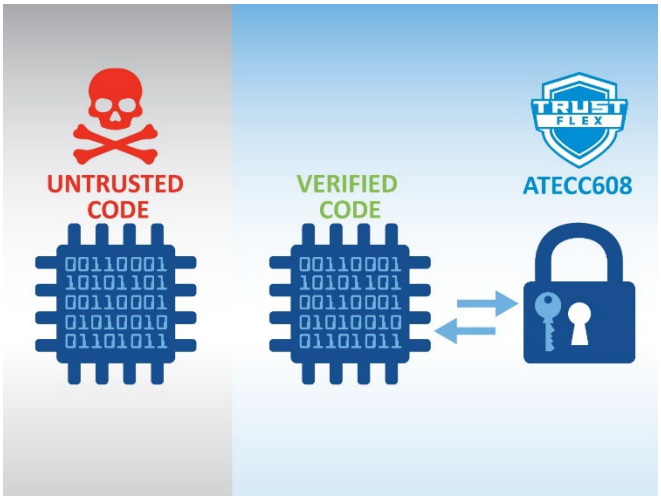
セキュア エレメントを使ったファームウェア検証 の 代表的な 3 つのユースケース (パート 1)

2 回シリーズのパート 1 では、アプリケーションのコードにファームウェア検証機能を実装する事の重要性について解説します。

今日の市場において、組み込み電子回路システムへの要求は大きく変化しており、それらの動作の重要度に応じて規格(例: 産業用アプリケーション向け IEC62443 セキュリティ規格)や法規則への対応がますます求められつつあります。しかし、規格や規則への対応だけがセキュリティではありません。セキュリティ機能の実装は基本的で重要な技術です。本シリーズのパート 1 では、各種の制約を受ける小型組み込みシステムでのファームウェア検証について解説します。組み込みシステムは内部のコード (ファームウェア、ソフトウェア等)を実行する事により動作します。パート 1 では組み込みシステム (ネットワーク接続の心臓ペースメーカー、産業用ゲートウェイ、IoT ベビーモニター、さらには産業機械内部のモータまで含む) のファームウェアが攻撃された場合に生じる問題について検討します。パート 2 では各種組み込みシステムにそれらのユースケースを実装する方法について解説しますので、そちらも是非ご覧ください。



製品に書き込まれたコードには企業の知的財産(IP)が大量に含まれます。ネットワーク接続型ペースメーカーの場合、人命に係わるためコードは極めて重要です。また、認証を取得する必要もあります。産業用ポンプのコードは、競合製品よりも優れた速度およびトルク制御を提供できる必要があります。産業用ゲートウェイのコードは、機械と人間が絡む複雑なスマートファクトリ ネットワーク内で、非常に大きなペイロードを競合製品よりも高速に処理できる必要があります。このようなシステムで機械の動作が不適切に変更されると、たちまち工場内の人々の安全に問題が生じかねません。IoT ベビーモニターのコードは、信頼性の高いネットワーク接続を維持して新生児に関する情報を保護者に提供し続ける必要があります。製品内のコードは企業の知的財産(IP)にとって極めて重要であるため、各種規格で定義されている以下の脅威モデルを考慮する必要があります。

1. **リモート デジタル攻撃:** コードへの不正なリモートアクセスにより、攻撃を受けた製品のみならず販売された全ての製品に被害が及ぶ恐れがあります。

2. **リモート論理攻撃:** コード内のバグを突いた不正行為により、ネットワークを介するリモート攻撃が容易になる可能性があります。
3. **ローカル物理攻撃:** 攻撃者が製品に対して物理的にアクセスする事で、バグを不正利用する事なくコードが攻撃される可能性があります。
4. **ローカル論理攻撃:** 攻撃者が製品に物理的にアクセスする事で、コードのバグが不正利用される可能性があります。

近年、これらの脅威に対応するため、政府および業界は IoT セキュリティ規則の規格化を進めています。例えば産業用アプリケーション市場では、ISA/IEC62443 規格 が産業用製品向けのセキュリティ対策を定義しています。EN303656 は、ヨーロッパのコンシューマ市場で販売される IoT 製品向けに適用される法規則に従います。UL2900 はソフトウェアセキュリティ対策を重視して策定され、現在コンシューマ市場向けの主な企業が注目しています。これら全ての規格または規則に共通する要件として、コードが純正である事を確認するための検証が求められます。

では、コードを保護するために何ができるでしょうか。また、そのためにどのようなトレードオフが生じるでしょうか。

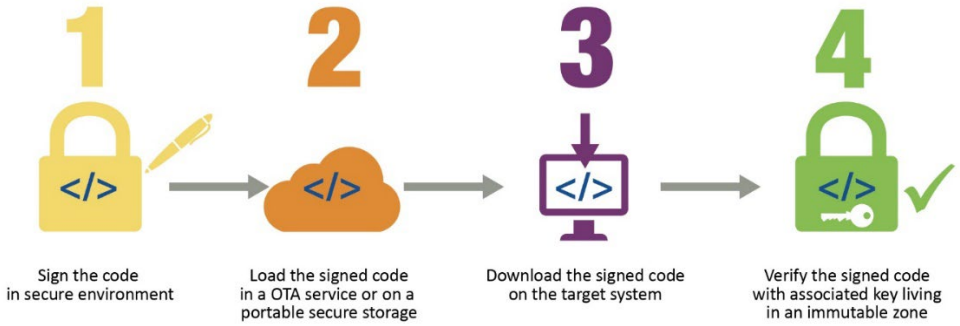
コードが純正である事を確認するには、暗号演算によってコードを検証する必要があります。組み込みシステムのコード検証は、以下のタイミングで実行できます。

1. **ブート時:** 一般的に「セキュアブート」と呼ばれ、各種の方法で実装できます。対称鍵または非対称鍵を使い、ダイジェストと署名のみの検証またはコードイメージ全体の検証が可能です。セキュアブートの目的は、組み込み回路で不正コードが実行されないよう保護する事です。
2. **実行時:** 一般的に「IP 保護」と呼ばれます。システム動作の任意タイミング (セキュアブートおよび OTA (Over-The-Air) 更新時を除く)でコード検証を実行し、コードが改ざんされていない事を確認します。
3. **OTA 更新後:** IoT アプリケーションでは、ネットワーク経由の OTA 更新により既存のコードを新しいコードに書き換えます。この新しいコードを実行する前に、コードが純正である事を検証する必要があります。

これらの 3 つのセキュリティ機能は全てコード検証に関係します。

以下では、コード検証機能の実装方法と、それによって生じるトレードオフについて解説します。基本的には、理想的にセキュアな企業環境内でコードに「署名」し、組み込みシステム内部でそのコードを「検証」する必要があります。「署名」と「検証」は、暗号アルゴリズムと対称鍵または非対称鍵のセットを使って実行します。この手順は、下図に示す通り 4 つの主要ステップで構成されます。

ステップ 1 の課題: 製造中に何が生じるか? どのようにコードの暗号化と署名を処理するか
ステップ 2 の課題: 署名済みのコードをどのように OTA 配布システムにアップロードするか
ステップ 3 の課題: コードをどのように組み込みシステムにダウンロードするか
ステップ 4 の課題: 製造後の組み込みシステム内部で何が発生するか
ターゲット アプリケーションで実行されているコードは本当に純正か



対称鍵を使った製造時の署名

対称暗号は共有鍵アーキテクチャに基づきます。これはメッセージの送信元と受信先で同じ鍵をペアにして使います(対称鍵は共有鍵とも呼ばれます)。この方法の大きな短所は、2 つの鍵が同じであるため 1 つの鍵にアクセスできればシステムを容易に攻撃できる事です。また、デバイスごとに異なる対称鍵を持たせるのが基本ですが、全てのデバイスに対して一意の対称鍵をプロビジョニングする必要があるため、管理上の課題が生じます。実装を容易にしてプロジェクトの日程を守るために全ての製品に同じ対称鍵を使った場合、鍵が暴露された時の被害が大きくなります。

製造時の手順は以下の通りです。最初の手順は企業内のセキュアな環境で行われます。

- OEM (Original End Manufacturer)では、コードが「ハッシュ」関数により処理されます。SHA256 を使う場合、ハッシュの出力はコードイメージの 32 バイトダイジェストです。
- このハッシュははコードに加えて対称鍵(署名用 OEM 鍵)を処理します。
- その出力はメッセージ認証コード(MAC)です。
- MAC は委託製造業者(CM)に提供されます。CM は工場でメイン コントローラをフラッシュし、MAC と対称鍵の両方を書き込みます。
- 鍵は、製造中およびマイクロコントローラ内部で決して暴露されない事が必要ですが、この CM での工程中にサプライチェーンの脆弱性が問題となる可能性があります。すなわち、工場内の従業員および MAC を生成する機器(OEM で未実行の場合) へ鍵が暴露される可能性あるという事です。

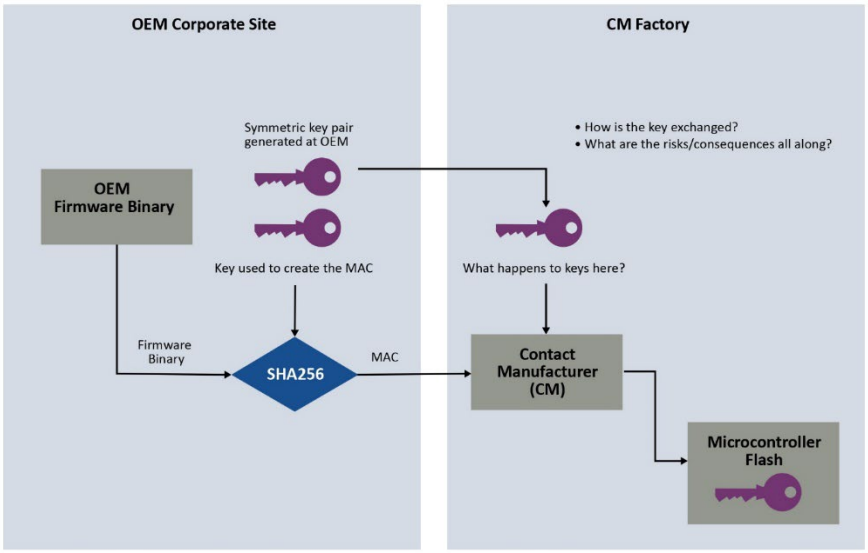


図 1: OEM 施設での対称署名

非対称鍵を使った製造時の署名

非対称鍵を使う事で信頼性と拡張性が向上します。非対称鍵は、異なる 2 つの鍵をペアで使い、それらは暗号アルゴリズムにより数学的に関連付けられます。例えば ECC-P256 は、組み込みシステム向けに最も一般的に使われているアルゴリズムの 1 つです。非公開鍵がコードに署名し、公開鍵(非公開鍵から算出) が署名またはダイジェスト(もしくはその両方)を検証します。

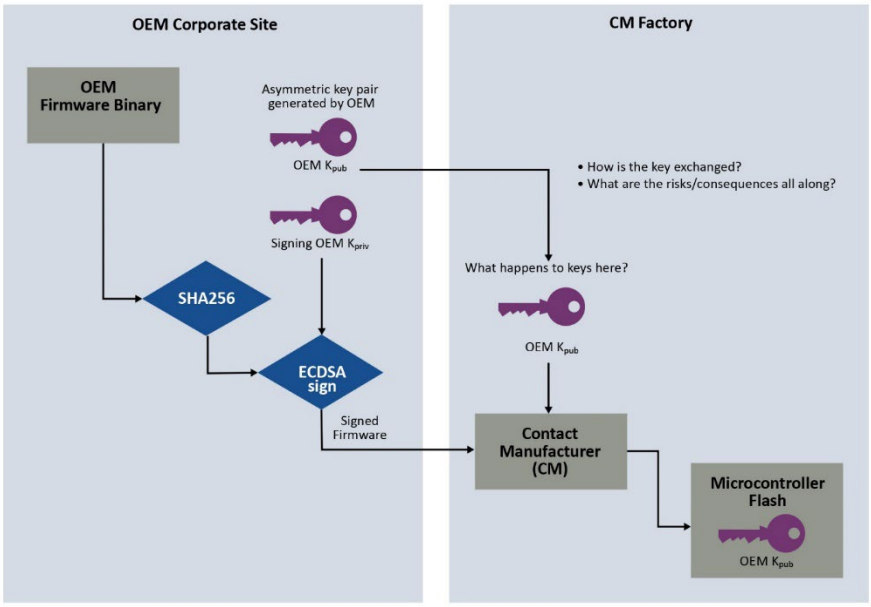


図 2: OEM 施設での非対称コード署名

製造時に鍵にアクセス可能なのは誰か?

対称鍵または非対称鍵のどちらを使っても、イメージ検証用の暗号鍵を組み込みシステム内に書き込む必要があるため、以下の点を考慮する必要があります。

- 署名済みコードを保護する鍵を扱う委託製造業者(CM)は信頼できるか?
  コードは企業の知的財産(IP)である事を思い出してください。鍵を渡された委託業者(CM)はコードにアクセスできます。
- CM は署名鍵または検証用の鍵にアクセスできるか?
  CM を変更する必要が生じた場合、鍵はどうなるか?
- 鍵処理の CM 委託により不明な依存性が生じていないか?
- 複数の CM と契約する場合、鍵ペアの各種セットをどのように管理するか?
- CM では鍵はどのように保護されているか?
  セキュリティ監査の結果はどうか?

製造中の鍵の管理を考える時、サプライチェーンにも様々な課題が存在します。署名鍵は常に最も重要な鍵であり、全ての暴露要因(機械および人員)から隔離する必要があります(理想的には HSM (Hardware Security Module)を使用)。鍵の隔離に加えて、企業の IP を保護するためにコード検証が極めて重要となります。鍵を委託製造業者に渡すと、鍵が暴露されるリスクが増大します。このため、以下の点を考慮する必要があります。

- 鍵はセキュアに保管されているか?
  鍵は従業員の間で共有されているか?
- 製造機器のネットワーク保護が脆弱なために鍵が外部からアクセスされる可能性はないか?
- 従業員が検証鍵を保存した USB スティックを工場の外に持ち出さないか?
  セキュリティ監査はどのように実施されているか?
  監査は信頼できるか?
- 委託業者の担当従業員が会社を辞めた時に鍵はどうなるか?

組み込みシステム内のどこに鍵を保存するか?

従来型のマイクロコントローラ内のフラッシュにファームウェア検証用の公開鍵を保存する場合、熟慮が必要です。この場合、暗号資産はコードのバイナリイメージに含まれます。従来のエンジニアなら、鍵(公開/対称)を単純にマイクロコントローラまたはプロセッサ内のフラッシュに保存するかもしれません。この場合、以下について考える必要があります。

この鍵にはどの程度の価値があるのか? 攻撃者はこの鍵に対して何ができるか?

フラッシュメモリの空き領域に鍵を保存した製品は現在でも多数存在します。攻撃者はバッファ オーバーフロー(例: Heartbleed)や HEX ファイルの抜き取り等、各種テクニックを使ってメモリ内の鍵にアクセスを試みます。これらの攻撃は実際に発生しており、多くの企業がそのようなシステムの脆弱性を報告しています。鍵が盗まれると、ネットワークを利用した攻撃の波及増大を防ぐ手立ては無くなります。全ての鍵がバイナリイメージ内に保存されているためにアクセス可能である場合、それらは読み取られて変更および複製される可能性があり、そうすると市場に出回っている大量の製品に対する不正リモートアクセスの危険性が増します。前述の通り、コード署名向けに対称鍵を使う方法は、あまり信頼性が高くありません。では、コードの検証に使う鍵をコントローラ内の OTP に保存するとどうでしょうか。OTP に保存する事で鍵は改変不可能なメモリ領域に保存されるため、システムの信頼性は少し向上します。しかし、改変不可能とは、アクセス不可能または読み出し不可能を意味するのではありません。単純に書き換えができないというに過ぎ

ません。コード内の IP の価値が高いほど、IP を保護するための鍵の重要性も増します。鍵がアクセス可能であれば、大切なコード(IP)が読み取られて複製および再使用されてしまう恐れがあります。悪意のあるユーザは、コード検証をパスさせるための MAC を生成できます。

- 対称鍵の場合(特にシステムがネットワークに接続されている場合) は最悪の状況となります。鍵にアクセスした攻撃者は不正コードの MAC を実行し、checkMAC を使って不正コードを容易に検証に合格させる事ができます。なぜなら、署名用の対称鍵はコード検証用の対称鍵と同じだからです。さらに悪い事に鍵派生を行わない場合、全ての製品の鍵を所持する人物を熟慮する事が極めて重要です。ネットワークに接続したベビーモニターや産業用機械の全てが不正な OTA 更新によりハッキングされ、状況はさらに悪化する可能性があります。
- コントローラのフラッシュ(OTP も含む) 内の公開鍵にアクセスした不正ユーザは、その鍵を複製して再使用する事でユーザ認証に成功し、不正コードをターゲット マイクロコントローラにインストールして実行させる事ができます。

公開鍵基盤を採用する事で、対称鍵アーキテクチャに比べて信頼性を大きく向上させる事ができます。なぜならコードに署名する非公開鍵は、コード検証用の公開鍵と(数学的に関連付けられてはいても)異なるからです。非公開鍵にアクセスした攻撃者はコードの検証を調べてトリガできるだけでなく、コードの実行前に復号もできます。その時点で彼らは「コードは読み出せたので、このコードを書き換えて、非公開鍵による検証をバイパスさせよう」と考えます。この問題に対処するには、適切な暗号アクセラレータ(Microchip 社製セキュアエレメント(ECC-P256)等)に加えてマイクロコントローラまたはプロセッサ内部の BootROM 機能が必要です。BootROM は、コントローラが検証コマンドを発行するメモリ領域を改変不可能かつバイパス不可能にします。しかし、たとえ BootROM を使っても、公開鍵へのアクセスが重要な問題として残ります。

ファームウェア検証における対称鍵と非対称鍵の短所と長所

鍵	長所	短所
対称鍵	少ない - シンプルな暗号処理 - 実装が容易	多い - 鍵を盗んで複製する事が非常に容易 - 被害が多数の製品に波及する危険性が高い - 製品全てに対して一意鍵を展開/管理する事が難しい
非対称	多い - 信頼性の高い暗号処理 - スケーラブルな鍵インフラストラクチャ - ECC 鍵によるメモリ消費量の最適化	少ない - ファームウェア検証に関する限り短所はない - ECDSA は SHA256 よりも処理時間が長い

パート 1 では、アプリケーションにファームウェア検証を実装する事の大切さについて解説しました。アプリケーション コードが純正である事を確認するには、暗号演算によってコードを検証する必要があります。検証は組み込みシステムの動作中に各種のタイミング (ブート時、実行時、ファームウェア更新時)で実行できます。非対称または対称暗号アルゴリズムを使ってファームウェア検証を実装する方法は各種存在します。各方法には長所と短所があります。パート 2 では、Microchip 社のセキュア エレメントとマイクロコントローラを使ってファームウェア検証を実装する方法を紹介します。

Xavier Bignalet, Dec 6, 2021