A Lightweight Implementation of NTRUEncrypt for 8-bit AVR Microcontrollers

Johann Großschädl

(joint work with Hao Cheng, Peter Roenne, and Peter Ryan)
PQC for the IoT

• AVRNTRU: NTRUEncrypt for 8-bit AVR
  – Compliant with EESS #1 version 3.1 (Sept. 2015)
  – Supports product-form parameter sets with SHA256, e.g. ees443ep1 (128b) and ees743ep1 (256b)
  – Scalable: change parameter set w/o re-compilation
  – Resistance against timing attacks

• Polynomial Arithmetic
  – Optimized for products of sparse ternary polynomials
  – “Hybrid” multiplication of Gura et al. (CHES 2004)

• Auxiliary Functions (SHA256)
IoT Connections Outlook

Source: Ericsson Mobility Report (Nov. 2017)

Billions of devices

- Fixed phones
- Mobile phones
- PCs/laptops/tablets
- Short-range IoT devices
- Wide-area IoT devices

Resource Constraints

- RFC7228: Three Categories of IoT Devices

<table>
<thead>
<tr>
<th>Category</th>
<th>RAM</th>
<th>Flash/ROM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 0 (C0)</td>
<td>≲ 10 kB</td>
<td>≲ 100 kB</td>
</tr>
<tr>
<td>Class 1 (C1)</td>
<td>~ 10 kB</td>
<td>~ 100 kB</td>
</tr>
<tr>
<td>Class 2 (C2)</td>
<td>~ 50 kB</td>
<td>~ 250 kB</td>
</tr>
</tbody>
</table>

- Examples of C1 Devices

**Memsic Iris**  
8-bit AVR ATmega1281  
8 kB RAM, 128 kB Flash

**Zolertia Z1**  
16-bit MSP430F2716  
10 kB RAM, 96 kB Flash

**Rexense Def10**  
32-bit STM32W108 (M3)  
12 kB RAM, 128 kB Flash
Ring Multiplication ("Convolution")

- **Main Arithmetic Operation: Ring Multiplication**
  - Truncated polynomial ring \( R = \mathbb{Z}_q[x]/(x^N-1) \)
  - Typical instantiation (128b): \( N = 443, \ q = 2^{11} = 2048 \)

- **Polynomial Multiplication with Reduction**
  - Operands of degree \( N-1 \), product has degree \( 2N-2 \)
  - Reduction modulo \( x^N-1 \) to get result of degree \( N-1 \)
  - Reduction of coefficients modulo \( q \)

- **Implementation Options**
  - Operand scanning, product scanning: \( O(N^2) \)
  - Karatsuba, Toom-Cook, etc: \( O(N \log N) \)
Optimization

• Convolution \( z(x) = u(x) \cdot v(x) \) in NTRU
  – \( v(x) \) is ternary polynomial, i.e. \( v_j = -1, 0, \) or \( 1 \)
  – Convolution: addition and subtraction of coefficients
  – Execution time depends on the number of non-0 coefficients of \( v(x) \)

• Product-Form Polynomials
  – \( v(x) = v_1(x) \cdot v_2(x) + v_3(x) \)
  – \( v_1(x), v_2(x), v_3(x) \) can be sparse (i.e. have few non-0 coefficients) since coefficients cross-multiply
  – Extremely efficient: \( O(N \log N) \)
  – No security implications (at least in theory!)
Problem: Timing Attacks

“The use of product-form parameter sets was originally intended to provide improved performance by allowing a specialized multiplication algorithm that used knowledge of the indices of the non-zero coefficients […]. However, this index-based multiplication proves to be very hard to implement in a constant-time fashion without losing the speed benefits, so in this paper we concentrate on other approaches of multiplication.”

Towards Timing-Attack Resistance

• Sources of Timing Leakage
  – Calculation of indices (i.e. pointer arithmetic) for accessing the coefficients $u_i$ of polynomial $u(x)$
  – Data-dependent RAM accesses (cache hits/misses)

• Constant-Time Multiplication
  – Microcontrollers used in C1 devices have no cache!
  – Implementation of index calculation without any conditional statements (e.g. if-then-else)

• Fast Constant-Time Multiplication
  – Hybrid method processing 8 coefficients at a time
Sparse Ternary Multiplication (CT)

```c
#define INTMASK(x) (~(x) - 1)

void mul_tern_sparse(uint16_t *r, const uint16_t *u, const uint16_t *v, int vlen, int N)
{
    int index[vlen], i, j, k;

    for (i = 0; i < vlen; i++) index[i] = INTMASK(v[i] != 0) & (N - v[i]);

    for (i = 0; i < N; i += 8) {
        for (j = 0; j < vlen/2; j++) {
            k = index[j];
            r[i   ] += u[k   ]; r[i+1] += u[k+1]; r[i+2] += u[k+2]; r[i+3] += u[k+3];
            r[i+4] += u[k+4]; r[i+5] += u[k+5]; r[i+6] += u[k+6]; r[i+7] += u[k+7];
            index[j] = k + 8 - (INTMASK(k + 8 >= N) & N);
        }
        for (j = vlen/2; j < vlen; j++) {
            k = index[j];
            r[i   ] -= u[k   ]; r[i+1] -= u[k+1]; r[i+2] -= u[k+2]; r[i+3] -= u[k+3];
            r[i+4] -= u[k+4]; r[i+5] -= u[k+5]; r[i+6] -= u[k+6]; r[i+7] -= u[k+7];
            index[j] = k + 8 - (INTMASK(k + 8 >= N) & N);
        }
    }
}
Auxiliary Functions of NTRU

• Index Generation Function (IGF)
  – Generates indices for a sparse ternary polynomial
  – Calls internally a hash function (SHA-2)

• Blinding Poly Generation Method (BPGM)
  – Generates $r(x)$ from a seed using IGF

• Mask Generation Function (MGF)
  – Generates a non-sparse ternary polynomial (mask)
  – Mask is added to message $m(x)$

• Efficient Implementation of SHA256
  – Compression function in ASM: 24k clock cycles
## Timings on 8-bit ATmega1281

<table>
<thead>
<tr>
<th>Operation</th>
<th>ees443 (128b)</th>
<th>ees743 (256b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poly-Arith</td>
<td>192,577</td>
<td>509,227</td>
</tr>
<tr>
<td>BPGM</td>
<td>308,801</td>
<td>492,940</td>
</tr>
<tr>
<td>MGF</td>
<td>189,525</td>
<td>293,138</td>
</tr>
<tr>
<td>Encryption</td>
<td>816,527</td>
<td>1,506,132</td>
</tr>
<tr>
<td>Decryption</td>
<td>1,022,093</td>
<td>2,037,124</td>
</tr>
</tbody>
</table>

- Ring mul only 23.6% – 33.8% of total encryption time
- Auxiliary functions (SHA-256) dominate execution time
- Code size: 8.9 kB (including two parameter sets)
- RAM footprint: 2.9 kB (128b encr) – 6.4 kB (256b decr)
## Comparison

<table>
<thead>
<tr>
<th>Reference</th>
<th>Algorithm</th>
<th>Sec.</th>
<th>Platform</th>
<th>Encryption</th>
<th>Decryption</th>
</tr>
</thead>
<tbody>
<tr>
<td>This work</td>
<td>NTRUEnc</td>
<td>128b</td>
<td>ATmega1281</td>
<td>816,527</td>
<td>1,022,093</td>
</tr>
<tr>
<td>This work</td>
<td>NTRUEnc</td>
<td>256b</td>
<td>ATmega1281</td>
<td>1,506,132</td>
<td>2,037,124</td>
</tr>
<tr>
<td>Boorghany [9]</td>
<td>NTRUEnc</td>
<td>128b</td>
<td>ATmega64</td>
<td>1,390,713</td>
<td>2,008,678</td>
</tr>
<tr>
<td>Boorghany [9]</td>
<td>NTRUEnc</td>
<td>128b</td>
<td>ARM7TDMI</td>
<td>693,720</td>
<td>998,760</td>
</tr>
<tr>
<td>Guillen [20]</td>
<td>NTRUEnc</td>
<td>128b</td>
<td>ARM CortexM0</td>
<td>588,044</td>
<td>950,371</td>
</tr>
<tr>
<td>Guillen [20]</td>
<td>NTRUEnc</td>
<td>192b</td>
<td>ARM CortexM0</td>
<td>1,040,538</td>
<td>1,634,821</td>
</tr>
<tr>
<td>Guillen [20]</td>
<td>NTRUEnc</td>
<td>256b</td>
<td>ARM CortexM0</td>
<td>1,411,557</td>
<td>2,377,054</td>
</tr>
<tr>
<td>Gura [21]</td>
<td>RSA-1024</td>
<td>80b</td>
<td>ATmega128</td>
<td>3,440,00</td>
<td>87,920,000</td>
</tr>
<tr>
<td>Düll [17]</td>
<td>ECC-255</td>
<td>254b</td>
<td>ATmega2560</td>
<td>13,900,397</td>
<td>13,900,397</td>
</tr>
<tr>
<td>Liu [37]</td>
<td>RingLWE</td>
<td>106b</td>
<td>ATxmega128</td>
<td>796,872</td>
<td>215,031</td>
</tr>
</tbody>
</table>
Some Final Words

- **Concluding Remarks**
  - Product-form parameters are useful in practice!
  - NTRUEncrypt is suitable for Class-1 IoT devices
  - Main problem: high RAM consumption

- **Ongoing Work: Masking for DPA Protection**
  - Polynomial arithmetic is easy to mask
  - SHA256 is extremely costly to mask

- **Future Work**
  - NTRUEncrypt for MSP430 and ARM Cortex-M3
  - NTRUPrime and Three Bears
The End

Thanks for your Attention!

Questions?