CL-004

セキュアプロセッサにおける楕円曲線暗号の評価
An Evaluation of Elliptic Curve Cryptography in Secure Processor

谷合 廣紀 † Hiroki Taniai
宮永 瑞紀 † Mizuki Miyanaga
入江 英嗣 † Hidetsugu Irie
坂井 修一 † Shuichi Sakai

1 はじめに

近年, 電子書籍や音楽データなどのデジタルコンテンツは急速に普及しており, ユーザは気軽にそれらをダウンロードして楽しめるようになった一方, デジタルコンテンツの製作者はデータの保護に頭を悩まされるようになった．紙の書籍を複製するためには手間がかかる上, オリジナルよりも質は劣化するが, 電子書籍のようなデジタルコンテンツでは簡単に同じものを複製することが可能であるためである．その複製の容易さ故, 法律で禁じられてはいるものの, 複製品の売買や配布などはあとをたたない.

このようにデジタルコンテンツが複製されることを防ぐため, コンテンツを暗号化し特定のソフトウェア上でのみ再生できるようにした DRM 技術が提案されてきた．しかしながら, 近年リバースエンジニアリング技術の発展やオープンソースの OS の普及により, プロセスや OS に改変が加えられた環境でコンテンツが実行される可能性がある．このような環境では, ソフトウェアに埋め込まれたコンテンツ復号鍵が読み取られたり, メモリ上に展開された平文のコンテンツが読み取られてしまう危険性がある．例として初期の DRM 技術である DVD を暗号化する CSS は, リバースエンジニアリングにより暗号鍵が流出し, 現在ではその実効性は失われている.

このような問題にはソフトウェアによる技術だけでは対処することはできない．そのためデジタルコンテンツ保護の方策として, ソフトウェアより強い権限を持つハードウェアによってセキュアな機能を付したセキュアプロセッサがあげられる．セキュアプロセッサによって OS やユーザプロセスの改変を検出することができ, デジタルコンテンツを不正な環境で実行することを防ぐことができる．そのような手法として TPM の TrustedBoot [1] や Intel SGX [2] のソフトウェア認証が提案されてきた.

プラットフォーム検証では OS やユーザプロセスのメモリイメージを取得し, 電子署名を施した上で外部サーバによる認証という手法をとるが, その際の電子署名には公開鍵暗号が用いられる．公開鍵暗号のアルゴリズムとしては現在,RSA 暗号と楕円曲線暗号が主に使われているが, 同じセキュリティ強度を実現するために必要な鍵の長さが楕円曲線暗号では RSA 暗号に比べ約 10 分の 1 で済むことが知られている [3]．鍵長が短くなることでメモリやメモリのバンド幅の使用率, さらには消費電力を抑えることができる [4].

既存研究におけるセキュアプロセッサ上での公開鍵暗号のハードウェア実装は Intel SGX での 3072bit の RSA 暗号 [2] のみであり, 楕円曲線暗号を実装したセキュアプロセッサはなかった．RSA 暗号ではなく楕円曲線暗号を実装することはセキュアプロセッサ全体の性能向上につながることが期待できる．そのため本研究では, セキュアプロセッサにおける公開鍵暗号の用途を考察し, プロセッサに楕円曲線暗号の電子署名である ECDSA と RSA 署名を実装, 評価を行った.

本論文の構成は, 第 2 章で公開鍵暗号, 第 3 章でセキュアプロセッサの既存研究を紹介し, 第 4 章で ECDSA と RSA 署名の実装, 評価を行い, 第 5 章でまとめと今後の展望を述べる.

2 公開鍵暗号

2.1 公開鍵暗号の概要

公開鍵暗号は暗号化と復号に非対称な鍵を用いる暗号方式である．公開鍵暗号では鍵は公開鍵と秘密鍵のペアによって構成され, その非対称性により通信の秘匿と電子署名の機能を提供する.

- 通信の秘匿
  (e, d) を Bob の公開鍵・秘密鍵ペア,E を暗号アルゴリズム,D を復号アルゴリズムとして,Alice が Bob にメッセージ m を暗号化して送ることを考える.
  1. Alice は Bob の公開鍵 e を用いて暗号文 $c = E(e, m)$ を作成し Bob に送信する.
  2. Bob は受信した暗号文 c を秘密鍵 d を用いて復号し, 平文 $m = D(d, c)$ を得る.

  このように Alice と Bob は事前に鍵の共有を行わずに通信を秘匿することができるため, 公開鍵暗号は共通鍵暗号の鍵の配送問題を解決できる [5].
- 電子署名
  (e, d) を Bob の公開鍵・秘密鍵ペア,S を署名アルゴリズム,V を検証アルゴリズムとして,Bob がメッセージ m に署名を行い Alice がその署名を検証することを考える.
  1. Bob は秘密鍵 d を用いて署名 $s = S(d, m)$ を作成
  2. Alice は公開鍵 e を用いて $v = V(e, m, s)$ を求めて署名 s が受理されるか検証する

公開鍵暗号は暗号鍵を公開しても対となる復号鍵を知らなければ暗号文の復号は計算量的に困難になるように

† 東京大学, The University of Tokyo

落とし戸付き一方向性関数を用いて構成される [5]．そのような落とし戸付き一方向性関数として離散対数問題, 素因数分解問題, 楕円曲線上の離散対数問題などがあり, それぞれ ElGamal 暗号,RSA 暗号, 楕円曲線暗号を構成している．

2.2 RSA 暗号

RSA 暗号 [6] は 1977 年に Rivest, Shamir, Adleman らが発明した公開鍵暗号である．RSA 暗号は非常に大きな整数に対する素因数分解問題の計算困難性に基づいて構成されている．

ここでは RSA 暗号の鍵生成と通信の秘匿, 電子署名の基本的な構成を述べる．すなわち下記アルゴリズム単体では脆弱であり, 暗号モジュールとして用いる際は他の関数と組み合わせ FIPS140-2 [7] で定義される暗号学的に安全なアルゴリズムを構成するべきであることには留意したい．

- 鍵生成
  p, q を相違なる素数,$n = pq$ とする．
  整数 e を選び,$de \equiv 1(mod(p-1)(q-1))$ となる整数 d を求める．
  公開鍵は (n, e), 秘密鍵は d となる．
- 通信の秘匿
  1. 暗号化
     公開鍵 e, メッセージ m に対し
     $E(e, m) := m^e mod n$
  2. 復号
     秘密鍵 d, 暗号文 c に対し
     $D(d, c) := c^d mod n$
- 電子署名 H をハッシュ関数とする．
  1. 署名生成
     秘密鍵 d, メッセージ m に対し
     $S(d, m) = H(m)^d mod n$
  2. 署名検証
     公開鍵 e, メッセージ m, 署名 s に対し
     $H(m) \equiv s^e mod n$ が成立すれば受理, そうでないなら棄却

2.3 楕円曲線暗号

楕円曲線暗号は 1985 年に Miller [8] と Koblitz [9] が独立に発明した楕円曲線を利用した公開鍵暗号である．他の公開鍵暗号に比べ, 短い鍵長で安全性を確保することができるため IC カードや組み込み機器など計算能力やメモリが制限されている環境を中心に利用されている．

楕円曲線暗号で使われる楕円曲線は, 素体 $GF(p)$ や 2 の拡大体 $GF(2^n)$ 上で定義されたものが多く用いられている．特に $GF(2^n)$ では加算や乗算が XOR とシフト演算によって構成できるため, ハードウェア上での実装に有利である．ここでは $GF(2^n)$ 上で定義される楕円曲線について述べる．

2.3.1 楕円曲線の定義

$GF(2^n)$ 上で定義される楕円曲線は　で表される．

$$y^2 + xy = x^3 + ax^2 + b$$

ここで,$a, b \in GF(2^n)$ である．上式を満たす $(x, y) \in GF(2^n) \times GF(2^n)$ を楕円曲線上の有理点と呼び, この有理点の演算により楕円曲線暗号は構成される．

2.3.2 有理点の加算

$P(x_1, y_1)$,$Q(x_2, y_2) \in GF(2^n) \times GF(2^n)$ の加算は次式のように表される．

$P \neq Q$ のとき

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a \\ y_3 &= \lambda(x_1 + x_3) + x_3 + y_1 \\ \lambda &= (y_1 + y_2)/(x_1 + x_2) \end{aligned}$$

$P = Q$ のとき

$$\begin{aligned} x_3 &= \lambda^2 + \lambda + a \\ y_3 &= \lambda(x_1 + x_3) + x_3 + y_1 \\ \lambda &= (x_1 + y_1)/x_1 \end{aligned}$$

また, 加算を繰り返すことで有理点のスカラー倍算 $Q = kP$ を定義することができる．

$$Q = kP = (...(((P + P) + P) + P)... + P)$$

2.3.3 楕円曲線上の離散対数問題

楕円曲線において,k と点 P から点 Q を求めることは容易であるが, 点 Q と点 P から k を求めることは k を大きくとると現実的な時間では求めることができない．これは楕円曲線上の離散対数問題と呼ばれ, 楕円曲線暗号はこの計算困難性に基づいて構成されている．

2.3.4 楕円曲線暗号の構成

楕円曲線暗号の鍵生成と通信の秘匿, 電子署名の基本的な構成を述べる．楕円曲線暗号に用いられる楕円曲線は, パラメタ次第で特殊な攻撃法があり大きく安全度を低くしてしまう可能性がある．そのため米国立標準技術研究所 (NIST:National Institute of Standards and Technology) が定めたパラメタ [10] などを用いる必要がある．ここで楕円曲線のパラメータと基準点となる有理点 P, 点 P の位数 l は公開されているパラメータである．

- 鍵生成 $d_A \in_R [1, l-1]$
  $P_A = d_A P$
  $d_A \in_R [1, l-1]$ は 1 から $l-1$ の範囲の乱数を生成することを表す．
  点 P_A が公開鍵,d_A が秘密鍵となる．また, 楕円曲線のパラメータと基準点となる有理点 P, 点 P の位数 l は公開パラメータである．
- 通信の秘匿
  1. 暗号化
     メッセージ M, 乱数 r に対して
     $E(m) := (rP, M + rP_A)$

2. 復号
   暗号文 c_1, c_2 に対して
   $D(c_1, c_2) := c_2 - d_A c_1$

- 電子署名
  1. 署名生成
     $r \in_R [1, l-1]$
     $U = rP = (x_U, y_U)$
     $H = Hash(m)$
     $u = x_U \pmod l$
     $v = r^{-1}(H + ud_A) \pmod l$
     $Hash(m)$ はメッセージ m に対してハッシュを生成することを表す.
     (u, v) が署名となる.
  2. 署名検証
     $s_1 = v^{-1} H \pmod l$
     $s_2 = v^{-1} u \pmod l$
     $V = s_1 P + s_2 P_A = (x_V, y_V)$
     $u = x_V \pmod l$ なら署名は受理される.

2.4 RSA 暗号と楕円曲線暗号

公開鍵暗号アルゴリズムとして RSA 暗号は広く普及してきたが, 近年 RSA 暗号の実装上の懸念材料が顕著になってきており, 楕円曲線暗号の暗号学的安全性の評価も相まって楕円曲線暗号が RSA 暗号に取って代わりつつある [11]. たとえば国家安全保障局 (NSA:National Security Agency) の機密情報の保護のために用いる暗号アルゴリズムのリストの SuiteB では RSA 暗号はリストに入っていない [12].

RSA の問題点として代表的なものが安全性を確保するための鍵長増加, 鍵の運用方法である.

2.4.1 鍵長

公開鍵暗号は前述したように計算量的な困難性に基づいて構成される. そのためムーア則により計算機性能が向上することに伴い, 公開鍵暗号の安全な運用に必要な鍵長は大きくなっている.

NIST によると 2030 年頃までの利用を想定する場合,RSA 暗号の鍵長は 2048bit 必要である. RSA と楕円曲線暗号で同じセキュリティ強度を実現するために必要な鍵長を Table1 に示す [3].

Table.1 から楕円曲線暗号では 2048bit の RSA 暗号と同程度の安全性を約 10 分の 1 の鍵長で保障することができることがわかる.

また, 素因数分解は一般数体篩法 [13] という準指数時間のアルゴリズムが発見されている一方, 楕円曲線上の離散対数問題には現在のところ指数時間のアルゴリズムしか発見されていない. そのため今後, より高い安全性が必要となったとき RSA 暗号は楕円曲線暗号に比べより大きく鍵長を倍増させる必要がある.

2.4.2 鍵の運用方法

RSA 暗号では鍵ペアの生成に 2 つの素数を必要とするが, それら素数の生成・運用方法を適切に行わなかった場合, 脆弱な鍵ペアを生成してしまう. すなわち 2 つの公開鍵が 1 つでも同じ秘密鍵を用いて生成されていた場合, それらをユークリッド互除法により最大公約数を求めることができ, 公開鍵から秘密鍵を多項式時間で計算することができる.

Table1　アルゴリズムの鍵長比較 (bit)

RSA	楕円曲線暗号
1024	160～223
2048	224～255
3072	256～383

RSA 暗号では仮に一方の秘密鍵に重複があっても公開鍵は異なるものとなるため, 脆弱な鍵であるか判別が難しく管理者による運用が難しい. 実際にインターネット上で利用されている RSA 公開鍵のうち, そのような脆弱な鍵が少数存在していることが確認されており [14, 15], このことが RSA 暗号の運用上のリスクとなっている.

一方, 楕円曲線暗号では秘密鍵の生成に 1 つの乱数を用いるだけでよく, また仮に秘密鍵に重複が生じた場合, 公開鍵は同一のものとなるため脆弱な鍵の判別は容易である. このような点で楕円曲線暗号の鍵運用は RSA に比較して軽減されている [14].

3 セキュアプロセッサ

3.1 セキュアプロセッサの概要

セキュアプロセッサとは OS やハイパーバイザなどの強い権限をもつソフトウェアやハードウェアタンパから, アクセス制御・暗号化・完全性検証などをハードウェアが行うことで特定のプロセスを保護することを目的とするプロセッサの総称である. セキュアプロセッサは想定する脅威モデル, すなわち「誰から何を保護するのか」によってセキュリティ要件が異なり, それに応じたセキュリティ機能を有する. セキュアプロセッサが持つセキュリティ機能は Fig.1 に示すような 4 つのレイヤーに分類することができる. 内側から暗号化機構, アクセス制御機構, プラットフォーム検証機構, 耐ハードウェアタンパ機構となっており, 外側のレイヤーほどセキュリティ要件が高くなっている.

3.1.1 暗号化機構

もっとも内側のレイヤーに位置する暗号化機構は外側のレイヤーを実現するための基本機能であり, 共通鍵暗号, ハッシュ関数, 乱数生成といった暗号アルゴリズムの実装が含まれる.

共通鍵暗号は主にプロセッサ外部のメモリやストレージに暗号化して書き出すときに用いられる. 代表的なアルゴリズムとして DES や AES がある.

ハッシュ関数は主にプロセッサ外部のメモリが改ざんされていないことを検証する完全性検証や電子署名を行うためのメモリの要約値を生成するために用いられる. 代表的なアルゴリズムには MD5 や SHA がある.

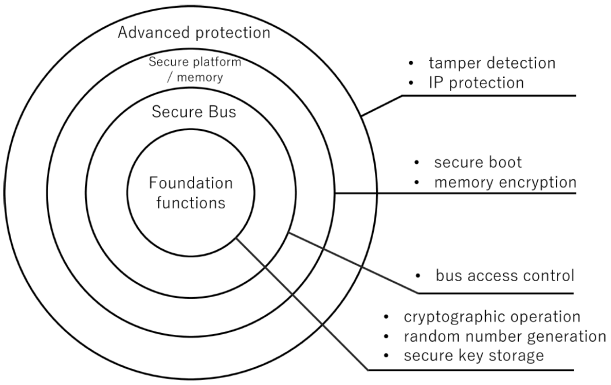

Fig.1　セキュアプロセッサの機能

乱数生成は主に共通鍵暗号の鍵生成や公開鍵暗号の暗号化・電子署名で用いられる．暗号における乱数生成器は乱数のばらつきだけでなく乱数列の傾向が推測できないことが重要である．そのため, セキュアプロセッサでは半導体製造時の素子遅延特性のばらつきを用いたデバイス固有の乱数生成器である PUF [16] や, 素因数分解の計算困難問題への帰着をもつ疑似乱数生成アルゴリズムである Blum-Blum-Shub [17] などが用いられる．

3.1.2 アクセス制御機構

プロセッサで動作する他のユーザプロセスや特権プロセスが信頼できない脅威モデルにおいては, プロセスのデータの平文での取得・改ざんが行われる危険性がある．そのようなモデルでプロセスを保護する機構がアクセス制御である．

たとえば広く普及している IA-32 アーキテクチャでは,OS が管理するページテーブルには他のユーザプロセスからのアクセスを制御するフラグが存在し, ユーザプロセスによる他のプロセス空間への不正アクセスは防ぐことができる．しかしながら, このアーキテクチャでは OS やハイパーバイザなどの特権プロセスを信頼したモデルを想定しているため, それら特権プロセスからの不正アクセスは防ぐことができない．

特権プロセスをも信頼できない脅威モデルを想定しているセキュアプロセッサとしては XOM [18],L-MSP [19],ARM TrustZone [20] があげられる．

XOM や L-MSP で実行されるプロセスはプロセスごとに一意な識別子が与えられタグ付けされており, キャッシュラインにはプロセスのタグの識別ビットが追加されている．プロセスがキャッシュへのアクセスをするたびにタグの比較を行うことで, アクセス可能なプロセスを制御している．

ARM TrustZone では normal world と secure world の 2 つの実行モードが用意されており,normal world では Android などのリッチな OS が動作し secure world では Trusted OS という小さなカーネルが動作する．ARM TrustZone において Trusted OS は脆弱性のない信頼できる OS としている．これらの動作モードはメモリ空間が分離されており,normal world のプロセスから secure world へのメモリ空間へアクセスができないように制御が行われている．

3.1.3 プラットフォーム検証機構

セキュアなシステムはセキュアなプラットフォーム上で動作していることを前提としており, プラットフォームそのものに改ざんが加えられていた場合, システムのセキュリティは無効化されてしまう可能性がある．たとえば OS を信頼している脅威モデルであれば OS そのものに改ざんが加えられていないか検証を行う必要がある．そのような検証を行うのがプラットフォーム検証機構である．

プラットフォーム検証は, システム起動時に最初にロードするソフトウェアイメージに電子署名をつけて外部サーバによる認証を行い完全性検証を行ったり, プロセッサや SoC 内部の改ざんが困難な領域にソフトウェアイメージを保存しておく手法がある．前者の手法ではセキュアプロセッサと外部サーバとの連携動作を行うために,3.2 で詳述する公開鍵暗号機構を必要とする．

3.1.4 耐ハードウェアタンパ機構

ハードウェアへのアクセスと高価な機材を保持している攻撃者であれば, プロセッサ外部のバスにプローブを当てることでデータを読み取るタッピング [21] や,DRAM を冷却し電荷減衰を遅くすることで RAM 上に残存するデータを読み取るコールドブートアタック [22] などのハードウェアタンパが可能である．このようなハードウェアタンパを検知・防止するものが耐ハードウェアタンパ機構である．

攻撃者がプロセッサ外部のハードウェアへのアクセスが可能で, プロセッサ外部のハードウェアを信頼できない脅威モデルでは, メモリの暗号化・完全性検証によってプロセスを保護する手法がある．メモリの暗号化・完全性検証を行うセキュアプロセッサとして AEGIS [23] と Intel SGX [24] があげられる．AEGIS ではデータがプロセッサから外部に出るとき, すなわちキャッシュから追い出されメインメモリに書き出されるタイミングでデータはプロセス固有の鍵で共通鍵暗号によって暗号化される．メインメモリからデータを読み込む際は同じくプロセス固有の鍵によって復号される．また, メモリ全体のイメージはツリー状にハッシュが計算されそのルートはプロセッサ内部で管理される．ハッシュの一致を比較することでメモリがプロセッサ外部による変更が行われた場合, プロセッサはそれを検知できる．メモリの暗号化によって攻撃者からのメモリへの平文のアクセスを防ぎ, 完全性検証によって攻撃者からのメモリへの改ざんの検知を行うことができる．

またハードウェアタンパとして, データそのものではなくデータを扱うときのプロセッサの挙動やメモリへのアクセスパターンを観測, 解析を行うことでデータの取得を試みるサイドチャネル攻撃がある．サイドチャネル攻撃にはプロセッサのキャッシュへのアクセスタイミングを利用する PRIME+PROBE 攻撃 [25]

や FLUSH+RELOAD 攻撃 [26] , プロセッサ動作時の消費電力を利用する攻撃 [27] があげられる．このようなサイドチャネル攻撃を防ぐことのできるプロセッサとして Ascend [28] が提案されている．Ascend は ObliviousRAM(ORAM) [29] を実装することでアクセスパターンを隠蔽し, ダミー命令を実行することでプロセッサの挙動を隠蔽する．

また, 完全準同型暗号 [30] を用いたセキュアプロセッサであればプロセッサにでさえデータの平文を渡さないため, いかなるハードウェアタンパに対しても情報漏えいは起こりえないが, 現在の完全準同型暗号ではオーバーヘッドが非常に大きく実用的ではない [31].

3.2 セキュアプロセッサにおける公開鍵暗号

セキュアプロセッサにおける公開鍵暗号の役割はプロセッサ外部との通信を前提とするものであり, プロセスの保護機能を表した Fig.1 とは別のセキュリティ機能である．公開鍵暗号の秘密鍵はプロセッサ固有のものでプロセッサ内に埋め込まれており耐ハードウェアタンパ性を持つとする．また, 対となる公開鍵にはプロセッサメーカによる電子署名がされている．これは SSL 通信の認証局に類似した仕組みであり, この場合の Root of Trust はプロセッサメーカとする．

このような公開鍵暗号をセキュアプロセッサに組み込むことで, プロセッサと外部モジュールとの認証, 遠隔地の計算環境への機密情報のアップロード, さらにはプラットフォーム検証やソフトウェア認証を行うことができる．ここではそれらの手法を詳述する．

3.2.1 遠隔地の計算環境への機密情報のアップロード

IaaS に代表されるクラウドコンピューティングなどの, 遠隔地で動作しているプロセッサにおける脅威モデルを考える．IaaS ではクラウドプロバイダーが管理する計算資源上でクラウドプロバイダーが管理するハイパーバイザが動作し, その上で動作する仮想マシン (VM) がユーザに提供される．そのため IaaS ではユーザ VM はクラウドプロバイダーにデータを取得・改ざんされるリスクがある．また, プロセッサ上で動作するほかのユーザ VM も信頼することはできない．そのため, この脅威モデルではセキュアプロセッサ以外を信頼しない．

このモデルで機密情報を扱うためには, セキュアプロセッサは 3 章で述べたアクセス制御機構と耐ハードウェアタンパ性の機構が必要である．しかしそれらの機構だけでは, クラウドに機密情報を安全にアップロードすることができない．

クラウドに機密情報を安全にアップロードするためには,SSL 通信があげられるが SSL 通信はクラウドプロバイダの公開鍵を用いるためこの脅威モデルでは不適切である．そのためセキュアプロセッサの公開鍵を用いて機密情報をアップロードする手法があげられる．暗号化された機密情報はセキュアプロセッサ上でのみ復号されるため安全にアップロードされる．

3.2.2 プロセッサと外部モジュールとの認証

セキュアプロセッサが別のセキュアプロセッサなどの外部モジュールと連携動作することを考える．このときセキュアプロセッサと外部モジュールはお互いを信頼するために認証を行う必要がある．梶原は SSL プロトコルを修正したプロセッサ間の認証手順を提案し, プロトコルの安全性を検証している [32]．この手法はプロセッサに埋め込まれた秘密鍵を利用する手法である．

3.2.3 プラットフォーム検証

プラットフォーム検証は 3.1.3 節で述べたように, プロセッサ上で動作しているプラットフォームのイメージを正しいイメージと比較して改ざんが行われていないか検証する機構である．正しいイメージをプロセッサや SoC 内部の改ざんが困難な領域に保存する手法では, 出荷後にシステムに全くの変更が加えられないため汎用性は高くない．そのため Trusted Platform Module(TPM) [1] で用いられている外部サーバとの連携動作による完全性検証の手法があげられる．

TPM は Trusted Computing Group によって策定されているセキュアなコプロセッサであり,Trusted Boot と呼ばれるプラットフォームの完全性検証の機能を持っている．Trusted Boot ではシステム起動時の BIOS から OS に至るまでソフトウェアイメージのハッシュ値を連鎖的にハッシュ計測していき, 最終的に得られたハッシュ値に TPM が署名を行い, 外部サーバが認証を行うことでプラットフォームの完全性を検証することができる

3.2.4 ソフトウェア認証

ソフトウェア認証はプロセッサ上で実行するソフトウェアイメージの完全性を検証する機構である．これはソフトウェアを起動する際にソフトウェアのメモリイメージ全体のハッシュを計測しプロセッサが署名を行い, 外部サーバが認証することで達成される．

プラットフォーム検証に近い機構であるが, プラットフォーム検証はシステム起動時に一度実行されるだけであるのに対して, ソフトウェア認証は要求があればソフトウェアを新しく起動する度に実行される．実装例として Intel SGX [2] があげられる．

4 公開鍵暗号のハードウェア実装と評価

4.1 実装と評価の概要

セキュアプロセッサにおいて公開鍵暗号は 3.2 節で述べたように, プロセッサ外部との通信路の確立やプラットフォーム検証, ソフトウェア認証などがあげられるが, いずれの用途においても公開鍵暗号はプロセッサ固有の秘密鍵を用いた電子署名の生成である．

プロセッサの秘密鍵を漏洩させないために, これらの用途における公開鍵暗号の使用は通常のプロセス空間から隔離されている必要があり, 特殊命令で実行される．また, 公開鍵暗号は比較的負荷がかかる計算であるため, プラットフォーム認証のような起動時に一度だけ実行されるものであれば許容できるが, ソフトウェア認証など

が頻繁に要求される場面ではプロセッサの性能低下を引き起こすことから公開鍵暗号の計算は暗号モジュールにオフロードされることが望ましい．

既存研究におけるセキュアプロセッサ上での公開鍵暗号のハードウェア実装は Intel SGX での 3072bit の RSA 暗号 [2] のみであり, 楕円曲線暗号を実装したセキュアプロセッサはなかった．RSA 暗号ではなく楕円曲線暗号を実装することは鍵長の短さからセキュアプロセッサ全体の性能向上につながることが期待できる．これらのことから本研究ではプロセッサに, 楕円曲線暗号の電子署名アルゴリズムである ECDSA と RSA 署名それぞれの生成モジュールを追加実装し評価を行う．

実装のベースとしたプロセッサはオープンソースのプロセッサである RocketCore である．RocketCore は ISA として 64bit の RISC-V が用いたシングルイシューのインオーダ 5 段パイプラインのプロセッサである．この RocketCore にペリフェラルの制御モジュールを追加し,FPGA 上で動作できる lowRISC が公開されており, そこに暗号モジュールを追加する形で実装をし, 計算速度と回路面積によって評価を行った．

4.2 ECDSA 生成モジュール

ECDSA には sect233r1 の楕円曲線を用いた．sect233r1 は $GF(2^{233})$ 上で定義される楕円曲線であり,2048bit の RSA 暗号に相当するセキュリティ強度を持っている．楕円曲線暗号の実装において必要となるモジュールは, 楕円曲線上の演算, モジュロ演算, ハッシュ関数, 乱数生成器である．これらの実装について詳述していく．

楕円曲線上の演算

ECDSA における計算の多くは楕円曲線上の演算である．その中でも楕円曲線上の除算と乗算は計算量が多く, それらの計算回数を減らす様々なアルゴリズムが開発されている．本実装は Rebeiro ら [33] の実装を参考にしており, 除算には Itoh-Tsujii [34] のアルゴリズム, 乗算には Karatsuba 法 [35] を用いている．

モジュロ演算

ECDSA においては剰余演算と積の剰余演算を要する．単なる剰余演算には拡張バイナリ GCD 法 [36] が, 積の剰余演算にはモンゴメリ乗算 [37] が広く用いられており, これらのアルゴリズムはハードウェアが不得手とする割り算をシフト演算に置き換えることができる．本実装では Kaihara らによる拡張バイナリ GCD 法とモンゴメリ乗算を共通リソースを用いて計算することができるアルゴリズム [38] を用いた．

ハッシュ関数

ハッシュ関数はメッセージを楕円曲線上で扱える bit 数, すなわち本実装においては 233bit にするために要する．最終的な実装には SHA-224 や SHA-256 を想定しているが, 本実装においてはメッセージの上位 233bit をとる脆弱な実装をとっている．

乱数生成器

乱数生成器には LFSR(Linear Feedback Shift Register) という比較的単純な疑似乱数生成器を用いている．LFSR はセキュアな疑似乱数生成器ではないため, 最終的な実装には暗号論的疑似乱数生成器を想定している．

4.3 RSA 署名生成モジュール

RSA 署名の生成モジュールには冪指数 3 で鍵長 2048bit の RSA を用いた．これは上述の sect233r1 の楕円曲線に相当するセキュリティ強度を持っている．RSA 署名の本実装では 2.2 節で述べた簡易なアルゴリズムを採用する．本来 RSA PKCS#1 v1.5 などの署名アルゴリズムで行うパディング処理などを省略しているため脆弱なアルゴリズムとなっていることに留意したい．そのため本実装において必要となるモジュールは, 冪剰余演算, ハッシュ関数である．

冪剰余演算

冪剰余演算には広く用いられているバイナリ冪剰余法 [39] によって実装した．バイナリ冪剰余法の内部ではモンゴメリ乗算が呼び出されている．

ハッシュ関数

ハッシュ関数は前述したように最終的な実装には SHA-224 や SHA-256 などのモジュールの実装を想定しているが, 本実装においてはメッセージの一部を切り取るような実装となっている．

4.3.1 速度評価

計算速度は論理レベルのシミュレーションにおけるサイクル数によって評価を行う．それぞれの電子署名の計算時間は Table2 となった．本実装は FPGA 上で 100MHz で動作させることを目標としているため, それに基づいた計算時間を算出している．

Table2　ソフトウェアによる電子署名の実行時間 (msec)

	RSA 署名	ECDSA
サイクル数	3700000	17000
計算時間	$37ms$	$170\mu s$

4.3.2 面積評価

回路面積は verilog による回路記述を論理合成･配置配線したあとの LUT や BRAM などの FPGA 上の回路資源によって評価を行う．Fig.2 はベースである lowRISC プロセッサとそれに ECDSA,RSA 署名をそれぞれ追加実装した場合の回路面積の比較である．

4.3.3 評価のまとめ

回路面積は RSA 署名と ECDSA ともにプロセッサの半分程度の回路面積となった．本実装におけるプロセッサは前述のとおり, シングルイシューのインオーダ 5 段パイプラインといった比較的軽量なプロセッサであり,

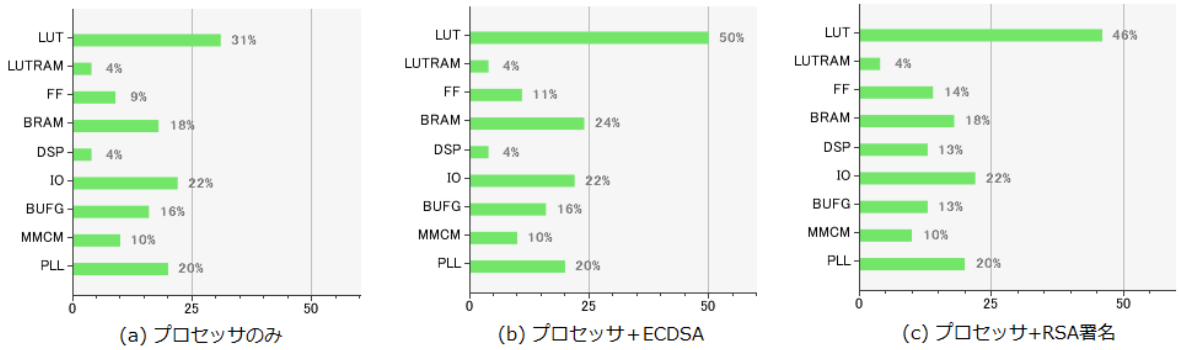

Fig.2　面積評価

スーパースカラやアウトオブオーダーなど, より性能を重視したプロセッサを用いた場合には署名モジュールが占める面積はより低くなると考えられる.

計算速度においては ECDSA が RSA 署名より著しく高速に計算できている. 回路面積も RSA 署名と ECDSA にはあまり差がないため, 署名生成が多く要求されるセキュアプロセッサにおいては ECDSA による実装が有利となることがわかった.

これら計算速度や回路面積は基本的にトレードオフの関係にあるため, 計算時間に余裕があるならば回路面積をより小さくすることができ, 一方より高速な演算が求められるなら回路面積を犠牲にすることで署名の高速化をすることが可能である. そのためセキュアプロセッサにおける公開鍵暗号において, ハードウェア実装のアルゴリズムやパラメータをチューニングすることは今後の課題である.

5 おわりに

5.1 まとめ

本論文ではセキュアプロセッサにおける公開鍵暗号の用途を考察し, 署名生成の実行頻度が高いことを論じた. 署名生成に適した公開鍵暗号アルゴリズムを模索するため, 実際にプロセッサ上に追加実装する形で RSA 署名と ECDSA のモジュールをハードウェア実装し, 評価を行った. その結果, シングルイシューのインオーダ 5 段パイプラインといった比較的軽量なプロセッサの半分程度の面積で RSA 署名,ECDSA モジュールそれぞれが実装できることがわかった. また, その実装においては RSA 署名と比較して ECDSA が高速に計算を行えることがわかった.

5.2 今後の課題

今後の課題のひとつとして, ハードウェア実装において回路面積と実行速度は基本的にはトレードオフの関係であるので, セキュアプロセッサの用途に即したハードウェア実装のアルゴリズムやパラメータの精査があげられる.

また, プラットフォーム認証やソフトウェア認証などはセキュアプロセッサが計測したハッシュ・署名を認証する外部のサーバが必要であり, そのような認証サーバを実装して認証システムを構成して評価することも課題である.

謝辞

本論文の研究の一部は，公益財団法人セコム科学技術振興財団の助成を受けたものである.

参考文献

[1] Trusted Computing Group. TPM main specification. *Technical report*, 2013.
[2] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, Vol. 2016, p. 86, 2016.
[3] Elaine Barker and Allen Roginsky. Transitions: Recommendation for transitioning the use of cryptographic algorithms and key lengths. *NIST Special Publication*, Vol. 800, p. 131A, 2011.
[4] Vipul Gupta, Sumit Gupta, Sheueling Chang, and Douglas Stebila. Performance analysis of elliptic curve cryptography for ssl. In *Proceedings of the 1st ACM workshop on Wireless security*, pp. 87–94. ACM, 2002.
[5] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, Vol. 22, No. 6, pp. 644–654, 1976.
[6] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, Vol. 21, No. 2, pp. 120–126, 1978.
[7] Entrust Authority Security Kernel. Fips 140-2 (level 2) cryptographic module security policy. 2016.
[8] Victor Miller. Use of elliptic curves in cryptography. Advances in Cryptology—CRYPTO' 85 Proceedings, pp. 417–426. Springer, 1986.
[9] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of computation*, Vol. 48, No. 177, pp. 203–209, 1987.
[10] NIST. Recommended elliptic curves for federal goverment use. July 1999.
[11] 清藤武暢, 四方順司. 公開鍵暗号を巡る新しい動き: Rsa から楕円曲線暗号へ. 金融研究, Vol. 32, No. 3, pp. 17–50, 2013.

[12] NSA. Suite b cryptography. 2005.
[13] Arjen K Lenstra, Hendrik W Lenstra Jr, Mark S Manasse, and John M Pollard. The number field sieve. In *The development of the number field sieve*, pp. 11–42. Springer, 1993.
[14] Arjen Lenstra, James P Hughes, Maxime Augier, Joppe Willem Bos, Thorsten Kleinjung, and Christophe Wachter. Ron was wrong, whit is right. Technical report, IACR, 2012.
[15] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *USENIX Security Symposium*, Vol. 8, 2012.
[16] G Edward Suh and Srinivas Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th annual Design Automation Conference*, pp. 9–14. ACM, 2007.
[17] Lenore Blum, Manuel Blum, and Mike Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on computing*, Vol. 15, No. 2, pp. 364–383, 1986.
[18] David Lie, Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. Architectural support for copy and tamper resistant software. *ACM SIGPLAN Notices*, Vol. 35, No. 11, pp. 168–177, 2000.
[19] 橋本幹夫, 春木洋美, 川端健. オープンソース os と共存可能なセキュリティプロセッサ技術. 東芝レビュー, Vol. Vol.60, No.6, pp. 44–47, 2005.
[20] Tiago Alves. Trustzone: Integrated hardware and software security. *White Paper*, 2004.
[21] Andrew Huang. Keeping secrets in hardware: The microsoft xboxtm case study. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 213–227. Springer, 2002.
[22] J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM*, Vol. 52, No. 5, pp. 91–98, 2009.
[23] G.Edward Suh, Dwaine Clarke, Blaise Gassend, Marten van Dijk, and Srinivas Devadas. AEGIS: Architecutre for tamperevident and tamper-resistant processing. *In International Conference on Supercomputing*, 2003.
[24] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. In *HASP@ ISCA*, p. 10, 2013.
[25] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of aes. Cryptographers' Track at the RSA Conference, pp. 1–20. Springer, 2006.
[26] Yuval Yarom and Katrina Falkner. Flush+ reload: a high resolution, low noise, l3 cache side-channel attack. In *23rd USENIX Security Symposium (USENIX Security 14)*, pp. 719–732, 2014.
[27] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, Vol. 1, No. 1, pp. 5–27, 2011.
[28] Christopher W Fletcher, Marten van Dijk, and Srinivas Devadas. A secure processor architecture for encrypted computation on untrusted programs. In *Proceedings of the seventh ACM workshop on Scalable trusted computing*, pp. 3–8. ACM, 2012.
[29] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, Vol. 43, No. 3, pp. 431–473, 1996.
[30] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.
[31] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, pp. 113–124. ACM, 2011.
[32] 梶原拓也. セキュアプロセッサを用いたマルチプロセッサシステム の構成. 2017.
[33] Chester Rebeiro and Debdeep Mukhopadhyay. High speed compact elliptic curve cryptoprocessor for fpga platforms. In *Progress in Cryptology-INDOCRYPT 2008*, pp. 376–388. Springer, 2008.
[34] Toshiya Itoh and Shigeo Tsujii. A fast algorithm for computing multiplicative inverses in gf (2m) using normal bases. *Information and computation*, Vol. 78, No. 3, pp. 171–177, 1988.
[35] André Weimerskirch and Christof Paar. Generalizations of the karatsuba algorithm for efficient implementations. *IACR Cryptology ePrint Archive*, Vol. 2006, p. 224, 2006.
[36] D.E. Knuth. *The art of computer programming: Seminumerical Alogirithms*, Vol. 2. Addison Wesley, 1998.
[37] Peter L Montgomery. Modular multiplication without trial division. *Mathematics of computation*, Vol. 44, No. 170, pp. 519–521, 1985.
[38] Marcelo E.Kaihara and Naofumi Takagi. A hardware algorithm for modular multiplication/division. *IEEE Transactions on Computers*, Vol. 54, p. 12, 2005.
[39] Donald E. Knuth. *The Art of Computer Programming*, Vol. Volume 2 of *Seminumerical Algorithms*. Addison-Wesley Professional.