MPLS JAPAN 2026

耐量子計算機暗号（PQC）の最新動向及びネットワーク分野における移行課題について

高橋 康（パナソニック ホールディングス株式会社）

2026年10月26日

Panasonic Business Development Division

自己紹介

高橋 康



業務内容

IoT機器の耐量子計算機暗号（PQC）移行支援を目的としたR&D

- **動作評価・規格調査**
- **移行課題抽出・解決技術開発**
- **移行支援コンサルティング**

本発表内容

① PQCとは何か › ② なぜ移行が必要 › ③ 移行課題 › ④ 実装改良

伝えたいこと

PQC移行は**複数の国・企業で開始 → 2030年代には出荷不可の恐れ**

PQCは**メモリ使用量が大きい → 省リソース機器では動作困難**

省メモリ（＆高速化）のための改良研究が盛ん

耐量子計算機暗号（PQC）とその動向

耐量子計算機暗号（PQC）とは

量子計算機の脅威に備えた、新しい公開鍵暗号アルゴリズムのこと

※PQC = Post-Quantum Cryptography

現在の公開鍵暗号

- 認証　：**RSA, ECDSA**
- 鍵共有：**ECDH**



PQC

- 認証　：**ML-DSA, FN-DSAなど**
- 鍵共有：**ML-KEM, HQC**



PQCは未来技術ではなく、既に標準化・利用されている技術

参考：量子鍵配送（QKD）とは別の技術

- QKD：量子計算機、量子通信などの新しいインフラが必要
- PQC：従来インフラで利用可能

耐量子計算機暗号（PQC）に関する各国・企業動向

米国

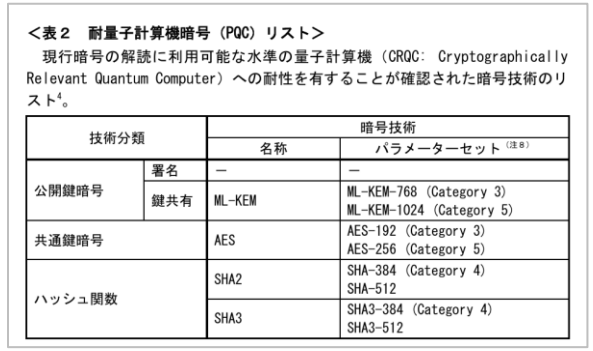
- 現在の公開鍵暗号を**2035年末に使用停止** (2024,NIST)
- 重要インフラを**2030年までにPQC移行** (2026, 大統領令)



Securing the Nation Against Advanced Cryptographic Attacks – The White House

日本

- **2035年までにPQC移行目標** (2025年 内閣府)
- **利用推奨PQCリスト発行** (2026年 CRYPTRECから)



＜表２　耐量子計算機暗号（PQC）リスト＞

現行暗号の解読に利用可能な水準の量子計算機（CRQC: Cryptographically Relevant Quantum Computer）への耐性を有することが確認された暗号技術のリスト[4]。

技術分類		暗号技術	
		名称	パラメーターセット（注８）
公開鍵暗号	署名	－	－
	鍵共有	ML-KEM	ML-KEM-768（Category 3） ML-KEM-1024（Category 5）
共通鍵暗号		AES	AES-192（Category 3） AES-256（Category 5）
ハッシュ関数		SHA2	SHA-384（Category 4） SHA-512
		SHA3	SHA3-384（Category 4） SHA3-512

電子政府における調達のために参照すべき暗号のリスト（CRYPTREC暗号リスト）

シンガポール

- 重要なインフラを**2031年までにPQC移行** (2026, CSA)



CSA Releases A Quantum-Safe Handbook And Quantum Readiness Index | Cyber Security Agency of Singapore

耐量子計算機暗号（PQC）に関する各国・企業動向

Google

- ChromeにてMLKEMを導入 (2024年)



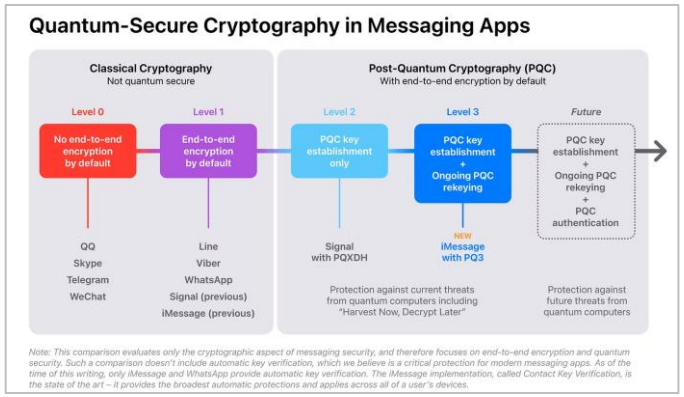
Security for the Quantum Era: Implementing Post-Quantum Cryptography in Android

Apple

- iMessageにてECC+Kyberを導入 (2024年)

Cloudflare

- Web/APIにてX25519+ML-KEM768を導入 (2022年)



AppleのiMessageがPQ3という耐量子計算機暗号(PQC)を導入。レベル2のSignalよりも更に量子セキュアなレベル3 に達した最初のメッセージングプロトコルとなった

Signal

- PQXDH機能にてKyberを導入 (2023年)

PQC移行の必要性と課題

なぜPQC移行が必要か？

量子計算機による暗号危殆化に備えるためもあるが…

移行しないと出荷・展開できなくなる恐れ

EU CRA

- 製品ライフサイクル向けのセキュリティ制度
- **「state-of-the-artな暗号を利用」の要件**

JC-STAR

- IoT製品セキュリティへの評価制度
- **「ベストプラクティスな暗号を利用」の要件**

CC (Common Criteria)

- 政府向けセキュリティ製品の認証制度
- **2027年1月からPQCを政府調達要件へ**

製品の価値を保つために、PQCが必須

出典: EU Regulation 2024/2847 Annex I; European Commission CRA; IPA JC-STAR

PQC移行における課題

現代暗号とPQCのスペック差がある ※方式によってその大きさは異なる

処理速度

- 現代暗号より**約2倍高速**
- 実用上は問題ない

メモリ使用量

- 現代暗号より**約60倍増加**
- **実装・理論改良余地あり**

鍵サイズ

- 現代暗号より**約50倍増加**
- 安全性強度に直結

メモリ使用量の低減を目指した研究が盛ん

参考：現代暗号との相互接続課題

- その他の課題として、PQC通信と現代暗号通信が相互接続できない問題がある
- これに対しては、ハイブリッド手法により対処可能だが、鍵・証明書保持コストが増える

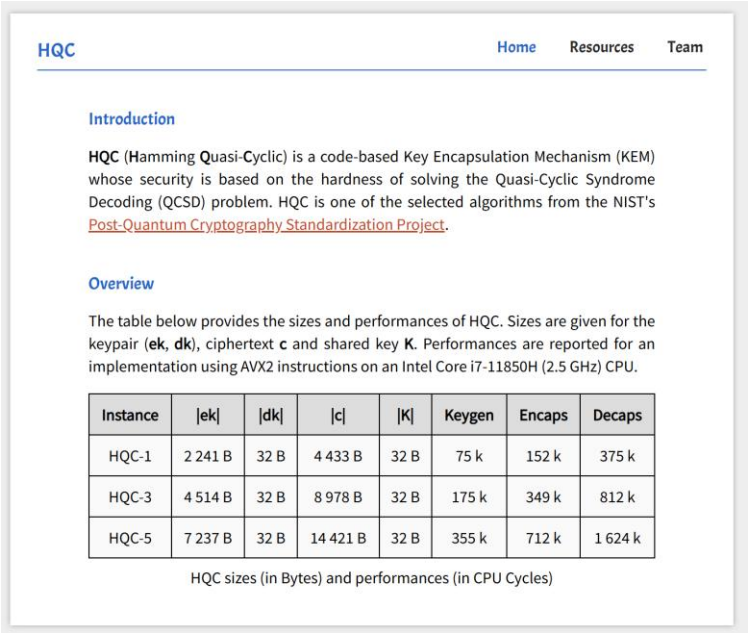
PQCに対する実装・理論改良紹介

今回発表：HQCの高速化・軽量改良技術

HQCとは：NIST標準のPQC鍵交換方式の一つ

数理的背景

- 誤り訂正符号（Quasi-Cyclic構造）
- Reed-Muller符号とReed-Solomon符号を併用



HQC　Home　Resources　Team

Introduction

HQC (**H**amming **Q**uasi-**C**yclic) is a code-based Key Encapsulation Mechanism (KEM) whose security is based on the hardness of solving the Quasi-Cyclic Syndrome Decoding (QCSD) problem. HQC is one of the selected algorithms from the NIST's Post-Quantum Cryptography Standardization Project.

Overview

The table below provides the sizes and performances of HQC. Sizes are given for the keypair (**ek**, **dk**), ciphertext **c** and shared key **K**. Performances are reported for an implementation using AVX2 instructions on an Intel Core i7-11850H (2.5 GHz) CPU.

Instance	|ek|	|dk|	|c|	|K|	Keygen	Encaps	Decaps
HQC-1	2 241 B	32 B	4 433 B	32 B	75 k	152 k	375 k
HQC-3	4 514 B	32 B	8 978 B	32 B	175 k	349 k	812 k
HQC-5	7 237 B	32 B	14 421 B	32 B	355 k	712 k	1 624 k

HQC sizes (in Bytes) and performances (in CPU Cycles)

https://pqc-hqc.org/

将来製品に搭載される可能性が高いPQCアルゴリズムの一つ

参考：他のPQC鍵交換アルゴリズム（ML-KEM）との関係

- 総合的にはML-KEMの方が優れた性能であり、主としてはこちらが使われる（と思われる）
- ML-KEMが破られた場合の代替方式として、HQCが標準方式として選ばれた

今回発表：HQCの高速化・軽量改良技術

HQCの性能と課題

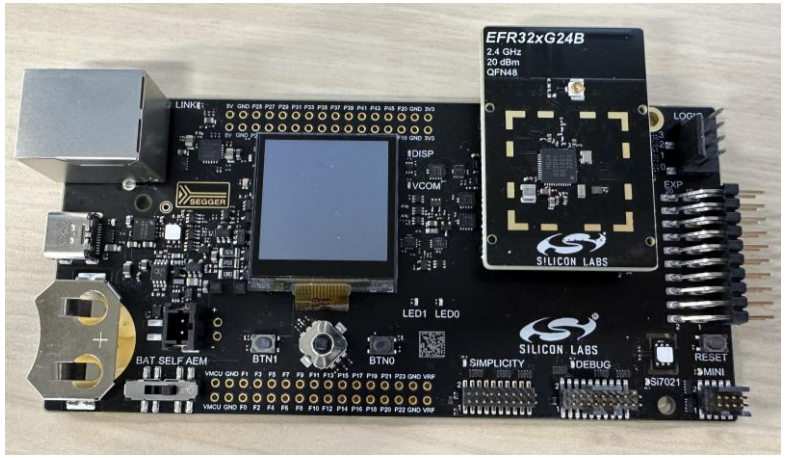
処理時間

鍵生成：　5.9 s
暗号化：11.8 s
復号　：17.8 s

メモリ使用量

鍵交換：114.8 KB
暗号化：178.2 KB
復号　：192.4 KB

処理時間：1sを超えると実用性が低い
メモリ使用量：256KBチップで実装困難



評価環境：Silicon Labs EFR32MG24
（ARM Cortex-M33, 39MHzで動作, 256KB RAM）

今回発表：HQCの高速化・軽量改良技術

貢献：HQCの省リソース環境向けソフトウェア実装”MUTUALISM”を提案

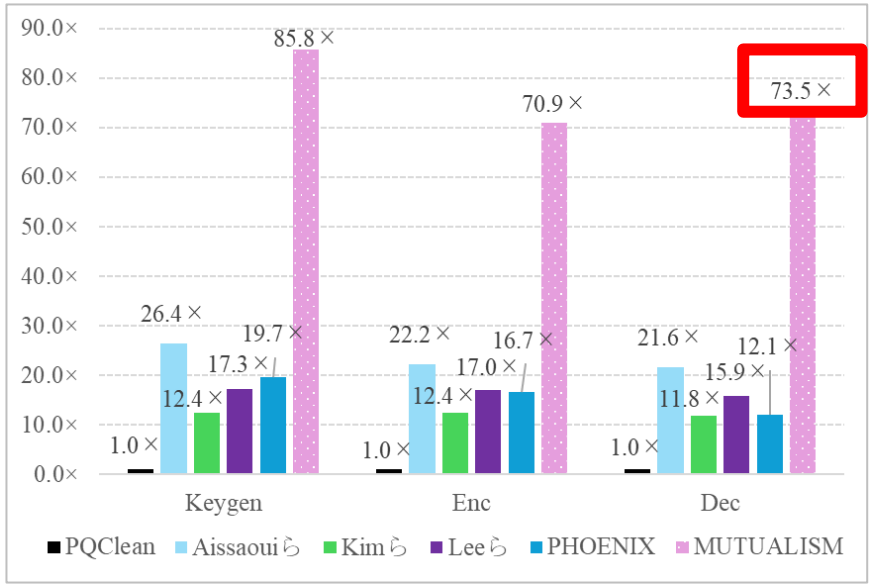
MUTUALISM: Low-Footprint and High-Throughput Software Implementation of HQC for Resource-Constrained Devices | Proceedings of the 19th ACM Conference on Security and Privacy in Wireless and Mobile Networks

改良方針①

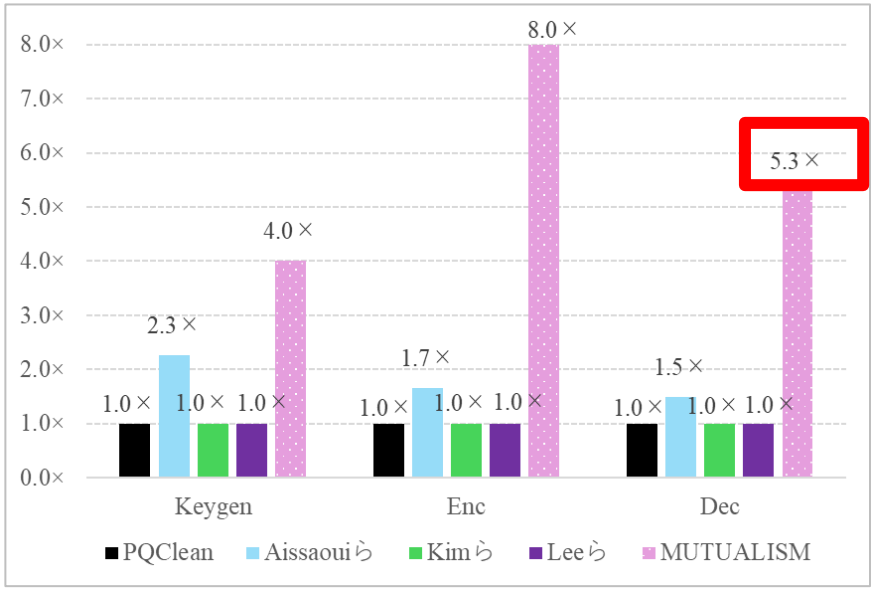
処理の中で**多項式乗算の割合が大きい**
-> **多項式乗算を改良（約70倍）**

改良方針②

既存研究で**多項式乗算改良以外が不足**
-> **HQC全体の処理最適化（約5倍）**



処理時間改善倍率（PQClean/各手法）



メモリ改善倍率（PQClean/各手法）

今回発表：HQCの高速化・軽量改良技術

貢献：HQCの省リソース環境向けソフトウェア実装”MUTUALISM”を提案

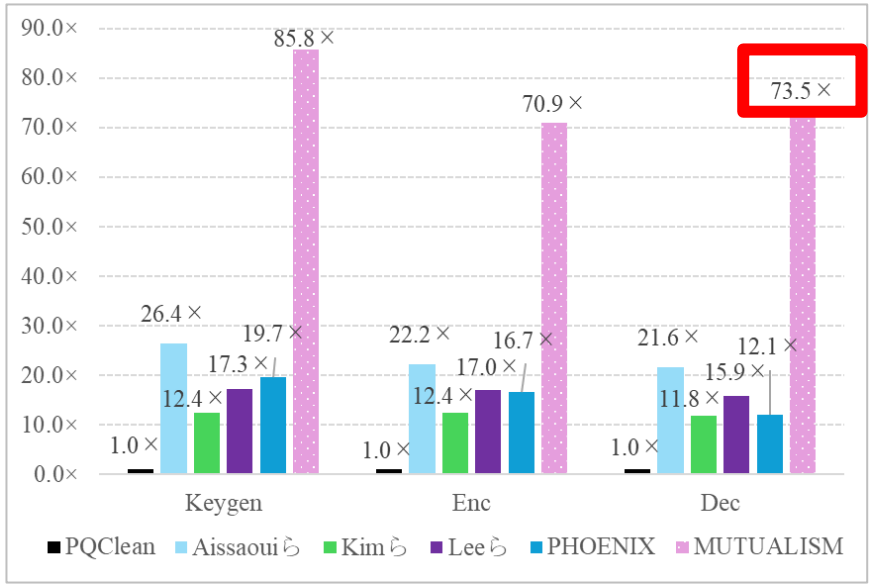
MUTUALISM: Low-Footprint and High-Throughput Software Implementation of HQC for Resource-Constrained Devices | Proceedings of the 19th ACM Conference on Security and Privacy in Wireless and Mobile Networks

改良方針①

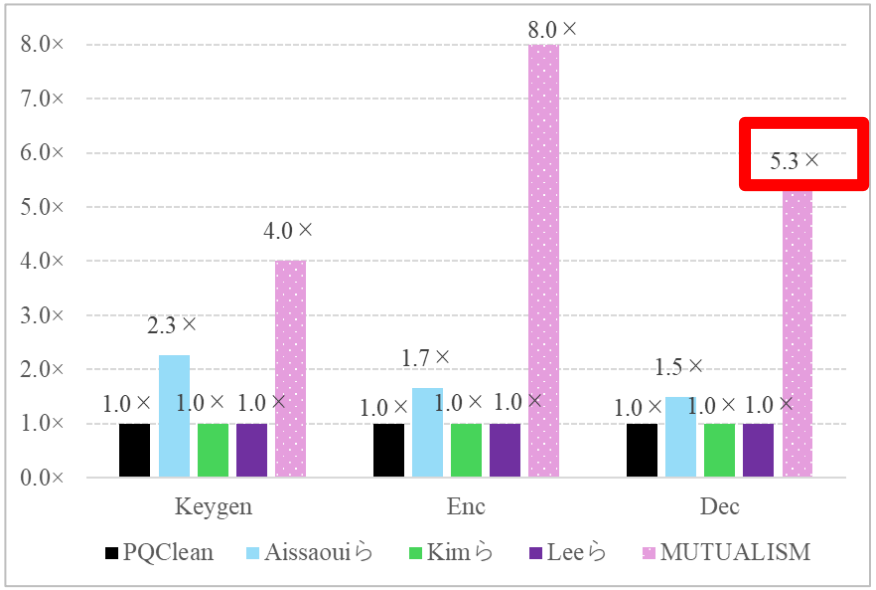
処理の中で**多項式乗算の割合が大きい**
-> **多項式乗算を改良（約70倍）**

改良方針②

既存研究で**多項式乗算改良以外が不足**
-> **HQC全体の処理最適化（約5倍）**



処理時間改善倍率（PQClean/各手法）



メモリ改善倍率（PQClean/各手法）

改良方針①：多項式乗算に対する改良

HQC復号の処理時間割合

処理	処理時間	割合
多項式乗算	**666.41**	**96.2%**
ベクトル生成	15.45	2.2%
Reed-Muller符号	1.76	0.25%
Reed-Solomon符号	2.68	0.39%
SHAKE-256	4.78	0.69%
Others	1.64	0.24%

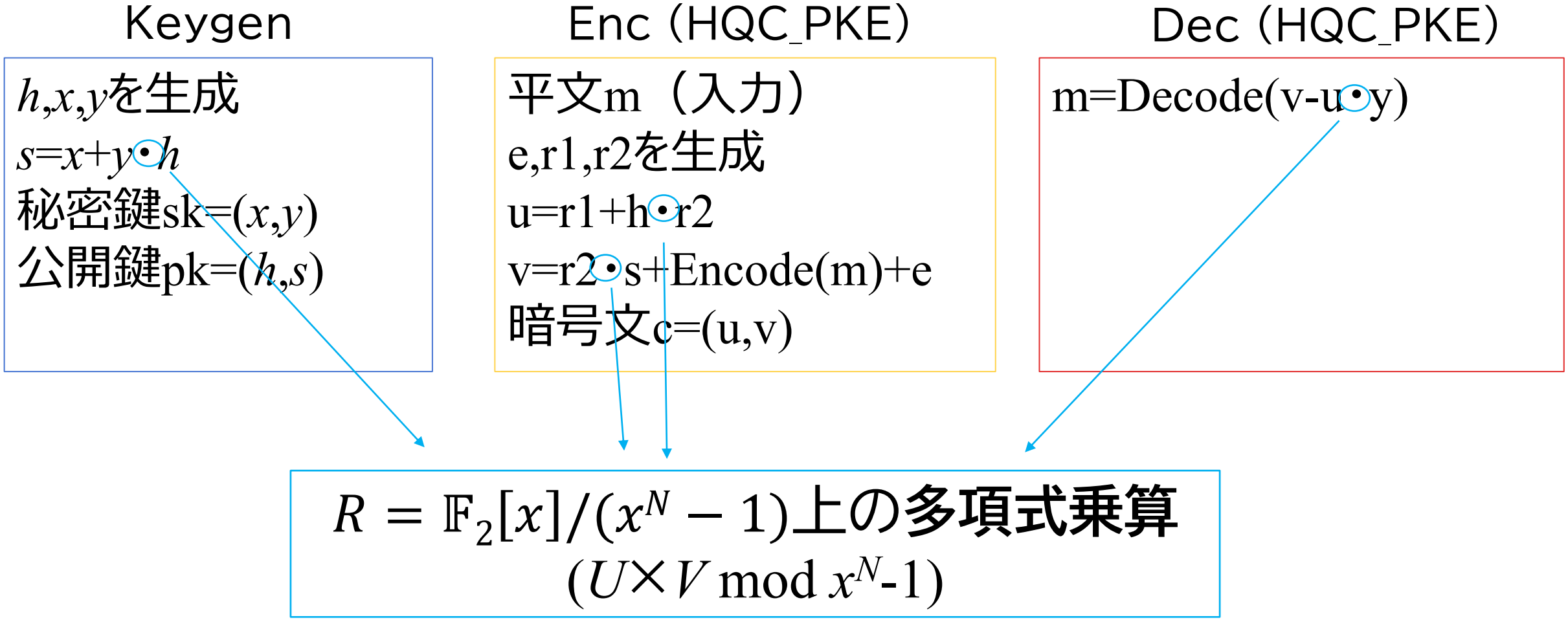


HQC-256復号処理時間プロファイリング(PQClean, 単位:Mcycles)

多項式乗算計算がボトルネック

【参考】HQCの処理概観

多項式乗算を行っている部分



【参考】HQC改良関連研究

多項式乗算の改良はまだ不十分

	実装	HQC-256多項式乗算**単体**(39MHz CPU)
スパース乗算	Aissaouiら(再現)	7.78 Mcycles (199ms) /7.3KB
FAFFT法	Leeら	6.55 Mcycles (168ms) /66KB
libNTL	Reference実装	24.74 Mcycles (634ms) /86KB
karatsuba法	PQClean	222.2 Mcycles (5697ms) /65KB

依然として多項式乗算処理時間は課題

- 1回の多項式乗算に、100ms以上かかっている
- 1回のHQC処理では、上記乗算を複数回行う (例:復号は3回)
  (スパース乗算方式以外は、メモリサイズが大きい)

改良方針①：多項式乗算に対する改良

HQCで用いられる多項式は「スパース多項式」

スパース多項式の利点

必要メモリが小さい
定数時間性：一部省リソース機器で保証可能

スパース多項式の非零位置だけ処理

$R = \mathbb{F}_2[x]/(x^N - 1)$上の多項式乗算

$O = U \times V$

$|U|=|V|=|O|=N$
$\omega(U) \leqq 149$
$N=57637(bits)$
} HQC-256の場合

$index := u_L[i]$ / BIT_WIDTH
$shift := u_L[i]$ mod BIT_WIDTH
for $j := 0$ **to** $N_{word} - 1$ **do**
  $temp[index + j] := temp[index + j]$ XOR lsl$(v[j], shift)$
  $temp[index + j + 1] := temp[index + j + 1]$ XOR lsr$(v[j], (\text{BIT_WIDTH} - shift))$

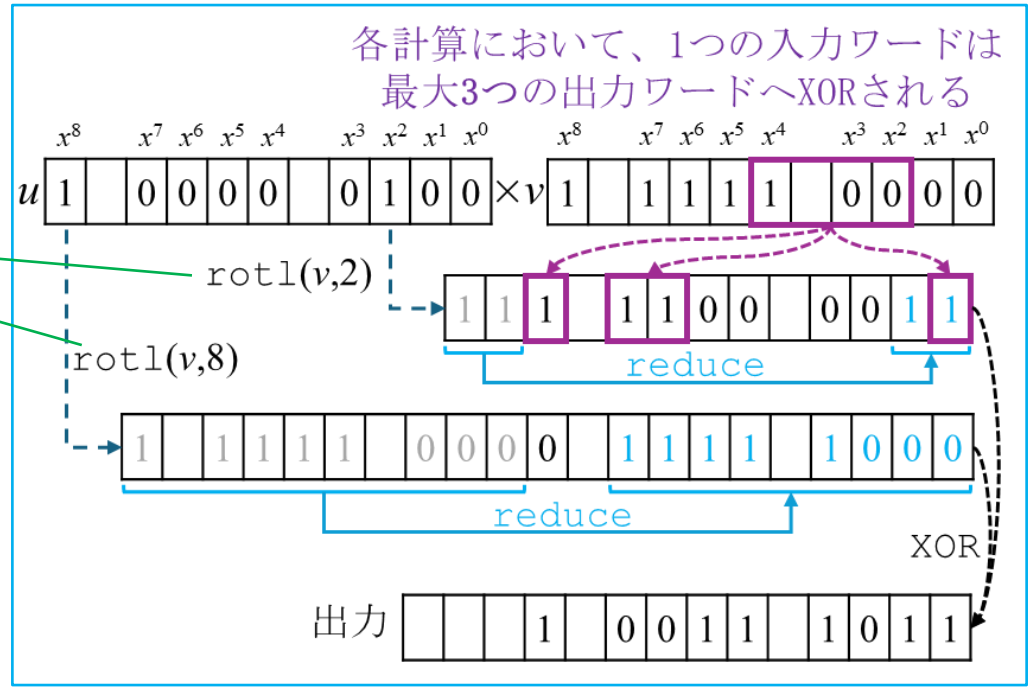
Basic乗算アルゴリズム(一部):
indexは入力データに依存

改良方針①：多項式乗算に対する改良

既存のスパース多項式乗算手法：rotl(左回転シフト)に基づく
⇒効率が悪い

rotlを使うことで、剰余還元(mod x^N-1計算)が自動的に行われる

uの全非零bitに対して、剰余還元が毎回発生する



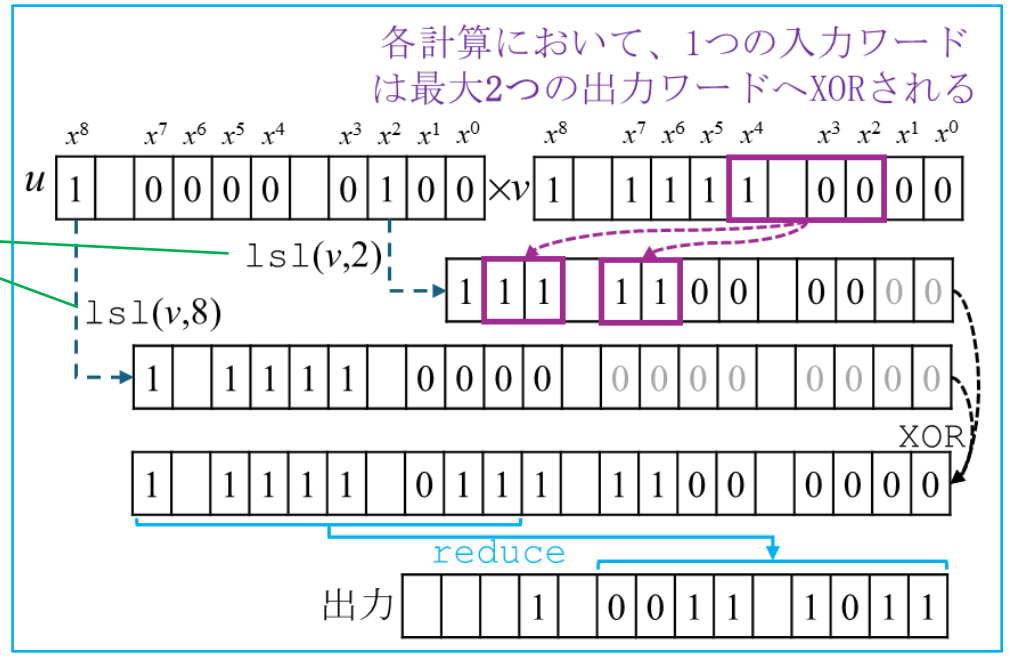
Aissaouiらが提案したrotlに基づく乗算方式

長さNの追加バッファが必要
(HQC-256では7KB)

改良方針①：多項式乗算に対する改良

提案手法(段階1)：多項式乗算と剰余還元を分離する方式を提案

乗算はlsl,lsr
(論理シフト)を使用



本研究が採用するMultiply-then-reduce方式
(Basic)

Basicアルゴリズム：先に乗算して、最後に剰余還元(reduce)する

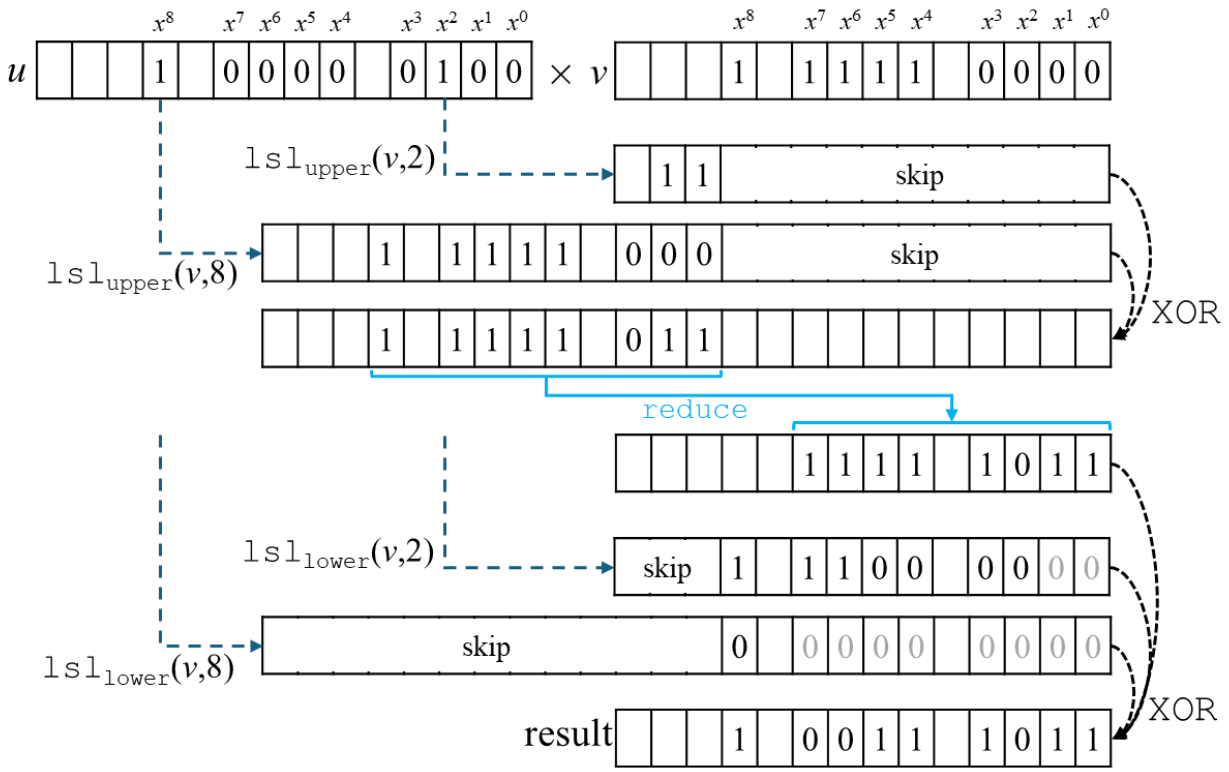
- メモリアクセス回数・分岐処理を削減可能
- さらなる高速化手法の適用も容易になる

長さNの追加バッファが必要
(HQC-256では7KB)

さらに削減可能！
(Zero-Buffer方式)

改良方針①：多項式乗算に対する改良

提案手法(段階２)：乗算結果の上位半分と下位半分を分割計算



① *Round1:* 中間結果$u \times v$の上位半分 $(x^N,\ldots,x^{2N-1})$を計算する

②上位半分の結果を剰余還元(reduce)する

③ *Round2:*中間結果$u \times v$の下位半分 $(x^0,\ldots,x^{N-1})$を計算する

処理速度を変えずに、必要な追加バッファを0にできた

改良方針①：多項式乗算に対する改良

その他の高速化手法

① スパース多項式乗算はsupport-set表現で削減
- 非零bitのindex⇒ベクトルの転換を省略できるため

② $\mathbb{F}_{2^8}$演算のルックアップテーブル化
- Reed-Solomon符号処理を高速化できる

③ アーキテクチャを意識した最適化
- if分岐を回避する（提案乗算アルゴリズムでは一切使わない）
- forループを展開する
- Cortex-M4向けshake256アセンブリ実装を導入
- 32bit単位でデータを処理する(PQCleanは64bit)

改良評価結果：多項式乗算単体の処理時間

実装	HQC-256	
本研究	**1.966**	**メモリ：0.3KB**
Aissaouiら	7.78	メモリ：7.3KB
Leeら	6.55	メモリ：66KB
libNTL	24.74	メモリ：86KB
PQClean	222.20	メモリ：65KB

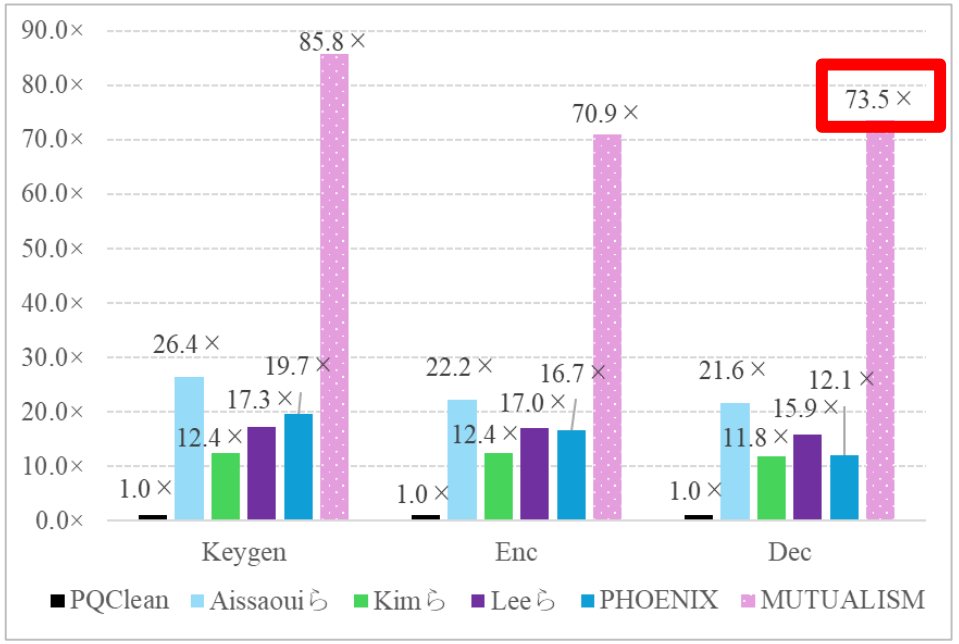
多項式乗算単体の処理時間（単位：Mcycles）

PQCleanより処理速度が100倍以上に
必要なメモリサイズがほぼゼロに

改良評価結果：HQC処理時間

単位:Mcycles

実装：HQC-256	Keygen	Enc	Dec
MUTUALISM	**2.676**	**6.495**	**9.421(242ms)**
Aissaouiら	8.70	20.73	32.08 (823ms)
Leeら	13.29	27.14	43.68 (1120ms)
Reference実装	31.40	63.04	94.32 (2.4s)
PQClean	229.49	460.36	692.72(17.8s)



処理時間改善倍率(PQClean/各手法)

一番重い処理も1s以内で処理可能

【再掲】今回発表：HQCの高速化・軽量改良技術

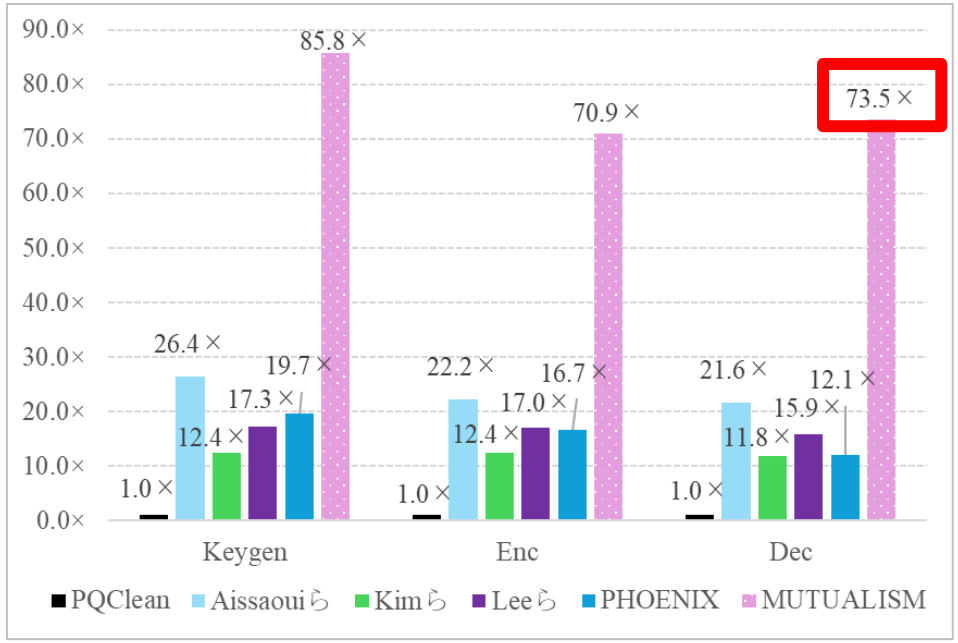
貢献：HQCの省リソース環境向けソフトウェア実装”MUTUALISM”を提案

改良方針①

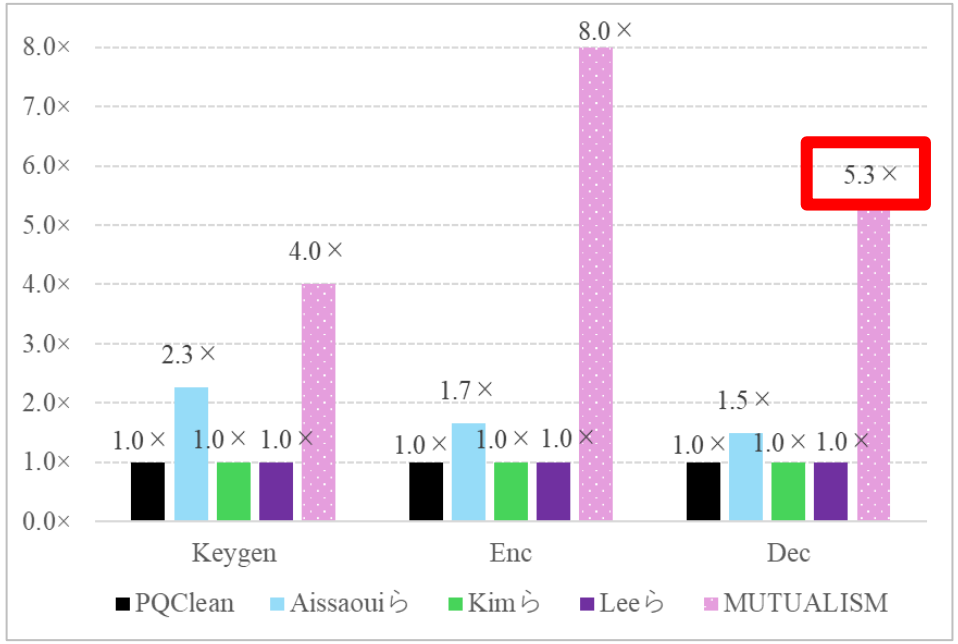
処理の中で**多項式乗算の割合が大きい**
-> **多項式乗算を改良（約70倍）**

改良方針②

既存研究で**多項式乗算改良以外が不足**
-> **HQC全体の処理最適化（約5倍）**



処理時間改善倍率（PQClean/各手法）



メモリ改善倍率（PQClean/各手法）

改良方針②：HQC全体の処理最適化

多項式乗算以外のHQC処理が軽視されている

【再掲】

処理	処理時間	割合
多項式乗算	**666.41**	**96.2%**
ベクトル生成	15.45	2.2%
Reed-Muller符号	1.76	0.25%
Reed-Solomon符号	2.68	0.39%
SHAKE-256	4.78	0.69%
Others	1.64	0.24%

HQC-256復号処理時間プロファイリング(PQClean, 単位:Mcycles)

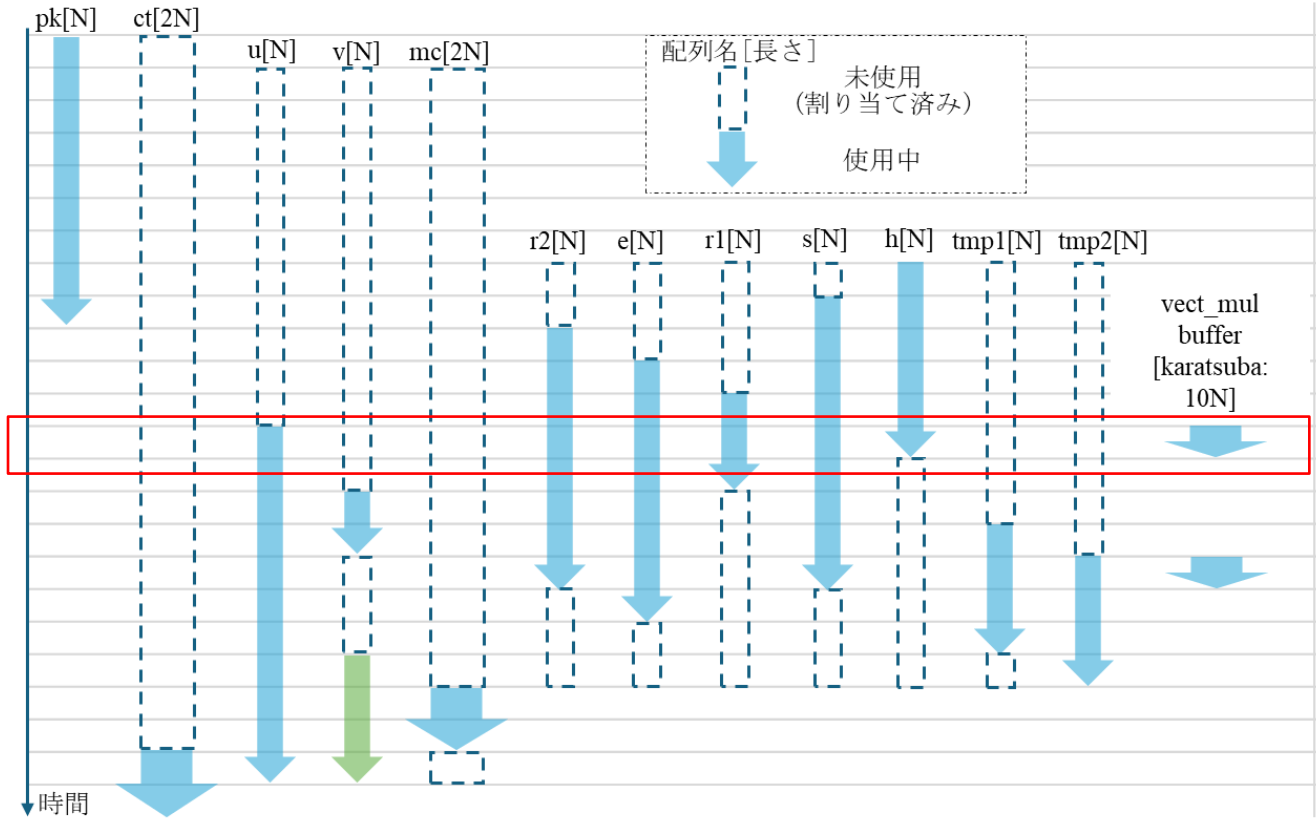
既存実装のメモリサイズ
100KBを超えている

多項式乗算以外の処理も含めた
HQC全体の性能をいかに最適化できるか？

改良方針②：HQC全体の処理最適化

HQCの全配列ライフサイクルを分析
➔ 衝突関係を把握し、スタックサイズが最大時の全配列の削減方法を検討
➔ いくつかの中間変数（h,tmp1,tmp2）を削減

スタックサイズ
最大時




```
crypto_kem_enc:
  m←rand_prng(SIZE_K)
  salt←rand_prng(SIZE_SALT)
  tmp←m||pk[0..31]||salt
  theta←shake256_512(tmp)
  u,v←hqc_pke_encrypt()
   hqc_pke_encrypt:
      h←gen_vector()
      s←pk[SEED_BYTES..]
      r2←gen_vector_fixed_weight(ω_r)
      e←gen_vector_fixed_weight(ω_r)
      r1←gen_vector_fixed_weight(ω_e)
      u←vect_mul(r2,h)
      u←vect_add(r1,u)
      v←code_encode(m)
      tmp1←resize(v)
      tmp2←vect_mul(r2,s)
      tmp2←vect_add(e,tmp2)
      tmp2←vect_add(tmp1,tmp2)
      v←resize(tmp2)
  mc←m||u||v
  ss←shake256_512(mc)
  ct←u||v||salt
```

HQC Reference実装の配列ライフサイクル(暗号化)

改良方針②：HQC全体の処理最適化

その他のメモリ削減手法

① 改良多項式乗算による計算

② スパース多項式乗算はsupport-set表現で削減
- 非零bitのindexのみを保存する

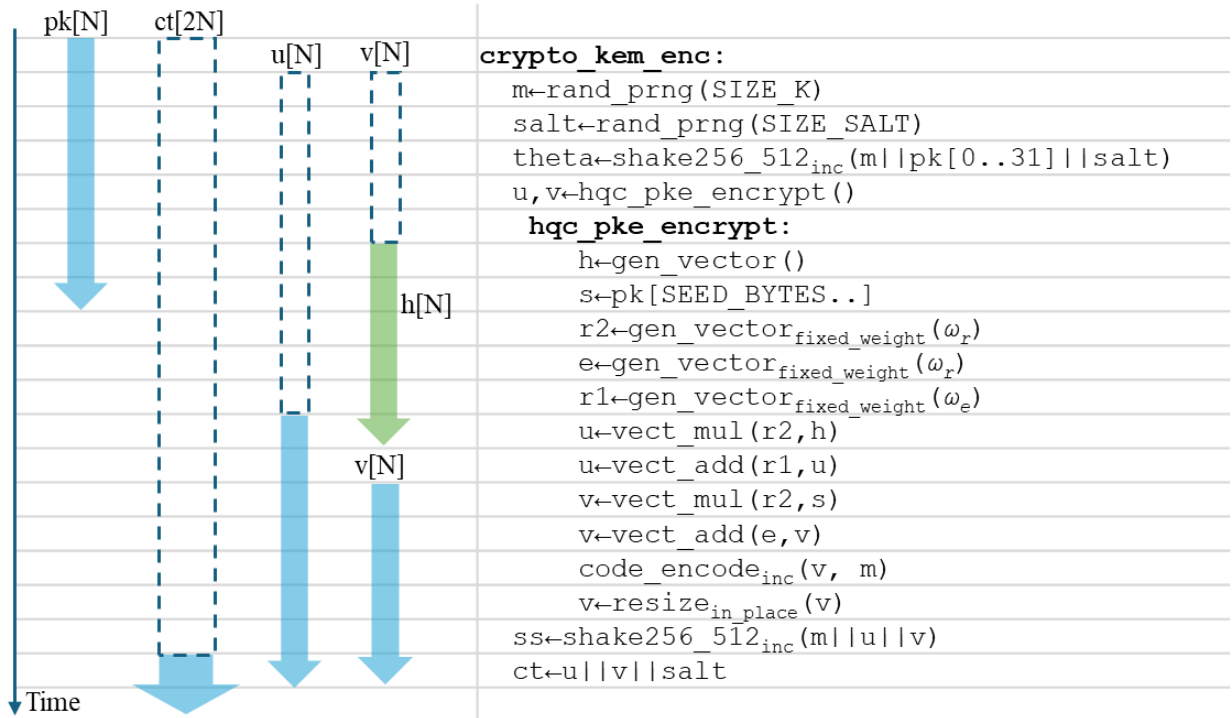
⇒r_2,e,r_1を削減

③ 中間バッファを用いない直接計算
- 例：shake256の入力バッファを使用せず、incremental APIで入力する

⇒mc,sを削減

④ ctのスペースを活用(カプセル化のみ)

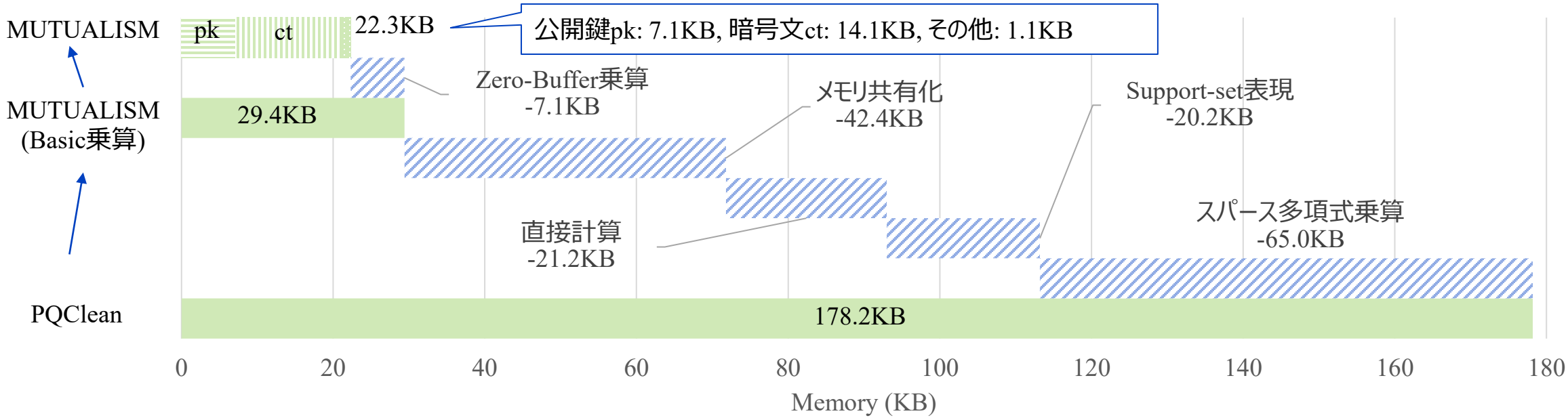
⇒u,vを削減



最適化①-④適用後の配列ライフサイクル(カプセル化)

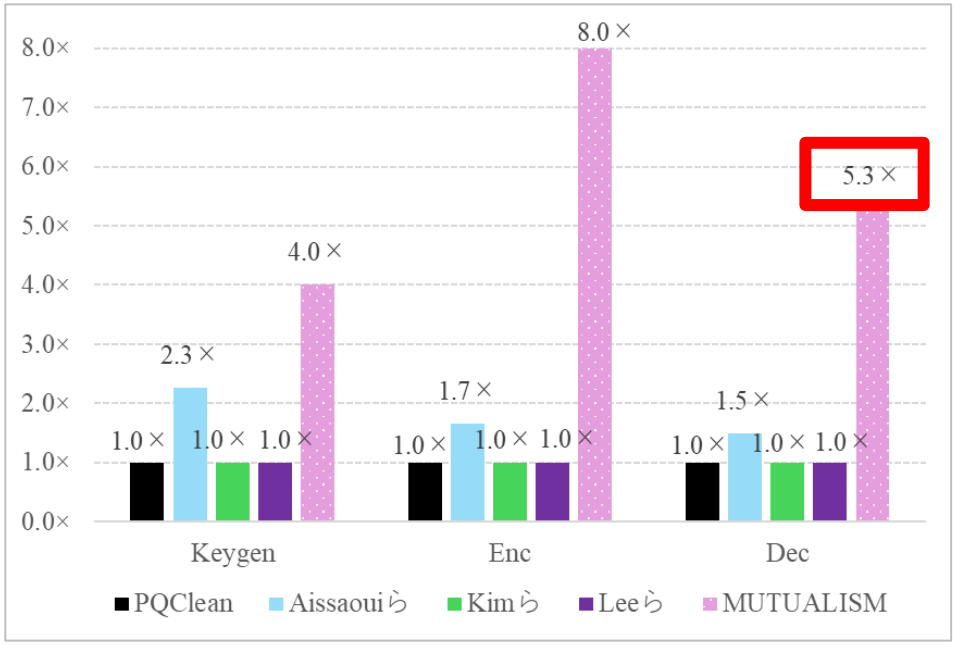
改良評価結果：HQC全体の処理最適化

HQC暗号化処理プロファイリング図



改良評価結果：HQCメモリサイズ

実装：HQC-256	Keygen	Enc	Dec
MUTUALISM	**29.0KB**	**22.3KB**	**36.6KB**
Aissaouiら	50.6KB	107.6KB	129.0KB
Leeら	115.8KB	179.2KB	193.4KB
Reference実装	135.4KB	191.9KB	213.5KB
PQClean	114.8KB	178.2KB	192.4KB



メモリ改善倍率(PQClean/各手法)

一番重い処理も64KB RAMチップで処理可能

【再掲】今回発表：HQCの高速化・軽量改良技術

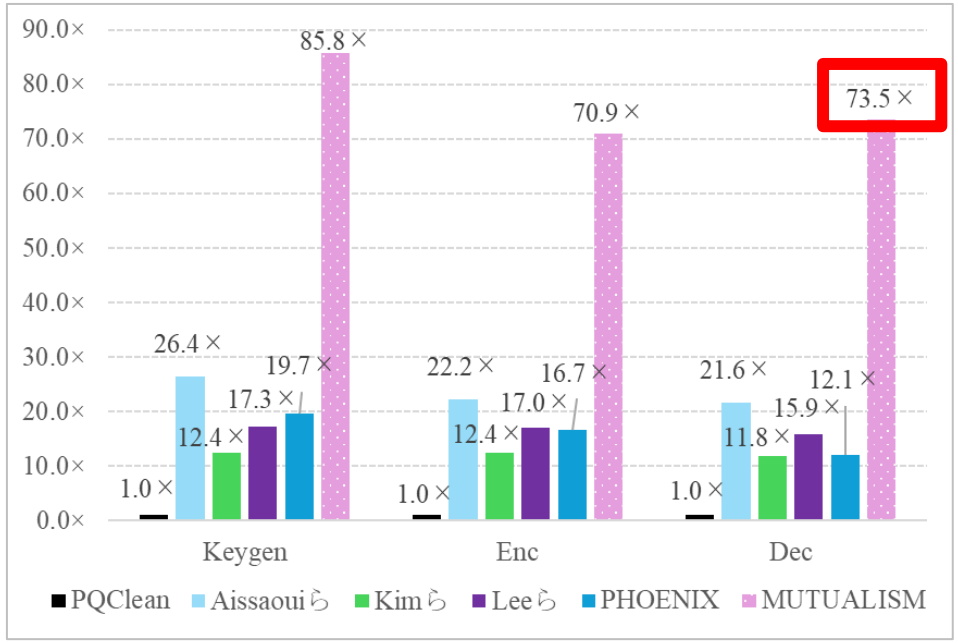
貢献：HQCの省リソース環境向けソフトウェア実装”MUTUALISM”を提案

改良方針①

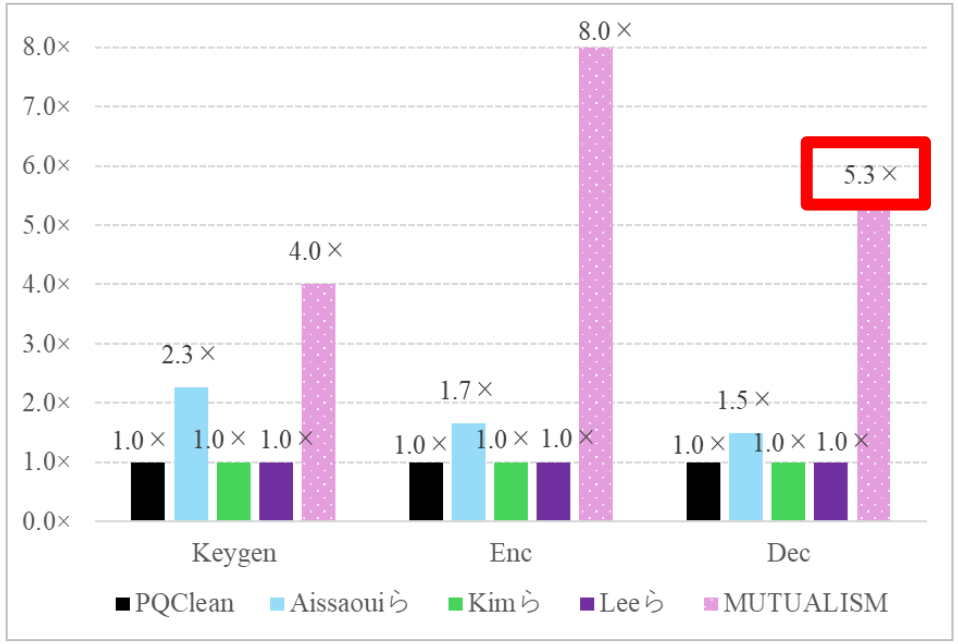
処理の中で**多項式乗算の割合が大きい**
-> **多項式乗算を改良（約70倍）**

改良方針②

既存研究で**多項式乗算改良以外が不足**
-> **HQC全体の処理最適化（約5倍）**



処理時間改善倍率（PQClean/各手法）



メモリ改善倍率（PQClean/各手法）

Business Development Division