Verified NTT Multiplications for NISTPQC KEM Lattice Finalists: KYBER, SABER, NTRU

Vincent Hwang, Jiaxiang Liu, Gregor Seiler, Xiaomu Shi, Ming-Hsien Tsai, Bow-Yaw Wang, Bo-Yin Yang
Postquantum Cryptography (PQC)

• A large-scale quantum computer breaks RSA and ECC by Shor’s algorithm
  • Postquantum Cryptography (PQC)
• PQC standardization process (NISTPQC) initiated by NIST
  • 7 finalists (KYBER, SABER, NTRU, ...) and 8 alternate candidates in the 3rd round
Implementation Issues

- Cryptography is always under a lot of pressure to be efficient
- Every round-3 submission in NISTPQC includes hand-optimized software
- PQC tends to be also more complex than pre-quantum public-key cryptography
- Bugs in PQC implementations?
• Consider the field multiplication over $\mathbb{F}_p$ with $p = 2^{255} - 19$.
• There are roughly $2^{510} (= 2^{255} \times 2^{255})$ different inputs.
• How many of them can be tested?
  • What about those inputs which are never tested?
  • “Testing shows the presence, not the absence of bugs.”
    E. W. Dijkstra (1969)
• Formal verification aims to prove the absence of bugs through logical or mathematical reasoning.
  • That is, the field multiplication is computed correctly for all inputs.
Functional Correctness

- Testing only checks that an implementation is correct on a fixed set of selected inputs
- Formal verification can reach a conclusion that the implementation computes the correct outputs for all possible inputs
- CRYPTOLine\(^1\) was developed to help programmers write correct cryptographic assembly programs
  - A domain-specific language for modeling cryptographic assembly programs and their specifications
  - A tool for verifying programs in the domain-specific language
  - Support two kinds of predicates
    - Algebraic predicates: non-linear (modular) equations over integers
    - Range predicates: bit-accurate comparisons, equations, or modular equations

\(^1\)https://github.com/fmlab-iis/cryptoline
Our Contributions

First verification of highly complex polynomial multiplications based on the Number Theoretic Transform (NTT)

<table>
<thead>
<tr>
<th></th>
<th>Intel AVX2</th>
<th>ARM Cortex-M4</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTRU</td>
<td>ntt-polymul² build 3e42fffa</td>
<td>pqm4³ build d26fee0</td>
</tr>
<tr>
<td>KYBER</td>
<td>PQClean⁴ build 688ff2f</td>
<td>pqm4³ build 688ff2f</td>
</tr>
<tr>
<td>SABER</td>
<td>ntt-polymul² build 3e42fffa</td>
<td>Strategy A by [ACC+22]⁵</td>
</tr>
</tbody>
</table>

Extension of the CRYPTO LINE tool

- Verification either much slower or impossible without these extensions

²https://github.com/ntt-polymul/ntt-polymul
³https://github.com/mupq/pqm4
⁴https://github.com/PQClean/PQClean
⁵https://github.com/multi-moduli-ntt-saber/multi-moduli-ntt-saber
AVX2 Kyber768 NTT

- The incomplete NTT in the Intel AVX2 implementation from PQClean does the following map:

\[
\begin{align*}
\mathbb{Z}_q[X]/\langle X^{256} + 1 \rangle & \to \mathbb{Z}_q[X]/\langle X^{128} - \omega_4 \rangle \times \mathbb{Z}_q[X]/\langle X^{128} + \omega_4 \rangle \quad \text{(level 0)} \\
& \to \ldots \\
& \to \mathbb{Z}_q[X]/\langle X^2 - \zeta_{6,0} \rangle \times \cdots \times \mathbb{Z}_q[X]/\langle X^2 - \zeta_{6,127} \rangle \quad \text{(level 6)}
\end{align*}
\]

where \( \zeta_{i,j} \) is the roots of unity used at the end of level \( i \) (counting up)

- Cut at each level to decompose the verification problem
Workflow of Verifying AVX2 KYBER768 NTT

\[ F \equiv G_{0,0} \mod [q, X^{128} - \omega_4] \]
\[ \vdots \]
\[ F \equiv G_{i,j} \mod [q, X^{2^{2i+1}} - \zeta_{i,j}], 0 \leq j < 2^i \]
\[ F \equiv G_{6,j} \mod [q, X^2 - \zeta_{6,j}], 0 \leq j < 64 \]

\[ F \equiv G_{0,1} \mod [q, X^{128} + \omega_4] \]
\[ \vdots \]
\[ F \equiv G_{i,j} \mod [q, X^{2^{2i+1}} + \omega_4 - \zeta_{i,j}], 2^i \leq j < 2^{i+1} \]
\[ F \equiv G_{6,j} \mod [q, X^2 + \zeta_{6,j}], 64 \leq j < 128 \]

- All 256 coefficients used in level 0; at most 128 needed at level 1 onwards
Verification of AVX2 KYBER768 NTT

Step 1: **running trace (in assembly)**
Extract from an executable by our script *itrace.py*

```
$ itrace.py test ntt PQCLEAN_KYBER768_AVX2_polyvec_ntt.gas
$ more PQCLEAN_KYBER768_AVX2_polyvec_ntt.gas
#PQCLEAN_KYBER768_AVX2_polyvec_ntt:
::
  # [some bookkeeping information]
::
    vmovdqa (%rsi),%ymm0  #! EA = L0x55555556395e0; Value = ...
    vpbroadcastq 0x140(%rsi),%ymm15  #! EA = L0x5555555639720; Value = ...
    vmovdqa 0x100(%rdi),%ymm8  #! EA = L0x7fffffffb080; Value = ...
::
    vpbroadcastq 0x148(%rsi),%ymm2  #! EA = L0x5555555639728; Value = ...
    vpmullw %ymm15,%ymm8,%ymm12  #! PC = 0x55555556eb85
::
    vpmulhw %ymm2,%ymm8,%ymm8  #! PC = 0x55555556eb99
```
Verification of AVX2 KYBER768 NTT

Step 2: Define translation between assembly and CRYPTO instructions (usually standard and reusable)

```c
#! $1c(%rsi) = %%EA
#! (%rsi) = %%EA
#! $1c(%rdi) = %%EA
#! (%rdi) = %%EA
#! %ymm$1c = %%ymm$1c

#! vpbroadcastq $1ea, $2v -> mov $2v_0 $1ea; \n
mov $2v_2 $1ea[+4]; \n
mov $2v_5 $1ea[+2]; \n
mov $2v_7 $1ea[+4]; \n
mov $2v_0 $1ea[+2]; \n
mov $2v_1 $1ea[+6]; \n
mov $2v_6 $1ea[+4]; \n
mov $2v_3 $1ea[+6]; ...

#! vmovdqa $1ea, $2v -> mov $2v_0 $1ea; \n
mov $2v_2 $1ea[+4]; \n
mov $2v_3 $1ea[+6]; ...

#! vmovdqa $1v, $2ea -> mov $2ea $1v_0; \n
mov $2ea[+4] $1v_2; \n
mov $2ea[+6] $1v_3; ...
```
Verification of AVX2 KYBER768 NTT

Step 3: to_zdsl.py translates running trace to CRYPTOINE program

```plaintext
proc main( [inputs] ) =
{ [precondition to be defined] }
(* vmovdqa (%rsi),%ymm0           ! EA = L0x5555556395e0; ... *)
mov ymm0_0 L0x5555556395e0;
:
mov ymm0_f L0x5555556395fe;
(* vpbroadcastq 0x140(%rsi),%ymm15  ! EA = L0x555555639720; ... *)
mov ymm15_0 L0x555555639720;
:
mov ymm15_f L0x555555639726;
(* vmovdqa 0x100(%rdi),%ymm8      ! EA = L0x7ffffffffb080; ... *)
{: 
{ [postcondition to be defined] }
```

CHES2022  Friends + B.-Y. Yang
Verification of AVX2 KYBER768 NTT iv

Step 4: Initialize constants used in the subroutine

(********** constants **********)

mov L0x55555556395e0 ( 3329)@sint16; mov L0x55555556395e2 ( 3329)@sint16;
... 
mov L0x5555555639600 ( -3327)@sint16; mov L0x5555555639602 ( -3327)@sint16;
... 
mov L0x5555555639620 ( 20159)@sint16; mov L0x5555555639622 ( 20159)@sint16;
... 
mov L0x5555555639adc ( 32)@sint16; mov L0x5555555639ade ( 32)@sint16;
Verification of AVX2 KYBER768 NTT

Step 5: pre-condition, the post-condition, and mid-conditions (mid-conditions not required for AVX2 KYBER768 NTT, easy to generate using a script, result in less verification time)

Precondition

\[-q < f_i < q \text{ for all } 0 \leq i < 256 \text{ where } f_i \text{'s are the inputs and } q = 3329\]

Midconditions and postcondition

\[F \equiv G_{i,j} \mod [q, X^{256/2^{i+1}} - \zeta_{i,j}] \text{ for all } 0 \leq j < 2^{i+1}\]

and

\[-(2 + i)q < g_{i,j,k} < (2 + i)q \text{ for all } 0 \leq j < 2^{i+1}, 0 \leq k < 256/2^{i+1}.\]

where \(i\) is the NTT level (from 0 to 6)
Verification of AVX2 KYBER768 NTT vi

Step 6: Run CRYPTOLINE, wait (human interaction no longer needed)

$ cv -v -isafety -jobs 24 -slicing -no_carry_constraint \ 
   PQCLEAN_KYBER768_AVX2_polyvec_ntt.cl

Parsing Cryptoline file: [OK] 0.089273 seconds
Checking well-formedness: [OK] 0.031599 seconds
Transforming to SSA form: [OK] 0.019121 seconds
Rewriting assignments: [OK] 0.020577 seconds
Verifying program safety: [OK] 183.994889 seconds
Verifying range assertions: [OK] 42.385435 seconds
Verifying range specification: [OK] 200.594131 seconds
Rewriting value-preserved casting: [OK] 0.001421 seconds
Verifying algebraic assertions: [OK] 0.007455 seconds
Verifying algebraic specification: [OK] 26.648724 seconds
Verification result: [OK] 453.802915 seconds
Classical Compositional Reasoning

- Consider the following program snippet:

\[
\text{cut : } P_0 \land P_1 \land \cdots \land P_{127}
\]

\[
\text{cut : } Q_0 \land Q_1 \land \cdots \land Q_{127}
\]

\[
\cdots
\]

- It happens in inverse NTT that \( Q_i \) only depends on \( P_i \) but \( Q_i, P_i \), and many other \( P_j \)'s involve common variables
  - Those \( P_j \)'s cannot be excluded systematically when verifying \( Q_i \)
  - Verification is quite inefficient or even impossible in such cases

- Proposed solution: nonlocal compositional reasoning
In Nonlocal Compositional Reasoning

- Each cut instruction is assigned to a number for reference
- Verifiers can add relevant premises by cut numbers

\[
\text{cut } 0: \quad P_0 \land P_1 \land \cdots \land P_{127} \\
\text{cut } 1: \quad P_0 \text{ prove with } 0 \\
\text{cut } 2: \quad P_1 \text{ prove with } 0 \\
\vdots
\text{cut } 128: \quad P_{127} \text{ prove with } 0 \\
\text{cut } 129: \quad \text{true} \\
\text{[code]}
\]

\[
\text{cut } 130: \quad Q_0 \text{ prove with } 1 \land Q_1 \text{ prove with } 2 \land \cdots \\
\text{[code]}
\]
Twisted NTT

- Mapping $X = aY$ from $\mathbb{F}[X]/\langle X^n - c \rangle$ to $\mathbb{F}[Y]/\langle Y^n - 1 \rangle$ is called twisting.

$$\frac{\mathbb{F}[X]}{\langle X^{2n} - 1 \rangle} \cong \frac{\mathbb{F}[X]}{\langle X^n - 1 \rangle} \times \frac{\mathbb{F}[X]}{\langle X^n + 1 \rangle}$$

- Two approaches of specifying twisted NTT:
  - With fresh variables $Y_{i,j}$ (ARM Cortex-M4 SABER)
  - Without fresh variables (Intel AVX2 SABER)
<table>
<thead>
<tr>
<th>KEM</th>
<th>architecture</th>
<th>direction</th>
<th>algebra</th>
<th>overflow</th>
<th>range</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kyber768</td>
<td>AVX2</td>
<td>normal</td>
<td>26.6</td>
<td>183.9</td>
<td>242.8</td>
<td>453.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inverse</td>
<td>761.7</td>
<td>781.0</td>
<td>6050.0</td>
<td>7593.5</td>
</tr>
<tr>
<td></td>
<td>Cortex M4</td>
<td>normal</td>
<td>134.3</td>
<td>173.7</td>
<td>191.0</td>
<td>499.4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inverse</td>
<td>1481.0</td>
<td>348.6</td>
<td>184.1</td>
<td>2014.3</td>
</tr>
<tr>
<td>ntru2048509</td>
<td>AVX2</td>
<td>normal</td>
<td>478.4</td>
<td>1229.8</td>
<td>1738.6</td>
<td>3447.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inverse</td>
<td>3868.6</td>
<td>1545.3</td>
<td>12170.3</td>
<td>17585.7</td>
</tr>
<tr>
<td></td>
<td>Cortex M4</td>
<td>normal</td>
<td>1353.0</td>
<td>5970.7</td>
<td>4810.2</td>
<td>12135.2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inverse</td>
<td>11315.1</td>
<td>3019.6</td>
<td>7813.7</td>
<td>22150.9</td>
</tr>
<tr>
<td>Saber</td>
<td>AVX2</td>
<td>normal</td>
<td>60.1</td>
<td>207.7</td>
<td>271.7</td>
<td>539.9</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inverse</td>
<td>436.2</td>
<td>443.8</td>
<td>859.4</td>
<td>1741.0</td>
</tr>
<tr>
<td></td>
<td>Cortex M4</td>
<td>normal</td>
<td>110.2</td>
<td>2731.9</td>
<td>2196.7</td>
<td>5039.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>inverse</td>
<td>3250.5</td>
<td>2754.0</td>
<td>853.4</td>
<td>6858.8</td>
</tr>
</tbody>
</table>

min: 453.8 seconds (≈ 8 minutes)  
max: 22150.9 seconds (≈ 6 hours)
Effectiveness of Cuts in Intel AVX2 KYBER768 NTT

- Number of cuts vs. time
- Green dotted line: algebra
- Red line: overflow
- Blue dashed line: range

Time (in milliseconds):
- 0
- 200
- 400
- 600
- 800
- 1000
- 1200
- 1400
- 1600
- 1800

Number of cuts:
- 1
- 3
- 4
- 6
- 8
- 10
- 12
- 14

CHES2022 Friends + B.-Y. Yang
Human Time

• Each of our verifications took less than a week of calendar time
• The majority of it was really communication with the programmer of the code, and secondly reading and gaining a basic understanding of the program at hand
Conclusion

• We demonstrate the feasibility for a programmer to verify his or her high-speed assembly code for PQC
• We demonstrate the feasibility for a verification specialist to verify someone else’s high-speed PQC software in assembly code, with some cooperation from the programmer
• Enhanced compositional reasoning techniques take full advantage of clearly demarcated stages in many cryptographic algorithms
• We did find a few bugs in high-speed software
Future Work

• The same technique applies to also
  • any implementation of small ideal-lattice-based cryptosystems that also has
    NTT-based arithmetic, e.g., the KEMs NTRU Prime, LAC, or NewHope and the
    signatures Dilithium and Falcon
  • a myriad of other architectures and other parameter sets

• Extend CRYPTOline to other PQCs such as Rainbow/UOV and Classic McEliece

• Watch out, we can do symmetric cryptography soon!
Thank you for listening