［ハードウェアセキュリティの最新動向］

②ハードウェアを用いた暗号処理の高速化

坂本純一 横浜国立大学　吉田直樹 横浜国立大学

IoT 時代の暗号技術

インターネット上を流れるデータには盗聴や改ざんの危険があるため，暗号技術を用いたデータ保護が求められる．すべての“モノ”がインターネットに接続される IoT（Internet of Things）時代には膨大な数のユーザがインターネット上で通信を行うため，利用される暗号技術にもそれぞれのユーザのニーズに合わせた“機能”が求められている．何らかの特徴的な機能を持つ暗号は“高機能暗号”と呼ばれ，IoT 時代に表出する課題を解決するものとして期待・開発されている．高機能暗号の 1 つの例として，集約署名を図 -1 に示す．この図は左側の LAN（Local Area Network）内の n 個の機器が送信したディジタル署名をゲートウェイが集約（1 つにまとめる）してから WAN（Wide Area Network）側に送信している．集約することで WAN 上を流れるデータ量が削減できるため通信時間の短縮やネットワーク帯域圧迫を回避することができる．またサーバ側では集約された署名を検証するだけで n 個のディジタル署名を一度に検証できるため，検証にかかる時間や電力を節約できる．

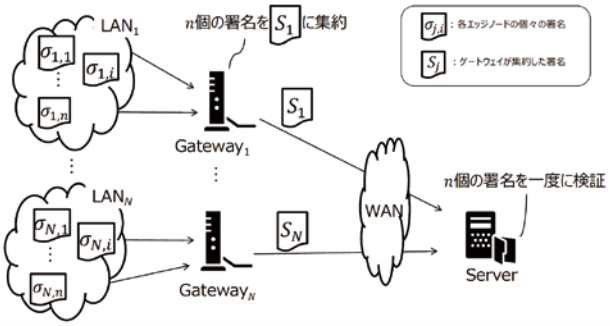

図 -1　高機能暗号の例：集約署名

集約署名のほかにも秘匿検索や代理人再暗号化など高機能暗号の例は枚挙にいとまがない．例に挙げた高機能暗号はペアリング演算と呼ばれる複雑な数学的操作を利用することで効率的に構築できることが知られている．しかしこのペアリング演算自体の計算コストは高く，高機能暗号の実用化に向けてペアリング演算の計算性能向上が必要とされている．

ペアリング演算に求められる性能

ペアリング演算のアプリケーション例として，図 -1 に示した集約署名プロトコルを考える． LAN 側の署名生成や集約は低コスト計算で構成され，サーバ側の署名検証にのみペアリング演算を必要とする集約署名プロトコルが知られている．署名生成・集約は計算リソースの乏しい IoT 機器でも達成可能である一方で，サーバ側の署名検証には潤沢な計算資源を用いた高速なペアリング演算が求められる．ここでいう高速性はレイテンシを指す．サーバは不定期に集約署名を受け取るため一度に大量のデータを処理するというよりも受け取った署名をいかに素早く検証するか―すなわちレイテンシ―が重要である．レイテンシを低減するためにはソフトウェア実装よりもハードウェア実装が有効である．

表 -1 に代表的なハードウェア実装ペアリング演算の計算時間を示す．この表の実装結果は実装対象や費やした回路リソース量が異なるため公平な比較ではないが，上記シナリオを考慮した場合最も適しているものは，計算時間が最も短い文献 1）である．

本稿ではハードウェアを用いたペアリング演算の高速化に焦点を置き，FPGA 実装において最速のレイテンシを達成しているハードウェア構成[1]を特に解説する．

ペアリング演算

暗号に用いられるペアリング演算は入力 P，Q に対する二項演算

$$e:G_1 \times G_2 \to G_3$$
$$e(P,Q) \mapsto R$$

で次の 3 つの性質

1. 非縮退性：任意の P（または Q）に対して $e(P,Q)=1$ なら $Q=0$（または $P=0$）
2. 双線形性：$e(P_1+P_2,Q)=e(P_1,Q)\cdot e(P_2,Q)$
   $e(P,Q_1+Q_2)=e(P,Q_1)\cdot e(P,Q_2)$
3. 計算可能性：多項式時間で計算可能

を満たすものである．G_1，G_2 を素数 p に対する有限体 F_p 上の楕円曲線群とすることで効率的なペアリング演算を構成できることが知られており，ペアリング演算は細かく分解していくと F_p 演算で構成されている．ペアリング演算の効率的な計算アルゴリズムを提案している文献 2）によれば，実用的なセキュリティレベルの（254 ビット素数位数体上で構成される）ペアリング演算には約 1 万回の F_p 乗算，約 5.8 万回の F_p 加算，4 回の F_p 逆元計算が必要とされている．したがってほとんどのペアリングプロセッサは剰余加算器，剰余乗算器，逆元演算器で構成されることが多い．またほとんどの F_p 演算は F_p 上の既約多項式で定義された 2 次拡大体 $F_{p^2}=F_p[i]/(i^2-\beta)$，$\beta=-1$ 演算に利用されるため，F_p 多項式演算の高速化も重要となる．

表 -1　代表的なハードウェア実装ペアリング演算の計算時間

	実装対象	回路リソース	時間
[1]	xcvu9p-2	5421 CLBs, 460 DSPs	61 μs
	xc6vlx240t-1	11672 slices, 500 DSPs	153 μs
[3]	xc6vlx240t-3	5163 slices, 144 DSPs	375 μs
[4]	xc6vlx240t-1	5237 slices, 64 DSPs	409 μs
[5]	65 nm ASIC	323k Gates	554 μs
[6]	65 nm ASIC	3205k Gates	62 μs

低レイテンシペアリングプロセッサの構成

文献 1）で提案されている低レイテンシペアリングプロセッサの概要図を図 -2 に示す．このプロセッサは前置加算器，剰余乗算器，定数倍演算器，後置加算器，逆元演算器の 5 つの演算器で構成されている．シーケンサにはプログラムに相当するものが格納されており，あらかじめ最適にスケジューリングされた計算順序に従って各演算器を駆動する．このように複数の演算器を持ち，それらをペアリング計算のために制御する機構を持つハードウェアのことをペアリングプロセッサと呼ぶ．逆元演算器以外の 4 つの演算器は直列に接続されており，F_p 多項式の演算に特化している．たとえば F_{p^2} の自乗算

$$z_0+z_1 i=(x_0+x_1 i)^2$$
$$=(x_0+x_1)\cdot(x_0-x_1)+2x_0x_1 i$$

を計算する際にはまず x_0+x_1 および x_0-x_1 が前置加算器で計算され，その結果の積が剰余乗算器で計算される．$2x_0x_1$ は剰余乗算器および定数倍演算器で計算される．このような小さな定数による乗算はシフト操作により低コストで実装できるため，剰余乗算器を 2 度利用するよりも効率的である．細かな差異はあるが，ほとんどのペアリングプロセッサがこのように演算器を直列に接続し，多項式演算を高速化する構成をとっている[1), 3)～6)]．

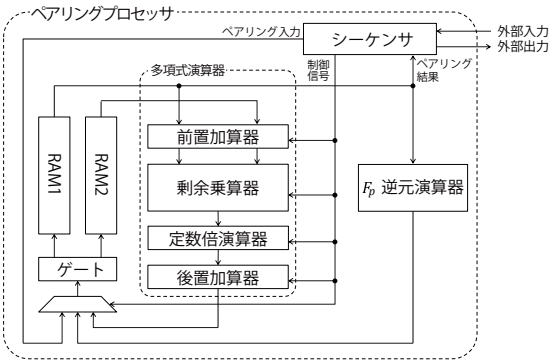

図 -2　ペアリングプロセッサの構成例

逆元演算はペアリング演算中に数回しか現れないため，専用の演算器を持たずにフェルマーの小定理に従い剰余乗算器を使って計算する構成も存在する[4), 5)]．しかしながら逆元演算は非常にコストがかかる計算であるため専用演算器による回路リソースを増やしてでも高速に実行したい．拡張ユークリッド互除法を利用して逆元を計算する専用演算器が一般的である[1), 3), 6)]．

剰余乗算器の高速化

F_p 演算の中でも，剰余乗算は最も計算コストが高くボトルネックとなる部分である．剰余乗算はペアリング演算のほかにも RSA や楕円曲線暗号などの中核を成す計算であり，高速な剰余乗算器は多くのシステムに望まれている．剰余乗算とは $a \times b \bmod N$ のように乗算結果の余りを求める演算である．単純には乗算後に除算（正確には余算）を行うことで計算できるが，除算は非常にコストのかかる演算であるためなるべく排除したい．そのような場合に利用されるのがモンゴメリ乗算に代表される剰余乗算アルゴリズムである．

表 -2 に示すようにモンゴメリ乗算は $a \times b \bmod N$ の代わりに $A \times B \times R^{-1} \bmod N$ を計算する．ここで gcd(·,·) は 2 つの入力の最大公約数を示している．ふつう R は 2 のベキに設定され，2 行目と 3 行目に現れる R による余算および除算はそれぞれビットマスクおよびシフト演算によって効率的に計算可能である．モンゴメリ乗算の入出力は計算したい数に R が乗じられた表現になっているため，最初に R を乗じる処理と最後に R で除する処理のオーバヘッドが生じるが，何度も剰余乗算を行うアプリケーションにおいては相対的にオーバヘッドが小さくなり，高速化に有効な手法である．

文献 3)，6) が表 -2 のモンゴメリ乗算アルゴリズムを採用している．このアルゴリズムは強力だが，ハードウェアに実装しようとすると乗算器の大きさが問題になる．実用的なセキュリティレベルのペアリング演算の場合 F_p の元のサイズが 254bit 程度になるため，表 -2 の 1 行目や 3 行目に現れる乗算には 254bit × 254bit という巨大な乗算器が必要になる．このように大きな乗算器を効率的に実装するのは容易ではないため，乗算幅をより小さな単位に分割したアルゴリズム（表 -3）が提案されている．ここで乗数は $B=\sum_{i=0}^{n-1}(2^k)^i b_i, b_i \in \{0,1,\cdots,2^k-1\}$ というふうに k ビットごとに分割して表現されている．このアルゴリズムはソフトウェア実装では一般的だが，計算依存性[☆1]が高いためにハードウェア実装ではあまり用いられない．図 -3 に k ビット分割モンゴメリ乗算の for ループ 1 段をハードウェア実装し

表 -2　モンゴメリ乗算アルゴリズム

Input:	被乗数 $A=aR \bmod N$, 乗数 $B=bR \bmod N$, 法 N ($N>2$, $\gcd(N,2)=1$), $NN' \bmod R=-1$
Output:	$A \times B \times R^{-1} \bmod N$
1:	$T=A \times B$
2:	$m=TN' \bmod R$
3:	$t=(T+m \times N)/R$
4:	**if** $t > N$ **then**
5:	$t=t-N$
6:	**endif**
7:	**return** t

表 -3　k ビット分割モンゴメリ乗算アルゴリズム

Input:	分割幅 k，分割数 n，被乗数 A，乗数 B, 法 N ($N>2$, $\gcd(N,2)=1$), $NN' \bmod R=-1$
Output:	$A \times B \times R^{-1} \bmod N$
1:	$S_0=0$
2:	**for** $i=0$ **to** $n-1$ **do**
3:	$q_i=(((S_i+b_iA) \bmod 2^k)N') \bmod 2^k$
4:	$S_{i+1}=(S_i+q_iN+b_iA)/2^k$
5:	**endfor**
6:	**return** S_n

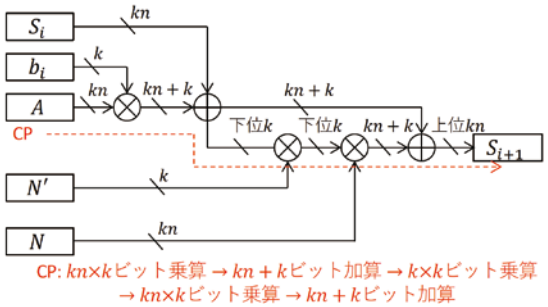

図 -3　k ビット分割モンゴメリ乗算のデータフローとクリティカルパス

☆1　計算依存性：ある計算を始めるために多くのほかの計算結果が必要な場合，計算依存性が高いという．

た際のデータフローを示す．乗算器と加算器が 5 つ直列に並んでおり，並列実行による高速化が見込めないことが分かる．このようなレジスタ間の計算パスのうちハードウェア上で最も長い部分をクリティカルパス（CP）と呼ぶ．長い CP ほど信号の伝達に時間がかかるため，回路を高速に動かすことができなくなる．CP 遅延の削減はハードウェア高速化に必須の要素である．

表 -3 のアルゴリズムの計算依存性を削減し，CP を短くしたものが表 -4 に示す Quotient Pipelining Montgomery Multiplication（QPMM）アルゴリズムである．図 -4 に QPMM の for ループ一段のデータフローを示す．明らかに計算個所が少なくなっており，クリティカルパスはわずか乗算と加算を 1 回ずつ通るのみである（2 つの乗算は依存関係がないため並列に実行できる）．しかしながら演算の幅は図 -3 と比べて増加しており，さらにループ段数も増加している．これは QPMM アルゴリズムが冗長計算を行うことで計算依存性を削減していることに起因する．このような欠点により QPMM は大量のハードウェアを利用してでも高速性が求められるアプリケーションに適しており，文献 1)，5) で提案されているペアリングプロセッサに利用されている．

表 -4　QPMM アルゴリズム

Input:	分割幅 k，遅延定数 d，ブロック数 n，被乗数 A，乗数 B，法 $N(N>2, \gcd(N,2)=1)$, $R=2^{kn}$, $NN' \bmod 2^{k(d+1)}=-1$, $\tilde{N}=(N' \bmod 2^{k(d+1)})N$, $N''=((\tilde{N}-1))/2^{k(d+1)}$
Output:	$S_{n+d+2} \equiv ABR^{-1} \bmod N$

1: $S_0=0, q_{-d}=0, \cdots, q_{-1}=0$
2: **for** $i=0$ **to** $n+d$ **do**
3: $\quad q_i=S_i \bmod 2^k$
4: $\quad S_{i+1}=S_i/2^k+q_{i-d}N''+b_iA$
5: **endfor**
6: $S_{n+d+2}=2^{kd}S_{n+d+1}+\sum_{j=0}^{d-1} q_{n+j+1}2^{kj}$
7: **return** S_{n+d+2}

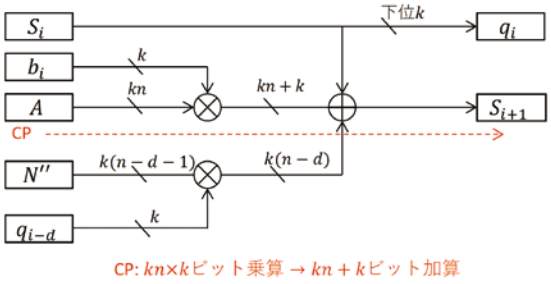

図 -4　QPMM のデータフローとクリティカルパス

今後の展望

本稿ではハードウェアによるペアリング演算の高速化について解説した．IoT 時代の暗号技術である高機能暗号を実現するためにはペアリング演算高速化は不可欠であり，特にサーバサイドアプリケーションでの利用が期待される．また本稿で解説した剰余乗算アルゴリズムは公開鍵暗号全般に利用されるものであるため，今後さらなる高速実装が提案されていくと考えられる．

参考文献

1) Sakamoto, J. et al. : Low-Latency Pairing Processor Architecture Using Fully-Unrolled Quotient Pipelining Montgomery Multiplier, in Proc. of 2019 Asian Hardware Oriented Security and Trust Symposium.
2) Aranha, D. F. et al. : The Realm of the Pairings, in Revised Selected Papers on Selected Areas in Cryptography - SAC 2013.
3) Ghosh, S. et al. : Core Based Architecture to Speed Up Optimal Ate Pairing on FPGA Platform, in Proc. of International Conference on Pairing-Based Cryptography (Pairing 2012) (2012).
4) Yao, G. X. et al. : Faster Pairing Coprocessor Architecture, in Proc. of International Conference on Pairing-Based Cryptography (Pairing 2012), pp.160-176 (2012).
5) Han, J. et al. : A 65 nm Cryptographic Processor for High Speed Pairing Computation, in IEEE Trans. on VLSI, Vol.23, Issue 4, pp.692-701 (2015).
6) Awano, H. et al. : An ASIC Crypto Processor for 254-Bit Prime-Field Pairing Featuring Programmable Arithmetic Core Optimized for Quadratic Extension Field, IEICE Trans. Fundamentals, Vol.E102-A (2019).

（2020 年 1 月 31 日受付）

坂本純一　sakamoto-junichi-ws@ynu.ac.jp
2020 年横浜国立大学大学院環境情報学府博士課程後期修了．博士（情報学）．現同大学院産学官連携研究員．サイドチャネル攻撃やレーザフォールト攻撃，高機能暗号実装の研究に従事．

吉田直樹　yoshida-naoki-jb@ynu.ac.jp
2017 年横浜国立大学大学院環境情報学府博士課程修了．博士（情報学）．2018 年より同大学特任助教．組込みセキュリティの研究に従事．