参考資料

Technical White Paper

MSPM0G MATHACL を使用した ED25519PH アルゴリズムの性能高速化

Gaspar Zoltan, Pengfei Xie

概要

本ドキュメントでは、MATHACL モジュールを搭載した MSPM0G マイコン向けに改善された、ED25519PH 楕円曲線暗号アルゴリズム (Libsodium ライブラリより) の効率的な実装を提示します。Arm® Cortex®-M0+ コアは 32 ビット ALU を備えていますが、ネイティブな 32x32 から 64 ビットへの乗算命令がないため、256 ビット演算にはソフトウェアベースの多倍長演算が必要になります。本ドキュメントでは、32x32 から 64 ビットへの乗算および積和演算 (MAC) をサポートする MATHACL ペリフェラルに重要な演算タスクをオフロードする、ハイブリッド実装を提案します。このアプローチにより実行時間とコード密度の両方が向上し、目標コード サイズの制約に応じて、Arm Cortex-M0+ コア上の純粋なソフトウェア実装の現在の速度と比較して 3 ～ 5 倍の高速化を達成します。MSPM0G3519 デバイスに基づく最適化された ED25519PH アルゴリズムの CCS サンプル プロジェクトには、ED25519PH_crypto_mspm0g3519 からアクセスできます。

目次

1 概要 2
2 ED25519PH アルゴリズムの概要 2
3 MSPM0G 演算アクセラレータ (MATHACL) 2
3.1 MATHACL モジュール 2
3.2 積和演算 (MAC) 動作 3
4 MATHACL の使用による ED25519PH アルゴリズムの高速化 4
4.1 ループ展開実装 4
4.2 ループ ロール実装 5
4.3 ベンチマーク測定 5
5 まとめ 6
6 参考資料 6

商標

Arm® and Cortex® are registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere.
すべての商標は、それぞれの所有者に帰属します。

Copyright © 2026 Texas Instruments Incorporated

1 概要

非対称 (公開鍵) 暗号は組込みデバイスのセキュリティ保護に不可欠であり、セキュア ブート、暗号化されたクラウド通信 (TLS)、デジタル ID 検証などの重要な機能を実現します。これらの手法の中で、楕円曲線暗号 (ECC) は従来の RSA よりもはるかに小さい鍵で高いセキュリティを提供するため、マイコンに適しています。この分野の主要な標準は ED25519PH アルゴリズム (RFC 8032 で標準化) であり、その速度とサイドチャネル攻撃への耐性から、TLS 1.3 などの最新プロトコルで広く使用されています。

ED25519PH アルゴリズムの利点にもかかわらず、ED25519PH の性能は効率的な 256 ビット多倍長乗算に大きく依存します。Arm Cortex-M0+ (ARMv6-M) マイコンでは、ARMv6-M 命令セットにネイティブな 32x32 から 64 ビットへの乗算命令がないため、これは大きな課題であり、32 ビット演算はソフトウェア エミュレーション (Clang の __aeabi_lmul ソフトウェアなど) を使用して処理する必要があります。この関数は関数呼び出しの準備とデジタル計算に 30 個以上の命令を必要とするため、この手法はきわめて非効率的です。

この問題を解決するために、本ドキュメントでは、ハードウェア アクセラレーション対応の MATHACL ペリフェラルを利用する、MSPM0G シリーズ向けに改善された実装を提案します。32x32 から 64 ビットへの乗算および積和演算 (MAC) タスクをハードウェアにオフロードすることにより、実行時間は数十命令からわずか 7 クロック サイクルに短縮されます。このアプローチにより、リソースに制約のある組込みシステムにおける ED25519PH アルゴリズムの性能が大幅に向上し、Arm Cortex-M0+ コア上の純粋なソフトウェア実装の現在の速度と比較して 3 ～ 5 倍の高速化を達成できます。

2 ED25519PH アルゴリズムの概要

ED25519PH アルゴリズムの実装については、Libsodium ライブラリを参照してください。ED25519PH デジタル署名スキームの性能は、主として高速な大整数演算によって左右されます。具体的には、このアルゴリズムでは大きな素数 (2^{255} - 19) を法とする 256 ビット整数の頻繁な乗算が必要です。Arm Cortex-M0+ のような 32 ビット アーキテクチャで広幅ワード演算を実装するには、256 ビットのオペランドをいわゆるリム (limb) と呼ばれる小さな単位に分解する必要があります。

基数ソートに基づくこれらの多倍長乗算を処理するには、主に 2 つのアプローチがあります。

A. Radix-32 実装:Radix-32 アプローチでは、大きな整数を 32 ビットのリムに分解します。512 ビットや 258 ビットの整数など、異なるビット幅をサポートするようにロジックを容易に拡張できるため、この手法は汎用性に優れています。この実装の主な利点はコード フットプリントが最小限であることであり、メモリに制約のある環境に適した実装となっています。しかし、この実装では数学的な整合性を維持するためにリム単位の演算ごとにオーバーフローを厳密に処理する必要があり、実行効率に影響を与える可能性があります。

B. Radix-25.5 実装:もう 1 つのアプローチは、『Curve25519：New Diffie-Hellman Speed Records』で詳述されているように、Radix-25.5 表現を利用します。この手法では、255 ビットの数値を 10 個のリムで表し、各リムは 32 ビット ワードのうち 25.5 ビットを占有します。この特定のビット幅により、中間積が 32 ビットのレジスタ容量を超えることなく、25.5 ビットの境界をオーバーフローすることが可能になります。この戦略により、桁上げ計算を最終段階まで効果的に遅延させることができ、乗算プロセスが大幅に高速化されます。ループ展開技術によりこの手法はきわめて高いパフォーマンスを発揮しますが、Radix-32 アプローチと比較してコード サイズが大幅に増大します。

本ドキュメントにおける ED25519PH アルゴリズムの実装は Radix-25.5 アプローチに基づいており、MSPM0G デバイスの MATHACL は、アルゴリズム内で頻繁に行われる 32x32 から 64 ビットへの乗算および積和演算 (MAC) を高速化できます。

3 MSPM0G 演算アクセラレータ (MATHACL)

3.1 MATHACL モジュール

演算アクセラレータ (MATHACL) は、ハードウェア アクセラレーションによる 32 ビット演算関数の集合体であり、システムの計算スループットを向上させます。CPU で実行されるソフトウェアは、MATHACL レジスタに書き込むことによって MATHACL を呼び出します。MATHACL ペリフェラルは、MSPM0Gx50x、MSPM0Gx51x、MSPM0G321x シリーズなどの MSPM0G シリーズ デバイスでサポートされており、本機能の詳細は、『MSPM0 G シリーズ 80MHz マイコン テクニカル リファレンス マニュアル』の「MATHACL」の章を参照してください。

MATHACL は、効率を向上させるために CPU が実行する数学的計算をオフロードします。MATHACL では、以下のハードウェア機能を使用できます。

- 正弦 / 余弦 (サイン / コサイン) (SINCOS)

Copyright © 2026 Texas Instruments Incorporated

- 逆正接 (アークタンジェント) (ATAN2)
- 平方根 (SQRT)
- 除算 (DIV)
- 乗算、32 ビットの結果 (MPY32)
- 二乗、32 ビットの結果 (SQUARE32)
- 乗算、64 ビットの結果 (MPY64)
- 64 ビット結果の二乗 (SQUARE64)
- 積和演算 (MAC)
- 二乗和演算 (SAC)

積和演算 (MAC) 動作は、ED25519PH アルゴリズムで使用されるものです。

3.2 積和演算 (MAC) 動作

積和演算 (MAC) 関数は、MATHACL が別の機能のために再プログラムされるまで、複数回の反復にわたって結果レジスタ RES1 および RES2 の内部に OP1 と OP2 の積の結果を累積します。MATHACL は、32 ビットの乗数および被乗数のオペランド サイズ、ならびに 64 ビットの累積結果サイズのみをサポートします。図 3-1 に、MAC モードでの MATHACL の動作を示します。

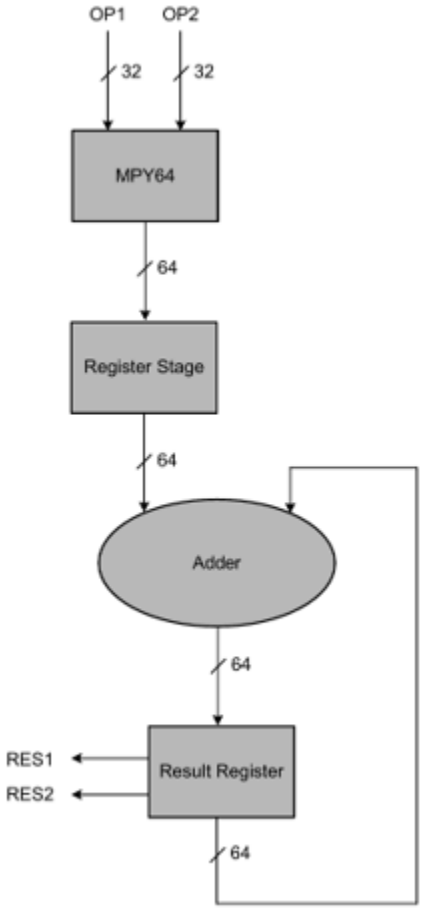

図 3-1. MATHACL の MAC 動作

注

正しい結果を得るには、MAC 動作を開始する前にユーザーが RES1 および RES2 に 0 を書き込む必要があり、MAC 動作の最終結果を取得する前に OP2 の書き込み後 2 サイクル待機する必要があります。

MATHACL の MAC (積和演算) 動作を使用することにより、CPU は 32 ビットの入力データを OP1 および OP2 レジスタに継続的にストリーミングできます。データのロードが完了すると、最終結果が RES1 および RES2 レジスタから直接取

Copyright © 2026 Texas Instruments Incorporated

得されます。この手法により、総 MAC 実行時間が入力ロード期間とほぼ同じ時間に短縮され、CPU ベースの標準的な動作と比較して大幅な性能向上が実現します。

4 MATHACL の使用による ED25519PH アルゴリズムの高速化

Radix-25.5 実装に基づいて、fe25519_mul ルーチン (サンプル コードの ed25519_ref10_fe_25_5.h に実装) は最適化されたスクールブック乗算を使用します。部分積は 10 個の 64 ビット一時変数に直接累積されるため、大きな多倍長中間バッファが不要になります。

恒等式 $2^{255} \equiv 19 \pmod p$ を使用することにより、ビット 255 を超える項は直ちに下位のリムへと折り返されます。したがって、この実装では乗算と剰余削減を単一の演算に統合しています。

fe25519_mul(fe25519 h, const fe25519 f, const fe25519 g) の引数定義:

- h:乗算出力、int32[10] フォーマット
- f:乗算入力、int32[10] フォーマット (f0 は f[0] を表す)
- g:乗算入力、int32[10] フォーマット (g0 は g[0] を表す)

4.1 ループ展開実装

4.1.1 MATHACL を使用しない最適化

乗算の開始前に、『Curve25519:New Diffie-Hellman Speed Records』ではいくつかのヘルパー値を計算します。

```
g1_19 = 19 * g1;
g2_19 = 19 * g2;
...
g9_19 = 19 * g9;
f1_2 = 2 * f1;
f3_2 = 2 * f3;
...
f9_2 = 2 * f9;
```

出力係数は 10 個の 64 ビット変数に累積されます (以下に h[0] の計算結果を示します)。

```
int64_t h0;
h0 = f0*g0 + f1_2*g9_19 + f2*g8_19 + f3_2*g7_19 + f4*g6_19 + f5_2*g5_19
+f6*g4_19+f7_2*g3_19+f8*g2_19+f9_2*g1_19;
```

4.1.2 MATHACL による最適化

h[0] の結果を得るために、MATHACL を使用して同じ演算を実行できます。

```
MATHACL->RES2 = 0;
MATHACL->RES1 = 0;
MATHACL->OP2 = g0;
MATHACL->OP1 = f0;
MATHACL->OP2 = g9_19;
MATHACL->OP1 = f1_2;
MATHACL->OP2 = g8_19;
MATHACL->OP1 = f2;
MATHACL->OP2 = g7_19;
MATHACL->OP1 = f3_2;
MATHACL->OP2 = g6_19;
MATHACL->OP1 = f4;
MATHACL->OP2 = g5_19;
MATHACL->OP1 = f5_2;
MATHACL->OP2 = g4_19;
MATHACL->OP1 = f6;
MATHACL->OP2 = g3_19;
MATHACL->OP1 = f7_2;
MATHACL->OP2 = g2_19;
MATHACL->OP1 = f8;
MATHACL->OP2 = g1_19;
MATHACL->OP1 = f9_2;
__asm__ __volatile__ ("nop");
hvect[0] = ((uint64)(MATHACL->RES2)<<32) + MATHACL->RES1;
```


Copyright © 2026 Texas Instruments Incorporated

累積後、桁上げは交互に配置された 26 ビットおよび 25 ビットのリムを通じて伝搬されます。最上位のリムからの最終的な桁上げはリム 0 に折り返されて 19 倍され、有効な体の元が維持されます。

MSPM0G デバイスの MATHACL モジュールにおける MAC 動作に基づき、プログラムは 32 ビットの演算データを OP1 および OP2 に継続的に入力して RES レジスタから MAC 結果を直接取得する必要があり、これにより 32 ビット乗算効率が明確に向上します。

4.2 ループ ロール実装

演算をループ内で実行すれば、前の演算のコード サイズを削減できます。この調整により、メモリ サイズがきわめて重要な状況でもアルゴリズムを適用できるようになります。

4.2.1 MATHACL を使用しない最適化

ヘルパー変数を配列に追加して、インデックス作成を改善できます。

```
int32 f_2[10];
int32 g_19[10];
int32 *f_combined[2];
f_combined[0] = f;
f_combined[1] = f_2;
for (i=0; i<10; i++){
g_19[i] = 19 * g[i];
if (i % 2 ==0) f_2[i]= f[i];
else f_2[i]= 2* f[i];
}
```

出力係数は 10 個の 64 ビット変数に累積されます。

```
int64_t hvect[10];
for (j=0; j<10; j++){
hvect[j] = 0;
for (i=0; i<j+1; i++)
hvect[j] += (int64) g[j-i]* (int64) f_combined[(j+1)%2][i];
for (i=0; i<10-j-1; i++)
hvect[j] += (int64) g_19[9-(i)] + (int64) f_combined[(j+1)%2][j+1+i];
}
```

4.2.2 MATHACL による最適化

MATHACL を使用する場合、乗算出力 (h[10]) を取得するためのコード実装は次のようになります。

```
for (j=0; j<10; j++){
MATHACL->RES2 = 0;
MATHACL->RES1 = 0;
for (i=0; i<j+1; i++){
MATHACL->OP2 = g_combined[0][j-i];
MATHACL->OP1 = f_combined[(j+1)%2][i];
}
for (i=0; i<10-j-1; i++){
MATHACL->OP2 = g_combined[1][9-(i)];
MATHACL->OP1 = f_combined[(j+1)%2][j+1+i];
}
hvect[j] = ((uint64)(MATHACL->RES2)<<32) + MATHACL->RES1;
}
```

累積後、桁上げはループ展開のシナリオと同様の方法で伝搬します。MATHACL を使用することにより、プログラムの実行時間を最適化できます。ループ展開した実装と比較して、ループ ロール + MATHACL 実装ではループ用の入力演算データのロード間にいくつかの命令が必要となるため、実行時間は長くなりますがコード サイズは小さくなります。

4.3 ベンチマーク測定

表 4-1 に、ED25519PH アルゴリズム検証プロセスのメモリおよびレイテンシのベンチマークを示します。選択した実装スタイル (ループ ロールまたはループ展開) において、MATHACL を使用することにより、純粋なソフトウェア実装の通常の効率と比較してコード実行効率を 3 ～ 5 倍向上させることができます。

ベンチ テスト条件を以下に示します。

Copyright © 2026 Texas Instruments Incorporated

- コンパイラ：TI Clang v4.0.4.LTS
- コンパイラ最適化レベル：-Os
- テスト ケース：サンプル プロジェクトの Sign_Util_Cfg.h ヘッダ ファイル内のマクロ USE_MATHACL および CRYPTO_OPT_SIZE によって、4 つの異なるテスト ケースを定義できます。

表 4-1. ED25519PH アルゴリズム検証ベンチマークの比較

	サイズ最適化 (ループ ロール)		速度最適化 (ループ展開)	
	実行時間 (ms)	コード サイズ (kB)	実行時間 (ms)	コード サイズ (kB)
CPU	488	11	428	16
CPU + MATHACL	135	11	85	15

5 まとめ

ED25519PH アルゴリズム向けにオープン ソースの実装 (Libsodium および LibTomCrypt) が公開されていますが、実行時間とコード サイズのトレードオフは良好ではありません。MSPM0G デバイスの MATHACL ペリフェラルは、楕円曲線暗号において大幅な速度向上 (約 3 ～ 5 倍) を提供します。

最大の速度向上には、ループ展開 (ループ アンローリング) によるコード サイズの増大が伴います。プロジェクトの制約 (実行時間対コード サイズ) に応じて、本ドキュメントで提示する手法により、きめ細かなトレードオフの実装が可能です。本ドキュメントで説明する手法は、セキュア ブートや署名検証などのケースに適用できる ED25519PH アルゴリズムの検証プロセスに焦点を当てています。

6 参考資料

1. Bernstein、D. J.(2006)。Curve25519：New Diffie-Hellman Speed Records。In M. Yung、Y. Dodis、A. Kiayias、& T. Malkin (Eds.)、Public Key Cryptography – PKC 2006 (Vol. 3958、pp. 207–228)
2. テキサス・インスツルメンツ (2026 年)。『MSPM0 G シリーズ 80MHz マイコン テクニカル リファレンス マニュアル』。
3. テキサス・インスツルメンツ (2025 年)。『CAN-FD インターフェイス搭載 MSPM0G350x ミックスド シグナル マイコン』データシート。

Copyright © 2026 Texas Instruments Incorporated

重要なお知らせと免責事項

TI は、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、TI 製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した TI 製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている TI 製品を使用するアプリケーションの開発の目的でのみ、TI はその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。TI や第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、 TI の販売条件 、 TI の総合的な品質ガイドライン 、 ti.com または TI 製品などに関連して提供される他の適用条件に従い提供されます。TI がこれらのリソースを提供することは、適用される TI の保証または他の保証の放棄の拡大や変更を意味するものではありません。 TI がカスタム、またはカスタマー仕様として明示的に指定していない限り、TI の製品は標準的なカタログに掲載される汎用機器です。

お客様がいかなる追加条項または代替条項を提案する場合も、TI はそれらに異議を唱え、拒否します。

Copyright © 2026, Texas Instruments Incorporated

最終更新日：2025 年 10 月