Lightweight Post-Quantum Key Encapsulation for 8-bit AVR Microcontrollers

Hao Cheng    Johann Großschädl    Peter B. Rønne    Peter Y. A. Ryan

University of Luxembourg

CARDIS 2020
IoT on the Rise

Source: Ericsson Mobility Report (June 2020)
IoT on the Rise

Source: Ericsson Mobility Report (June 2020)

IoT needs lightweight cryptographic schemes and protocols
8-bit AVR Microcontrollers
8-bit AVR Microcontrollers

8-bit AVR Architecture

- RISC philosophy and modified Harvard memory model
- 32 general-purpose working registers of 8-bit width
8-bit AVRs are microcontrollers designed by Atmel.

**8-bit AVR Architecture**

- RISC philosophy and modified Harvard memory model
- 32 general-purpose working registers of 8-bit width
- Bitwise logical and most arithmetic instructions take 1 clock cycle
- Multiplication and RAM accessing instructions take 2 clock cycles
Quantum Cryptanalyses

Quantum Computing exploits quantum-mechanical phenomena

Shor’s Algorithm solves IFP and DLP in polynomial time
Post-Quantum Cryptography Standardization

The Round 3 candidates were announced July 22, 2020. NISTIR 8309, Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process is now available. NIST has developed Guidelines for Submitting Tweaks for Third Round Finalists and Candidates.

- Solicit, evaluate and standardize one or more quantum-resistant PKC algorithms
- Evaluate candidates’ performance also on resource-constrained devices
- Now is Round 3
Lattice-Based KEMs in IoT

- Benchmarking results collected in pqm4 \(^1\) (ARM Cortex-M4)
  - faster than Curve25519
  - RAM footprint often between 5 kB ~ 30 kB (vs 500 bytes of Curve25519)

\(^1\)https://github.com/mupq/pqm4
Lattice-Based KEMs in IoT

- Benchmarking results collected in pqm4\(^1\) (ARM Cortex-M4)
  - faster than Curve25519
  - RAM footprint often between 5 kB ~ 30 kB (vs 500 bytes of Curve25519)

- Deployment in AVR devices
  - AVR devices feature only a few kB of RAM (e.g. MICAz mote has only 4 kB RAM)
  - need low-memory implementations

\(^1\)https://github.com/mupq/pqm4
A Fairy Tale

- Designer
  - Mike Hamburg

- Ed448-Goldilocks [Ham15]
  - RFC7748 and TLS 1.3
  - “Golden-ratio” Solinas prime
    \( 2^{448} - 2^{224} - 1 \) (Goldilocks)

- ThreeBears
  - NIST PQC Round 2 candidate (KEM)
  - Integer Module Learning With Errors (I-MLWE) [Gu17]
  - BabyBear (II), MamaBear (IV), PapaBear (V)
  - has both CCA and CPA instances
This Work

- Analyzes the performance of ThreeBears on AVR
- Studies its flexibility to achieve different trade-offs between RAM footprint and execution time
Our Implementation

- First highly-optimized software implementations of BabyBear for AVR platform (constant-time)
Our Implementation

- First highly-optimized software implementations of BabyBear for AVR platform (constant-time)
  - Memory-Efficient ME-BBear (CCA) ME-BBear-Eph (CPA)
    based on low-memory implementation in NIST package
    most memory-efficient software implementation of Round 2 candidate
Our Implementation

- First highly-optimized software implementations of BabyBear for AVR platform (constant-time)
  - **Memory-Efficient** ME-BBear (CCA) ME-BBearer-Eph (CPA)
    based on low-memory implementation in NIST package
  - **High-Speed** HS-BBear (CCA) HS-BBearer-Eph (CPA)
    based on optimized implementation in NIST package
Our Implementation

- First highly-optimized software implementations of BabyBear for AVR platform (constant-time)
  - Memory-Efficient ME-BBear (CCA) ME-BBear-Eph (CPA) based on low-memory implementation in NIST package
    most memory-efficient software implementation of Round 2 candidate
  - High-Speed HS-BBear (CCA) HS-BBear-Eph (CPA) based on optimized implementation in NIST package

- Memory-optimized and speed-optimized
  Multiply-ACcumulate (MAC) operations $r = r + a \times b$
ThreeBears KEM

- The underlying field
  - $\mathbb{Z}/N$

- Prime ("golden-ratio" Solinas prime [Ham15])
  - $N = 2^{3120} - 2^{1560} - 1$
  - $N = \phi(x) = x^D - x^{D/2} - 1$
  - $N = \lambda^2 - \lambda - 1$
ThreeBears KEM

- The underlying field
  - \( \mathbb{Z}/N \)

- Prime ("golden-ratio" Solinas prime [Ham15])
  - \( N = 2^{3120} - 2^{1560} - 1 \)
  - \( N = \phi(x) = x^D - x^{D/2} - 1 \)
  - \( N = \lambda^2 - \lambda - 1 \)

- Field operations
  - \((+, \cdot)\) are conventional integer addition and multiplication
  - addition (+) \( a + b := a + b \mod N \)
  - multiplication (*) \( a \cdot b := a \cdot b \cdot \lambda^{-1} \mod N \)
ThreeBears KEM (CCA)

**Key Generation**

- $sk \leftarrow \text{random}()$
- $a, b \leftarrow \text{noise\_sampler}(sk)$
- $r \leftarrow \text{hash}(sk)$
- $M \leftarrow \text{uniform\_sampler}(r)$
- $z \leftarrow z_i = b_i + \sum_{j=0}^{d-1} M_{i,j} \cdot a_j$

---

private key $sk$
public key $(r, z)$

Dimension $d$ is 2 for BabyBear, 3 for MamaBear, 4 for PapaBear
ThreeBears KEM (CCA)

**Key Generation**

\[ sk \leftarrow \text{random}( ) \]
\[ \alpha, \beta \leftarrow \text{noise\_sampler}(sk) \]
\[ r \leftarrow \text{hash}(sk) \]
\[ M \leftarrow \text{uniform\_sampler}(r) \]
\[ z \leftarrow z_i = b_i + \sum_{j=0}^{d-1} M_{i,j} * \alpha_j \]

Private key: \( sk \)
Public key: \( (r, z) \)

**Encapsulation**

\[ g \leftarrow \text{random}( ) \]
\[ \hat{\alpha}, \hat{\beta}, c \leftarrow \text{noise\_sampler}(r, g) \]
\[ M \leftarrow \text{uniform\_sampler}(r) \]
\[ y \leftarrow y_i = \hat{b}_i + \sum_{j=0}^{d-1} M_{j,i} * \hat{\alpha}_j \]
\[ x = c + \sum_{j=0}^{d-1} y_j * \hat{\alpha}_j \]
\[ f \leftarrow \text{FEC\_encode}(g, x) \]
\[ ss \leftarrow \text{hash}(r, g) \]

Shared secret: \( ss \)
Cipher text: \( (f, y) \)

Dimension \( d \) is 2 for BabyBear, 3 for MamaBear, 4 for PapaBear
ThreeBears KEM (CCA)

**Key Generation**

\[ sk \leftarrow \text{random}( ) \]
\[ \mathbf{a}, \mathbf{b} \leftarrow \text{noise} \_\text{ sampler}(sk) \]
\[ r \leftarrow \text{hash}(sk) \]
\[ \mathbf{M} \leftarrow \text{uniform} \_\text{ sampler}(r) \]
\[ \mathbf{z} \leftarrow z_i = b_i + \sum_{j=0}^{d-1} M_{i,j} \ast a_j \]

- private key \( sk \)
- public key \( (r, \mathbf{z}) \)

**Encapsulation**

\[ g \leftarrow \text{random}( ) \]
\[ \hat{a}, \hat{b}, c \leftarrow \text{noise} \_\text{ sampler}(r, g) \]
\[ \mathbf{M} \leftarrow \text{uniform} \_\text{ sampler}(r) \]
\[ y \leftarrow y_i = \hat{b}_i + \sum_{j=0}^{d-1} M_{j,i} \ast \hat{a}_j \]
\[ x = c + \sum_{j=0}^{d-1} z_j \ast \hat{a}_j \]
\[ f \leftarrow \text{FEC} \_\text{ encode}(g, x) \]
\[ ss \leftarrow \text{hash}(r, g) \]

- shared secret \( ss \)
- ciphertext \( (f, y) \)

**Decapsulation**

\[ a \leftarrow \text{noise} \_\text{ sampler}(sk) \]
\[ x = \sum_{j=0}^{d-1} y_j \ast a_j \]
\[ g \leftarrow \text{FEC} \_\text{ decode}(f, x) \]
\[ (r', \mathbf{z}') \leftarrow \text{KeyGen}(sk) \]
\[ (f', y') \leftarrow \text{Encaps}(g, (r', \mathbf{z}')) \]
\[ (f', y') \overset{?}{=} (f, y) \]

- shared secret \( ss' \)
- ciphertext \( (f', y') \)

Dimension \( d \) is 2 for BabyBear, 3 for MamaBear, 4 for PapaBear
### ThreeBears KEM (CPA)

#### Key Generation

\[ \text{sk} \leftarrow \text{random()} \]
\[ \text{a}, \text{b} \leftarrow \text{noise}\_\text{ampler(sk)} \]
\[ r \leftarrow \text{hash(sk)} \]
\[ M \leftarrow \text{uniform}\_\text{ampler(r)} \]
\[ z \leftarrow z_i = b_i + \sum_{j=0}^{d-1} M_{i,j} \times a_j \]

| private key | \( \text{sk} \) |
|-------------|----------------|
| public key  | \((r, z)\)      |

#### Encapsulation

\[ \text{g} \leftarrow \text{random()} \]
\[ \hat{\text{a}}, \hat{\text{b}}, \text{c} \leftarrow \text{noise}\_\text{ampler(r, g)} \]
\[ M \leftarrow \text{uniform}\_\text{ampler(r)} \]
\[ y \leftarrow y_i = \hat{b}_i + \sum_{j=0}^{d-1} M_{j,i} \times \hat{a}_j \]
\[ x = c + \sum_{j=0}^{d-1} z_j \times \hat{a}_j \]
\[ t \leftarrow \text{hash(r, g)} \]
\[ f \leftarrow (\text{FEC}\_\text{encode}(t), x) \]
\[ ss \leftarrow \text{hash(r, t)} \]

| shared secret | \( ss \) |
|---------------|---------|
| ciphertext    | \((f, y)\) |

#### Decapsulation

\[ \text{a} \leftarrow \text{noise}\_\text{ampler(sk)} \]
\[ x = \sum_{j=0}^{d-1} y_j \times a_j \]
\[ t \leftarrow \text{FEC}\_\text{decode}(f, x) \]
\[ r \leftarrow \text{hash(sk)} \]
\[ ss \leftarrow \text{hash}(r, t) \]

| shared secret | \( ss \) |

---

Dimension \( d \) is 2 for BabyBear, 3 for MamaBear, 4 for PapaBear
Implementation View Point

- Auxiliary functions
  - samplers: noise/uniform sampler → cSHAKE256 → Keccak permutation
  - forward error correction (FEC): Melas BCH code

- Arithmetic components
  - MAC operation: \( r = r + a \times b \mod N \)

\(^{2}\text{https://github.com/XKCP/XKCP/tree/master/lib/low/KeccakP-1600/AVR8}\)
Implementation View Point

- **Auxiliary functions**
  - samplers: noise/uniform sampler $\rightarrow$ cSHAKE256 $\rightarrow$ Keccak permutation
    - open-source highly-optimized AVR Assembler
  - forward error correction (FEC): Melas BCH code

- **Arithmetic components**
  - MAC operation: $r = r + a \times b \mod N$

---

²https://github.com/XKCP/XKCP/tree/master/lib/low/KeccakP-1600/AVR8
Implementation View Point

- Auxiliary functions
  - samplers: noise/uniform sampler → cSHAKE256 → Keccak permutation
    - open-source highly-optimized AVR Assembler
  - forward error correction (FEC): Melas BCH code
    - small memory/code requirements, constant time and runtime is almost negligible

- Arithmetic components
  - MAC operation: $r = r + a \times b \mod N$

---

²https://github.com/XKCP/XKCP/tree/master/lib/low/KeccakP-1600/AVR8
Implementation View Point

- **Auxiliary functions**
  - Samplers: noise/uniform sampler → cSHAKE256 → Keccak permutation
    - open-source highly-optimized AVR Assembler
  - Forward error correction (FEC): Melas BCH code
    - Small memory/code requirements, constant time and runtime is almost negligible

- **Arithmetic components**
  - MAC operation: $r = r + a \times b \mod N$
    - Dominate both the RAM footprint and the execution time!!

---

2[https://github.com/XKCP/XKCP/tree/master/lib/low/KeccakP-1600/AVR8](https://github.com/XKCP/XKCP/tree/master/lib/low/KeccakP-1600/AVR8)
Field Element Representation

\[ N = 2^{3120} - 2^{1560} - 1 \rightarrow 3120\text{-bit integer} \]
Field Element Representation

\[ N = 2^{3120} - 2^{1560} - 1 \rightarrow 3120\text{-bit integer} \]

### Reduced-radix representation \((w = 32)\)

| Description           | Value                        |
|-----------------------|------------------------------|
| Format                | \(120 \times 26\) bits = 3120 bits |
| RAM usage             | \(120 \times 4\) bytes = 480 bytes |
Field Element Representation

\[ N = 2^{3120} - 2^{1560} - 1 \rightarrow 3120\text{-bit integer} \]

**Reduced-radix representation \((w = 32)\)**

- Format \(120 \times 26\text{ bits} = 3120\text{ bits}\)
- RAM usage \(120 \times 4\text{ bytes} = 480\text{ bytes}\)

**Full-radix representation \((w = 32)\)**

- Format \(97.5 \times 32\text{ bits} = 3120\text{ bits}\)
- RAM usage \(98 \times 4\text{ bytes} = 392\text{ bytes}\)
- Why?
  - Sequential order in AVR
  - Reduce RAM consumption
MAC Operation $r = r + a \times b \mod N$

$$N = \lambda^2 - \lambda - 1 \quad \rightarrow \quad \lambda^{-1} = \lambda - 1$$
MAC Operation $r = r + a \times b \mod N$

$$N = \lambda^2 - \lambda - 1 \rightarrow \lambda^{-1} = \lambda - 1$$

$$z := a \times b = a \times b \times \lambda^{-1} = (a_L + a_H \lambda)(b_L + b_H \lambda) \times \lambda^{-1} \mod N$$

$$= a_L b_L \lambda^{-1} + (a_L b_H + a_H b_L) + a_H b_H \lambda \mod N$$

$$= a_L b_L (\lambda - 1) + (a_L b_H + a_H b_L) + a_H b_H \lambda \mod N$$

$$= (a_L b_H + a_H b_L - a_L b_L) + (a_L b_L + a_H b_H) \lambda \mod N$$

$e_L/e_H$ stands for the lower/higher half of element $e$
MAC Operation $r = r + a \ast b \mod N$

\[ N = \lambda^2 - \lambda - 1 \quad \rightarrow \quad \lambda^{-1} = \lambda - 1 \]

\[ z := a \ast b = a \cdot b \cdot \lambda^{-1} = (a_L + a_H \lambda)(b_L + b_H \lambda) \cdot \lambda^{-1} \mod N \]

\[ = a_L b_L \lambda^{-1} + (a_L b_H + a_H b_L) + a_H b_H \lambda \mod N \]

\[ = a_L b_L (\lambda - 1) + (a_L b_H + a_H b_L) + a_H b_H \lambda \mod N \]

\[ = (a_L b_H + a_H b_L - a_L b_L) + (a_L b_L + a_H b_H) \lambda \mod N \]

\[ = (a_H b_H - (a_L - a_H)(b_L - b_H)) + (a_L b_L + a_H b_H) \lambda \mod N \] (1)

$e_L/e_H$ stands for the lower/higher half of element $e$
MAC Operation $r = r + a * b \mod N$

\[
r := r + a \ast b \mod N
\]

\[
= (r_L + a_H b_H - (a_L - a_H)(b_L - b_H)) + (r_H + a_L b_L + a_H b_H)\lambda \mod N \quad (2)
\]

\[
= (r_L + a_H b_L - a_L(b_L - b_H)) + (r_H + (a_L + a_H)b_H + a_L(b_L - b_H))\lambda \mod N \quad (3)
\]

\[
= (r_L + a_H b_L - 2a_L(b_L - b_H)) + (r_H + (a_L + a_H)b_H)\lambda + a_L(b_L - b_H)\lambda^2 \mod N \quad (4)
\]
Memory-Optimized MAC

\[ r := (r_L + a_H b_L - 2a_L(b_L - b_H)) + (r_H + (a_L + a_H)b_H)\lambda + a_L(b_L - b_H)\lambda^2 \mod N \]

**Algorithm 1** Memory-optimized MAC operation

**Input:** Aligned \( s \)-word integers \( A = (A_{s-1}, \ldots, A_1, A_0) \), \( B = (B_{s-1}, \ldots, B_1, B_0) \), and \( R = (R_{s-1}, \ldots, R_1, R_0) \), each word contains \( \omega \) bits; \( \beta \) is a parameter of alignment

**Output:** Aligned \( s \)-word product \( R = R + A \cdot B \cdot \lambda \mod N = (R_{s-1}, \ldots, R_1, R_0) \)

1. \( Z_0 \leftarrow 0, Z_1 \leftarrow 0 \)
2. \( l \leftarrow s/2 \)
3. for \( i \) from 0 to \( l - 1 \) by 1 do
   4. \( Z_2 \leftarrow 0, k \leftarrow i + 1 \)
   5. for \( j \) from 0 to \( i \) by 1 do
      6. \( Z_0 \leftarrow Z_0 + A_j \cdot B_k \)
      7. \( Z_1 \leftarrow Z_1 + (A_j + A_{j+1}) \cdot B_k \)
      8. \( Z_2 \leftarrow Z_2 + A_j \cdot (B_k - B_{k+1}) \)
   9. end for
10. \( Z_0 \leftarrow Z_0 - 2 \cdot Z_2 \)
11. \( k \leftarrow l \)
12. for \( j \) from \( i + 1 \) to \( l - 1 \) by 1 do
13. \( k \leftarrow k - 1 \)
14. \( Z_1 \leftarrow Z_1 + 2^\beta \cdot A_{j+1} \cdot B_k \)
15. \( Z_2 \leftarrow Z_2 + 2^\beta \cdot (A_j + A_{j+1}) \cdot B_{k+1} \)
16. \( Z_0 \leftarrow Z_0 + 2^\beta \cdot A_j \cdot (B_k - B_{k+1}) \)
17. end for
18. \( Z_0 \leftarrow Z_0 + Z_3 + R_i \)
19. \( Z_1 \leftarrow Z_1 + Z_2 + R_{i+1} \)
20. \( R_i \leftarrow Z_0 \mod 2^\omega \)
21. \( Z_0 \leftarrow Z_0/2^\omega \)
22. end for
23. \( R_{i+1} \leftarrow Z_1 \mod 2^\omega \)
24. \( Z_1 \leftarrow Z_1/2^\omega \)
25. end for
26. \( Z_0 \leftarrow 2^\beta \cdot Z_0 + R_{i-1}/2^\omega - \beta \)
27. \( Z_1 \leftarrow 2^\beta \cdot Z_1 + R_{i-1}/2^\omega - \beta \)
28. \( R_{i-1} \leftarrow R_{i-1} \mod 2^\omega - \beta \)
29. \( R_{i-1} \leftarrow R_{i-1} \mod 2^\omega - \beta \)
30. \( Z_0 \leftarrow Z_0 + Z_1 \)
31. for \( i \) from 0 to \( l - 1 \) by 1 do
32. \( Z_1 \leftarrow Z_1 + R_i \)
33. \( R_i \leftarrow Z_1 \mod 2^\omega \)
34. \( Z_1 \leftarrow Z_1/2^\omega \)
35. end for
36. \( Z_0 \leftarrow 2^\beta \cdot Z_0 + R_{i-1}/2^\omega - \beta \)
37. \( R_{i-1} \leftarrow R_{i-1} \mod 2^\omega - \beta \)
38. for \( i \) from \( l \) to \( s - 1 \) by 1 do
39. \( Z_0 \leftarrow Z_0 + R_i \)
40. \( R_i \leftarrow Z_0 \mod 2^\omega \)
41. \( Z_0 \leftarrow Z_0/2^\omega \)
42. end for
43. return \( (R_{s-1}, \ldots, R_1, R_0) \)

RAM consumption

- Three 80-bit accumulators
- Some local variables
- No more levels of Karatsuba (reduce RAM usage)
Memory-Optimized MAC

\[ r := (r_L + a_H b_L - 2a_L (b_L - b_H)) + (r_H + (a_L + a_H) b_H) \lambda + a_L (b_L - b_H) \lambda^2 \mod N \]

**Algorithm 1** Memory-optimized MAC operation

**Input:** Aligned \( s \)-word integers \( A = (A_{s-1}, \ldots, A_1, A_0) \), \( B = (B_{s-1}, \ldots, B_1, B_0) \), and \( R = (R_{s-1}, \ldots, R_1, R_0) \), each word contains \( \omega \) bits; \( \beta \) is a parameter of alignment.

**Output:** Aligned \( s \)-word product \( R = R + A \cdot B \cdot \lambda \mod N = (R_{s-1}, \ldots, R_1, R_0) \)

```plaintext
1: \( Z_0 \leftarrow 0, Z_1 \leftarrow 0 \)
2: \( l \leftarrow s/2 \)
3: for \( i \) from 0 to \( l - 1 \) by 1 do
4: \( Z_2 \leftarrow 0, k \leftarrow i + 1 \)
5: for \( j \) from 0 to \( i \) by 1 do
6: \( k \leftarrow k - 1 \)
7: \( Z_0 \leftarrow Z_0 + A_{j+l} \cdot B_k \)
8: \( Z_1 \leftarrow Z_1 + (A_j + A_{j+l}) \cdot B_{k+l} \)
9: \( Z_2 \leftarrow Z_2 + A_j \cdot (B_k - B_{k+l}) \)
10: end for
11: \( Z_0 \leftarrow Z_0 - 2 \cdot Z_2 \)
12: \( k \leftarrow l \)
13: for \( j \) from \( i + 1 \) to \( l - 1 \) by 1 do
14: \( k \leftarrow k - 1 \)
15: \( Z_1 \leftarrow Z_1 + 2^\beta \cdot A_{j+l} \cdot B_k \)
16: \( Z_2 \leftarrow Z_2 + 2^\beta \cdot (A_j + A_{j+l}) \cdot B_{k+l} \)
17: \( Z_0 \leftarrow Z_0 + 2^\beta \cdot A_j \cdot (B_k - B_{k+l}) \)
18: end for
19: \( Z_0 \leftarrow Z_0 + Z_2 + R_i \)
20: \( Z_1 \leftarrow Z_1 + Z_2 + R_{i+l} \)
21: \( R_i \leftarrow Z_0 \mod 2^\omega \)
22: \( Z_0 \leftarrow Z_0/2^\omega \)
23: \( R_{i+l} \leftarrow Z_1 \mod 2^\omega \)
24: \( Z_1 \leftarrow Z_1/2^\omega \)
25: end for
26: \( Z_0 \leftarrow 2^\beta \cdot Z_0 + R_{i-1}/2^{\omega-\beta} \)
27: \( Z_1 \leftarrow 2^\beta \cdot Z_1 + R_{i-1}/2^{\omega-\beta} \)
28: \( R_{i-1} \leftarrow R_{i-1} \mod 2^{\omega-\beta} \)
29: \( R_{i-1} \leftarrow R_{i-1} \mod 2^{\omega-\beta} \)
30: \( Z_0 \leftarrow Z_0 + Z_1 \)
31: for \( i \) from 0 to \( l - 1 \) by 1 do
32: \( Z_1 \leftarrow Z_1 + R_i \)
33: \( R_i \leftarrow Z_1 \mod 2^\omega \)
34: \( Z_1 \leftarrow Z_1/2^\omega \)
35: end for
36: \( Z_0 \leftarrow 2^\beta \cdot Z_0 + R_{i-1}/2^{\omega-\beta} \)
37: \( R_{i-1} \leftarrow R_{i-1} \mod 2^{\omega-\beta} \)
38: for \( i \) from \( l \) to \( s - 1 \) by 1 do
39: \( Z_0 \leftarrow Z_0 + R_i \)
40: \( R_i \leftarrow Z_0 \mod 2^\omega \)
41: \( Z_0 \leftarrow Z_0/2^\omega \)
42: end for
43: return \((R_{s-1}, \ldots, R_1, R_0)\)
```

**Main MAC loop**

- Product-scanning
- Interleaved with modular reductions \( \lambda^2 = \lambda + 1 \) (lines from 19 to 24)
Memory-Optimized MAC

\[ r := (r_L + a_H b_L - 2a_L (b_L - b_H)) + (r_H + (a_L + a_H) b_H) \lambda + a_L (b_L - b_H) \lambda^2 \mod N \]

**Algorithm 1** Memory-optimized MAC operation

**Input:** Aligned \( s \)-word integers \( A = (A_{s-1}, \ldots, A_1, A_0) \), \( B = (B_{s-1}, \ldots, B_1, B_0) \), and \( R = (R_{s-1}, \ldots, R_1, R_0) \), each word contains \( \omega \) bits; \( \beta \) is a parameter of alignment

**Output:** Aligned \( s \)-word product \( R = R + A \cdot B \cdot \lambda \mod N = (R_{s-1}, \ldots, R_1, R_0) \)

```plaintext
1: \( Z_0 \leftarrow 0, Z_1 \leftarrow 0 \)
2: \( l \leftarrow s/2 \)
3: for \( i \) from 0 to \( l - 1 \) by 1 do
4: \( Z_2 \leftarrow 0, k \leftarrow i + 1 \)
5: for \( j \) from 0 to \( i \) by 1 do
6: \( k \leftarrow k - 1 \)
7: \( Z_0 \leftarrow Z_0 + A_{j+i} \cdot B_k \)
8: \( Z_1 \leftarrow Z_1 + (A_j + A_{j+i}) \cdot B_{k+l} \)
9: \( Z_2 \leftarrow Z_2 + A_j \cdot (B_k - B_{k+i}) \)
10: end for
11: \( Z_0 \leftarrow Z_0 - 2 \cdot Z_2 \)
12: \( k \leftarrow l \)
13: for \( j \) from \( i + 1 \) to \( l - 1 \) by 1 do
14: \( k \leftarrow k - 1 \)
15: \( Z_1 \leftarrow Z_1 + 2^\beta \cdot A_{j+i} \cdot B_k \)
16: \( Z_2 \leftarrow Z_2 + 2^\beta \cdot (A_j + A_{j+i}) \cdot B_{k+l} \)
17: \( Z_0 \leftarrow Z_0 + 2^\beta \cdot A_j \cdot (B_k - B_{k+i}) \)
18: end for
19: \( Z_0 \leftarrow Z_0 + Z_2 + R_i \)
20: \( Z_1 \leftarrow Z_1 + Z_2 + R_{i+l} \)
21: \( R_i \leftarrow Z_0 \mod 2^\omega \)
22: \( Z_0 \leftarrow Z_0^2 \mod 2^\omega \)
23: \( R_{i+1} \leftarrow Z_1 \mod 2^\omega \)
24: \( Z_1 \leftarrow Z_1 / 2^\omega \)
25: end for
26: \( Z_0 \leftarrow 2^\beta \cdot Z_0 + R_{i-1} / 2^\omega - \beta \)
27: \( Z_1 \leftarrow 2^\beta \cdot Z_1 + R_{i-1} / 2^\omega - \beta \)
28: \( R_{i-1} \leftarrow R_{i-1} \mod 2^\omega - \beta \)
29: \( R_{s-1} \leftarrow R_{s-1} \mod 2^\omega - \beta \)
30: \( Z_0 \leftarrow Z_0 + Z_1 \)
31: for \( i \) from 0 to \( l - 1 \) by 1 do
32: \( Z_1 \leftarrow Z_1 + R_i \)
33: \( R_i \leftarrow Z_1 \mod 2^\omega \)
34: \( Z_1 \leftarrow Z_1 / 2^\omega \)
35: end for
36: \( Z_0 \leftarrow 2^\beta \cdot Z_0 + R_{l-1} / 2^\omega - \beta \)
37: \( R_{l-1} \leftarrow R_{l-1} \mod 2^\omega - \beta \)
38: for \( i \) from \( l \) to \( s - 1 \) by 1 do
39: \( Z_0 \leftarrow Z_0 + R_i \)
40: \( R_i \leftarrow Z_0 \mod 2^\omega \)
41: \( Z_0 \leftarrow Z_0^2 \mod 2^\omega \)
42: end for
43: return \( (R_{s-1}, \ldots, R_1, R_0) \)
```

Final reduction modulo \( N \)

Carry propagation
Memory-Optimized MAC

\[ r := (r_L + a_H b_L - 2a_L (b_L - b_H)) + (r_H + (a_L + a_H) b_H) \lambda + a_L (b_L - b_H) \lambda^2 \mod N \]

**Algorithm 1** Memory-optimized MAC operation

**Input:** Aligned \( s \)-word integers \( A = (A_{s-1}, \ldots, A_0) \), \( B = (B_{s-1}, \ldots, B_1, B_0) \), and \( R = (R_{s-1}, \ldots, R_1, R_0) \), each word contains \( \omega \) bits; \( \beta \) is a parameter of alignment

**Output:** Aligned \( s \)-word product \( R = R + A \cdot B \cdot \lambda \mod N = (R_{s-1}, \ldots, R_1, R_0) \)

1. \( Z_0 \leftarrow 0, Z_1 \leftarrow 0 \)
2. \( l \leftarrow s/2 \)
3. **for** \( i \) from 0 to \( l - 1 \) by 1 **do**
4. \( Z_2 \leftarrow 0, k \leftarrow i + 1 \)
5. **for** \( j \) from 0 to \( i \) by 1 **do**
6. \( k \leftarrow k - 1 \)
7. \( Z_0 \leftarrow Z_0 + A_{j+i} \cdot B_k \)
8. \( Z_1 \leftarrow Z_1 + (A_j + A_{j+i}) \cdot B_{k+l} \)
9. \( Z_2 \leftarrow Z_2 + A_j \cdot (B_k - B_{k+l}) \)
10. **end for**
11. \( Z_0 \leftarrow Z_0 - 2 \cdot Z_2 \)
12. \( k \leftarrow l \)
13. **for** \( j \) from \( i + 1 \) to \( l - 1 \) by 1 **do**
14. \( k \leftarrow k - 1 \)
15. \( Z_1 \leftarrow Z_1 + 2^{\beta} \cdot A_{j+i} \cdot B_k \)
16. \( Z_2 \leftarrow Z_2 + 2^{\beta} \cdot (A_j + A_{j+i}) \cdot B_{k+l} \)
17. \( Z_0 \leftarrow Z_0 + 2^{\beta} \cdot A_j \cdot (B_k - B_{k+l}) \)
18. **end for**
19. \( Z_0 \leftarrow Z_0 + Z_3 + R_i \)
20. \( Z_1 \leftarrow Z_1 + Z_2 + R_{i+l} \)
21. \( R_i \leftarrow Z_0 \mod 2^\omega \)
22. \( Z_0 \leftarrow Z_0/2^\omega \)
23. \( R_{i+l} \leftarrow Z_1 \mod 2^\omega \)
24. \( Z_1 \leftarrow Z_1/2^\omega \)
25. **end for**
26. \( Z_0 \leftarrow 2^\beta \cdot Z_0 + R_{i-1}/2^{\omega-\beta} \)
27. \( Z_1 \leftarrow 2^\beta \cdot Z_1 + R_{i-1}/2^{\omega-\beta} \)
28. \( R_{i-1} \leftarrow R_{i-1} \mod 2^{\omega-\beta} \)
29. \( R_{s-1} \leftarrow R_{s-1} \mod 2^{\omega-\beta} \)
30. \( Z_0 \leftarrow Z_0 + Z_1 \)
31. **for** \( i \) from 0 to \( l - 1 \) by 1 **do**
32. \( Z_1 \leftarrow Z_1 + R_i \)
33. \( R_i \leftarrow Z_1 \mod 2^\omega \)
34. \( Z_1 \leftarrow Z_1/2^\omega \)
35. **end for**
36. \( Z_0 \leftarrow 2^\beta \cdot Z_0 + R_{i-1}/2^{\omega-\beta} \)
37. \( R_{i-1} \leftarrow R_{i-1} \mod 2^{\omega-\beta} \)
38. **for** \( i \) from \( l \) to \( s - 1 \) by 1 **do**
39. \( Z_0 \leftarrow Z_0 + R_i \)
40. \( R_i \leftarrow Z_0 \mod 2^\omega \)
41. \( Z_0 \leftarrow Z_0/2^\omega \)
42. **end for**
43. **return** \((R_{s-1}, \ldots, R_1, R_0)\)
Memory-Optimized MAC

\[ r := (r_L + a_H b_L - 2a_L (b_L - b_H)) + (r_H + (a_L + a_H) b_H) \lambda + a_L (b_L - b_H) \lambda^2 \mod N \]

**Algorithm 2** First triple MAC loop

1: \( Z_2 \leftarrow 0, \ k \leftarrow i + 1 \)

2: **for** \( j \) from 0 to \( i \) by 1 **do**

3: \( \ k \leftarrow k - 1 \)

4: \( \ Z_0 \leftarrow Z_0 + A_{j+l} \cdot B_k \)

5: \( \ Z_1 \leftarrow Z_1 + (A_j + A_{j+l}) \cdot B_{k+l} \)

6: \( \ Z_2 \leftarrow Z_2 + A_j \cdot (B_k - B_{k+l}) \)

7: **end for**

8: \( \ Z_0 \leftarrow Z_0 - 2 \cdot Z_2 \)
Memory-Optimized MAC

\[ r := (r_L + a_H b_L - 2a_L (b_L - b_H)) + (r_H + (a_L + a_H) b_H) \lambda + a_L (b_L - b_H) \lambda^2 \mod N \]

**Algorithm 3** Second triple MAC loop

1: \[ k \leftarrow l \]
2: \[ \textbf{for } j \text{ from } i + 1 \text{ to } l - 1 \text{ by } 1 \text{ do} \]
3: \[ k \leftarrow k - 1 \]
4: \[ Z_1 \leftarrow Z_1 + 2^\beta \cdot A_{j+i} \cdot B_k \]
5: \[ Z_2 \leftarrow Z_2 + 2^\beta \cdot (A_j + A_{j+i}) \cdot B_{k+l} \]
6: \[ Z_3 \leftarrow Z_3 + 2^\beta \cdot A_j \cdot (B_k - B_{k+l}) \]
7: \[ \textbf{end for} \]
8: \[ Z_1 \leftarrow Z_1 - 2 \cdot Z_3 \]

Product of \( A \cdot B \)
\[ r := (r_L + a_H b_L - 2a_L (b_L - b_H)) + (r_H + (a_L + a_H) b_H) \lambda + a_L (b_L - b_H) \lambda^2 \mod N \]

**Algorithm 3** Second triple MAC loop

1: \( k \leftarrow l \)

2: **for** \( j \) from \( i + 1 \) to \( l - 1 \) **do**

3: \( k \leftarrow k - 1 \)

4: \( Z_1 \leftarrow Z_1 + 2^\beta \cdot A_{j+l} \cdot B_k \)

5: \( Z_2 \leftarrow Z_2 + 2^\beta \cdot (A_j + A_{j+l}) \cdot B_{k+l} \)

6: \( Z_3 \leftarrow Z_3 + 2^\beta \cdot A_j \cdot (B_k - B_{k+l}) \)

7: **end for**

8: \( Z_1 \leftarrow Z_1 - 2 \cdot Z_3 \)

\[ \lambda^3 = (\lambda + 1) \cdot \lambda = \lambda^2 + \lambda = (\lambda + 1) + \lambda = 2\lambda + 1 \mod N \]

\[ Z_0 \leftarrow Z_0 + Z_3 \]

\[ Z_1 \leftarrow Z_1 - 2 \cdot Z_3 + 2 \cdot Z_3 = Z_1 \]
Memory-Optimized MAC

\[ r := (r_L + a_H b_L - 2a_L(b_L - b_H)) + (r_H + (a_L + a_H) b_H) \lambda + a_L(b_L - b_H) \lambda^2 \mod N \]

**Algorithm 3** Second triple MAC loop

1: \( k \leftarrow l \)

2: for \( j \) from \( i + 1 \) to \( l - 1 \) by 1 do

3: \( k \leftarrow k - 1 \)

4: \( Z_1 \leftarrow Z_1 + 2^\beta \cdot A_{j+l} \cdot B_k \)

5: \( Z_2 \leftarrow Z_2 + 2^\beta \cdot (A_j + A_{j+l}) \cdot B_{k+l} \)

6: \( Z_0 \leftarrow Z_0 + 2^\beta \cdot A_j \cdot (B_k - B_{k+l}) \)

7: end for

\( \lambda^3 = \lambda^2 \cdot \lambda = (\lambda + 1) \cdot \lambda = \lambda^2 + \lambda = (\lambda + 1) + \lambda = 2\lambda + 1 \mod N \)

\( Z_0 \leftarrow Z_0 + Z_3 \)

\( Z_1 \leftarrow Z_1 - 2 \cdot Z_3 + 2 \cdot Z_3 = Z_1 \)

product of \( A \cdot B \)
Reverse Product Scanning (RPS) multiplication [LSGK14]

- Enhanced variant of conventional hybrid multiplication [GPW+04]
- Fast, small-code-size, fewer-registers and parameterized

Inner-loop operations of RPS multiplication (middle)
Speed-Optimized MAC

\[ r := (r_L + a_H b_H - (a_L - a_H)(b_L - b_H)) + (r_H + a_L b_L + a_H b_H) \lambda \mod N \]
\[ = (r_L + h + m) + (r_H + l + h) \lambda \mod N \]
\[ = (r_L + (h_L + h_H \lambda) + (m_L + m_H \lambda)) + (r_H + (l_L + l_H \lambda) + (h_L + h_H \lambda)) \lambda \mod N \]
\[ = (r_L + h_L + m_L) + (r_H + l_L + h_L + m_H + h_H) \lambda + (l_H + h_H) \lambda^2 \mod N \]
\[ = (r_L + m_L + \underbrace{h_L + h_H + l_H}) + (r_H + m_H + h_H + l_L + \underbrace{h_L + h_H + l_H}) \lambda \mod N \] (5)
Speed-Optimized MAC

\[ r := (r_L + m_L + h_L + h_H + l_H) + (r_H + m_H + h_H + l_L + h_L + h_H + l_H) \lambda \mod N \]

**Experiments for speed-optimized MAC**

- **Combination**
  - Subtractive Karatsuba method \([HS14]\) \(\Theta(n^{\log_2 3})\)
  - RPS multiplication \([LSGK14]\) \(\Theta(n^2)\)

- **Result**
  - 3-level Karatsuba with \((390 \times 390)\)-bit RPS multiplication underneath for the entire MAC
MAC Optimization Strategies

- Memory-optimized MAC operation
  - Equation (4)
  - one-level Karatsuba multiplication (product-scanning)
  - RPS technique for inner-loop operation

- Speed-optimized MAC operation
  - Equation (5)
  - three-level Karatsuba multiplication
  - RPS multiplication
Measurement Environment

**Experiment Setup**

- Target MCU: ATmega1284 (16 kB RAM; 128 kB flash memory)
- Development tool: Atmel Studio v7.0
- Compiler: avr-gcc 5.4.0

**Our source code**

- AVR Assembler: MAC operation; Keccak permutation
- C code: other components
Performance Evaluation

Execution time (in clock cycles) of our implementations on AVR

| Implementation     | Security  | MAC       | KeyGen    | Encaps    | Decaps   |
|--------------------|-----------|-----------|-----------|-----------|----------|
| ME-BBear           | CCA-secure| 1,033,728 | 8,746,418 | 12,289,744| 18,578,335|
| ME-BBear-Eph       | CPA-secure| 1,033,728 | 8,746,418 | 12,435,165| 3,444,154 |
| HS-BBear           | CCA-secure| 604,703   | 6,123,527 | 7,901,873 | 12,476,447|
| HS-BBear-Eph       | CPA-secure| 604,703   | 6,123,527 | 8,047,835 | 2,586,202 |

HS version is \(1.5x\) faster compared to ME
# Performance Evaluation

RAM usage and code size (both in bytes) of our implementations on AVR

| Implementation     | MAC RAM  | KeyGen RAM | Encaps RAM | Decaps RAM | Total RAM |
|--------------------|----------|------------|------------|------------|-----------|
| ME-BBear           | 82       | 1,715      | 1,735      | 2,368      | 2,368     |
| ME-BBear-Eph       | 82       | 1,715      | 1,735      | 1,731      | 1,735     |
| HS-BBear           | 934      | 2,733      | 2,752      | 4,559      | 4,559     |
| HS-BBear-Eph       | 934      | 2,733      | 2,752      | 2,356      | 2,752     |

ME version is **1.5x** RAM-efficient compared to HS.
Comparison – AVR Implementations

Comparison with other key-establishment algorithms (all of which target 128-bit security) on 8-bit AVR (Encaps and Decaps in clock cycles; RAM and code size in bytes)

| Implementation         | Algorithm | Encaps   | Decaps   | RAM  | Size  |
|------------------------|-----------|----------|----------|------|-------|
| This work (ME-CCA)     | ThreeBears| 12,289,744 | 18,578,335 | 2,368 | 12,264 |
| This work (ME-CPA)     | ThreeBears| **12,435,165** | **3,444,154** | **1,735** | 10,998 |
| This work (HS-CCA)     | ThreeBears| 7,901,873 | 12,476,447 | 4,559 | 11,568 |
| This work (HS-CPA)     | ThreeBears| 8,047,835 | 2,586,202 | 2,752 | 10,296 |
| [CDG+19]               | NTRU Prime| 8,160,665 | 15,602,748 | n/a  | 11,478 |
| [DHH+15] (ME)          | Curve25519| 14,146,844 | 14,146,844 | 510  | 9,912 |
| [DHH+15] (HS)          | Curve25519| **13,900,397** | **13,900,397** | **494** | 17,710 |

Encaps $1.12x$ faster; Decaps $4.0x$ faster; RAM $3.5x$ more; compared to Curve25519
Comparison – RAM Footprint

Comparison of RAM consumption (in bytes) of NIST PQC implementations (all of which target NIST security category 1 or 2) on AVR and Cortex-M4 microcontrollers

| Implementation | Algorithm | Platform | KeyGen | Encaps | Decaps |
|----------------|-----------|----------|--------|--------|--------|
| **CCA-secure schemes** | | | | | |
| This work (ME) | ThreeBears | AVR | 1,715 | 1,735 | **2,368** |
| [Ham19] pqm4 | ThreeBears | Cortex-M4 | 2,288 | 2,352 | 3,024 |
| pqm4 | ThreeBears | Cortex-M4 | 3,076 | 2,964 | 5,092 |
| pqm4 | Kyber | Cortex-M4 | 2,388 | 2,476 | 2,492 |
| pqm4 | NTRU | Cortex-M4 | **11,848** | 6,864 | 5,144 |
| pqm4 | Saber | Cortex-M4 | 9,652 | 11,388 | 12,132 |
| **CPA-secure schemes** | | | | | |
| This work (ME) | ThreeBears | AVR | 1,715 | **1,735** | 1,731 |
| [Ham19] pqm4 | ThreeBears | Cortex-M4 | 2,288 | 2,352 | 2,080 |
| pqm4 | ThreeBears | Cortex-M4 | **3,076** | 2,980 | 2,420 |
| pqm4 | NewHope | Cortex-M4 | 3,836 | **4,940** | 3,200 |
| pqm4 | Round5 | Cortex-M4 | 4,052 | **4,500** | 2,308 |

The most RAM-efficient software implementation of Round 2 candidates
The first highly-optimized Assembler implementation of ThreeBears for AVR

Many trade-offs between execution time and RAM consumption are possible

A new record for memory efficiency among second-round candidates

Very well suited for a hybrid pre/post-quantum key agreement protocol

An excellent candidate for a post-quantum cryptosystem to secure the IoT
Thank you for your attention!