Cortex-M デバイスのメモリ領域を考慮したマルチスカラ倍算の事前計算法の提案

飯島 悠介[†1]　永井 彰[†1]　田村 桜子[†1]　安田 幹[†1]

概要：今後，IoT 機器の普及に従い，データに対する柔軟なアクセス制御と高い秘匿性が求められる．本稿では，IoT 機器の汎用例として，Cortex-M を利用したデバイスを利用し，関数型暗号を実装する際の高速化手法を検討する．特に，関数型暗号では暗号化に楕円スカラ倍算を用いていることに着目し，事前計算テーブルを用いたマルチスカラ倍算による高速化の実装手法を検討する．IoT 機器では，PC などと比較して，メモリ量が少なく数 Kbyte しかないが，ROM 領域は数 Mbyte 確保されることを利用し，マルチスカラ倍算の事前計算テーブルを ROM に実装することで，マルチスカラ倍算の高速化を行う手法を提案し，実際に実装評価を行うことで，ROM を活用した事前計算テーブルを用いた高速化実装が有効であることを確かめた．

キーワード：IoT，マルチスカラ倍算，ARM Cortex-M，関数型暗号

On the Implementation of Multiple Scalar Multiplication Using ROM Region of ARM Cortex-M

Yusuke Iijima[†1*]　Akira Nagai[†1]　Sakurako Tamura[†1]　Kan Yasuda[†1]

Abstract: In few year, more varions IoT devices appear, and IoT devices also can utilize date sensing from another IoT devices. Thousands of IoT devices are connect to internet and IoT devices are send or received impotant data such as personal information or confidential information and so on. Therefore,any data send from IoTdevices are prevented leakage, and realized a frexible access control.The Functional encryption achieve above contributions. In this paper, we propose that an implementation method of improving processing speed on ARM Cortex-M. We implimemt a precomputation method proposed by Lim and Lee using ROM region. IoT devices have a few RAM region but a lot of ROM region (order Mbyte). We prove that improving calurate speed of IoT device for using saved precompution with large byte into ROM region.

Keywords: IoT, multi scalar multiplication, ARM Corete-M, functional encryption

1. はじめに

2020 年までにインターネットに接続されるモノは，およそ 403 億台に達すると予測されており [1]，今後多種多様な IoT (Internet of Things) 機器が市場に出回り，各機器が取得するセンサデータの種類も拡大すると予測できる．今後様々な機器がインターネットにつながること を想定すると，取得したセンサデータの見える化やデータ解析を行うだけでなく，IoT 機器同士がセンサデータを授受し，IoT 機器が他の IoT 機器を制御するなど，IoT 機器同士の連携も可能となる．実際に特定のスマートフォンからメーカの異なる家電の動作の制御を実施したり[3]，タクシーと自動販売機間での センサデータのやり取りを行うなど，異業種間でのデータ活用方法の検討 [4] もはじめられている．一方で，各 IoT 機器間でやり取りするデータには，生体情報や位置情報などの個人に紐付く情報または企業の機微な情報に 関連するデータも増えていくことが予測される． 実際に[2]では，産業用 IoT 機器がセキュリティ上重要な個人データを大量に生成し，他の IoT 機器とデータをやり取りしあうことが示されている．IoT システムでは産業以外でも，同様の事象が起こりうることが示唆される．さらに．そのようなデータの処理は．GDPR などの情報保護に関する法律やガイドラインによって制限される場合がある．そのため，IoT 機器間でやり取りされる機密データを不正アクセスカラ保護するためのデータの暗号化や，効率的で柔軟なアクセス制御メカニズムが必要となる．従来の共通鍵や公開鍵を用いて実現する場合には，暗号化や復号に利用する鍵を IoT 機器ごとに共有し，各 IoT 機器でそれら鍵の管理が必要となる．しかし，1 つの IoT システムに対して，システム内で管理されている IoT 機器が数千単位で存在していることが珍しくない．すなわち，機器の半数がやり取りしようとしても，数百単位の鍵を各 IoT 機器で管理する必要が生じることとなり，鍵管理が煩雑になる

†1　日本電信電話株式会社 NTT セキュアプラットフォーム研究所，〒180- 8585 東京都武蔵野市緑町 3-9-11,
NTT Secure Platform Laboratories, NTT Corporation, 3-9-11, Midoricho, Musashino-shi, Tokyo 180-8585, Japan.

* yuusuke.iijima.he@hco.ntt.co.jp

ことが予想される．

それらに対する，暗号を用いた解決方法として，関数型暗号[7]を用いたシステムの構築が挙げられる[5,6]．関数型暗号とは，公開鍵暗号をより高度にした暗号方式であり，暗号文，属性鍵に属性情報や条件式を導入することで，暗号/復号のロジックを規定している．[4]では，暗号化鍵に復号ロジックを埋め込む Key Policy の関数型暗号を用いたシステムを提案している．このシステムを利用すると，複数の IoT 機器と暗号化したセンサデータを共有する際に，IoT 機器は，それらのセンサデータが閲覧可能という情報を含んだロジックにより生成した復号鍵を 1 つ保有することで，膨大な IoT 機器における鍵の管理が容易になる．

一方で．IoT 機器の性能も様々であり，ARM Cortex-M のようなリソースの限られたデバイスも多く存在する．CPU 性能も PC 等の汎用コンピュータに比べて低く，PC での実装に比べ，低速での実装速度となるため，高速化していく必要がある．また，RAM は Kbyte 単位でしかないが，ROM は Mbyte 単位で保持している．

[6]では，データを活用する側は，ある程度リソースがあることを想定し，逆にデータを送信する側は，一般的な IoT 機器同様，リソースが限られたデバイスであることを想定し，フィージビリティの検証を行っていた．本稿でも同様の想定で，データ送信側，すなわちデータの暗号化を行うデバイスのリソースが限られていることを想定して，高速化を図る．関数型暗号の暗号化にはマルチスカラ倍算が多用されており，属性の増加や属性表現の複雑化によって．必要となるマルチスカラ倍算の数も増加する．

本稿では，関数型暗号の IoT 機器への応用を想定し，マイコンでマルチスカラ倍算の事前計算テーブルを用いた高速化手法の実装法を提案する．これは，関数型暗号の暗号化アルゴリズムにおいて，楕円点は公開パラメータとして，鍵生成サーバ等から配布されたものを用いるため，マスター鍵が変わらない限り，固定値として考えてよいためである．関数型暗号の暗号化に適用した場合，どの程度の高速化が見込めるか見積もりを行う．

2. 準備

本章では，本稿で用いる技術，機器の詳細説明を示す．

2.1 マルチスカラ倍算事前計算のアルゴリズム

スカラ倍算とは，整数 n に対し，楕円曲線上の点 P を n 個加算した $nP = P + P + \cdots + P$ の計算のことであ る．マルチスカラ倍算とは，スカラ倍算の組み合わ せ (例えば，楕円点 P, Q，整数 n, m に対して，$nP + mQ$) のことである．関数型暗号の暗号化処理ではこのマルチスカラ倍算が，主要な演算となる．よって，暗号化処理の高速化を実現するためには，マルチスカラ倍算の効率化を行うことが必要である．今回の実装では，事前計算テーブルの作成による高速化手法の一つである， Lim らが提案した方式を用いた．

スカラ倍算の Lim らの方式は， 位数をビット長とする楕円点 P' をある整数 h個のブロックに分割し，更に分割により生成されたh個の楕円点 P' を各々v ビットごとの組に分け，事前計算テーブル を生成する手法である．v は h を割り切るようにとる．

楕円点 P'に対する事前計算テーブルの生成は以下のように行う．

まず，楕円点 P' から h個の要素を持つ楕円点配列を生成する．ここで，$a = \lceil n/h \rceil$ である．

$$(P'_{h-1}, \ldots, P'_0) = \left(2^{(h-1)a} \cdot P', \ldots, P'\right)$$

更に v ビットごとに，P' を分割すると以下の楕円点 P' の 配列が $b = h/v$個できる．

$$P'[j] = \left(P'_{jv}, \ldots, P'_{jv+j-1}\right)$$

これらの配列を $e_i \in \{0,1\}$ に対して，

$$Ptbl[j][e_{v-1}2^{v-1} + \cdots + e_1 2^1 + e_0 2^0]$$
$$= e_{v-1}P'_{jv+v-1} + \cdots + e_1 P'_{jv+1} + e_0 P'_{jv}$$

を事前計算し，これらのテーブルを，$Ptbl[j][i]$ $(0 \le j < b,$ $0 \le i < 2^{v-1})$とする．

事前計算テーブル$Ptbl$ に対して，kP' は以下のように計算する．スカラ値 k を楕円点で利用したパラメータを用いて，以下のように分解する．

$$k = \sum_{i=0}^{h-1} 2^{ia}\, k^{(i)}$$

$$k^{(i)} = \sum_{j=0}^{a-1} 2^{ia+j}\, k_{ia+j}$$

k_i は，k の i ビット目とする． k_i に対する楕円加算として，スカラ倍値を以下のように，v ごとの組にする．

$$k'[j] = (k'_{jv}, \ldots, k'_{v+v-1})$$

次のアルゴリズムにより，スカラ倍算を求める．

```
1.  Input a, b, kij, Ptblij
2.  FOR j∈[0, …, b − 1]
3.    FOR i∈[0, …, a − 1]
4.      dsval[j][i] = Σ_{k=0}^{v−1}[2^k · k_{jv+k}]
5.  r = 0
6.  FOR j∈[a − 1, …, 0]
7.    FOR i∈[0, …, b]
8.      IF dsval[j][i] > 0
9.        r = r + Ptbl[j][dsval[j][i] − 1]
10. Output r
```

上記は，一回のスカラ倍算の事前計算テーブルを用いた，処理を示している．m 個の楕円点を足し合わせる場合には，各楕円点を h 個に分割した後，分割した後の全ての楕円点を一列に整列させて，それ以降の処理を行う．そして，マ

ルチスカラ倍算を行う際には，各スカラー値をビット列に変換し，これも全て一列に整列させて，スカラー値に戻したものを利用する．この一列に整列させる際には，事前計算で，並べる順番と，スカラー値を並べる順番を一致させなければ，正しい値を出力できない．

本実装では，事前計算テーブルの用いた演算の効率を重視し，事前計算テーブルの大きさを指定すると以下で定めるコスト[Cost]が最も小さくなるような, v, h の値をとる．ここで，ad_ratio は楕円点の加算演算と二倍演算にかかる時間の比率を示す．

$$cost = \left\lceil \frac{n}{h} \right\rceil \cdot \left(\frac{\mathrm{hm}}{\mathrm{v}} \right) \cdot \left(1 - \frac{1}{2^v} \right) + \left(\left\lceil \frac{n}{h} \right\rceil - 1 \right) \cdot [ad\_ratio]$$

2.2 ARM　Cortex-M デバイス

我々が，IoT 機器として想定している，ARM CortexM 系のデバイスについて述べる．まず，我々が ARM 社 の Cortex に着目した理由としては，世の中の組み込み 機器において CPU コアは様々存在するが, 2014 年 6 月 までに出荷実績は累計で 160 億プロセッサに達し，今後 IoT 機器に利用するチップとしてもシェアをとる事が予 測されるためである．“Cortex” とは, ARM 社の CPU ファミリーの名称であり, Cortex-A，Cortex-R，CortexM の三種類からなる. Cortex-A は 3 つの中で最も高性能な CPU であり，高度なオペレーションシステム (例えば LinuxOS) 向けのアプリケーションプロセッサである．一方，Cortex-M が 3 つの中では最も機能が限定されている CPU に位置づけられ，コストを重視した組み込み向けのプロセッサである．よって，IoT 機器として想定される機器としてはコスト面も考慮しリソースの限られた Cortex-M 系が市場に出回ると予測し, 今回の測 定ターゲットとした. Cortex-M には，M0，M0+，M1，M3，M4，M7，M22，M33 があり，それぞれアーキテクチャ(ARMv6-M，ARMv7-M，ARMv7E-M)や, クロック数や命令セットなどが異なる. 今回は, Cortex-M 系のデバイスの中で高性能な Cortex-M7F のアーキテクチャを搭載した市販の評価ボードを使用する．

3. 実装

3.1 実装内容

今回は，2.1 で紹介した，Lim らによる高速化手法を用いて生成した事前計算テーブルを，マイコンの ROM に保存し，マイコンは，事前計算テーブルを ROM から読み出して，マルチスカラ倍算を実行するプログラムの実装を行った．詳細を以下に記す．

まず, 事前計算テーブルの保存方法について述べる. Lim らの方法を用いて生成した事前計算テーブルは，各関数が読み出しをしやすいようにビット列に変換する．事前計算テーブル $Ptbl$ に対して, 各要素 $P_{j,i} = Ptbl[j][i]$ $(0 \le j < b, 0 \le i < 2^{v-1})$ は，楕円点であるため，普段は射影座標を用いて保存されるが，マイコンでの計算回数を減らすため，事前計算テーブルは，アフィン座標に加え，楕円加算演算で用いる値を事前に計算した結果を，1 つの要素に保存している．各要素は，事前計算値をビット列に変換し，並べたもので，構成される．具体的には図 2 のように事前計算のビット列を生成する． 生成した事前計算ビット列は，ROM に書き込む.

従来のライブラリの実装では，事前計算ビット列は，全て RAM に保存され，使用する際には，コピーし，利用する楕円点を事前計算ビット列から取り出し，楕円点演算を行うというものだった.

本実装では，書き込まれた ROM を読み込むにあたり，事前計算テーブルの読み出し方法次のように変えた.
事前計算テーブル内の座標を指定すると，指定された座標内の楕円点を出力する内部関数を用意した. 内部関数では，ROM に書き込まれており，その中で事前計算テーブルが定義されている．この内部関数は，要素の座標をと，RAM に保存される変数を引数で受け取ると，事前計算テーブル内の座標を検索し，楕円点を RAM に書きこむ.

アルゴリズム内の 9 行目でこの関数を呼び出し, RAM に書き込むことで，ROM からの読み出しを実現している.

事前計算テーブルに書き込む前に有限体上での事前計算を行うことが出来るが, その場合, 有限退場での演算は，アセンブラで行われるために，実行する CPU の Bit 数に依存した事前計算テーブルが生成される． Cortex-M7 は 32bitCPU であるため，事前計算テーブルも 32bit 用のものを用意した.

3.2 実装環境

実装に利用した機器や環境を記述する

3.2.1 ハードウェア

本実装では．Cortex-M7 を搭載した STMicroelectronics 社製の開発ボード STM32F767ZI-Nucleo　を利用した. 性能は表のとおりである.

表 1 ハードウェア性能

STM32F767ZI-Nucleo	
CPUコア	ARM Cortex-M7
OS	–
Clock	216MHz
RAM	512kByte
ROM	2MByte

$Ptbl[0][0]$	$Ptbl[0][1]$	…	$Ptbl[0][2^v-1]$	$Ptbl[1][0]$	$Ptbl[1][1]$	…	$Ptbl[1][2^v-1]$	…	$Ptbl[b-1][0]$	$Ptbl[b-1][1]$	…	$Ptbl[b-1][2^v-1]$

図 2 事前計算ビット列の構成

3.2.2 開発環境

実装の開発環境には．IAR 社の EWARM を利用する．

EWARM には，一般的なコンパイラと同様に，セクションに対してデータを書き込むアドレスを割り当てることができ. かつ const 装飾子を付けて宣言した値を ROM 領域を示すアドレスに指定するように設定できる．

3.2.3 ソフトウェア構成

以下に示すライブラリを用いて，実装を行った．

・FreeRTOS
・mbedTLS
・STM32Cube

マイコン用の OS である，FreeRTOS を用いて，ジョブの管理を行った．mbetTLS を用いて，暗号演算に利用した．特に mbedTLS 内の可変多倍長整数のライブラリを用いる．暗号演算を行うライブラリは．文献[6]でも利用している独自ライブラリを利用した．動作させている．また，有限体上の加算，乗算などはアセンブラを用いて実装し．開発ボードに付属するハードウェアの制御には．STM32CubeLibrary を利用する．
mbedTLS 上でのペアリングライブラリの動作については，文献を参照．

mbedTLS は, C 言語で記述された，ARM 社が提供する暗号・TLS/SSL 関連のオープンソースライブラリである．mbedTLS は組み込みシステム向けに利用可能なように設計されているため，PC における実装で広く利用される OpenSSL と比較し,より軽量に動作する．今回の Cortex-M 系における実装では本ライブラリを関数型暗号の演算の一部に利用する．

FreeRTOS は. 組み込み用に小さく単純であるように設計されたリアルタイム OS である．ほとんど C 言語でコードが書かれている．Amazon.com によって提供されている．

STM32Cube ライブラリは．STMicroelectoronics 社によって提供される開発ボードの付属ハードウェアを制御するためのライブラリである．

3.3 測定条件

マルチスカラ倍算に利用する楕円点の楕円曲線パラメータは 462bit 素数有限体上の$y^2 = x^3 + 5$ によって定義される楕円曲線である．

RAM に事前計算テーブルを仕込んだ場合, テーブルの大きさが 128kbyte を超えるときメモリ違反となり．動作のしなくなる. これ以上の大きさの事前計算テーブルを用意し，計算時間を測定する．

マルチスカラ倍算において，足し合わせる楕円点の数を．20．50．70 個と変化させた場合の実行速度の測定結果を比較する．また．楕円点の数を 20，50，70 個に固定し．指定する事前計算テーブルの大きさを変化させたときの実装時間の測定結果を比較する．

また．事前計算テーブルを ROM に書き込んだことによる実行速度の変化を考慮する必要がある．具体的には，ROM 一般的に RAM からの読込速度よりも ROM からの読込速度のほうが遅いため，今回の実装方法で，実行時間が遅くなる可能性がある．したがって．ROM に事前計算テーブルを書き込んだ際にマルチスカラ倍算の演算に対する影響を確認する．すなわち，事前計算テーブルの大きさを 64kByte に指定して．RAM にそのまま載せた場合と ROM に書き込んだ場合（本稿での実装）での実行速度の比較を行う．

マルチスカラ倍算用に用意する楕円点は．事前計算テーブルを作成する際にランダムで生成し．スカラ値はマルチスカラ倍算する際にランダムに生成する．

4. 実装結果

4.1 RAM 実装，ROM 実装の比較

事前計算テーブルを RAM に実装した場合（RAM 実装）と．ROM に実装した場合（ROM 実装）のマルチスカラ倍算の実行速度の比較結果を表に記述する．

50 個の楕円点を加算するマルチスカラ倍算では．RAM 実装に比べて ROM 実装での実行時間は，1.2%の増加が見られた．一方で，RAM 実装と ROM 実装での実行時間の差は，事前計算テーブルなし(非高速化)でのマルチスカラ倍算の実行時間の 0.4% 程になる．したがって，実行時間の差は，高速化に対して十分無視できるほど小さいため，ROM 実装による高速化は RAM 実装による高速化と変わりないと思われる．

20, 70 個の場合も同様に，ROM 実装と RAM 実装による実行時間の差は，十分無視できるほど小さいといえる．

表 3 ROM 実装．RAM 実装の実行速度の比較

楕円点の個数	ROM 実装（msec）	RAM 実装（msec）	非高速化（msec）
20	826	817	2731
50	2102	2076	6626
70	5103	4481	9207

4.2 事前計算テーブルの大きさを変えた際の計測結果

事前計算テーブルの指定値を RAM に入らない大きさである，512Kbyte に固定した際の実装結果を表に示す．以下のように ROM に事前計算テーブルを保存した場合でも，マルチスカラ倍算の計算速度の向上を確認できた．

更に，事前計算テーブルの大きさを変化させた際の実行速度を比較すると，事前計算テーブルをおおきくすることで，実行時間が増加していることを確かめた．（表 5,6）

表 4 個数を変えた際の実行速度の比較

楕円点の個数	実際の事前テーブルの大きさ（Byte）	ROM 実装（msec）	事前計算なし(msec)
20	368296	540	2731
70	457216	2072	9327

表 5 事前計算テーブルの大きさを変えた際の実行速度の比較(楕円点数 20)

指定事前計算テーブルの大きさ（kByte）	実際の事前テーブルの大きさ（Byte）	ROM 実装（msec）
64	44656	854
256	229516	570
512	368296	540

表 6 事前計算テーブルの大きさを変えた際の実行速度の比較(楕円点数 70)

指定事前計算テーブルの大きさ（kByte）	実際の事前テーブルの大きさ（Byte）	ROM 実装（msec）
64	56716	5103
256	228612	2050
512	457216	2072

5. 関数型暗号への適応に関する考察

本章では. マルチスカラ倍算の高速化による適用により，どの程度高速化を図ることが可能か見積もる．

5.1 関数型暗号のアルゴリズム

関数型暗号とは，公開鍵暗号をより高度にした暗号方式であり，暗号文，属性鍵に属性情報や条件式を導入することで，暗号/復号のロジックを規定している．この暗号と復号のメカニズムには，AND ゲート，OR ゲート，NOT ゲート，しきい値ゲートにより表現されるすべての関係式を組み込むことが可能である．関数型暗号には，暗号文に属性情報，復号鍵に条件式を組み込む，Key Policy 方式と，暗号文に条件式，復号鍵に属性情報を組み込む，Cipher text Policy 方式がある．本稿では，復号鍵に属性情報の OR 条件式を持たせることで，復号する機器側で保有する鍵の個数を削減することができると考え，Key Policy 方式を用いる．関数型暗号のアルゴリズムの詳細に関しては記述を省略する．処理の詳細は OT-10[7]の文献を参照されたい．

5.2 実装結果からの考察

関数型暗号において，公開パラメータには， d+2 個の楕円点行列が含まれる．個々で，d は表現できる属性の個数である．各楕円点行列を$\mathbb{B}_t$（$1 \le t \le d+2$）と表すと，$\mathbb{B}_t$（$1 \le t \le d$）は$(3n_t+1)\times(3n_t+1)$行列，$\mathbb{B}_{d+1}$は5×5行列，$\mathbb{B}_{d+1}$は7×7行列である．n_tは各属性を表現するための次元数である．

関数型暗号の暗号化処理においては，行列とスカラ列を用いて次のような行列演算が行われる．

$$c_t = \left(x_{t,1}, x_{t,2}, \dots, x_{t,n_t}, \overbrace{0^{n_t}}, \overbrace{0^{n_t}}, \overbrace{\varphi_t^{1}}\right)\mathbb{B}_t$$

$$(1 \le t \le d)$$

$$c_{d+1} = (\delta_1, 0, \zeta_1, 0, \varphi_1)\mathbb{B}_{d+1}$$

$$c_{d+2} = (\delta_2, \zeta_2, 0, 0, 0, 0, \varphi_2)\mathbb{B}_{d+2}$$

ここで，$x_{t,1}, x_{t,2}, \dots, x_{t,n_t}, \delta_s, \zeta_s, \varphi_s$, は整数値であり，暗号化の際に指定するランダムな値である．ここで，各楕円点の行列の列に注目すると，上記の行列演算は，マルチスカラ倍算を行っていることがわかる．

すなわち，暗号化処理全体で $D = \left(\sum_{t=1}^{d} 3n_t + 1\right) + 12$ 回のマルチスカラ倍算が行われることになる．

詳細に記述すると，各 t に対して，$n_t + 1$個の楕円点によるマルチスカラ倍算がそれぞれ $3n_t + 1$回，3 個の楕円点によるマルチスカラ倍算が 12 回行われる．

例えば. 属性が 3 個あり. それぞれ次元が 2 である場合. を考える．このような状況は以下のような設定考えることができる．

・IoT システムに対して．IoT 機器が数千台あり．各 IoT 機器に対して．ユニークな ID が振られている

・各 IoT 機器はグループに所属しており．各グループにはユニークな ID が振られている．

・データに対して閲覧可能な有効期限（日付）を設定する．

この場合．属性の種類として．IoT 機器の ID．IoT グループの ID. 有効期限の 3 つを想定できる. かつ. 各属性は. 重複して，条件式を設定する可能性を考えると(たとえば，2 つのグループに対して復号を許可したい場合など)，次元として．最低でも 2 もつ必要がある．

暗号化処理の中で，楕円点 3 個のマルチスカラ倍算が 33 回行われる. 事前に計算した値（表 4 を参照）を用いると，522(msec)×33 回=17226(msec) の実行時間が必要であると見積もることが出来る．

ここで，各マルチスカラ倍算に対して，事前計算テーブルを作成し，本稿での実装法を適用することを考える．楕円点の数が少ないため，各事前計算テーブルの大きさは小さめにとって，32Kbyte で見積もる．このとき，事前計算テーブルの大きさは，約 730kbyte となる．なお，今回用いたにおいては，この事前計算テーブルは ROM にしか実装できない. このとき, 暗号化処理の時間を見積もると, 4158msec となる．

楕円点の個数	事前計算テーブルの大きさ (Byte)	ROM 実装 (msec)	非高速化（msec）
3	22696	128	522
20	368296	540	2731

表 7 マルチスカラ倍算実行時間（一部）

暗号化演算においては．ハッシュ値の計算や可変多倍長整数の生成．和や積も行われるが．これらの演算は楕円点の演算に対して．演算時間は無視できるほど小さい．

上記においても．暗号化に数秒の時間を要するが．秒単位のリアルタイム性を求めない．たとえばスマートハウスやスマートアグリなどの分野であれば．十分な性能を確保できていると思われる．

6. まとめと今後の課題

本稿では．IoT 機器として今後の利用が想定される Cortex-M 系デバイスでのマルチスカラ倍算の事前計算テーブルを ROM に書き込んで．ROM から読み出すように実装した結果..関数型暗号へ適用する際の高速化について見積もった．その結果．ROM へ事前計算テーブルを書き込んで．ROM から読み出してマルチスカラ倍算を行う実装方法は．マルチスカラ倍算の高速化に有効であることが確かめられた．また．マルチスカラ倍算を多用する関数型暗号に対しても．属性の表現が複雑であっても．暗号化時間を短縮可能であることを示した．

以上により．計算能力やメモリリソースの少ない Cortex-M 系デバイスのような．センサデータの収集を行う IoT 機器であっても．リアルタイム性を求めない IoT システムにおいては十分な性能を確保できていると考えられる

今後の課題としては．実際に関数型暗号の暗号化を実装し．性能を確かめる．その際に．効率の良い事前計算テーブルの大きさを決定す方法を検討する．さらに．今回は ROM に保存するという方法をとることで．性能に制限がある IoT デバイスであっても．高速化を実現した．

参考文献

[1] 総務省 “情報通信白書” http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h30/html/nd111200.html

[2] Ahmad-Reza Sadeghi , et.al. “Security and privacy challenges in industrial internet of things”, **ACM/EDAC 2015**

[3] 坂村健，“IoT 動く”，http://www.tronshow.org/2016-tron-symposium/session-pdf/ja/pdf/20161214-1-1.pdf ， 2016 TRON Symposiom.

[4] IPA，“IoT 開発におけるセキュリティ設計の手引き”，p. 18-19.

[5] 田村桜子，永井彰,”関数型暗号を適用した IoT 機器データ活用方式の一考察”， CSS2017

[6] 田村桜子，永井彰， 早川和宏,“CortexM 系での関数型暗号の実装評価”， SCIS2018

[7] T．Okamoto，K．Takashima，“Fully Secure functional Encryption with General Relations from the Decisional Linear Assumption，” CRYPTO2010，LNCS6223，pp．191-208，2010．.

[8] 高島克幸，岡本龍明，“アクセス権限を制御できる最新技術「関数型暗号」，” 日経 BP 社，p．87-95，2012

[9] C.H. Lim, P.J. Lee, “More Flexible Exponentiation with Precomputation,” CRYPT0 ’94, LNCS 839, pp. 95–107, 1994.

[10] 川原祐人，小林鉄太郎，”効率的な固定点マルチスカラ倍算の提案”, SCIS2015 1B-1