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

Hao Cheng, Daniel Dinu, Johann Grossschädl, Peter B. Rønne, Peter Ryan

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], and Peter Y. A. Ryan[1]

[1] SnT and CSC, University of Luxembourg
6, Avenue de la Fonte, L-4364 Esch-sur-Alzette, Luxembourg
{hao.cheng,johann.groszschaedl,peter.roenne,peter.ryan}@uni.lu
[2] IPAS, Intel, Chandler, AZ 85226, USA
daniel.dinu@intel.com

Abstract. The dawning era of quantum computing has initiated various initiatives for the standardization of post-quantum cryptosystems with the goal of (eventually) replacing RSA and ECC. NTRU Prime is a variant of the classical NTRU cryptosystem that comes with a couple of tweaks to minimize the attack surface; most notably, it avoids rings with "worrisome" structure. This paper presents, to our knowledge, the first assembler-optimized implementation of Streamlined NTRU Prime for an 8-bit AVR microcontroller and shows that high-security lattice-based cryptography is feasible for small IoT devices. An encapsulation operation using parameters for 128-bit post-quantum security requires 8.3 million clock cycles when executed on an 8-bit ATmega1284 microcontroller. The decapsulation is approximately twice as costly and has an execution time of 15.8 million cycles. We achieved this performance through (i) new low-level software optimization techniques to accelerate Karatsuba-based polynomial multiplication on the 8-bit AVR platform and (ii) a fast implementation of the SHA-512 hash function written in assembly language. The execution time of both the encapsulation and decapsulation is independent of secret data, which makes our software resistant against timing attacks. Finally, we assess the performance one could potentially gain by using a so-called product-form polynomial as secret key and discuss the consequential security implications.

Keywords: Lightweight cryptography · Post-quantum cryptography · Key encapsulation mechanism · NTRU Prime · Efficient implementation

1 Introduction

The advent of quantum computing is a technological revolution that will soon have a massive impact on our daily life and may even disrupt whole industries [20]. In short, a quantum computer operates on so-called qubits (the "quantum analog" of bits), which can not only take the two states 0 and 1, but also be in a superposition of both states. A quantum computer with n qubits can be in an arbitrary superposition of up to 2^n states simultaneously, enabling it to process

2^n values in parallel or to store 2^n values in one step. For example, a quantum computer with about 50 logical qubits could solve certain complex optimization problems a lot faster than the most advanced classical supercomputer today. In the not-so-distant future, our daily life will start to get affected by large-scale quantum computers that are powerful enough to aid the discovery of new drugs or materials, organize the routes of millions of self-driving cars in metropolitan areas without introducing traffic jams, and improve the efficiency of national power grids [20]. Unfortunately, quantum computing has also a destructive side because a large-scale quantum computer with a few thousand qubits would be able to break essentially every public-key cryptosystem in use today. This was discovered in the mid-90s by Peter Shor, who also developed a polynomial-time quantum algorithm to factor large integers, which could break the widely-used RSA cryptosystem [26]. Later, it was also found that a generalization of Shor's algorithm would enable one to take discrete logarithms in a large elliptic curve groups, thereby breaking Elliptic Curve Cryptography (ECC).

Estimates as to when the first large-scale quantum computer will become available vary significantly, but optimistic predictions suggest it could happen by the end of the 2020s [22]. Given the real-world threat posed by quantum computing, it is not surprising that research in the area of Post-Quantum Cryptography (PQC) [3], i.e. cryptography that is able to withstand cryptanalytic attacks carried out using a large-scale quantum computer, has gained momentum in recent years. In December 2016, the NIST officially announced a process to solicit, evaluate, and standardize one or more quantum-resistant public-key cryptographic algorithms and also published a call for proposals. This call, whose submission deadline was 30 November 2017, covered all public-key cryptosystems that are currently standardized by the NIST: public-key encryption, key agreement, and digital signature schemes. A total of 72 candidates were submitted, of which 69 satisfied the minimum acceptability requirements of a "complete and proper" submission and entered the first round of evaluation, which took roughly one year. In January 2019, the NIST announced 26 second-round candidates, of which 17 are public-key encryption and key-establishment algorithms, while the remaining nine are digital signature schemes. The 17 encryption/key-agreement algorithms include nine that are based on hard problems in lattices, seven whose security rests on classical problems in coding theory, and one that claims security from the (supersingular) isogeny walk problem on elliptic curves.

NTRU Prime [4] is a family of lattice-based cryptosystems developed by Bernstein, Chuengsatiansup, Lange, and van Vredendaal, who took inspiration from the 20-year old classical NTRU cryptosystem [14]. There exist two variants of NTRU Prime, one called *Streamlined NTRU Prime*, which can be seen as a variant of the classical NTRU, while the other, *NTRU LPRime*, shares a similar structure with the NewHope key exchange mechanism [1], whose security is based on the Ring Learning With Errors (RLWE) problem [21]. Roughly speaking, NTRU Prime is an attempt to improve the security of classical NTRU and other lattice-based cryptosystems by avoiding rings with "worrisome" structure and using extension fields of the form $(\mathbb{Z}/q)[x]/(x^p - x - 1)$ instead, where p is a

prime. Multiplication in such fields can be implemented using several layers of Karatsuba's technique [18], which makes NTRU Prime relatively fast on 64-bit Intel processors with vector instructions. Concretely, the designers of NTRU Prime describe a highly-optimized implementation of the field multiplication using AVX2 vector instructions capable to execute 16 separate multiplications of integers modulo 2^{16} in a SIMD-parallel fashion. NTRU Prime belongs to the 26 candidates that made it into the second round of NIST's evaluation process. This second round will focus heavily on evaluating the candidates' performance across a wide variety of systems, which includes not only big computers and smartphones, but also devices that have limited processor power [23].

Research on software optimization techniques that lead to efficient implementations of Streamlined NTRU Prime and NTRU LPRime has, until now, been limited to 64-bit Intel processors with AVX vector extensions. When using a parameter set for 128-bit post-quantum security, the AVX implementation reported in [4] requires 59600 clock cycles for encryption (resp. "encapsulation" of a 256-bit session key) on an Intel Haswell processor, while the decryption ("decapsulation") is slightly slower and takes 97452 cycles. The only performance figures of NTRU Prime for smaller platforms (e.g. 8, 16, or 32-bit microcontrollers) that we are aware of, were reported in a recent paper on `pqm4` [17], a testing and benchmarking framework for NIST PQC candidates on ARM Cortex-M4 devices. Due to the lack of an optimized ARM implementation, the authors of [17] resorted to the reference C code provided by the designers of NTRU Prime, which requires 55 million clock cycles for encapsulation and about 166.5 million cycles for decapsulation, whereby both cycle counts were determined with Streamlined NTRU Prime parameterized for 128 bits of security. However, these results do not allow one to reason about the actual performance of NTRU Prime since the purpose of a reference implementation is to promote the understanding of an algorithm rather than achieving high speed. Consequently, there is currently very little known about how to optimize NTRU Prime for small microcontrollers and what performance an optimized assembler implementation could achieve.

In this paper we introduce a carefully optimized implementation of NTRU Prime for 8-bit AVR microcontrollers that we developed from scratch to achieve high speed and resistance against timing attacks. We chose 8-bit AVR as evaluation platform for two reasons. First, the 8-bit AVR architecture is widely used in devices where security plays a crucial role, e.g. smart cards, wireless sensor nodes, and various kinds of other IoT gadgets. Second, 8-bit AVR microcontrollers are among the most resource-constrained computing platforms that exist today, which means that if NTRU Prime can be implemented to run with acceptable speed on an AVR device, it can also be implemented to run satisfactorily on more powerful 16 and 32-bit microcontrollers (e.g. ARM Cortex-M), whereas the opposite is not necessarily true. We discovered that the overall performance of NTRU Prime on AVR depends heavily on the polynomial arithmetic. In order to reduce the execution time of the arithmetic component of NTRU Prime, we developed optimizations for two approaches of polynomial multiplication: the first is based on a combination of Karatsuba's algorithm and the school-

book method, while the second uses so-called product-form polynomials, which were originally introduced to speed up the classical NTRU cryptosystem [15]. Furthermore, we describe optimized Assembler implementations of the reduction modulo q and the reduction modulo 3, and demonstrate that optimizing C compilers can generate code with operand-dependent execution time for the modulo-3 reduction, thereby enabling timing attacks. We verified the correctness of our implementation using the reference implementation of NTRU Prime and made extensive tests to ensure that the execution time of all security-critical functions is independent of the operands being processed.

2 A Brief Overview of NTRU Prime

NTRU Prime is described by its inventors as a high-security *prime-degree large-Galois-group inert-modulus* ideal-lattice-based key-establishment algorithm [4]. Together with NTRU (a merger of NTRUEncrypt and NTRU-HRSS-KEM), NTRU Prime has survived the first round of NIST's ongoing PQC standardization project and is one of the 26 candidates in the second round. As stated in the previous section, the NTRU Prime specification describes two public-key cryptosystems, both providing standard IND-CCA2 security. In this paper, our work only focuses on Streamlined NTRU Prime, which has a strong link with classic NTRU cryptosystem. The brief description of the Streamlined NTRU Prime algorithm is shown as follows.

Notation and Parameters

A parameter set for NTRU Prime includes the triple (p, q, w), which defines the underlying algebraic structures [5]. The prime number p is the number of coefficients for a polynomial, which is offered as 653, 761, and 857 in the three official parameter sets (i.e. `sntrup653`, `sntru761`, and `sntru857`). Modulus q is specified to be a prime number, which is 4621, 4591, and 5167 respectively for the three p-values mentioned above. The *weight* w means that there are precisely w nonzero coefficients in a polynomial. Furthermore, let $(\mathbb{Z}/m)$ represent the set of integers in $(-m/2, m/2]$. We abbreviate the ring $\mathbb{Z}[x]/(x^p - x - 1)$, the ring $(\mathbb{Z}/3)[x]/(x^p - x - 1)$, the field $(\mathbb{Z}/q)[x]/(x^p - x - 1)$ as $\mathcal{R}$, $\mathcal{R}/3$, and $\mathcal{R}/q$, respectively. The notation *small* means an element of $\mathcal{R}$ if all of its coefficients are in $\{-1, 0, 1\}$. Define Short as the set of *small* weight-w elements of $\mathcal{R}$. At last, the parameters of Streamline NTRU Prime have restrictions: $2p \geqslant 3w$; $q \geqslant 16w + 1$; and $x^p - x - 1$ is irreducible in the polynomial ring $(\mathbb{Z}/q)[x]$ (see [5] for all details).

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 $f(x) \in \mathsf{Short}$.
4. Compute $h(x) = g(x)/(3f(x))$ in $\mathcal{R}/q$.
5. Output *public key* $h(x)$ and *private key* $(f(x), v(x))$.

Encapsulation

1. Generate a uniform random polynomial $r(x) \in \mathsf{Short}$.
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 $e(x) = (3f(x) \star c(x) \in \mathcal{R}/q) \bmod 3$.
2. Compute $r'(x) = e(x) \star v(x) \in \mathcal{R}/3$.
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 $\in \mathsf{Short}$.
5. Hash (`SHA-512`-based) $r'(x)$ together with $c(x)$ to obtain *session key* $k(x)$.

3 Polynomial Arithmetic

Since Streamlined NTRU Prime is closely related to the classical NTRU scheme (i.e. NTRUEncrypt), it is not surprising that they share a number of implementation aspects, in particular they have in common that the overall performance depends heavily on the polynomial arithmetic. However, the underlying algebraic structures are different: NTRUEncrypt is based on the residue class ring $\mathcal{R} = (\mathbb{Z}/q)[x]/(x^N - 1)$ where q is a power of two, while NTRU Prime uses the field $(\mathbb{Z}/q)[x]/(x^p - x - 1)$ where q is a prime (e.g. $q = 4591, 4621, 5167$). The reduction modulo q is essentially free in the former case, but relatively costly for NTRU Prime, especially when one aims for constant execution time so as to thwart timing attacks. Furthermore, the irreducible polynomial of NTRU Prime contains an additional non-0 coefficient, which makes the convolution more costly. In addition, most performance-optimized implementations of the classical NTRUEncrypt cryptosystem use parameter sets with so-called product-form polynomials to minimize the computational cost of the polynomial multiplication. Product-form parameter sets were not included in the NTRU Prime specification, but could be added as described in the following subsection.

3.1 Product-Form Polynomials

In our implementation of NTRU Prime we provide the option of using a product-form representation for the private key polynomial $f(x) \in \mathsf{Short}$ to speed up the decryption, an approach originally presented in [16] and analysed in detail in [13] for the classical NTRU. The idea is simply to replace $f(x)$ with three *sparse*

ternary polynomials of the form $f = f_1 \star f_2 + f_3$. Indeed, when f_1, f_2, and f_3 are sparse ternary polynomials, then the polynomial multiplication of $c(x)$ by $f(x)$ in the decryption phase can be optimized to reach very high speed, even on small microcontrollers. Furthermore, for target platforms that do not have a data cache (which is the case for virtually all 8 and 16-bit microcontrollers, and also for many 32-bit models, e.g. most members of the ARM Cortex-M series), it is possible to implement the polynomial multiplication to have constant execution time. That is, the execution time only depends on the number of non-zero coefficients of the sparse polynomials, which is fixed, but not on the coefficients of $f(x)$ themselves (i.e. the execution time does not depend on *which* coefficients are $+1$ and which -1).

Our software implementation represents the ciphertext-polynomial $c(x)$ as an array of words of type `uint16_t`, similar to [2], whereas the ternary polynomials f_1, f_2, and f_3 are not stored in the form p-element arrays, but can simply be represented as arrays that contain the *indices* of the non-0 coefficients. This representation of the sparse polynomials f_i has two advantages: firstly, it is easy to load the corresponding coefficients of $c(x)$ by adding the offset to the start address of the array in which $c(x)$ is stored. Secondly, the polynomial f_1 does not consume much space in RAM since we only need to consider the non-zero coefficients. When using a straightforward multiplication technique for polynomials then the sparse nature of $f_1(x)$ causes most of partial products to be zero. So rather than employing a traditional polynomial multiplication algorithm that wastes a lot of time by computing zero terms, we scan the array $F1$ containing the offsets of the non-zero coefficients of f_1 and calculate only those partial-product terms which are be non-zero.

Note that for NTRU Prime, Bernstein et al [4] chose not to support product-form polynomials due to the threat of timing-attacks. However, for *cache-less* microcontrollers like 8-bit AVR, 16-bit MSP, and 32-bit ARM Cortex-M, the multiplication of sparse polynomials in product form can be easily implemented to have constant execution time as was recently demonstrated in [7]. Besides, in the official Google group for the ongoing NIST PQC Standardization, a few researchers have already suggested adding a *product-form* option for the parameter sets in NTRU Prime. One of the main contributions of this paper is to derive product-form parameters for NTRU Prime and to demonstrate that they enable substantial savings in execution time without compromising security against timing attacks on cache-less microcontrollers.

In classical NTRU, a *product-form* parameter set extends a conventional parameter set by three weight parameters that specify the number of non-0 coefficients in each of the three sub-polynomials a product-form polynomial is composed of. Since the designers of NTRU Prime decided to not provide a product-form option for their parameter sets, we need to determine the parameters for the non-0 coefficients (w_1, w_2, w_3) for a product-form ternary polynomial ourselves. In NTRU Prime, the *weight* w is one of the three primary parameters (p, q, w), and Short represents a ternary polynomial with exactly w non-0 coefficients. The following parameter generation method is inspired by [24] for the product-form

ternary polynomial $\mathcal{R}/3$ with weight w (i.e. Short). Similar as in NTRUEncrypt, we split Short into three *sparse* polynomials with small fixed weights $2w_1$, $2w_2$ and $2w_3$, specifically f_i is ternary with w_i coefficients being $+1$ and -1 respectively. We take the `sntrup653` parameters as example to show how to calculate (w_1, w_2, w_3) from $w = 288$, which means the number of non-0 coefficients of $f = f_1 \star f_2 + f_3$ has to amount to roughly 288. The following calculation shows the overall number of non-0 coefficients of f is expected to be $\approx 6w_1w_2 + 2w_3$. Since the irreducible polynomial of NTRU Prime is $P = x^p - x - 1$, a reduction of a product-polynomial modulo P introduces more non-0 coefficients in the convolution than a reduction modulo $x^p - 1$, the irreducible polynomial of NTRU. For example, a term of the form $a_p x^p$ will be reduced to $a_p + a_p x$ in NTRU Prime, but to just a_p in classical NTRU. Since half of the terms of a product-polynomial $f_1(x) \star f_2(x)$ have an exponent above $p - 1$, the expected number of non-0 coefficients of this convolution is $1.5 \cdot 2w_1 \cdot 2w_2 = 6w_1w_2$. Let $\alpha \in \mathbb{R}$ be the unique positive root of

$$3x^2 + x - w/2 = 0 \tag{1}$$

and let w_1 be $\lceil \alpha \rceil$. In this way, we obtain $w_1 = 7$. The selection of w_2 and w_3 is relatively arbitrary. For example, if we first approximate $w_3 = w_1$, we get

$$3w_1w_2 + w_1 = w/2. \tag{2}$$

Then, we can obtain

$$w_2 = \lceil \frac{w}{6w_1} - \frac{1}{3} \rceil = 7 \tag{3}$$

and finally write w_3 as

$$w_3 = \texttt{max}(\lceil w/2 - 3w_1w_2 \rceil, \lceil w_1/2 + 1/2 \rceil) = 4. \tag{4}$$

Consequently, the product-form parameters for `sntrup653` are $(w_1, w_2, w_3) = (7, 7, 4)$. In the same way, we obtain the product-form parameters for `sntrup761` as $(7, 7, 4)$ and for `sntrup857` as $(8, 7, 5)$.

Security. We note that there there is no security analysis for using product-form polynomials in NTRU Prime. Indeed, the product form $f = f_1 \star f_2 + f_3$ will have a linear distribution of non-zero values instead of a uniform distribution as in the classical case, if the sparse polynomial are uniformly distributed. We leave the security analysis as an important piece of future work and additionally provide a less efficient, but safer implementation without product-form polynomials.

3.2 Product-Form Polynomial Multiplication

The NTRUEncrypt implementation for 8-bit AVR microcontrollers described in [7] contains a ring multiplication function where one operand is an element of $\mathcal{R}/q$ (i.e. a polynomial of degree N with coefficients in the range $[0, q-1]$) and

the other operand is a ternary polynomial (i.e. an element of $\mathcal{R}/3$) in product-form. We extended this method to suit the requirements of Streamlined NTRU Prime, which uses the field $\mathbb{Z}[x]/(x^p - x - 1)$ as underlying algebraic structure. We refer to [7] for a detailed description of the original multiplication technique. In the following, we give a brief overview of how this multiplication technique can be adapted for NTRU Prime.

Similar to classical NTRUEncrypt, also NTRU Prime requires to multiply a polynomial that is an element of $\mathcal{R}/q$ and a ternary polynomial with a specified weight (i.e., Short, see Section 2 and [5]). Concretely, this multiplication is performed in Step 1 of the decryption operation. As discussed in the previous subsection, we can represent $f(x) \in \mathsf{Short}$ as a product of the form $f_1 \star f_2 + f_3$. In order to use the sparse polynomial multiplication described in [7] for the convolution in Streamline NTRU Prime, two modifications are necessary; one affects the reduction modulo q and the other the reduction modulo the irreducible polynomial.

Modification 1: Reduction Modulo Prime q. The so-called large modulus q in classical NTRUEncrypt is always a power of 2; the most recent parameter sets use $q = 2^{11} = 2048$. This choice of q is motivated by arithmetic efficiency since any integer can be reduced modulo 2048 by simply extracting the 11 least significant bits of the integer using e.g. a logical AND operation with an appropriate bit-mask. However, in NTRU Prime, the modulus q is a prime, and each parameter set has its own q; typical values are $q = 4621$, $q = 4591$, and $q = 5167$ [5]. When using product-form polynomials as described in the previous subsection, the maximum weight of any of the sparse ternary polynomials of the three parameter sets we consider is $2w_1 = 16$ of `sntrup857`, which means there are at most eight $+1$ coefficients and eight -1 coefficients. Consequently, when performing the product-form convolution as described in [7], at most eight coefficients in the range of $[0, q-1]$ have to be added or subtracted to form an intermediate result, which is then, at the very end, reduced modulo q to obtain a coefficient of the product-polynomial. However, since a product-form multiplication $c(x) \star f(x) = c(x) \star [f_1(x) \star f_2(x) + f_3(x)]$ involves two sparse ternary polynomial multiplications (one by by f_1 and the other by f_2) and one addition by f_3, the maximum value that an intermediate result can have is $17 \cdot 5167 = 87839$, which fits into 17 bits (5167 is the q parameter of `sntrup857`). Consequently, it makes sense to develop an optimized algorithm for reducing a 17-bit unsigned integer modulo a 13-bit unsigned integer since all three q values of the NTRU Prime parameter sets we consider have a length of 13 bits.

Algorithm 1 shows a generic algorithm for reducing an arbitrary 17-bit unsigned integer a modulo an arbitrary 13-bit modulus q. It first extracts the five most-significant bits of a (referred to as m_5) and then computes the remainder $r = 2^{12} m_5 \bmod q$ using a pre-computed look-up table that contains 32 entries. Then, r is added to the 12 least-significant bits of a, which are obtained through a logical AND operation with the bit-mask `0xfff`. The intermediate result b in line 3 is always less than $2q$ since the 12 least-significant bits of a are always

Algorithm 1 Lookup-table-based constant-time modular reduction

Input: integer a of a length of (up to) 17 bits, modulus q of a fixed length of 13 bits
Output: $b \equiv a \bmod q$

1: $m_5 \leftarrow a \gg 12$ ▷ get the 5 most significant bits of a
2: $r \leftarrow \texttt{LUT}[m_5]$ ▷ fast reduction of $m_5 2^{12}$ modulo q via look-up table
3: $b \leftarrow r + a\&\texttt{0xfff}$ ▷ keep the 12 least-significant bits of a
4: $b \leftarrow b - q \cdot (b \geqslant q)$
5: **return** b

smaller than $2^{12} = 4096$ (and, therefore, also smaller than q) and r is in the range $[0, q-1]$. Consequently, a single subtraction of q suffices to get a fully-reduced result. In order to perform subtraction in constant time, we first compare the intermediate result b with q in such a way that we get 1 as result when $b \geq q$ and 0 otherwise. This comparison result is multiplied by q, yielding a product that is either q or 0, which is then subtracted from b. Algorithm 1 works for any of the q values of the NTRU Prime parameter sets, though each of them requires its own look-up table. These three look-up tables can be pre-computed and have a size of 192 bytes altogether.

Modification 2: Reduction Modulo Irreducible Polynomial. As mentioned before, the irreducible polynomial of NTRU Prime is $x^p - x - 1$, where p is a prime, whereas classical NTRU uses $X^N - 1$. Therefore, the reduction modulo the irreducible polynomial is not as cheap as in NTRU, where it is just substitution (i.e., every x^{k+N} with $0 \leq k < N-1$ is replace by x^k). In NTRU Prime, on the other hand, each term x^{k+p} with $0 \leq k < p-1$ is replaced by two terms, namely the sum $x^k + x^{k+1}$.

3.3 Karatsuba-Based Polynomial Multiplication

Besides the product-form convolution described in the previous subsection, which can be applied in Step 1 of the decryption operation, there are two further convolutions in NTRU Prime, namely (i) the multiplication of a polynomial in $\mathcal{R}/q$ with a ternary polynomial (i.e. an element of Short) carried out in Step 1 of the encryption, and (ii) the multiplication of two ternary polynomials (i.e. a multiplication of two elements of $\mathcal{R}/3$) in Step 3 of the decryption. Even though one of the operands of these two convolutions is a ternary polynomial, the product-form technique can not be applied due to the way NTRU Prime is designed. Consequently, an implementer has to resort to "conventional" multiplication techniques for polynomials, many of which are based on well-known algorithms for multiple-precision multiplication of integers, such as needed for RSA and ECC. From a high-level perspective, polynomial multiplication algorithms can be divided into two main categories, namely basic techniques that require n^2 coefficient multiplications to get the product of two polynomials consisting of n coefficients each, and advanced techniques with sub-quadratic complexity, e.g.

Karatsuba multiplication [18]. Examples of the former category are the so-called *operand-scanning* and *product-scanning* methods, which compute the coefficient-products in a row-wise or column-wise fashion and differ with respect of the number of load and store instructions they need to execute [12]. The *hybrid technique* introduced in [11] combines both operand and product scanning and computes $d > 1$ coefficient-products in each iteration of the inner loop, thereby reducing the number of load and store instructions.

Multiplication techniques with sub-quadratic complexity are known since the early 1960s when Karatsuba published his seminal paper [18]. Karatsuba's approach reduces a multiplication of two operands consisting of n coefficients to three multiplications of $(n/2)$-coefficient polynomials and a couple of additions. The half-size multiplications can be performed with any multiplication technique, including the conventional operand-scanning and product-scanning method. Alternatively, it is possible to apply Karatsuba's idea recursively until the operands consist of just one single word, in which case the asymptotic complexity becomes $\theta(n^{log_2(3)})$. A further variant is the Arbitrary Degree Karatsuba (ADK) algorithm described [25]. Also multiplication algorithms with even better asymptotic complexity have been studied in the literature, e.g. an optimized Toom-Cook 4-way algorithm in [19], which was originally designed for the Saber KEM in a 32-bit ARM Cortex-M4 Microcontrollers, to perform the polynomial multiplication for degree 256.

Finding the optimal multiplication strategy for the two convolutions mentioned at the beginning of this subsection is a non-trivial task. Intuitively, one would assume that a combination of multiplication techniques with sub-quadratic and quadratic complexity will yield the best performance. However, the concrete implementation of such a combined strategy poses a number of questions. Asymptotic complexity bounds are not always meaningful in the real world, especially when the operands are relatively short. Therefore, it is necessary to find out which sub-quadratic multiplication algorithm is the best choice for the convolutions in NTRU Prime, which depends besides the length of the operands also on certain characteristics of the target platform. A second important question is how many iterations of Toom-Cook and/or Karatsuba should be performed and what is the ideal "cut-off" point at which one should switch to a basic multiplication technique with quadratic complexity. Finally, a third questions is which one of the basic multiplication methods should be used. In order to answer these research questions, we conducted an extensive number of experiments with combinations of different sub-quadratic algorithms, different numbers of iterations of the sub-quadratic algorithms (i.e. different cut-off points) and different basic multiplication techniques.

The results of these experiments show that for both convolutions, four levels of Karatsuba multiplication provide the best performance across all parameter sets we consider in this paper. However, the "low-level" optimization strategy differs. The convolution in Line 1 of the encryption ($\mathcal{R}/q$ with Short) reaches peak performance when the hybrid method with $d = 2$ is used at the lower level. The modulo-q reduction of the coefficient-products can use Algorithm 1

described in the previous subsection. On the other hand, the convolution in Line 3 of the decryption ($\mathcal{R}/3$ with $\mathcal{R}/3$) reaches the best results when the hybrid method with $d = 4$ is used at the lower level, which is possible because more free registers are available. The coefficient-products need to be reduced modulo 3 at the end of the convolution. This modulo reduction is less costly than the modulo q reduction, but deserves special attention with respect to timing attacks, as will be discussed below.

Constant-Time Modulo-3 Reduction. One of the operands of the convolution in Line 3 of the decryption operation is the ternary polynomial $v(x)$, which is a part of the secret key. Hence, one has to pay attention that the polynomial multiplication, including the reduction of the coefficient-products modulo 3, has constant execution time to prevent timing attacks. This modulo-3 reduction can be simply implemented in C through an operation of the form `y = x % 3`, whereby in our case `x` is a 16-bit integer. However, in the course of our experiments we found that one can not take it for granted that a C compiler generates constant-time code for this operation. More concretely, we found out that certain versions of `avr-gcc` generate code with operand-dependent execution time for certain AVR models, which can leak information about the secret key $v(x)$. For example, we measured the execution time of a modulo-3 reduction compiled with `avr-gcc` 4.8.2 for an ATtiny45 device using the cycle accurate simulator Avrora [27, 28]. For target devices that do not have a hardware multiplier, such as ATtiny45, `avr-gcc` uses the `__udivmodhi4` function from the runtime library `libgcc` to perform the modulo-3 reduction. The same function was also used for devices that have a hardware multiplier until version 4.7.0 of `avr-gcc` (released on March 2012), when it was replaced with the `__umulhisi3` function [9, 10]. The former function has an operand-dependent execution time, while the latter function has a constant execution time (54 cycles) for all input values. Concretely, the execution time of the `__udivmodhi4` varies between 193 cycles for input values 0, 1, and 2 to 207 cycles for input values 49149, 49150, and 49151. Thus, the time difference between the longest and shortest execution is 14 cycles. A summary of all possible execution times is given in Table 1 and in Figure 1.

In order to ensure that the resistance against timing attacks does not depend on the compiler, we implemented the modulo-3 reduction in Assembly language following the approach described in [7].

4 Results and Comparison

Target Platform. The target platform for our software is the 8-bit AVR ATmega1284 microprocessor, which is equipped with 16 kB SRAM and 128 kB program memory. Our NTRU Prime software is written in a mix of ANSI C and Assembly language (all functions that are not performance-critical or security-critical are written in C for portability reasons). In particular, we implement the core arithmetic operations in Assembly to achieve fast and operand-independent

Table 1. The execution time in cycles of the `__udivmodhi4` function for all possible 16-bit unsigned integer inputs.

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			

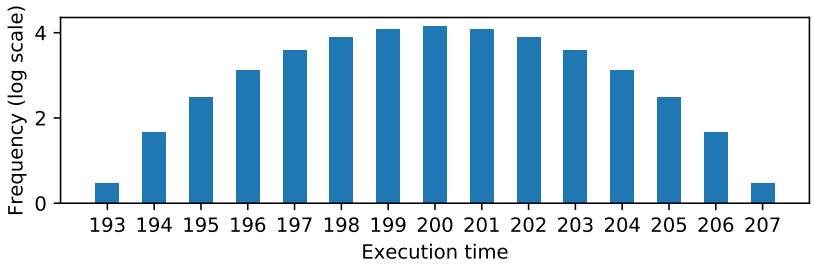

Fig. 1. The execution time in cycles of the `__udivmodhi4` function for all possible 16-bit unsigned integer inputs.

execution times. Our software uses the highly optimized Assembler implementation of the SHA512 hash function described in [6] to reduce the execution time of some auxiliary functions that are performance-critical. When executed on our target device, the compression function takes slightly less than 60k clock cycles, which corresponds to a compression rate of roughly 467 cycles per byte. Our NTRU Prime implementation can be complied with Atmel Studio v7.0 under the `-O2` optimization option, which produces an executable that, according to our timing simulations, does not leak any secret information and can, therefore, withstand timing attacks.

Experimental Results. Table 2 summarizes the execution time and code size of the main arithmetic operations of encryption/decryption and the full encapsulation and decapsulation of our NTRU Prime software. It is interesting to compare the Karatsuba multiplication and the product-form multiplication, which both multiply an element of $\mathcal{R}/q$ by an element of Short. The latter uses a product-form representation of the element of Short, while the former requires the Short-element to be represented in the standard way. Our results show that product-form based multiplication saves 77.5% execution time compared to the optimized Karatsuba multiplication. On the other hand, the Karatsuba multi-

Table 2. 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

plication for two general ternary polynomials requires only 1.28 million cycles, which can be attributed to the fact that the reduction modulo 3 is much faster than the reduction modulo q and that the "low-level" hybrid method with $d = 4$ is relatively efficient. The by far most costly operation of NTRU Prime is obviously the Karastusba multiplication for an element in $\mathcal{R}/q$ and Short, which represents 68.8% of the overall running time of the encapsulation and 52.4% of the running time in decapsulation, respectively. Even though the decapsulation has to perform both a decryption and a full encapsulation, it is only slightly slower than the encapsulation because the two convolutions carried out in the decryption are relatively fast and need only around two million cycles altogether. Other costly components not explicitly listed in Table 2 include some auxiliary functions, e.g. the SHA-512 hash function and the basic encode/decode functions of NTRU Prime. Furthermore, because most operations of the encapsulation are used in the decapsulation, the code size of whole NTRU Prime is just slightly more than that of its decapsulation.

Comparison. Our work is to the best of our knowledge the first highly-optimized software implementation of NTRU Prime for constrained devices. Only one previous software implementation of NTRU Prime for microcontrollers is published in the literature, namely the implementation of `pqm4` [17], which is essentially reference C code without any optimizations. Compared with the `pqm4` results on the ARM Cortex-M4, our implementation is 6.6 times faster for encapsulation and 15.3 times faster for decapsulation. However, it has to be taken into account that a 32-bit ARM Cortex-M4 is much more powerful than an 8-bit AVR microcontroller. The comparison with the reference implementation confirms that, without product-form optimization in the decryption, the decapsulation is significantly slower than the encapsulation. Table 3 also specifies the performance of other NIST PQC candidates on the Cortex-M4. Due to the very limited number of implementations of post-quantum cryptosystems for 8-bit AVR microcontrollers, we further compare NTRU Prime with traditional public-key schemes RSA and ECC. The decapsulation time of NTRU Prime is just 12.4% of the decryption time of RSA. Furthermore, the results show that

Table 3. Execution time (in clock cycles) of our NTRU Prime software and other implementations of post-quantum key encapsulation schemes, as well as RSA and ECC, on microcontrollers. 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 [17]	NTRU Prime	Cortex M4	54,942,173	166,481,625
Kannwischer et al [17]	Frodo	Cortex M4	45,883,334	45,366,065
Kannwischer et al [17]	NewHope	Cortex M4	1,903,231	1,927,505
Kannwischer et al [17]	Kyber	Cortex M4	652,769	621,245
Kannwischer et al [17]	NTRU	Cortex M4	645,329	542,439
Gura et al [11] *	RSA-1024	ATmega128	3,440,000	87,920,000
Düll et al [8]	ECC-255	ATmega2560	27,800,794	23,900,397

* To the best of our knowledge, no RSA implementation providing 128-bit security on an 8-bit processor exists. Thus, we use the 80-bit security level implementation for comparison.

NTRU Prime also outperforms ECC by a factor of between 2.2 (decapsulation) and 3.4 (encapsulation).

5 Conclusions

We presented the first optimized microcontroller implementation of NTRU Prime that is capable to resist timing attacks. When executed on an 8-bit AVR ATmega1284 device, the encapsulation takes roughly 8.3 million cycles, while the decapsulation has an execution time of 15.8 million cycles, where both results are based on a parameter set providing 128 bits of post-quantum security. For comparison, the reference C implementation requires 54.9 and 166.5 million cycles for encapsulation and decapsulation, respectively, on a much more powerful 32-bit ARM Cortex-M4 microcontroller. Our results clearly show that any effort spent on optimizing the polynomial arithmetic is well spent. We developed several software optimization techniques for the three convolutions performed in the encryption and decryption. Furthermore, we discussed how to adapt the concept of product-form polynomials from the classical NTRU cryptosystem to NTRU Prime and demonstrated that product-form multiplication is very efficient on 8-bit AVR microcontrollers. However, given the (potential) security implications, we do not recommend to use product-form polynomials in real-world deployments of NTRU Prime. An alternative is the optimized implementation of non-product-form multiplication that combines four levels of Karatsuba multiplication with the hybrid method with $d = 4$ at the lowest level. In addition, we demonstrated that one can not trust C compilers to generate constant-time code for the modulo-3 reduction, which generally raises concerns about the security (i.e. resistance against timing attacks) of C implementations of NTRU Prime. In summary, our results show that NTRU Prime can be well optimized to run

efficiently on small microcontrollers, which makes NTRU Prime an interesting candidate for securing the post-quantum IoT.

Acknowledgements. This work was supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No. 779391 (FutureTPM). This work was conducted before Daniel Dinu joined Intel and may not reflect the views of his current or previous employers.

References

1. E. Alkim, L. Ducas, T. Pöppelmann, and P. Schwabe. Post-quantum key exchange – A new hope. In T. Holz and S. Savage, editors, *Proceedings of the 25th USENIX Security Symposium (USS 2016)*, pages 327–343. USENIX Association, 2016.
2. D. V. Bailey, D. Coffin, A. J. Elbirt, J. H. Silverman, and A. D. Woodbury. NTRU in constrained devices. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 262–272. Springer Verlag, 2001.
3. D. J. Bernstein, J. Buchmann, and E. Dahmen, editors. *Post-Quantum Cryptography*. Springer Verlag, 2009.
4. D. J. Bernstein, C. Chuengsatiansup, T. Lange, and C. van Vredendaal. NTRU prime. Cryptology ePrint Archive, Report 2016/461, 2016. Available for download at `http://eprint.iacr.org`.
5. D. J. Bernstein, C. Chuengsatiansup, T. Lange, and C. van Vredendaal. NTRU prime: Round 2 specification, 2016. Available for download at `http://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions`.
6. H. Cheng, D. Dinu, and J. Großschädl. Efficient implementation of the SHA-512 hash function for 8-bit AVR microcontrollers. In J.-L. Lanet and C. Toma, editors, *Innovative Security Solutions for Information Technology and Communications — SecITC 2018*, volume 11359 of *Lecture Notes in Computer Science*, pages 273–287. Springer Verlag, 2019.
7. H. Cheng, J. Großschädl, P. B. Rønne, and P. Y. A. Ryan. A lightweight implementation of NTRUEncrypt for 8-bit AVR microcontrollers. In *Proceedings of the 2nd NIST PQC Standardization Conference*, 2019. Available online at `http://csrc.nist.gov/Events/2019/second-pqc-standardization-conference`.
8. M. Düll, B. Haase, G. Hinterwälder, M. Hutter, C. Paar, A. H. Sánchez, and P. Schwabe. High-speed Curve25519 on 8-bit, 16-bit and 32-bit microcontrollers. *Designs, Codes and Cryptography*, 77(2–3):493–514, Dec. 2015.
9. GCC Team. AVR-GCC Wiki. Available online at `http://gcc.gnu.org/wiki/avr-gcc#Exceptions_to_the_Calling_Convention`. Accessed: June 2019.
10. GCC Team. GCC Releases. Available online at `http://gcc.gnu.org/releases.html`. Accessed: June 2019.
11. N. Gura, A. Patel, A. S. Wander, H. Eberle, and S. Chang Shantz. Comparing elliptic curve cryptography and RSA on 8-bit CPUs. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems — CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 119–132. Springer Verlag, 2004.
12. D. R. Hankerson, A. J. Menezes, and S. A. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer Verlag, 2004.

13. J. Hoffstein, J. Pipher, J. M. Schanck, J. H. Silverman, W. Whyte, and Z. Zhang. Choosing parameters for NTRUEncrypt. Cryptology ePrint Archive, Report 2015/708, 2015. Available for download at `http://eprint.iacr.org`.
14. J. Hoffstein, J. Pipher, and J. H. Silverman. NTRU: A ring-based public key cryptosystem. In J. P. Buhler, editor, *Algorithmic Number Theory, Third International Symposium (ANTS-III)*, volume 1423 of *Lecture Notes in Computer Science*, pages 267–288. Springer Verlag, 1998.
15. J. Hoffstein and J. H. Silverman. Optimizations for NTRU. In K. Alster, J. Urbanowicz, and H. C. Williams, editors, *Public-Key Cryptography and Computational Number Theory*, De Gruyter Proceedings in Mathematics, pages 77–88. Walter de Gruyter, 2001.
16. J. Hoffstein and J. H. Silverman. Random small hamming weight products with applications to cryptography. *Discrete Applied Mathematics*, 130(1):37–49, 2003.
17. M. J. Kannwischer, J. Rijneveld, P. Schwabe, and K. Stoffelen. `pqm4`: Testing and benchmarking NIST PQC on ARM Cortex-M4. Cryptology ePrint Archive, Report 2019/844, 2019. Available for download at `http://eprint.iacr.org`.
18. A. A. Karatsuba and Y. P. Ofman. Multiplication of multidigit numbers on automata. *Soviet Physics - Doklady*, 7(7):595–596, Jan. 1963.
19. A. Karmakar, J. M. Bermudo Mera, S. S. Roy, and I. Verbauwhede. Saber on arm. In *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 243–266, 2018(3).
20. P. R. Kaye, R. Laflamme, and M. Mosca. *An Introduction to Quantum Computing*. Oxford University Press, 2007.
21. V. Lyubashevsky, C. Peikert, and O. Regev. On ideal lattices and learning with errors over rings. *Communications of the ACM*, 60(6):43:1–43:35, June 2013.
22. M. Mariantoni. Building a superconducting quantum computer. Invited presentation given at the 6th International Conference on Post-Quantum Cryptography (PQCrypto 2014), Waterloo, ON, Canada, Oct. 2014. Available online at `http://www.youtube.com/watch?v=wWHAs--HA1c`.
23. National Institute of Standards and Technology (NIST). NIST Reveals 26 Algorithms Advancing to the Post-Quantum Crypto 'Semifinals'. Available online at `http://www.nist.gov/news-events/news/2019/01/nist-reveals-26-algorithms-advancing-post-quantum-crypto-semifinals`. Accessed: Oct. 2019.
24. J. M. Schanck. *Practical Lattice Cryptosystems: NTRUEncrypt and NTRUMLS*. M.Sc. Thesis, University of Waterloo, Waterloo, ON, Canada, 2015.
25. M. Scott. Missing a trick: Karatsuba variations. *Cryptography and Communications*, 10(1):5–15, Jan 2018.
26. P. W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS '94)*, pages 124–134. IEEE Computer Society Press, 1994.
27. B. L. Titzer, D. K. Lee, and J. Palsberg. Avrora: The AVR simulation and analysis framework. Available online at `http://compilers.cs.ucla.edu/avrora/`. Accessed: June 2019.
28. B. L. Titzer, D. K. Lee, and J. Palsberg. Avrora: Scalable sensor network simulation with precise timing. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN 2005)*, pages 477–482. IEEE, 2005.