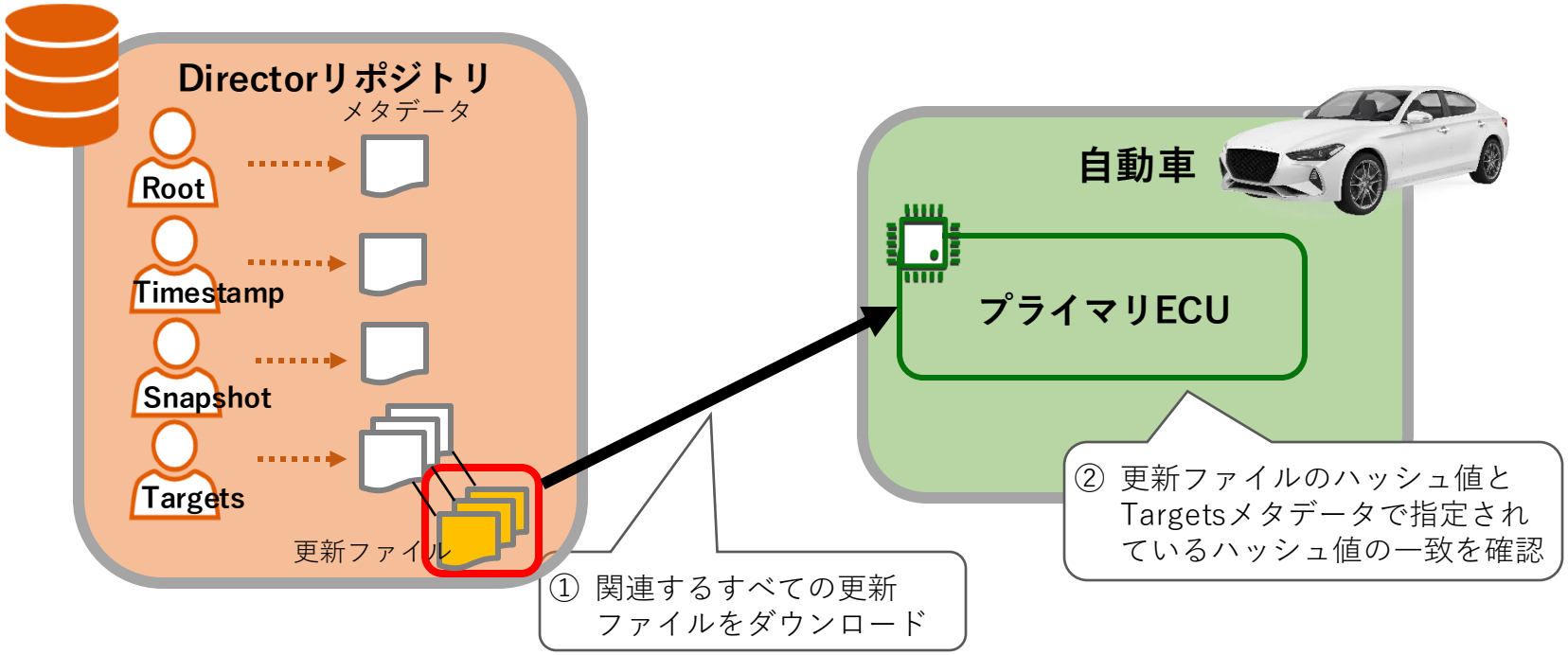
メタデータのダウンロードと検証

プライマリECUの振舞い

4. 更新ファイルのダウンロードと検証

① 関連するすべてのセカンダリECUの更新ファイルをダウンロード

② 更新ファイルはそのハッシュ値と，Targetsメタデータで指定されているハッシュ値と一致することを確認

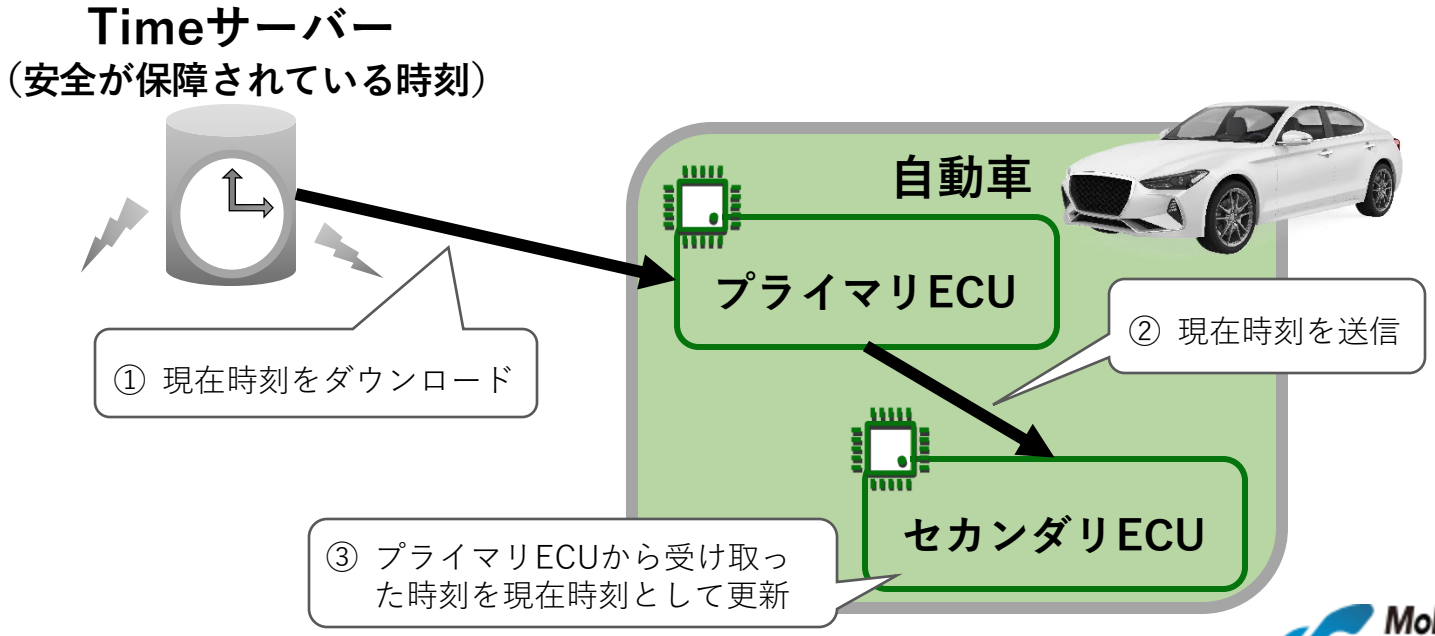


更新ファイルのダウンロードと検証

プライマリECUの振舞い

5. 現在時刻をセカンダリECUに送信

① 現在時刻をダウンロード
② 条件付きで各セカンダリECUに現在時刻を送信
※セカンダリECUが時刻を確認する方法を持っている場合は除く
③ セカンダリECUは，受信した時刻を現在時刻として上書き

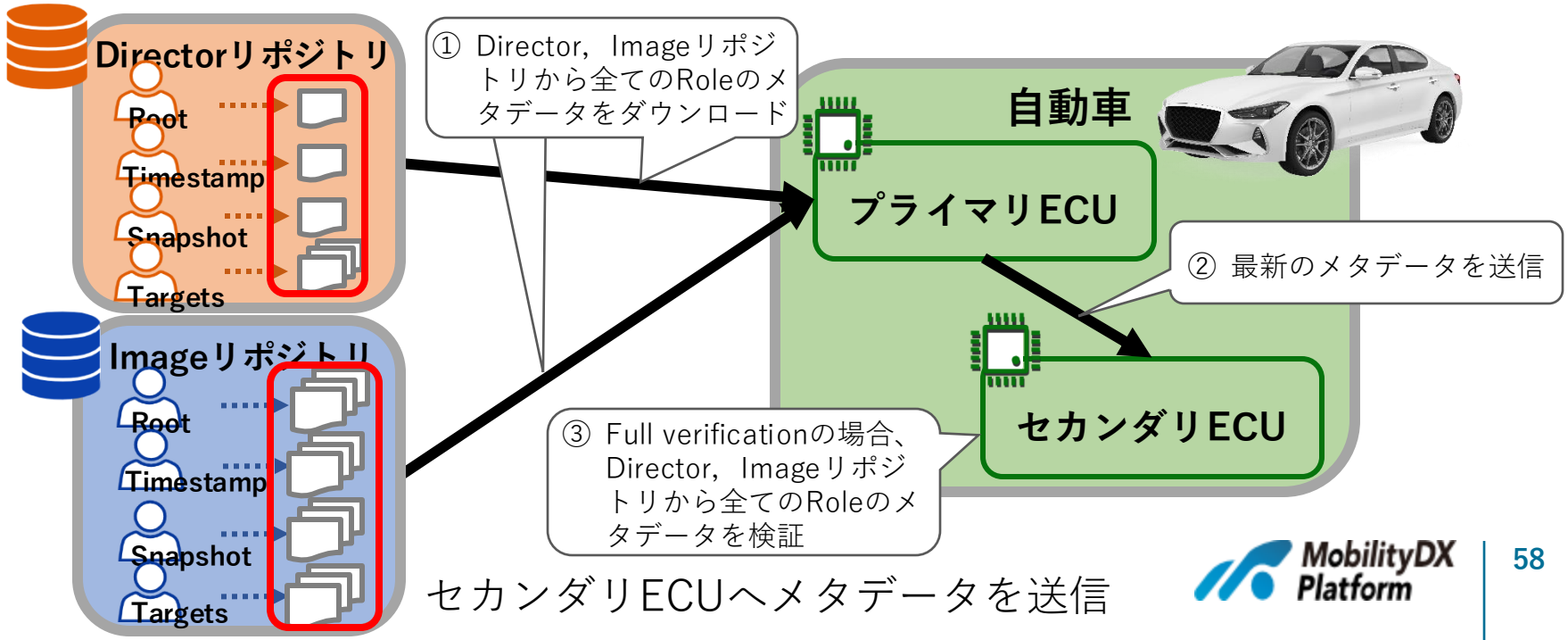


現在時刻をセカンダリECUに送信

プライマリECUの振舞い

6. セカンダリECUへメタデータを送信

① プライマリECUは，全てのRoleのメタデータをダウンロード
② プライマリECUは，ダウンロードした最新のメタデータを関連するすべてのセカンダリECUに送信
③ セカンダリECU側でFull verificationする場合，Directorリポジトリと Imageリポジトリの両方のリポジトリから4つすべてのRoleのメタデータを検証

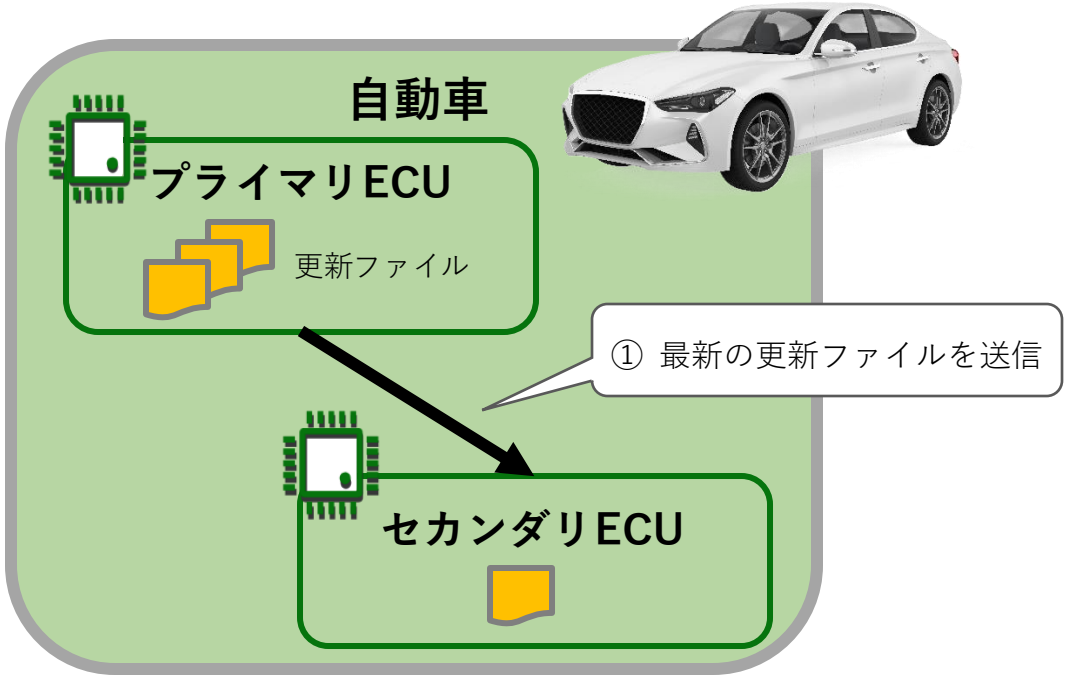


セカンダリECUへメタデータを送信

プライマリECUの振舞い

7. セカンダリECUへ更新ファイルを送信

① プライマリECUは，最新の更新ファイルを関連する各セカンダリECUに送信



セカンダリECUへ更新ファイルを送信

ECUの更新ファイルのインストール

ECUは新しい更新ファイルをインストールする際，以下の手順を実施.

1. 現在時刻のダウンロードと検証
2. メタデータの検証
3. 最新の更新ファイルのダウンロード
4. 更新ファイルの検証
5. 更新ファイルのインストール
6. ECUバージョンレポートの作成と送信
   （セカンダリECUはプライマリECUへ送信、プライマリECUは自身で保管）
7. プライマリECUが車両バージョンマニフェストを更新し、Directorリポジトリへ送信

メタデータの検証

Verification(検証)の種類

「Full verification」と「Partial verification」のいずれかを選択.
※UptaneはFull verificationを推奨

- Full Verification
  ImageリポジトリとDirectorリポジトリからダウンロードした更新ファイルとメタデータを全て検証.

- Partial Verification
  最低限DirectorリポジトリのTargetsメタデータのみを検証.

★プライマリはFull verificationの実装が必須

Full verification

検証内容

1. 現在時刻（もしくは証明された最新時刻）の読み込みと検証

2. Directorリポジトリの4つ（Root, Timestamp, Snapshot, Targets）のメタデータのダウンロードと検証

3. Imageリポジトリの4つ（Root, Timestamp, Snapshot, Targets）のメタデータのダウンロードと検証

4. DirectorリポジトリとImageリポジトリのTargetsメタデータの更新ファイルの情報が一致するかを確認

メタデータの検証

４つのメタデータ間（Root, Timestamp, Snapshot, Targets）で共通の検証と各メタデータ固有の検証について説明する.

- メタデータ共通の検証
- 各メタデータ固有の検証
  - Root
  - Timestamp
  - Snapshot
  - Targets

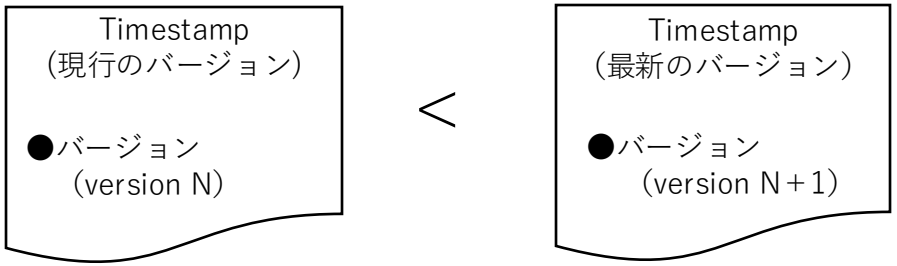
メタデータ共通の検証

４つのメタデータで共通する検証は以下（１～３）を行う.

1. メタデータファイルのバージョンの検証

   現行メタデータファイル*1のバージョン（N番）より最新メタデータファイル*2のバージョンが新しいこと（N＋1番）を確認

例（Timestampメタデータ）：



最新メタデータファイルのバージョンが現行メタデータファイルのバージョンより古い場合は更新サイクルを中止し，ロールバック攻撃の可能性を通知

＊1：現行メタデータファイル：現在使用しているECUのバージョンに紐づくメタデータファイル

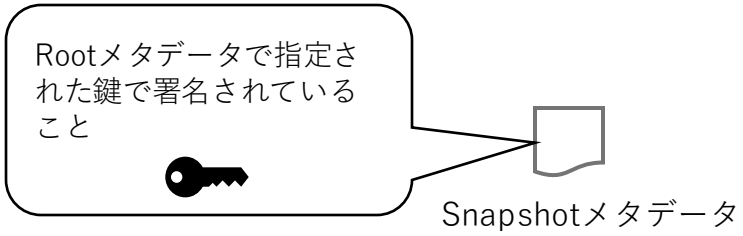
＊2：最新メタデータファイル：今回更新するファイルのバージョンに紐づくメタデータファイル

メタデータ共通の検証

2. 署名の検証

Rootメタデータで指定された鍵によって署名されていることを確認

例（Snapshotメタデータ）：



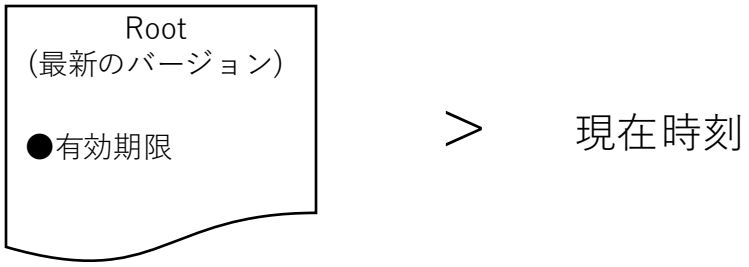
いずれの場合も，正しく署名されていない場合は更新サイクルを中止し，署名検証の失敗を通知（任意のソフトウェアへの攻撃をチェック）

メタデータ共通の検証

3. 有効期限の検証

現在時刻（もしくは証明された最新時刻）が最新メタデータファイルの有効期限内の時刻であることを確認

例（Rootメタデータ）：

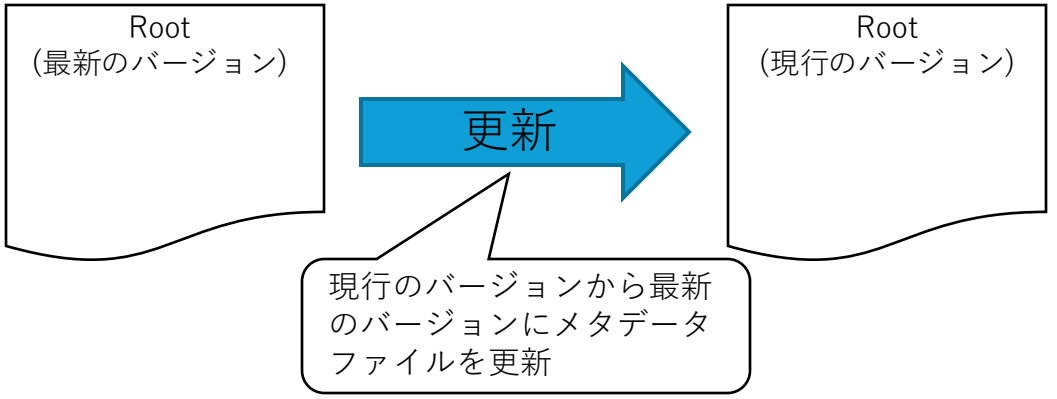


最新メタデータファイルの有効期限が切れている場合，これを破棄し，更新サイクルを中止，フリーズ攻撃の可能性を通知

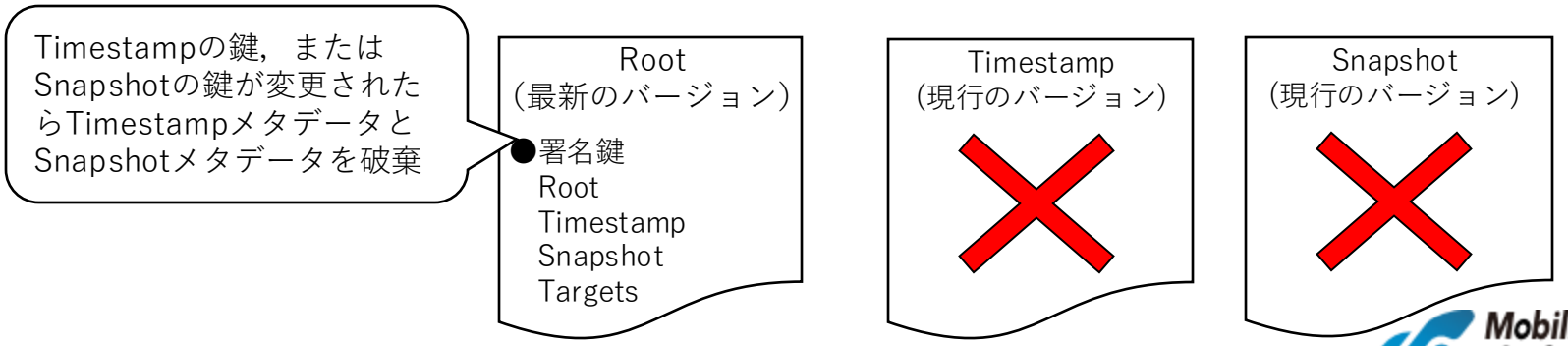
Rootメタデータ固有の検証

Rootメタデータ固有の検証は以下のことを行う.

1. 共通の検証１，２を行った後，最新のRootメタデータファイルに更新



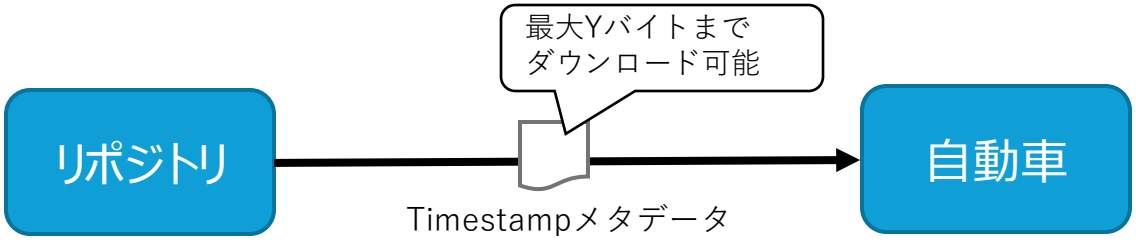
2. Timestampの鍵，またはSnapshotの鍵が変更された場合，現行のバージョンのTimestampメタデータとSnapshotメタデータを破棄



Timestampメタデータ固有の検証

Timestampメタデータ固有の検証は以下のことを行う.

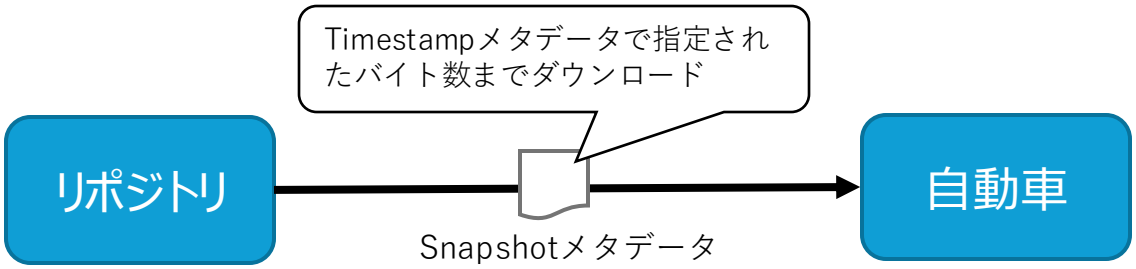
1. TimestampメタデータをYバイト数までダウンロード（Yの値は実装者が設定）. その後メタデータ共通の検証を行う.



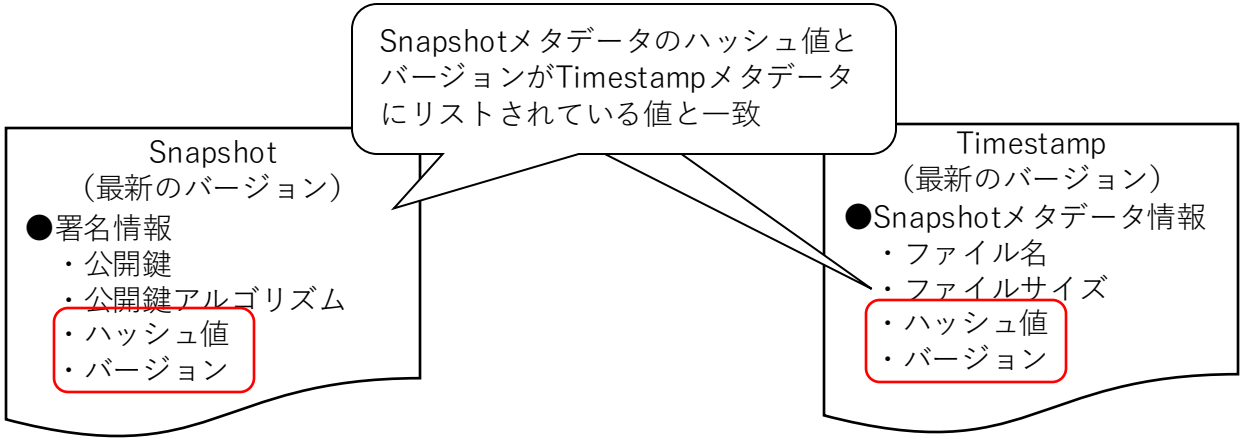
Snapshotメタデータ固有の検証

Snapshotメタデータ固有の検証は以下のことを行う.

1. Timestampメタデータで指定されたバイト数までSnapshotメタデータをダウンロード．その後メタデータ共通の検証を行う.



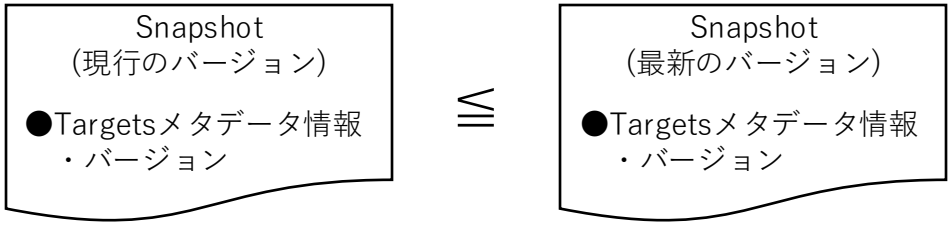
2. ハッシュ値とバージョンがTimestampメタデータにリストされているSnapshotメタデータ情報と一致することを確認.



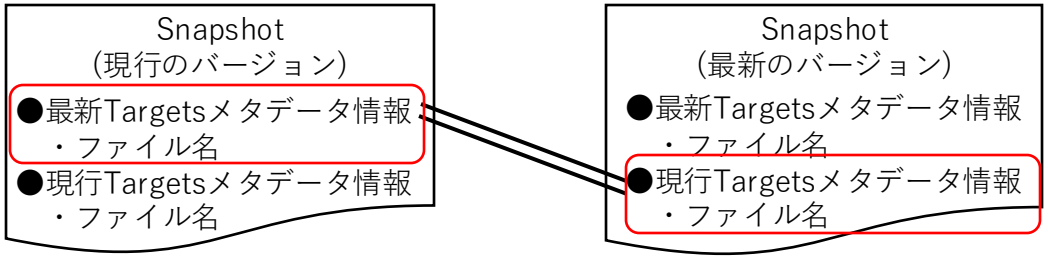
一致しない場合は最新のSnapshotメタデータを破棄し，更新サイクルを中止，検証の失敗を通知（ミックスアンドマッチ攻撃のチェック）

Snapshotメタデータ固有の検証

3. Snapshotメタデータに含まれるTargetsメタデータ情報の検証
   1. リストされているTargetsメタデータのバージョンが，現行メタデータファイルにリストされているバージョンよりも新しい(もしくは同等)であることを確認



   2. 現行のSnapshotメタデータファイルに記載されている各Targetsメタデータのファイル名が、最新のSnapshotメタデータにも記載されていることを確認。

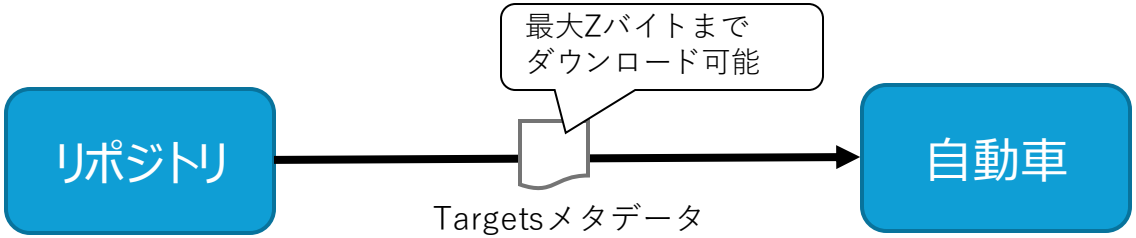


この条件が満たされない場合，最新のSnapshotメタデータを破棄し，更新サイクルを中止，検証の失敗を通知（ロールバック攻撃をチェック）

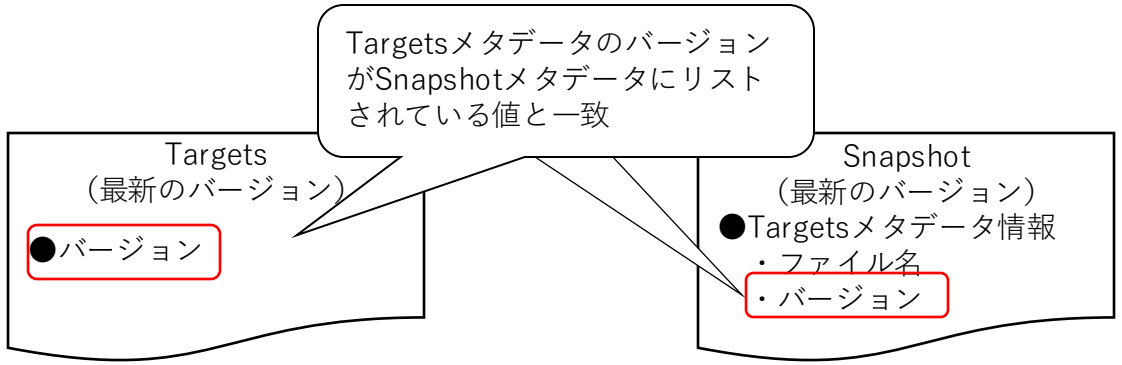
Targetsメタデータ固有の検証

Targetsメタデータ固有の検証は以下のことを行う.

1. TargetsメタデータをZバイト数までダウンロード（Zの値は実装者が設定）．その後メタデータ共通の検証を行う.



2. バージョンが，最新のSnapshotメタデータにリストされている値と一致していることを確認



バージョンが一致しない場合，最新のTargetsメタデータを破棄し，更新サイクルを中止，検証の失敗を通知
（ミックスアンドマッチ攻撃をチェック）

メタデータの検証

メタデータの検証まとめ

検証事項		メタデータの種類			
		Root	Timestamp	Snapshot	Targets
共通－1	バージョンの検証	○	○	○	○
共通－2	署名の検証	○	○	○	○
共通－3	有効期限の検証	○	○	○	○
固有	Timestampの鍵とSnapshotの鍵の検証	○			
固有	最新メタデータファイルと現行メタデータファイルに記載されたTargetsメタデータ情報の検証			○	
固有	依存関係にあるメタデータファイルに記載されている情報が一致しているかを検証			○ (バージョンとハッシュ値)	○* (バージョン)

＊Targetsメタデータの検証においてバージョンのみを検証する理由は、Snapshotメタデータが持つTargetsメタデータ情報にハッシュ値が含まれていないため。