# 暗号鍵の HSM による管理

著者：関 勝寿　公開日：2026 年 5 月 6 日　ソース：https://sekika.github.io/2026/05/06/HSM/

HSM (Hardware Security Module) は、暗号鍵の生成・保管・使用を専用のハードウェア上で行うための装置である。銀行の ATM ネットワーク、インターネットの TLS 証明書、電子政府システムなど、高いセキュリティが求められる場面で標準的に採用されている。一般に HSM は高価なエンタープライズ機器であるが、USB 接続型の軽量 HSM である YubiHSM 2 のように、運用機能やスケーラビリティの範囲を限定することで、より低コストで提供される製品も存在する。

通常のサーバーや PC では、秘密鍵はファイルシステムやメモリ上に存在するため、マルウェアや不正アクセスによって鍵そのものが盗まれるリスクがある。HSM はこの問題に対して根本的な解決策を提供する。鍵は HSM の内部にのみ存在し、外部に平文で出ることはない。この記事では、HSM について調べたことをまとめる。

## 1.HSM が防げること

HSM の強みは以下である。

- **秘密鍵の物理的な保護**：HSM は耐タンパー性を持つ専用ハードウェアとして設計されており、秘密鍵は内部のセキュア領域に保持されるため、通常のサーバーのように OS やメモリダンプ、ディスクコピーなどから直接取得することはできない。多くの製品では改ざんや異常状態を検知した場合に内部データを消去する仕組みを備えている。
- **鍵のライフサイクル管理**：鍵の生成から廃棄までを HSM 内部で管理でき、外部のシステムは鍵そのものではなく署名や復号などの操作のみを要求する形になる。また、アクセス制御や操作ログの記録機能を備えており、誰がどの操作を行ったかを追跡できる。ただし、不正使用の検知や監査機能の具体的な強度は製品や構成に依存する。

## 2.HSM でも防げないこと

HSM は「鍵を盗まれない」ための道具であり、以下のような攻撃は防げない。

- **署名命令そのものへの攻撃**：HSM は「何に署名するか」については関知しない。攻撃者がシステムに侵入し、正規の署名インターフェースを通じて「偽のデータに署名させる」命令を送ることは技術的に可能だ。HSM は命令が正規のルートから来ているかを確認するが、その命令の内容が業務上正当かどうかは判断できない。これを防ぐには、HSM への署名要求を生成するアプリケーション層のセキュリティが別途必要だ。
- **内部不正（インサイダー脅威）**：HSM を操作する権限を持つ人間が悪意を持って行動した場合、技術的な防御だけでは限界がある。正規の手順でアクセスし、正規の命令を送ることで、攻撃者が意図したデータに署名させることができる。金融機関では「デュアルコントロール」と呼ばれる、複数人の承認なしに重要な操作ができない運用設計が義務付けられている。
- **ソーシャルエンジニアリング**：技術的な防御がどれだけ強固でも、人間を騙すことはできる。管理者権限を持つ担当者が偽の問い合わせに応じてしまえば、HSM は無力である。
- **HSM 自体の脆弱性への攻撃**：過去には特定メーカーの HSM に対してサイドチャネル攻撃が成立した事例や、ファームウェアの脆弱性が報告された事例がある。多くの場合、これらの攻撃は高度な物理アクセスや専門的な装置を前提とするが、理論上は鍵の抽出につながる可能性も否定できない。
- **「鍵を使う権限」への攻撃**：HSM に接続されたサーバーが侵害された場合、そのサーバーに付与されている「HSM への署名要求権限」が悪用される可能性がある。

## 3. 監査ログを確実に残すには

こうした攻撃に対抗するために、HSM が署名操作を行ったという記録を、攻撃者が無効化できない形で残すことは、セキュリティ設計の重要な課題である。これは単純に見えて、実は難しい問題である。

攻撃者は様々な手段でログの記録・保全を妨害できる。たとえば、ネットワークを切断してログ送信を止める、ログサーバーへの通信をブロックする、ログサーバー自体を攻撃する、HSM の設定を変更してログ機能を無効化する、あるいはログファイルを直接改ざんするといった方法が考えられる。これらに対して、AI は以下のアプローチを提案した。単一の対策ではなく、複数の独立した仕組みを組み合わせることが重要である。

### 3.1. 基本レイヤー：ログの真正性と完全性の担保

#### 3.1.1. アプローチ 1：ログエントリ自体を HSM が署名する

最も基本的かつ重要な対策である。署名操作が発生したとき、HSM はその操作の記録（署名対象のハッシュ、タイムスタンプ、操作者 ID など）を含むログエントリを生成し、別の専用鍵でそのログエントリ自体に署名する。これにより、ログが後から改ざんされた場合に検知できる。「ログが届いたかどうか」は保証できないが、「届いたログが本物かどうか」は保証できる。

#### 3.1.2. アプローチ 2：ハッシュチェーンによるログの連鎖

ログエントリを独立に保存するのではなく、前のエントリのハッシュを次のエントリに含める構造にする。

```
エントリ 1: { 操作内容 , 時刻 , ハッシュ (genesis)} → HSM 署名
エントリ 2: { 操作内容 , 時刻 , ハッシュ ( エントリ 1)} → HSM 署名
エントリ 3: { 操作内容 , 時刻 , ハッシュ ( エントリ 2)} → HSM 署名
```

この構造により、途中のエントリの削除や改ざんは連鎖的に検知可能になる。攻撃者がログを「なかったことにする」には、それ以降のすべてのエントリを再生成する必要があるが、HSM の署名鍵がなければそれは不可能である。

### 3.2. 可用性レイヤー：ログ消失への耐性

#### 3.2.1. アプローチ 3：複数の独立したログサーバーへのリアルタイム転送

ログを複数の独立したサーバーに同時送信する。物理的に分離され、可能であれば異なる管理主体が運用する構成が望ましい。一部のサーバーが攻撃されても、他のサーバーにログが残るため、完全な消去は困難になる。金融機関では複数拠点への冗長化が一般的である。

### 3.3. 改ざん耐性レイヤー：保存後の変更防止

#### 3.3.1. アプローチ 4：WORM（Write Once Read Many）ストレージ

一度書き込んだら変更・削除できないストレージにログを保存する。ハードウェアまたはサービスレベルで不変性が保証されるため、攻撃者による事後改ざんを防ぐことができる。クラウド環境でも、不変ストレージ（例：オブジェクトロック機能など）を利用することで同様の効果が得られる。

### 3.4. 外部検証レイヤー：組織外への証跡固定

#### 3.4.1. アプローチ 5：パブリックブロックチェーンへのアンカリング

一定期間ごとにログのルートハッシュをパブリックブロックチェーンに

記録する。これにより、ログの存在証明が外部に固定される。後からログを改ざんすると、ブロックチェーン上のハッシュと一致しなくなるため検知可能となる。詳細なログはオフチェーンに保持し、その要約のみを記録することでコストを抑えられる。