

一般化メルセンヌ素数が法の時の還元法とその楕円曲線暗号の 共通言語基盤実装への適用について

及川一樹 王家宏 児玉英一郎 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, 短い鍵長で高い安全性を確保できる公開鍵暗号方式である楕円曲線暗号の普及が進んでいる。楕円曲線を構成する有限体としては, 素体 $GF(p)$ や標数 2 の有限体 $GF(2^m)$ などが利用可能であるが, 素体上の楕円曲線は標数 2 の有限体よりもソフトウェア実装に向いている他, アメリカ国家安全保障局 (NSA) が 2005 年に発表した次世代暗号規格である Suite B [1] において, 素体上での楕円曲線暗号のみが採用されているため, 今後特に普及が進むと考えられる。楕円曲線暗号の奨励パラメータでは, 法として一般化メルセンヌ素数を採用している。一般化メルセンヌ素数は $2^{n_1} - 2^{n_2} - \dots - 2^{n_m} - 1$ ($n_1 > n_2 > \dots > n_m$) という形をした特殊な素数となっており, この素数を法とした場合, 剰余演算が高速に行えることが知られている。

そこで, 本稿では一般化メルセンヌ素数が法の時の還元法について説明した後に, 効率的に動作する実装手法を提案する。そして, 昨年度に我々が提案した共通言語基盤上で有用な高速化手法 [2] を利用した暗号アルゴリズムの実装に本提案手法を適用し, その効果を評価する。

2. 一般化メルセンヌ素数が法の時の還元法 一般化メルセンヌ素数が法の時の還元法について, 以下に例を挙げて説明する。この例では法 $p = 2^{192} - 2^{64} - 1$ を利用し, 法 p における数値同士の乗算結果を対象とするため, p^2 未満の値が還元の対象となっている。還元を行う値を x とすると, x は以下のように表現できる。

$$x = x_5 2^{320} + x_4 2^{256} + x_3 2^{192} + x_2 2^{128} + x_1 2^{64} + x_0 \quad (x_i < 2^{64})$$

また, 法 p においては次の合同式が成り立つ。

$$2^{320} \equiv 2^{128} + 2^{64} + 1 \pmod{p}$$

$$2^{256} \equiv 2^{128} + 2^{64} \pmod{p}$$

$$2^{192} \equiv 2^{64} + 1 \pmod{p}$$

よって, x は法 p において以下のように表現できる。

$$x \equiv (x_5 + x_4 + x_2) 2^{128} + (x_5 + x_4 + x_3 + x_1) 2^{64} + (x_5 + x_3 + x_0) \pmod{p}$$

以上により, 法が一般化メルセンヌ素数の時, x は加算のみで還元が行えるため, 他の還元アルゴリズムよりもかなり少ない計算量で可能となる。

3. 一般化メルセンヌ素数が法の時の乗算の実装手法の提案 一般に乗算の剰余を求める際は, 乗算を行い, その結果に対して還元演算を行うという二段階を踏む。また, 法 p は 192 ~ 384bit 程度の長さであるため, Karatsuba 法や FFT を用いた高速乗算法を利用せず, 式 (1) に示すような古典的な乗算法 (いわゆる筆算) を利用する。古典的乗算法では乗算する 2 つの値 (a, b) の各桁同士の乗算 $(a_i b_i)$ を行った後, その解 $(c_0 \sim c_{17})$ の同じ桁同士の数値を加算することで, 最終的な解を求めている。

そこで我々は, 古典的乗算法で行われている加算と, 2 節で示した還元演算に含まれる加算を同時に行うことで, より高速に乗算の剰余を求める実装手法を提案する。

2 節の例を用いて提案手法を説明すると, x_i は基数 2^{64} の桁として考えることができ, 同様に式 (1) を基

数 2^{64} の 3 桁の乗算と考えることができる。式 (1) の $c_5, c_9, c_{10}, c_{13}, c_{14}$ は $2^{192} \equiv 2^{64} + 1$ より, 式 (2) のように, 4 桁目から 1 桁目と 2 桁目に移動する。同様に式 (1) の 5, 6 桁目も 1 ~ 3 桁目に移動し, 最終的に式 (2) のように 1 ~ 3 桁に収まる形になる。最後に, 式 (2) の c_i の同じ桁同士を加算し解を求める。以上により, 従来よりも計算量を抑えて乗算の剰余を求める事ができるため, 法 p における乗算の高速化が可能となる。

\times	a_2	a_1	a_0	\times	a_2	a_1	a_0
	b_2	b_1	b_0		b_2	b_1	b_0
		c_1	c_0			c_1	c_0
	c_3	c_2			c_3	c_2	
	c_5	c_4	c_6		c_4	c_5	c_5
		c_7			c_7	c_6	
	c_9	c_8			c_8	c_9	c_9
c_{11}	c_{10}				c_{11}	c_{10}	c_{10}
	c_{13}	c_{12}			c_{11}	c_{11}	
	c_{15}	c_{14}			c_{12}	c_{13}	c_{13}
c_{17}	c_{16}				c_{14}	c_{14}	c_{14}
					c_{15}	c_{15}	
					c_{16}	c_{16}	
					c_{17}	c_{17}	c_{17}

(1)

(2)

4. 楕円曲線暗号への適用と評価 昨年度, 我々が実装した, 仮想マシンを実行基盤とする共通言語基盤上で動作する暗号ライブラリ [2] に本提案手法を適用し, 楕円曲線暗号の署名方式である ECDSA の署名生成・検証速度を評価した。結果を図 1 に示す。評価環境は, OS: Windows Vista, CPU: AMD Opteron 2212, 共通言語基盤: .NET Framework 2.0 である。Original が本提案手法適用前, OpenSSL はオープンソースプロジェクトである OpenSSL の ECDSA 実装, Proposed が本提案手法適用後を表している。図 1 のとおり, 従来の実装と比較し 18% の速度の向上が確認できた。

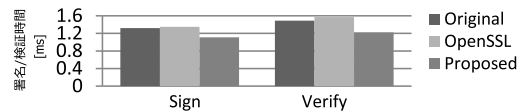


図 1: ECDSA 速度の比較

5. 終わりに 本稿では楕円曲線暗号などに使われる一般化メルセンヌ素数が法の時の乗算を効率的に行う実装手法を提案し, 本提案手法を ECDSA に適用したときの性能評価結果について報告を行った。現在は法となる一般化メルセンヌ素数ごとに乗算部分を実装しなければならないが, 本評価に利用した共通言語基盤はプログラムの動的生成が可能であるため, その機能を利用した乗算部分の自動生成手法を考案する予定である。

本提案手法を利用した暗号ライブラリは Web 上で公開しており, 修正 BSD ライセンスのもと自由に利用することができる。より詳しい情報やソースコードは <http://trac.panicode.com/crypto/> より入手可能である。

参考文献

- [1] Fact Sheet NSA Suite B Cryptography: http://www.nsa.gov/ia/industry/crypto_suite_b.cfm
- [2] 及川一樹, 児玉英一郎, 王家宏, 高田豊雄: 共通言語基盤上における暗号アルゴリズムの効率的な実装手法, 2C2-5, SCIS2008.