

共通言語基盤上での Camellia と Rijndael の高速な実装手法について

及川一樹 児玉英一郎 王家宏 高田豊雄

岩手県立大学ソフトウェア情報学部

1. はじめに 近年, Microsoft が設計し, 国際規格でもある共通言語基盤を利用した開発が数多く行われている. この共通言語基盤は, 中間言語を用いて仮想マシン上で実行する仕組みを採っている. このため, 直接機械語にコンパイルする場合と比べるとパフォーマンスの点で劣っているが, 言語に依存しないため, 言語間での相互運用性が高く, 開発も容易といった利点を持っている.

一方, 近年のアプリケーション開発において, 個人情報や機密情報保護の観点から, データを暗号化する必要性が高まっているが, 高速かつ強度の高い対称鍵アルゴリズムとして知られる Camellia[1] や Rijndael[2] でも, CPU に強く依存するアセンブラによる実装でなければ, ギガビットイーサネットワークなどの広帯域を生かすような, 高速な暗号化処理は望めない. 即ち, 共通言語基盤上で Camellia や Rijndael を単純に実装した場合, 高速に動作させることは困難である.

そこで, 本稿では共通言語基盤上で Camellia と Rijndael を高速化する手法を提案する. また, 評価の項では共通言語基盤の実装として, .NET Framework を利用し, 本提案手法を C# により実装した場合の性能結果を示す.

2. 既存の高速化手法 Camellia と Rijndael には以下のような高速化手法が既に提案されている.

1. 鍵を副鍵に展開する処理を, 暗号化する処理単位ごとに行うのではなく, 事前にメモリ上に副鍵を展開し, それを処理時に参照することで計算量を削減する
2. 暗号処理に利用する複数の関数の代わりに, 予め用意しておいた置換表を利用して, 等価な結果を得られるようにし, 処理時の計算量を削減する

但し, これらは事前計算を行うことで, 計算量を減らすというアルゴリズムレベルでの改良であり, 実際に実装する場合, この手法だけでは完全とはいえない.

3. 共通言語基盤向けの高速化手法の提案 我々は, 上述の既存の高速化手法に加えて利用できる, 共通言語基盤上での実装レベルの高速化手法を提案する. 以下に本提案手法を示す.

1. 共通言語基盤における配列では, アクセスするたびに添え字の値が配列サイズの範囲内かチェックする処理 (境界チェック) を行うため, パフォーマンスが若干低下する. これを回避するために境界チェックが行われないポインタ型に配列型を変換してから, アクセスを行うようにする.
2. 一時的に利用する配列のメモリ確保は, 通常ならばヒープ領域だが, ガベージコレクションの対象外であり, 境界チェックの入らないスタック領域に確保することで高速化を図る.
3. 参照頻度の高い置換表などは静的メンバ変数よりも少ない命令数でアクセスできるローカル変数やメソッドの引数を経由してアクセスする.
4. n-bit の数値型から 8bit の数値を取得するときには,

0xFF との論理積ではなく, byte 型へのキャスト変換を利用する.

5. 命令語長を短くするために, メソッドの引数の順序を参照頻度の高い順に並べる.

但し, 4 は共通言語基盤の仕様では特に定義されておらず, 実装依存となるが Microsoft による実装である .NET Framework とオープンソースの実装である Mono の二つでは, 有用であった.

4. 評価 既存の高速化手法に加えて, 本提案手法の高速化手法を実装したプログラムを, AMD Opteron 270 搭載の PC で実行し, 性能評価を行った. 結果を図 1 に示す. なお, ここでは鍵長・ブロック長は共に 128bit, ブロック暗号モードは ECB を利用した.

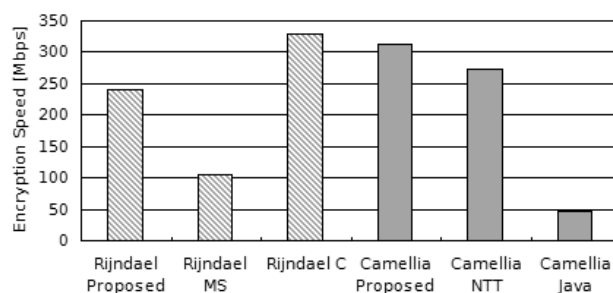


図 1: 暗号化速度の比較

図 1 の Rijndael Proposed と Camellia Proposed が本提案手法に従った実装による結果となっている. Rijndael MS は .NET Framework に付属の Rijndael アルゴリズムの実装, Rijndael C は OpenSSL で利用されている C 言語による実装, Camellia NTT と Camellia Java は NTT による実装で, 前者は C 言語による実装となっている.

図 1 で示したとおり, Camellia の場合は C 言語による実装よりも本実装の方が高い性能を示しているが, Rijndael の場合は, .NET Framework に付属しているものよりは優れているものの, C 言語による実装よりは劣った結果となった.

5. おわりに 本稿では, 共通言語基盤上での Camellia と Rijndael の高速化手法の提案を行った. また, 本提案手法の実装を利用した性能評価結果について報告を行った. その結果, 仮想マシン上で動くプログラムながら, 直接, 環境固有の機械語にコンパイルされる C 言語で記述されたプログラムに近い暗号化速度を達成できた.

今後は, スタックマシン向けに最適化できる箇所がないか, 調査する予定である.

参考文献

- [1] 市川哲也, 松井充, 中嶋純子, 時田俊雄, 青木和麻呂, 神田雅透, 盛合志帆: 128 ビットブロック暗号 Camellia アルゴリズム仕様書, <http://info.isl.ntt.co.jp/crypt/camellia/dl/01jspec.pdf>
- [2] Joan Daemen, Vincent Rijmen: AES submission document on Rijndael, Version 2, <http://csrc.nist.gov/CryptoToolkit/aes/rijndael/Rijndael-ammended.pdf>