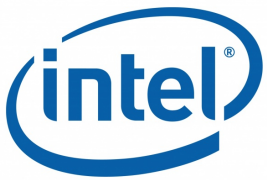
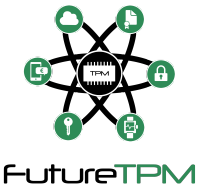
A Lightweight Implementation of NTRU Prime for the Post-Quantum Internet of Things

Hao Cheng [1] Daniel Dinu [2] Johann Großschädl [1]
Peter B. Rønne [1] Peter Y. A. Ryan [1]

[1]SnT and CSC, University of Luxembourg

[2]IPAS, Intel



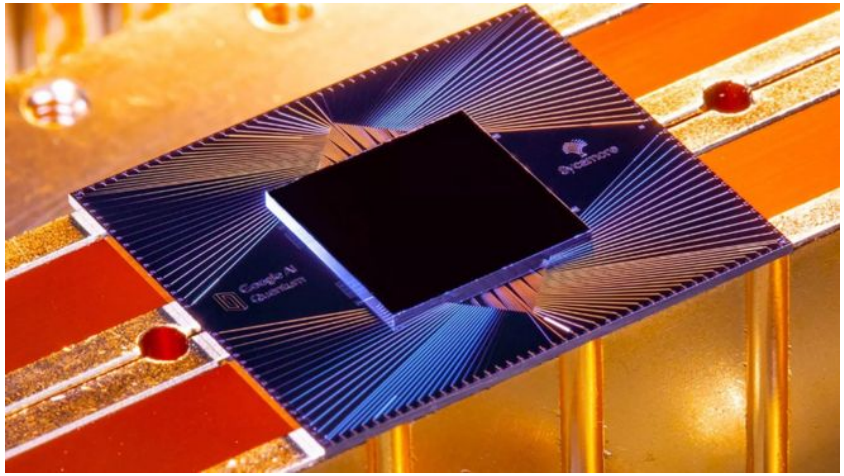


WISTP 2019, 11-12 December 2019, Paris, France

Outline

Outline

Quantum Cryptanalysis



Algorithms for Quantum Computation:
Discrete Logarithms and Factoring

Peter W. Shor
AT&T Bell Labs
Room 2D-149
600 Mountain Ave.
Murray Hill, NJ 07974, USA

Abstract

A computer is generally considered to be a universal computational device; i.e., it is believed able to simulate any physical computational device with a cost in computation time of at most a polynomial factor. It is not clear whether this is still true when quantum mechanics is taken into consideration. Several researchers, starting with David Deutsch, have developed models for quantum mechanical computers and have investigated their computational properties. This paper gives Las Vegas algorithms for finding discrete logarithms and factoring integers on a quantum computer that take a number of steps which is polynomial in the input size, e.g., the number of digits of the integer to be factored. These two problems are generally considered hard on a classical computer and have been used as the basis of several proposed cryptosystems. (We thus give the first examples of quantum cryptanalysis.)

[1, 2]. Although he did not ask whether quantum mechanics conferred extra power to computation, he did show that a Turing machine could be simulated by the reversible unitary evolution of a quantum process, which is a necessary prerequisite for quantum computation. Deutsch [9, 10] was the first to give an explicit model of quantum computation. He defined both quantum Turing machines and quantum circuits and investigated some of their properties.

The next part of this paper discusses how quantum computation relates to classical complexity classes. We will thus first give a brief intuitive discussion of complexity classes for those readers who do not have this background. There are generally two resources which limit the ability of computers to solve large problems: time and space (i.e., memory). The field of analysis of algorithms considers the asymptotic demands that algorithms make for these resources as a function of the problem size. Theoretical computer scientists generally classify algorithms as effi-

- Quantum Computing
  - Exploits quantum-mechanical phenomena (superposition and entanglement)
  - Can solve certain hard problems efficiently
- Shor's Algorithm[1]
  - Integer Factorization, Discrete Logarithm in polynomial time
- Google publishes landmark quantum supremacy claim[2]

[1] Peter W. Shor. "Algorithms for Quantum Computation: Discrete Logarithms and Factoring". In: *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS '94)*. IEEE Computer Society Press, 1994, pp. 124–134.

[2] F. Arute et al. "Quantum Supremacy using a Programmable Superconducting Processor". In: *Nature* 574 (2019), pp. 505–510.

NIST PQC Standardization[3]


NIST

Information Technology Laboratory

COMPUTER SECURITY RESOURCE CENTER

PROJECTS | POST-QUANTUM CRYPTOGRAPHY

Post-Quantum Cryptography

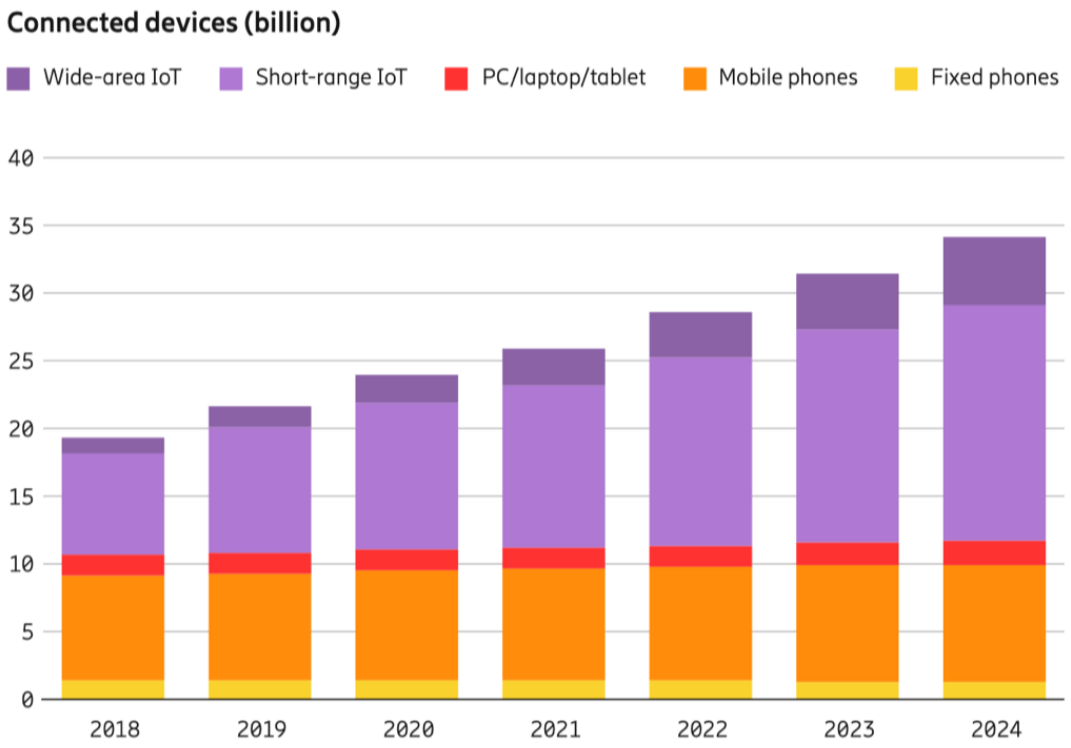
Post-Quantum Cryptography Standardization

The Round 2 candidates were announced January 30, 2019. NISTIR 8240, Status Report on the First Round of the NIST Post-Quantum Cryptography Standardization Process is now available.

- Solicit, evaluate and standardize **one or more** quantum-resistant PKC algorithms.
- 26 candidates in Round 2, 17 KEM/Encryption and 9 Signature schemes.
- NTRU Prime is the KEM candidate in Round 2.
- Performance (hardware + software) will play more of a role.

[3] https://csrc.nist.gov/Projects/post-quantum-cryptography/Post-Quantum-Cryptography-Standardization

Internet of Things



IoT connections (billion)

IoT	2018	2024	CAGR
Wide-area IoT	1.1	4.5	27%
Cellular IoT[2]	1.0	4.1	27%
Short-range IoT	7.5	17.8	15%
Total	8.6	22.3	17%

Figure: Internet of Things on the risel[4]

- IoT needs **lightweight** cryptosystems and protocols

[4] Ericsson Mobility Report (Jan 2019): https://www.ericsson.com/490532/assets/local/mobility-report/documents/2019/ericsson-mobility-report-world-economic-forum.pdf

8-bit AVR Microcontrollers

- 8-bit AVR Architecture
  - 8-bit RISC, 133 instructions
  - 32 general-purpose registers
  - Three 16-bit pointer registers: `X`, `Y`, and `Z`
  - Two-operand instruction format, e.g. "`ADD R0, R1`"
  - Most arithmetic/logic instructions take 1 cycle
  - Loads/Stores to/from RAM take 2 cycles
- ATmega1284 microcontroller: 16 KB RAM, 128 KB program memory
- One of the most constrained microcontrollers

Outline

Parameters and Rings

- NTRU Prime (Key-establishment Algorithms)
  - **Streamlined NTRU Prime**: Variant of classic NTRU
  - NTRU LPRime: Similar structure with NewHope (based on RLWE)
- Parameters and Notations
  - p: the number of coefficients (must be **prime**), e.g. 653, 761 and 857
  - q: modulus of the ring (must be **prime**), e.g. 4621, 4591 and 5167
  - w: weight of the polynomial (the number of non-0 coefficients)
  - *small*: an element of R if all of its coefficients are in $\{$-1, 0, 1$\}$.
  - Short: the set of *small* weight-w elements of $\mathcal{R}$.
- Rings: $(\mathbb{Z}/m)$ means the set of integers in $(-m/2, m/2]$
  - $\mathcal{R}$: ring $\mathbb{Z}[x]/(x^p - x - 1)$
  - $\mathcal{R}/3$: ring $(\mathbb{Z}/3)[x]/(x^p - x - 1)$
  - $\mathcal{R}/q$: field $(\mathbb{Z}/q)[x]/(x^p - x - 1)$

Key Generation

1. Generate a uniform random *small* polynomial $g(x) \in \mathcal{R}$ that is invertible in $\mathcal{R}/3$ (Repeat this step if $g(x)$ is not qualified).
2. Compute $v(x) = 1/g(x)$ in $\mathcal{R}/3$.
3. Generate a uniform random polynomial $f(x) \in$ Short.
4. Compute $h(x) = g(x)/(3f(x))$ in $\mathcal{R}/q$.
5. Output **public key $\boldsymbol{h(x)}$** and **private key $(\boldsymbol{f(x)}, \boldsymbol{v(x)})$**.

Encapsulation

1. Generate a uniform random polynomial $r(x) \in$ Short.
2. Compute $hr(x) = h(x) \star r(x) \in \mathcal{R}/q$ and then round each coefficient of $hr(x)$ to the nearest multiple of 3, the generated polynomial is **ciphertext $c(x)$**.
3. Hash (SHA-512-based) $r(x)$ together with $c(x)$ to obtain **session key $k(x)$**.

Decapsulation

1. Compute $e(x) = (3f(x) \star c(x) \in \mathcal{R}/q) \bmod 3$
   $= 3f(x) \star h(x) \star r(x) = 3f(x) \star (g(x)/3f(x)) \star r(x) = g(x) \star r(x)$.
2. Compute $r'(x) = e(x) \star v(x) \in \mathcal{R}/3$
   $g(x) \star r(x) \star v(x) = g(x) \star r(x) \star g^{-1}(x) = r(x)$.
3. Repeat the Step 2 of Encapsulation to generate $c'(x)$ by $r'(x)$.
4. Check whether $c'(x) = c(x)$: if they are not equal, set $r'(x)$ to be a new uniform random polynomial $\in$ Short.
5. Hash (`SHA-512`-based) $r'(x)$ together with $c(x)$ to obtain **session key $\boldsymbol{k(x)}$**.

Scheme Performance Analysis

- Arithmetic Operations
  - Multiplication between an element in $\mathcal{R}/q$ and Short
    - ★ $hr(x) = h(x) \star r(x) \in \mathcal{R}/q$ (Encap. step 2, Decap. step 3)
    - ★ $f(x) \star c(x) \in \mathcal{R}/q$ (Decap. step 1)
  - Multiplication between two elements in $\mathcal{R}/3$
    - ★ $r'(x) = e(x) \star v(x) \in \mathcal{R}/3$ (Decap. step 1)
- Auxiliary Functions
  - SHA-512 hash function
    - ★ Optimization of SHA-512 is based on our previous work[5], which sets the speed record of SHA-512 on 8-bit AVR platform
  - Encoding/Decoding
    - ★ Decoding the private key $f(x)$ is constant-time

Our work has the constant running time for the security-critical part, that is resistant for timing attacks.

[5] Hao Cheng, Daniel Dinu, and Johann Großschädl. "Efficient Implementation of the SHA-512 Hash Function for 8-Bit AVR Microcontrollers". In: *Innovative Security Solutions for Information Technology and Communications — SecITC 2018*.

Outline

Karatsuba-Based Polynomial Multiplication in $\mathcal{R}/3$

4-Level Karatsuba Multiplication (`mul_kara`) in $\mathcal{R}/3$ for `sntrup653`

$$\begin{aligned}(a+bX)*(c+dX) &= ac+(ad+bc)X+bdX^2 \\ &= ac+[(a-b)(d-c)+ac+bd]X+bdX^2\end{aligned}$$

1. Padding $p = 653$ coefficients to 656 (a multiple of 2^4) coeffcients
2. `mul_kara`($len = 656$)
3. 3 * `mul_kara`($len = 328$) + polynomial additions/subtractions
4. 3^2 * `mul_kara`($len = 164$) + polynomial additions/subtractions
5. 3^3 * `mul_kara`($len = 82$) + polynomial additions/subtractions
6. 3^4 * `school_book`($len = 41$) + polynomial additions/subtractions
7. Final polynomial reduction (mod $x^p - x - 1$)

Karatsuba-Based Polynomial Multiplication in $\mathcal{R}/3$

Hybrid School Book Multiplication[6] ($d = 4$) in $\mathcal{R}/3$ for `sntrup653`

Outer layer: product–scaning

$$R_i = \sum_{i=j+k} A_j * B_k$$

- $Z_i(z_0 \dots z_6)$ += $A_j(a_0 \dots a_3) * B_k(b_0 \dots b_3)$
- $R_i \leftarrow z_0, z_1, z_2, z_3$
- $z_0 \leftarrow z_4$; $z_1 \leftarrow z_5$; $z_2 \leftarrow z_6$; $z_3, z_4, z_5, z_6 \leftarrow 0$

Inner layer: operand–scaning

- $z_0 \mathrel{+}= a_0 * b_0$; $z_1 \mathrel{+}= a_1 * b_0$; $z_2 \mathrel{+}= a_2 * b_0$; $z_3 \mathrel{+}= a_3 * b_0$;
- $z_1 \mathrel{+}= a_0 * b_1$; $z_2 \mathrel{+}= a_1 * b_1$; $z_3 \mathrel{+}= a_2 * b_1$; $z_4 \mathrel{+}= a_3 * b_1$;
- $z_2 \mathrel{+}= a_0 * b_2$; $z_3 \mathrel{+}= a_1 * b_2$; $z_4 \mathrel{+}= a_2 * b_2$; $z_5 \mathrel{+}= a_3 * b_2$;
- $z_3 \mathrel{+}= a_0 * b_3$; $z_4 \mathrel{+}= a_1 * b_3$; $z_5 \mathrel{+}= a_2 * b_3$; $z_6 \mathrel{+}= a_3 * b_3$;

Perform modulo-3 reduction at the end of each hybrid school book multiplication. Maximal intermediate value is $2 * 2 * 41 = 164$ (8-bit).

[6] Nils Gura et al. "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs". In: *Cryptographic Hardware and Embedded Systems — CHES 2004*.

Karatsuba-Based Polynomial Multiplication in $\mathcal{R}/3$

Modulo-3 Reduction

`avr-gcc 4.8.2` for ATtiny45 (no hardware multiplier) ← `__udivmodhi4`

Cycles	Frequency	Percent (%)	Cycles	Frequency	Percent (%)
193	3	0.005	201	12244	18.683
194	45	0.069	202	7956	12.140
195	312	0.476	203	3825	5.836
196	1323	2.019	204	1323	2.019
197	3825	5.836	205	312	0.476
198	7956	12.140	206	45	0.069
199	12243	18.681	207	3	0.005
200	14121	21.547			

Figure: The execution time in cycles of the `__udivmodhi4` function for all possible 16-bit unsigned integer inputs.

Karatsuba-Based Polynomial Multiplication in $\mathcal{R}/3$

Modulo-3 Reduction

1. $b \leftarrow a \bmod 255$ (lines 1 to 2)
2. $c \leftarrow b \bmod 15$ (lines 3 to 11)
3. $d \leftarrow c \bmod 3$ (lines 12 to 21)
4. Final subtraction of 3 (lines 22 to 25)

Algorithm 1 Constant-Time Modulo 3 Reduction for 16-bit Unsigned Integer

Input: 16-bit unsigned integer $a = (\texttt{HIBYTE}, \texttt{LOBYTE})$, where HIBYTE represents the higher byte and LOBYTE represents the lower byte; ZERO is initially 0

Output: LOBYTE $\equiv a \bmod 3$

```
 1: ADD  LOBYTE, HIBYTE
 2: ADC  LOBYTE, ZERO

 3: MOV  HIBYTE, LOBYTE
 4: SWAP HIBYTE
 5: ANDI LOBYTE, 0x0F
 6: ANDI HIBYTE, 0x0F
 7: ADD  LOBYTE, HIBYTE
 8: MOV  HIBYTE, LOBYTE
 9: SWAP HIBYTE
10: ADD  LOBYTE, HIBYTE
11: ANDI LOBYTE, 0x0F

12: MOV  HIBYTE, LOBYTE
13: LSR  HIBYTE
14: LSR  HIBYTE
15: ANDI LOBYTE, 0x03
16: ADD  LOBYTE, HIBYTE
17: MOV  HIBYTE, LOBYTE
18: LSR  HIBYTE
19: LSR  HIBYTE
20: ANDI LOBYTE, 0x03
21: ADD  LOBYTE, HIBYTE

22: SUBI LOBYTE, 0x03
23: SBC  ZERO, ZERO
24: ANDI ZERO, 0x03
25: ADD  LOBYTE, ZERO

26: CLR  ZERO
```

Karatsuba-Based Polynomial Multiplication in $\mathcal{R}/q$

4-Level Karatsuba Multiplication (`mul_kara`) in $\mathcal{R}/q$ for `sntrup653`

Multiplication between an element in $\mathcal{R}/q$ and Short:

- $hr(x) = h(x) \star r(x) \in \mathcal{R}/q$ (Encap. step 2, Decap. step 3)
- $f(x) \star c(x) \in \mathcal{R}/q$ (Decap. step 1)

1. Padding $p = 653$ coefficients to 656 (a multiple of 2^4) coeffcients
2. `mul_kara`($len = 656$)
3. 3 * `mul_kara`($len = 328$) + polynomial additions/subtractions
4. 3^2 * `mul_kara`($len = 164$) + polynomial additions/subtractions
5. 3^3 * `mul_kara`($len = 82$) + polynomial additions/subtractions
6. 3^4 * `school_book`($len = 41$) + polynomial additions/subtractions
7. Final polynomial reduction (mod $x^p - x - 1$)

Karatsuba-Based Polynomial Multiplication in $\mathcal{R}/q$

Modulo-q Reduction for `sntrup653` ($q = 4621$)

- Perform modulo-q reduction at the end of each school book multiplication
- Maximal intermediate value's length is 30-bit ($4620 * 4620 * 41$)
- 30-bit unsigned integer modulo-q reduction
  1. $tmp \leftarrow \text{LUT1}(\texttt{b24}\ldots\texttt{b29}) + \text{LUT2}(\texttt{b16}\ldots\texttt{b23}) + (\texttt{b0}\ldots\texttt{b15})$
  2. $r \leftarrow \text{LUT3}(\texttt{t12}\ldots\texttt{t16}) + tmp\,\&\ \texttt{0xfff}$
  3. $r \leftarrow r - q \cdot (r \geqslant q)$

This polynomial multiplication in $\mathcal{R}/q$ occupies 70% of the whole execution time.

Product-Form Polynomial

- Product-form polynomial is in the fashion of $f(x) = f_1(x) \star f_2(x) + f_3(x)$
- Widely used in the classic NTRU[7]
- Proved to have constant running time in cache-less devices[8]
- A few researchers appeal to use this technique in NTRU Prime
- Multiplication between an element in $\mathcal{R}/q$ and Short:
  - $hr(x) = h(x) \star r(x) \in \mathcal{R}/q$ (Encap. step 2, Decap. step 3)
  - $\boldsymbol{f(x) \star c(x) \in \mathcal{R}/q}$ **(Decap. step 1)**
- The weight of sparse polynomial $f_1(x), f_2(x), f_3(x)$ is $(18, 16, 8)$
- 7.7 times faster than Karatsuba-based multiplication, just costs less than 1 million clock cycles

---

[7] Jeffrey Hoffstein and Joseph H. Silverman. "Optimizations for NTRU". In: *Public-Key Cryptography and Computational Number Theory*. 2001, pp. 77–88.

[8] Hao Cheng et al. "A Lightweight Implementation of NTRUEncrypt for 8-bit AVR Microcontrollers". In: *Proceedings of the 2nd NIST PQC Standardization Conference*. Available online at http://csrc.nist.gov/Events/2019/second-pqc-standardization-conference. 2019.

Product-Form Polynomial Multiplication

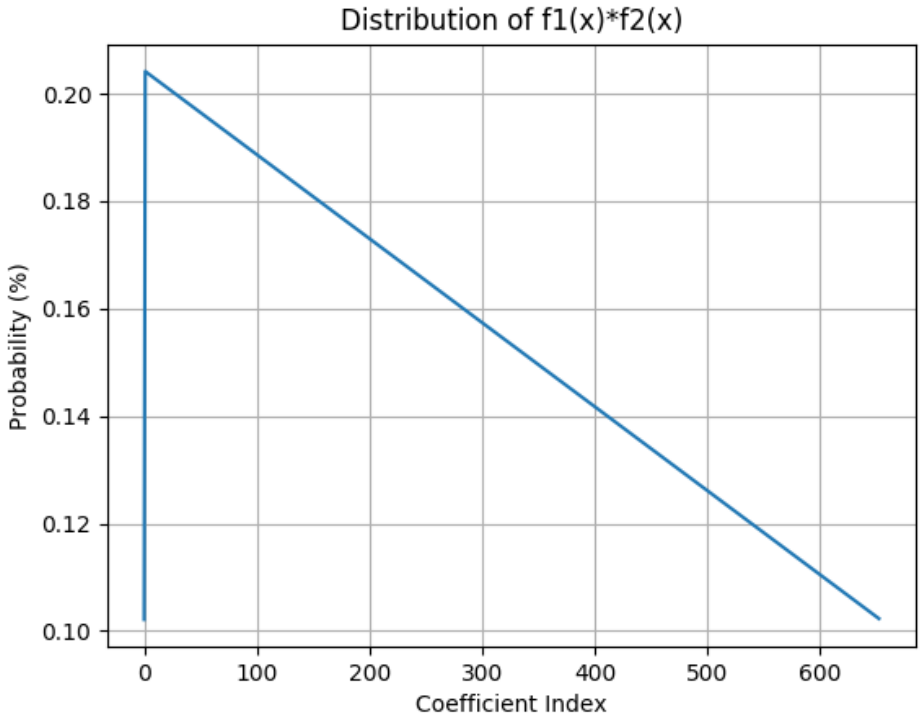
```
for (j = 0; j < blen; j ++)
{
  if (b[j] == 0) {b[j+9] = 0x0000;} else {b[j+9] = 0xFFFF; b[j] = N-b[j];}
}

while (i < loop_cnt)  // loop_cnt must be >= N and a multiple of 5
{
  sum0 = r[i  ]; sum1 = r[i+1]; sum2 = r[i+2]; sum3 = r[i+3];
  sum4 = r[i+4]; sum0 += sumx;  sumx = 0;

  for (j = 0; j < blen; j ++)
  {
    idx = b[j];                                 sum1 += a[idx]&b[j+9]; sum0 += a[idx++];
    sum2 += a[idx]&b[j+9]; sum1 += a[idx++]; sum3 += a[idx]&b[j+9]; sum2 += a[idx++];
    sum4 += a[idx]&b[j+9]; sum3 += a[idx++]; sumx += a[idx]&b[j+9]; sum4 += a[idx++];
    if (idx >= N) { b[j] = idx-N; b[j+9] &= 0x0000; }
    else          { b[j] = idx;   b[j+9] &= 0xFFFF; }
  }

  r[i++] = uint17_mod_q(sum0); r[i++] = uint17_mod_q(sum1); r[i++] = uint17_mod_q(sum2);
  r[i++] = uint17_mod_q(sum3); r[i++] = uint17_mod_q(sum4); sumx   = uint17_mod_q(sumx);
}
```

Please see details in our previous work[8]

[8] Hao Cheng et al. "A Lightweight Implementation of NTRUEncrypt for 8-bit AVR Microcontrollers". In: *Proceedings of the 2nd NIST PQC Standardization Conference*.

Security Weakness of the Product-Form Polynomial

$f(x) = f_1(x) \star f_2(x) + f_3(x) \bmod (x^p - x - 1)$



- The distribution of $f_1(x) * f_2(x)$ is not uniform
- Could use a more complicated fashion to have the uniform distribution, but it will increase the time cost

Outline

Experiment Setup

- Tools: Atmel Studio v7.0
- Simulator: ATmega1284 simulator in Atmel Studio v7.0
- Compiler: 8-bit AVR GNU toolchain avr-gcc version 5.4.0
- Optimization Level: -O2 option
- Source code:
  - Assembler (performance/security-critical operations + SHA-512 compression)
  - Others are written in C language

Performance Evaluation

Table: Execution time (in clock cycles) and code size (in bytes) of the main components of two Streamlined NTRU Prime implementations: Karatsuba multiplication based (KA) version and product-form (PF) based version

Operation	KA version		PF version	
	Time	Code	Time	Code
Karatsuba Mul. (in $\mathcal{R}/q$)	5,691,117	2,230	5,691,117	2,230
Product-Form Mul.	n/a	n/a	740,980	2,812
Karatsuba Mul. (in $\mathcal{R}/3$)	1,277,675	1,510	1,277,675	1,510
Encapsulation	8,276,001	8,694	8,276,001	8,694
Decapsulation	15,838,978	11,478	10,869,879	14,370
Encapsulation + Decapsulation	24,114,979	11,634	19,145,880	14,530

Comparision

Table: Execution time (in clock cycles) of our NTRU Prime software, compared with other post-quantum key encapsulation schemes, RSA and ECC. All cryptosystems (except RSA) provide 128-bit security.

Implementation	Algorithm	Platform	Encap.	Decap.
This work	NTRU Prime	ATmega1284	8,276,001	15,838,978
This work (PF)	NTRU Prime	ATmega1284	8,276,001	10,869,879
Kannwischer et al[9]	NTRU Prime	Cortex M4	54,942,173	166,481,625
Kannwischer et al[9]	Frodo	Cortex M4	45,883,334	45,366,065
Kannwischer et al[9]	NewHope	Cortex M4	1,903,231	1,927,505
Kannwischer et al[9]	NTRU	Cortex M4	645,329	542,439
Gura et al[10] *	RSA-1024	ATmega128	3,440,000	87,920,000
Düll et al[11]	ECC-255	ATmega2560	27,800,794	23,900,397
Cheng et al [8]	NTRU	ATmega1281	847,973	1,051,871

[9] Matthias J. Kannwischer et al. *pqm4: Testing and Benchmarking NIST PQC on ARM Cortex-M4*. Cryptology ePrint Archive, Report 2019/844. Available for download at `http://eprint.iacr.org`. 2019.

[10] Gura et al., "Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs".

[11] Michael Düll et al. "High-Speed Curve25519 on 8-bit, 16-bit and 32-bit Microcontrollers". In: *Designs, Codes and Cryptography* 77.2–3 (Dec. 2015), pp. 493–514.

Outline

Conclusion

- The first optimized microcontroller implementation of NTRU Prime (Timing Attacks resistant)
- Optimization of multiplication that combines four levels of Karatsuba multiplication with the hybrid method at the lowest level
- Can not trust C compilers to generate constant-time code for the modulo-3 reduction, which generally raises security concerns
- Adapt the concept of product-form polynomials to NTRU Prime, and show its performance and security weakness
- NTRU Prime can be well optimized to run efficiently on small microcontrollers

Thanks for your attention!

Questions?