剰余テーブルを循環する高速 RSA 暗号プロセッサのパイプラインアーキテクチャ

苫米地　宣　裕*

A Pipelined Architecture of High-Speed RSA Encryption Processor Using Rotation of Residue Table

Nobuhiro TOMABECHI*

Abstract

In the preceding study, the author presented a design of a high-speed RSA encryption processor using redundant binary number arithmetic and table-look-up. The purpose of this study is to improve the encryption rate of the processor when the plaintext data is continuously input. This paper newly presents a pipelined architecture in which the table data is circulated over all pipeline units in order to make use of a single residue table for all these units. It is demonstrated that the encryption rate for continuous input data of the presented processor is approximately 2.4 times that of conventional processors.

Key words: RSA cryptosystem, processor, pipeline, architecture, redundant binary number system, rotation, residue table

1. まえがき

近年のコンピュータネットワーク通信の発展とともに，通信のセキュリティを確保する高速暗号プロセッサの開発が期待されている。高速 RSA 暗号プロセッサの構成に関しては，これまで多くの報告がなされているが [1]-[6]，一層の高速性を有するプロセッサの実現が望まれている。RSA 暗号演算は，乗算と剰余演算から成る。通常，剰余演算はモントゴメリのアルゴリズムによって行われているが [3]，本方法は本質的に逐次演算であり，この方法では高速性の達成は困難と考えられる。

本研究者は，先に，冗長 2 進演算とテーブルルックアップに基づく高速 RSA 暗号プロセッサの構成法を提案した[7]。冗長 2 進演算とテーブルルックアップは，ともに本質的に並列演算であり，高速 RSA 暗号演算が可能となる。しかし，平文データが連続的に入力する場合，提案したプロセッサの暗号処理速度は，通常のプロセッサと比較してそれほど大きくはならない。

本論文では，先に提案したプロセッサのパイプラン構成，とくに，テーブルデータをパイプラインユニット上に循環させるというパイプライン構成を提案している。提案したプロセッサの連続入力データに対する暗号処理速度は，鍵の長さが 1,024 ビットの場合，通常のプロセッサの 2.4 倍となることが明らかとなった。

平成 16 年 12 月 17 日受理

* システム情報工学科・教授

2. 高速 RSA 暗号化アルゴリズム

はじめに，冗長 2 進演算とテーブルルックアップに基づく高速 RSA 暗号化アルゴリズムについて概観する[7]。鍵を (e, n)，鍵の長さを $N>=1{,}024$ ビット，平文を M，暗号文を C と表すと，RSA 暗号では，C は次式で与えられ

る。ただし，($x \bmod n$) は，x を n で割った剰余を表している。

$$C = M^e \bmod n$$

1 桁の冗長 2 進数を x^* のように表記する。x^* は，1, 0, −1 の 3 つの値をとる。N 桁の冗長 2 進数 x^* を次のように表す。

$$x^* = x^*_{N-1}2^{N-1} + x^*_{N-2}2^{N-2} + \cdots + x^*_0 = (x^*_{N-1}, x^*_{N-2}, \cdots, x^*_0).$$

鍵 e を，$e = e_{N-1}2^{N-1} + e_{N-2}2^{N-2} + \cdots + e_0$ と表すとき，C の値は以下のように求められる。

［**アルゴリズム 1**］**高速 RSA 暗号化アルゴリズム**

(Step 1-1) $y^*_1 = 1$

(Step 1-2) Step 1-3 から Step 1-7 までを $i = 1, 2, \cdots, N$ の間くり返す。

(Step 1-3) $y^*_2 = y^*_1 \times y^*_1$

(Step 1-4) $y^*_1 = y^*_2 \bmod n$

(Step 1-5) $e_{N-i} = 0$ ならば，Step 1-2 に戻る。
$e_{N-i} \neq 0$ ならば，Step 1-6 と Step 1-7 を実行する。

(Step 1-6) $y^*_2 = y^*_1 \times M$

(Step 1-7) $y^*_1 = y^*_2 \bmod n$.

(Step 1-8) $C = y^*_1$

剰余計算はテーブルルックアップによって，以下のように行われる。

［**剰余テーブルの構成**］

［構成 1］テーブルは H 個の RAM で構成される。RAM を RAM_i ($i = 0, 1, 2, ..., H-1$) と表す。

［構成 2］RAM のアドレスは L 桁の冗長 2 進数で表される。ただし，$L = N/H$ とする。

［構成 3］RAM_i ($i = 0, 1, 2, ..., H-1$) のアドレス $A(y^*_{L-1}, y^*_{L-2}, ..., y^*_0)$ に，次のデータ T_i をストアする。

$$T_i = (y^*_{L-1}2^{N+iL+L-1} + y^*_{L-2}2^{N+iL+L-2} + , ..., + y^*_0 2^{N+iL}) \bmod n$$

［**アルゴリズム 2**］**剰余計算**

入力データを $x^* = (x^*_{2N-1}, x^*_{2N-2}, ..., x^*_0)$ と表すと，$x^* \bmod n$ は以下のように計算される。

(Step 2-1) 以下の step を $j = 1, 2, 3, ..., R$ の間くり返す。

(Step 2-2) RAM_i ($i = 0, 1, 2, ..., H-1$) にアドレス $A(x^*_{N+iL+L-1}, x^*_{N+iL+L-2}, ..., x^*_{N+iL})$ を与え，出力 T_i を得る。

(Step 2-3) $y^* = \sum_{i=0}^{C(j)-1} T_i + (x^*_{N-1}, x^*_{N-2}, ..., x^*_0)$

ただし，$C(1) = H$，$C(j) = \log_2(C(j-1) + 1)/L$, ($j = 2, 3, 4, ..., R$) である。

Step 2-2 と Step 2-3 のくり返しの数 R は，N と L の値に依存する。例として，$N = 1{,}024$，$L = 2$ とすると，$R = 5$ となる。

3. 剰余テーブルを循環するパイプライン構成

アルゴリズム 1 では，乗算と剰余計算が $2N$ 回くり返される。(Step 1-3 および Step 1-4) または（Step 1-6 および Step 1-7）を実行する 1 組の乗算回路と剰余計算回路をユニットセルと呼ぶこととする。

連続データ入力に適したパイプラインアーキテクチャとして，次のような構成を提案する。

① $2N$ 個のユニットセルをパイプライン形式に直列接続する。

② 剰余テーブルデータを $2N$ 個のユニットセル上で循環させる。これによって，単一の剰余テーブルを $2N$ 個のユニットセルで共用する。

3.1 乗算回路の構成

［構成 4］ユニットセル内の乗算回路は，N ビット 1×ビット冗長 2 進乗算回路 1 個，$2N$ ビット冗長 2 進加算回路 1 個，$2N$ ビットラッチ，および，シフトレジスタで構成される。シフトレジスタは，被乗数と乗数をストアする。

［構成 5］乗算と加算を，演算数，被演算数をシ

フトしながら N 回くり返す。

3.2 パイプライン化剰余計算アルゴリズム

RAM のアドレスのビット数 $L=2$ ととる。アルゴリズム 2 を以下のように修正する。

［**アルゴリズム 3**］**修正剰余計算アルゴリズム**

入力データを $x^*=(x^*_{2N-1}, x^*_{2N-2}, ..., x^*_0)$ と表し，$x^* \bmod n$ を求める。

(Step 3-1) RAM_i $(i=0, 1, 2, ..., N/2-1)$ にアドレスデータ $A(x^*_{N+2i+1}, x^*_{N+2i})$ を与え，出力 T_i を得る。

(Step 3-2) $y^*=\sum_{i=0}^{N/2-1} T_i+(x^*_{N-1}, x^*_{N-2}, ..., x^*_0)$

(Step 3-3) Step 3-4 から Step 3-6 までを 4 回くり返す。

(Step 3-4) $x^*=(y^*_{N+10}, y^*_{N+9}, ..., y^*_N, ..., y^*_0)$.

(Step 3-5) RAM_i $(i=0, 1, 2, ..., N/2-1)$ にアドレスデータ $A(x^*_{N+2i+1}, x^*_{N+2i})$ を与え，出力 T_i を得る。

(Step 3-6) $y^*=\sum_{i=0}^{4} T_i+(x^*_{N-1}, x^*_{N-2}, ..., x^*_0)$

3.3 剰余計算回路の構成

アルゴリズム 3 に基づき，剰余計算回路を次のように構成する。

［構成 6］ユニットセル内の剰余計算回路は，RAM, $2N$ ビット冗長 2 進加算回路，ラッチ，および，被乗数をストアするシフトレジスタから成る。RAM は剰余テーブルの 1 部をストアする。

［構成 7］RAM のアドレスは冗長 2 進 2 ビット，出力は N ビットとする。

［構成 8］$2N$ 個のユニットセルに対して，剰余テーブルは 1 個だけ用意する。すなわち，剰余テーブルは，各ユニットセルに分散された形式でストアされる。

［構成 9］各ユニットセルでは，テーブルルックアップは，$N/2+20$ 回くり返される。

［構成 10］Step 3-1 から Step 3-2 までの演算をプロセス 1，Step 3-3 から Step 3-6 までの演算をプロセス 2 と呼ぶ。

(1) プロセス 1

［構成 11］各ユニットセルは 1 個の RAM を含み，プロセス 1 に対応する剰余テーブルの 1 部をストアする。

［構成 12］最初のタイミングにおいて，i 番目のユニットセル $(i=0, 1, 2, \cdots, (2N-1))$ は，$((y^*_1 2^{N+2k+1}+y^*_0 2^{N+2k}) \bmod n)$ の値をストアする。ただし，$k=(2i \bmod N)$ とし，y^*_1 および y^*_0 は任意の冗長 2 進数とする。

［構成 13］テーブルデータは，タイミングクロックに同期しながら，$2N$ 個のユニットセル上を循環する。すなわち，j 番目のタイミングにおいて $(j=0, 1, 2, \cdots, N/2)$，i 番目のユニットセルに，$((y^*_1 2^{N+2k+1}+y^*_0 2^{N+2k}) \bmod n)$ の値をストアする。ただし，$k=((i+j) \bmod N)$ とする。

［構成 14］i 番目のユニットセルは，j 番目のタイミングにおいて，剰余計算を被演算数シフトレジスタから (x^*_{N+k+1}, x^*_{N+k}) をシフトアウトし，これを テーブルデータをストアする RAM のアドレスに与えることによって実行する。ただし，$k=((i+j) \bmod N)$ とする。

(2) Process 2

プロセス 2 に対応する剰余テーブルは，プロセス 1 と同様に構成する。ただし，テーブルは 5 個の RAM で構成される。

［構成 15］各ユニットセルは，プロセス 1 とは別のプロセス 2 に対応する 1 個の RAM を有する。

［構成 16］最初のタイミングにおいて，i 番目のユニットセル $(i=0, 1, 2, \cdots, (2N-1))$ は，$((y^*_1 2^{N+2k+1}+y^*_0 2^{N+2k}) \bmod n)$ をストアする。ただし，$k=(2i \bmod 5)$ とする。

［構成 17］テーブルデータは，タイミングクロックに同期して，$2N$ 個のユニットセル上を循環する。すなわち，j 番目のタイミング $(j=0, 1, 2, \cdots, 19)$ において，i 番目のユニットセルは，$((y^*_1 2^{N+2k+1}+y^*_0 2^{N+2k}) \bmod n)$ をストアす

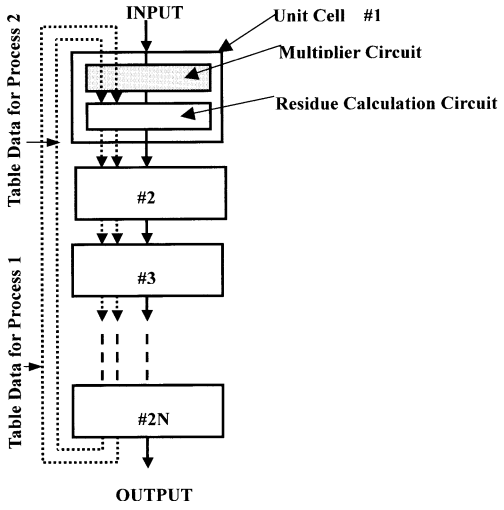

図 1　プロセッサのブロック図
Fig. 1　Circuit diagram of the processor

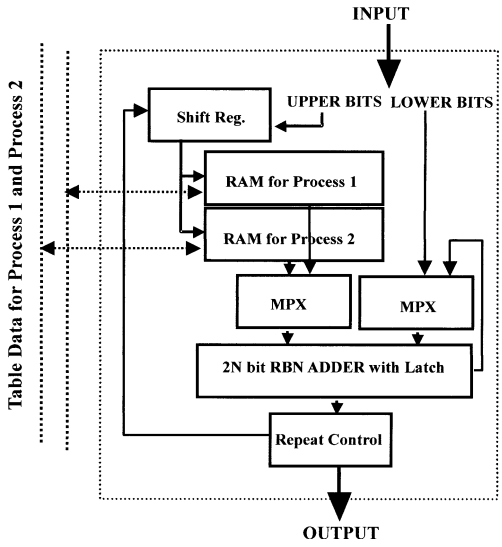

図 2　剰余計算回路のブロック図
Fig. 2　Block diagram of a residue calculation circuit

る。ただし，$k=((i+j) \bmod 5)$ とする。

［構成 18］i 番目にユニットセルは，j 番目のタイミングにおいて，剰余計算は，被演算数シフトレジスタから (x^*_{N+k+1}, x^*_{N+k}) をシフトアウトし，これを テーブルデータをストアする RAM のアドレスに与えることによって実行する。ただし，$k=((i+j) \bmod 5)$ とする。

［構成 19］剰余計算回路の演算時間を乗算回路の演算時間と等しくするために，プロセス 2 の後に，遅延素子を挿入する。

図 1 に，プロセッサのブロック図を示している。

提案したプロセッサはパイプライン動作をとるので，その暗号処理速度は，ユニットセル 1 個の処理速度と等しくなる。

4.　暗号化速度

4.1　提案したプロセッサの暗号化速度

連続暗号化速度 V_c を，平文データが連続的に入力された場合の 1 秒当たりの処理可能データ量と定義する。

ゲート 1 個の遅延時間を t_g，1 ビット冗長 2 進乗算回路の遅延を t_{M0}，1 ビット冗長 2 進加算回路の遅延を t_{A0}，RAM の遅延を t_{R0}，ラッチの遅延を t_{F0}，パイプライン動作のクロック間隔を t_0，乗算回路または剰余計算回路で処理されるデータ量を D と表す。

$t_{M0} \cong t_{R0}$ なので，次のようにとる。

$$t_0 = t_{M0} + t_{A0} + t_{F0}.$$

$D=N$ なので，V_c は次のようになる。

$$V_c = D/(Nt_0) = N/(Nt_0) = 1/t_0.$$

文献[7]より $t_0 \cong 10\ t_g$ が得られるので，V_c は次となる。

$$V_c \cong 1/(10\ t_g)$$

よって，次の結果が得られる。

［結果 1］提案したプロセッサの暗号化速度は，連続データ入力の場合，約 $1/(10\ t_g)$ (bits/s) となる。ただし，t_g はゲート 1 個の遅延時間を表している。

今，典型的な値として，t_g=0.1ns を用いると，V_c=1 (Gbits/s) となる。

4.2 従来の方法との比較

RSA 暗号プロセッサの構成に関しては，これまで多くの報告がなされている。典型的なプロセッサとして，ここでは文献 [3] に記載されたプロセッサをとる。文献［3］のプロセッサは，本研究と同じ種類のデバイスを用いており，かつ，処理速度を支配するクリティカルパスに含まれるゲートの数が，本質的に，最小であると考えられる。

文献 [3] のプロセッサの連続データに対する暗号化速度は，$1/(24\ t_g)$ とすることができる。従って，次の結果が得られる。

［結果 2］提案したプロセッサの連続データに対する暗号化速度は，従来のプロセッサの約 2.4 倍である。

5. ま と め

本論文では，剰余テーブルデータをパイプラインユニット上に循環させるという高速 RSA 暗号プロセッサの構成法を提案した。

本プロセッサの連続データに対する暗号化速度は，鍵の長さを 1,024 ビットとすると，従来のプロセッサの約 2.4 倍となることが明らかとなった。

提案したアーキテクチャの特長は，VLSI チップの集積度が許容できるならば，ユニットセルをより小さなパイプライン構造に容易に分割できる点にある。このことは，連続データに対する高い暗号化速度の実現可能性を示している。今後，プロセッサ VLSI チップの試作を行う予定である。

本研究は，平成 16 年度文部科学省科学技術研究費基盤研究（C）(2) の補助を受けたことを付記する。

参 考 文 献

[1] Brickell E.F. “A survey of hardware implementations of RSA encryption processors,” Advances in Cryptology–CRYPTO ’89, Springer–Verlag, pp. 368–370, 1990.

[2] Kameyama M., Wei S., Higuchi T. “Design of a RSA encryption processor based on signed–digit multivalued arithmetic circuits,” Systems and Computers Japan, Vol. 21, pp. 21–31, 1990.

[3] Eldridge S.E., Walter C.D. “Hardware implementation of Montgomery’s modular multiplication algorithm,” IEEE Trans. on Computers, Vol. 42, No. 6, pp. 693–699, June, 1993.

[4] Chen P.S., Hwang S.A., Wu C.W. “A systolic RSA public key cryptosystem,” Proc. ISCAS, Vol. 4, pp. 408–411, 1996.

[5] Ishii S., Oyama K., Yamanaka K. “High–speed public key encryption processor,” IEICE Japan Trans. (D–I), Vol. J80–D–I, No. 8, pp. 725–735, Aug., 1997.

[6] Yang C.C., Chang T.S., Jen C.W. “A new RSA cryptosystem hardware design based on Montgomery’s algorithm,” IEEE Trans. Circuits and Systems, Vol. 45, No. 7, pp. 908–913, July 1998.

[7] Tomabechi N., Ito T. “Design of a high–speed RSA encryption processor based on the residue table for redundant binary numbers,” Systems and Computers in Japan, Vol. 33, No. 5, pp. 1–10, May 2002.