High-Performance Ideal Lattice-Based Cryptography on 8-bit AVR Microcontrollers

Zhe Liu, University of Waterloo, Canada
Thomas Pöppelmann, Infineon Technologies AG, Germany
Tobias Oder, Ruhr-University Bochum, Germany
Hwajeong Seo, Institute for Inforcomm Research, Singapore
Sujoy Sinha Roy, Katholieke Universiteit Leuven, Belgium
Tim Güneysu, University of Bremen and DFKI, Germany
Johann Großschädl, University of Luxembourg, Luxembourg
Howon Kim, Pusan National University, Republic of Korea
Ingrid Verbauwhede, Katholieke Universiteit Leuven, Belgium

Over the last years lattice-based cryptography has received much attention due to versatile average-case problems like Ring-LWE or Ring-SIS that appear to be intractable by quantum computers. In this work we evaluate and compare implementations of Ring-LWE encryption and the bimodal lattice signature scheme (BLISS) on an 8-bit Atmel ATxmega128 microcontroller. Our implementation of Ring-LWE encryption provides comprehensive protection against timing side-channels and takes 24.9 ms for encryption and 6.7 ms for decryption. To compute a BLISS signature, our software takes 317 ms and 86 ms for verification. These results underline the feasibility of lattice-based cryptography on constrained devices.

CCS Concepts: •**Security and privacy** → **Public key (asymmetric) techniques; Digital signatures; Public key encryption;** •**Computer systems organization** → *Embedded software;*

Additional Key Words and Phrases: Ideal lattices, NTT, RLWE, BLISS, ATxmega

ACM Reference Format:
Zhe Liu, Thomas Pöppelmann, Tobias Oder, Hwajeong Seo, Sujoy Sinha Roy, Tim Güneysu, Johann Großschädl, Howon Kim and Ingrid Verbauwhede, 2016. High-Performance Ideal Lattice-Based Cryptography on 8-bit AVR Microcontrollers. *ACM Trans. Embedd. Comput. Syst.* ?, ?, Article ? (February 2016), 24 pages.
DOI: 0000001.0000001

1. INTRODUCTION

RSA and Elliptic Curve Cryptography(ECC)-based schemes are the most popular asymmetric cryptosystems to date, being deployed in billions of security systems and applications. Despite their predominance, they are known to be susceptible to attacks using quantum computers [Shor 1994] on which significant resources are spent to boost their further development [Rich and Gellman 2013]. Moreover, even standardization bodies like NIST [NIST 2015] and secret services like NSA's IAD [NSA 2015] acknowledge the need for discussions on the development, standardization, and deployment of efficient *post-quantum* public-key cryptography. Additionally, RSA and ECC have been shown to be quite inefficient on very small and constrained devices

Author's addresses: Hwajeong Seo, Institute for Infocomm Research A Star; Zhe Liu (Corresponding Author:sduliuzhe@gmail.com), University of Waterloo; Jongseok Choi and Howon Kim (Corresponding Author:howonkim@pusan.ac.kr), Computer Engineering, Pusan National University; ; Yasuyuki Nogami, Okayama University;

DOI: 0000001.0000001

like 8-bit AVR microcontrollers [Gura et al. 2004; Hutter and Schwabe 2013]. A possible alternative are asymmetric cryptosystems based on hard problems on ideal lattices. The special algebraic structure of ideal lattices [Lyubashevsky et al. 2010a] defined in $\mathcal{R} = \mathbb{Z}_q[\mathbf{x}]/\langle x^n + 1\rangle$ allows a significant reduction of key and ciphertext sizes and enables efficient arithmetic using the number theoretic transform (NTT)[1] [Nussbaumer 1982; Winkler 1996; Blahut 2010]. To realize lattice-based public key encryption several proposals exist (see [Cabarcas et al. 2014] for a comparison) like classical NTRU [Hoffstein et al. 1998] (defined in $\mathbb{Z}_q[\mathbf{x}]/\langle x^n - 1\rangle$), provably secure NTRU [Stehlé and Steinfeld 2011] (defined in $\mathbb{Z}_q[\mathbf{x}]/\langle x^n + 1\rangle$), or a scheme based on the ring learning with errors (RLWE) problem [Lindner and Peikert 2011; Lyubashevsky et al. 2010a] (from now on referred to as RLWEenc). From an implementation perspective, the RLWEenc scheme is currently one of the best-studied lattice-based public key encryption schemes (see [de Clercq et al. 2014; Chen et al. 2015; Roy et al. 2014; Pöppelmann and Güneysu 2014; Pöppelmann and Güneysu 2013; Boorghany et al. 2014; Liu et al. 2015a]). The scheme is also highly related to recently proposed key exchange protocols [Peikert 2014; Bos et al. 2015; Alkim et al. 2016a] which basically use key generation, encryption, and decryption of RLWEenc. When using a RLWEenc-based key exchange scheme it is possible to rely on a so-called error reconciliation mechanism for further efficiency. This way a key can be derived that has not explicitly been chosen by one party. However, using a public key encryption scheme for key transport (i.e., one party explicitly chooses a symmetric key) is a standard technique and thus RLWEenc could be also used for post-quantum key exchange.

Concerning signature schemes, several proposals exist like GLP [Güneysu et al. 2012] (derived from [Lyubashevsky 2012]), BG [Bai and Galbraith 2014], PASSSign [Hoffstein et al. 2014], a modified NTRU signature scheme [Melchor et al. 2014], or a signature scheme derived from a recently proposed IBE scheme [Ducas et al. 2014]. However, so far instantiations of the bimodal lattice signature schemes (BLISS) [Ducas et al. 2013a] seem superior in terms of signature size, performance, and security. Despite their popularity, implementation efforts so far mainly led to very efficient hardware designs for RLWEenc [Chen et al. 2015; Roy et al. 2014; Pöppelmann and Güneysu 2014] and BLISS [Pöppelmann et al. 2014] and fast software on 32-bit microcontrollers [Oder et al. 2014] and 64-bit microprocessors [Güneysu et al. 2013; Ducas et al. 2013a] but only few works cover constrained 8-bit architectures [Boorghany et al. 2014; Boorghany and Jalili 2014; Liu et al. 2015a]. Additionally, current works usually rely on the straightforward Cooley-Tukey radix-2 decimation-in-time algorithm (e.g., [Roy et al. 2014; de Clercq et al. 2014; Pöppelmann et al. 2014; Boorghany et al. 2014]) to implement the NTT and thus to realize polynomial multiplication $\mathbf{c} = \mathbf{a} \cdot \mathbf{b}$ for $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathcal{R}$ as $\mathbf{c} = \mathsf{INTT}(\mathsf{NTT}(\mathbf{a}) \circ \mathsf{NTT}(\mathbf{b}))$. However, by taking a closer look at works on the implementation [Crandall and Pomerance 2001; Chu and George 2000] of the highly related fast Fourier transform (FFT) it becomes evident that the sole focus on Cooley-Tukey radix-2 decimation-in-time algorithms prevents further optimizations of the NTT, especially given the constraints of an 8-bit architecture.

Contribution. In this work we present different optimization techniques to increase the performance of RLWEenc and BLISS. We review different approaches and varieties of NTT algorithms and improvements compared to previous work are mainly achieved by merging certain operations into the NTT itself (multiplication by n^{-1}, powers of ψ and ψ^{-1}) and by removing the expensive bit-reversal step. Additionally, we propose two methods to improve the performance of modular coefficient multiplication. For $q = 7681$, we describe the "MOV-and-ADD" technique for coefficient multiplication and the

[1] The NTT can be described as fast Fourier transform (FFT) over $\mathbb{Z}_q$.

SAMS2 technique for modular reduction. For $q = 12289$, we propose a constant-time tiny Montgomery multiplication based on the observation that the modulus belongs to the family of optimal prime fields (OPFs) [Liu et al. 2013].

This work is based on two previous conference versions [Liu et al. 2015b; Pöppelmann et al. 2015]. The main additional contributions are constant-time NTTs and noise samplers as components of the (up to our knowledge) first RLWEenc with comprehensive protection against timing attacks. Due to our improvements using Montgomery multiplication and the SAMS2 technique our constant-time NTTs achieve similar performance than the unprotected implementations in [Liu et al. 2015b; Pöppelmann et al. 2015]. Moreover, we show a new approach for constant-time noise generation based on a low-precision CDT sampler that combines the advantages of relieved precision requirements (see [Saarinen 2015]) for Ring-LWE with the performance of a table-based sampling approach. This sampler even outperforms the recently proposed very simple approach using the Binomial distribution (see [Alkim et al. 2016a]) as it deals better with larger standard deviations than the approach in [Alkim et al. 2016a]. All in all, our results shows that lattice-based cryptography can be used to realize the two most basic asymmetric primitives (public key encryption and signatures) on very constrained devices with high performance.

Outline. In Section 2 we review the number theoretic transform (NTT), RLWEenc, and BLISS. We then show NTT algorithms that are better suited for polynomial multiplication in Section 3. Our AVR ATxmega128 implementation is described in detail in Section 4 and we discuss our results in Section 5.

2. BACKGROUND

In this section we introduce the NTT and explicitly describe its application in the algorithms of the RLWEenc public key encryption scheme and the BLISS signature scheme.

2.1. The Number Theoretic Transform and Negacyclic Convolutions

The number theoretic transform (NTT) [Nussbaumer 1982; Winkler 1996; Blahut 2010] is similar to the discrete Fourier transform (DFT) but all complex roots of unity are exchanged for integer roots of unity and arithmetic is also carried out modulo an integer q in the field $GF(q)$[2]. The forward transformation $\tilde{\mathbf{a}} = \text{NTT}(\mathbf{a})$ of a length n sequence $(\mathbf{a}[0], \ldots, \mathbf{a}[n-1])$ to $(\tilde{\mathbf{a}}[0], \ldots, \tilde{\mathbf{a}}[n-1])$ with elements in $\mathbb{Z}_q$ is defined as $\tilde{\mathbf{a}}[i] = \sum_{j=0}^{n-1} \mathbf{a}[j]\omega^{ij} \bmod q,\ i = 0, 1, \ldots, n-1$ where ω is an n-th primitive root of unity. The inverse transform $\mathbf{a} = \text{INTT}(\tilde{\mathbf{a}})$ is defined as $\mathbf{a}[i] = n^{-1} \sum_{j=0}^{n-1} \tilde{\mathbf{a}}[j]\omega^{-ij} \bmod q,\ i = 0, 1, \ldots, n-1$ where ω is exchanged by ω^{-1} and the final result scaled by n^{-1}. For an n-th primitive root of unity ω_n it holds that $\omega_n^n = 1 \bmod q$, $\omega_n^{n/2} = -1 \bmod q$, $\omega_{\frac{n}{2}} = \omega_n^2$, and $\omega_n^i \neq 1 \bmod q$ for any $i = 1, \ldots, n-1$.

The main operation in ideal lattice-based cryptography is polynomial multiplication[3]. Schemes are usually defined in $\mathcal{R} = \mathbb{Z}_q[\mathbf{x}]/\langle x^n + 1\rangle$ with modulus $x^n + 1$ where n is a power of two and one can make use of the *negacyclic convolution* property of the NTT that allows carrying out a polynomial multiplication in $\mathbb{Z}_q[\mathbf{x}]/\langle x^n + 1\rangle$ using length-n transforms and no zero padding. When $\mathbf{a} = (\mathbf{a}[0], \ldots \mathbf{a}[n-1])$ and

[2]Actually, this is overly restrictive and the NTT is also defined for certain composite numbers (n has to divide $p - 1$ for every prime factor p of q). However, for the given target parameter sets common in lattice-based cryptography we can restrict ourselves to prime moduli and refer to [Nussbaumer 1982] for further information on composite moduli NTTs.

[3]Similar to exponentiation being the main operation of RSA or point multiplication being the main operation of ECC.

$\mathbf{b} = (\mathbf{b}[0], \ldots \mathbf{b}[n-1])$ are polynomials of length n with elements in $\mathbb{Z}_q$, ω be a primitive n-th root of unity in $\mathbb{Z}_q$ and $\psi^2 = \omega$, then we define $\mathbf{d} = (\mathbf{d}[0], \ldots \mathbf{d}[n-1])$ as the negative wrapped convolution of $\mathbf{a}$ and $\mathbf{b}$ so that $\mathbf{d} = \mathbf{a} \cdot \mathbf{b} \mod x^n + 1$. We then define $\bar{\mathbf{a}} = \mathsf{PMul}_{\psi}(\mathbf{a}) = (\mathbf{a}[0], \psi\mathbf{a}[1], \ldots, \psi^{n-1}\mathbf{a}[n-1])$ as well as the inverse multiplication by powers of ψ^{-1} denoted as $\mathbf{a} = \mathsf{PMul}_{\psi^{-1}}(\bar{\mathbf{a}})$. Then it holds that $\mathbf{d} = \mathsf{PMul}_{\psi^{-1}}(\mathsf{INTT}(\mathsf{NTT}(\mathsf{PMul}_{\psi}(\mathbf{a}) \circ \mathsf{NTT}(\mathsf{PMul}_{\psi}(\mathbf{b})))))$ [Winkler 1996], [Crandall and Fagin 1994; Crandall and Pomerance 2001], where $\circ$ denotes point-wise multiplication. For simplicity, we do not always explicitly apply PMul_{ψ} or $\mathsf{PMul}_{\psi^{-1}}$ when it is clear from the context that a negacyclic convolution is computed.

2.2. The RLWEenc Cryptosystem

The semantically secure public key encryption scheme RLWEenc was proposed in [Lindner and Peikert 2011; Lyubashevsky et al. 2010a; Lyubashevsky et al. 2010b] and is also used as a building block in the identity-based encryption scheme (IBE) by Ducas, Lyubashevsky, and Prest [Ducas et al. 2014]. We provide the key generation procedure $\mathsf{RLWEenc}_{\mathrm{GEN}}$ in Algorithm 1, the encryption procedure $\mathsf{RLWEenc}_{\mathrm{ENC}}$ in Algorithm 2, and the decryption procedure $\mathsf{RLWEenc}_{\mathrm{DEC}}$ in Algorithm 3. All algorithms explicitly use calls to the NTT and function names used later on during the evaluation of our implementation (see Section 5). The exact placement of NTT transformations is slightly changed compared to [Roy et al. 2014], which saved one transformation compared to [Pöppelmann and Güneysu 2013], as $\mathbf{c}_2$ is not transmitted in NTT form and thus removal of least significant bits is still possible (see [Ducas et al. 2014; Pöppelmann and Güneysu 2013]).

Algorithm 1 RLWEenc Key Generation

Precondition: Access to global constant $\tilde{\mathbf{a}} = \mathsf{NTT}(\mathbf{a})$
1: **function** $\mathsf{RLWEenc}_{\mathrm{GEN}}()$
2: $\quad \tilde{\mathbf{r}}_1 \leftarrow \mathsf{NTT}(\mathsf{Sampler}())$
3: $\quad \tilde{\mathbf{r}}_2 \leftarrow \mathsf{NTT}(\mathsf{Sampler}())$
4: $\quad \tilde{\mathbf{p}} = \tilde{\mathbf{r}}_1 - \tilde{\mathbf{a}} \circ \tilde{\mathbf{r}}_2$
5: $\quad$ **return** $(pk, sk) = (\tilde{\mathbf{p}}, \tilde{\mathbf{r}}_2)$
6: **end function**

Algorithm 2 RLWEenc Encryption

Precondition: Access to global constant $\tilde{\mathbf{a}} = \mathsf{NTT}(\mathbf{a})$
1: **function** $\mathsf{RLWEenc}_{\mathrm{ENC}}(\tilde{\mathbf{a}}, \tilde{\mathbf{p}}, \mu \in \{0,1\}^n)$
2: $\quad \tilde{\mathbf{e}}_1 = \mathsf{NTT}(\mathsf{Sampler}())$
3: $\quad \tilde{\mathbf{e}}_2 = \mathsf{NTT}(\mathsf{Sampler}())$
4: $\quad \tilde{\mathbf{c}}_1 = \tilde{\mathbf{a}} \circ \tilde{\mathbf{e}}_1 + \tilde{\mathbf{e}}_2$
5: $\quad \tilde{\mathbf{h}}_2 = \tilde{\mathbf{p}} \circ \tilde{\mathbf{e}}_1$
6: $\quad \mathbf{e}_3 \leftarrow \mathsf{Sampler}()$
7: $\quad \mathbf{c}_2 = \mathsf{INTT}(\tilde{\mathbf{h}}_2) + \mathbf{e}_3 + \mathsf{Encode}(m)$
8: $\quad$ **return** $(\tilde{\mathbf{c}}_1, \mathbf{c}_2)$
9: **end function**

Algorithm 3 RLWEenc Decryption

1: **function** $\mathsf{RLWEenc}_{\mathrm{DEC}}(\mathbf{c} = [\mathbf{c}_1, \mathbf{c}_2], \tilde{\mathbf{r}}_2)$
2: **return** $\mathsf{Decode}(\mathsf{INTT}(\mathbf{c}_1 \circ \tilde{\mathbf{r}}_2) + \mathbf{c}_2)$.
3: **end function**

The main idea of the scheme is that during encryption the n-bit encoded message $\bar{\mathbf{m}} = \mathsf{Encode}(m)$ is added to $\mathbf{pe}_1 + \mathbf{e}_3$ (in NTT notation $\mathsf{INTT}(\tilde{\mathbf{h}_2}) + \mathbf{e}_3$) which is uniformly random and thus hides the message. Decryption is only possible with knowledge of $\mathbf{r}_2$ since otherwise the large term $\mathbf{ae}_1\mathbf{r}_2$ cannot be eliminated when computing $\mathbf{c}_1\mathbf{r}_2 + \mathbf{c}_2$. The encoding of the message of length n is necessary as the noise term $\mathbf{e}_1\mathbf{r}_1 + \mathbf{e}_2\mathbf{r}_2 + \mathbf{e}_3$ is still present after calculating $\mathbf{c}_1\mathbf{r}_2 + \mathbf{c}_2$ and would prohibit the retrieval of the binary message after decryption. With the simple threshold encoding $\mathsf{Encode}(m) = \frac{q-1}{2}m$ the value $\frac{q-1}{2}$ is assigned only to each binary one of the string m. The corresponding decoding function needs to test whether a received coefficient $z \in [0 \ldots q-1]$ is in the interval $\frac{q-1}{4} \leq z < 3\frac{q-1}{4}$ which is interpreted as one and zero otherwise. As a consequence, the maximum error added to each coefficient must not be larger than $\lfloor \frac{q}{4} \rfloor$ in order to decrypt correctly. The probability for decryption errors depends on the shape (i.e., std. deviation) of the RLWEenc error/noise distribution that is sampled by Sampler. The usage of the discrete Gaussian distribution for RLWEenc, which comes natural from certain worst-case to average case reductions, has led to some controversy regarding the required precision and tail-cut for practical and theoretical security. The motivation to move away from high-precision discrete Gaussians is that the implementation of a high-precision discrete Gaussian sampler is costly (see [Bos et al. 2015]), especially when constant-time operation is required. Moreover, from a practical perspective it is extremely unlikely, that any instantiation of RLWEenc using a high-precision sampler will ever sample a value close to the tail cut (i.e. 10σ) as the probability, e.g., to sample 45 for $\sigma = 4.51$ is $< 2^{-75}$. Moreover, the concrete shape of a distribution is not exploited by lattice reduction algorithms or other approaches to cryptanalysis of RLWEenc and the main parameter of a distribution is only variance and entropy. The observations have also been made in other works [Saarinen 2015; Alkim et al. 2016a] and in [Alkim et al. 2016a] as additional safeguard a reduction using Rényi divergence is provided that shows that the usage of a Binomial distribution instead of a rounded continuous Gaussian will not lead to a significant advantage of an attacker when RLWEenc is used as a key exchange scheme.

It is also worth noting that decreasing σ reduces the error probability but also negatively affects the security of the scheme [Göttert et al. 2012; Lindner and Peikert 2011]. Increasing q on the other hand increases the key size, ciphertext expansion, and reduces performance (on certain devices). To support the NTT, Göttert et al. [Göttert et al. 2012] proposed parameter sets (n, q, s) where $\sigma = s/\sqrt{2\pi}$ denoted as RLWEenc-Ia $(256, 7681, 11.31)$ and RLWEenc-IIa $(512, 12289, 12.18)$. Lindner and Peikert [Lindner and Peikert 2011] originally proposed the parameter sets RLWEenc-Ib $(192, 4093, 8.87)$, RLWEenc-IIb $(256, 4093, 8.35)$ and RLWEenc-IIIb $(320, 4093, 8.00)$. The security levels of RLWEenc-Ia and RLWEenc-IIb are roughly comparable and RLWEenc-IIb provides 105.5 bits of pre-quantum security, according to a refined security analysis by Liu and Nguyen [Liu and Nguyen 2013] for standard LWE and the original parameter sets. The RLWEenc-IIa parameter set uses a larger dimension n and should thus achieve even higher security than the 156.9 bits obtained by Liu and Nguyen for RLWEenc-IIIb. For the IBE scheme in [Ducas et al. 2014] the parameters $n = 512$, $q \approx 2^{23}$ and a trinary error/noise distribution are used. An approach to estimate the security of Ring-LWE based encryption against quantum computers (in a pessimistic model) is

provided in [Alkim et al. 2016a]. Applying this approach to our parameter set, the estimated post-quantum security of the scheme is approximately 46 bits for RLWEenc-Ia and 104 bits for RLWEenc-IIa. Thus, for long-term security applications we recommend to use the parameter set RLWEenc-IIa. RLWEenc-Ia is still useful for applications that do not require post-quantum security. For RLWEenc-IIa the ciphertext size is 14,336 bits.

2.3. The BLISS Cryptosystem

In this work we only consider the efficient ring-based instantiation of BLISS [Ducas et al. 2013a]. We recall the key generation procedure $\text{BLISS}_{\text{GEN}}$ in Algorithm 4, the signing procedure $\text{BLISS}_{\text{SIGN}}$ in Algorithm 5, and the verification procedure $\text{BLISS}_{\text{VER}}$ in Algorithm 6. Key generation requires uniform sampling of sparse and small polynomials $\mathbf{f}, \mathbf{g}$, rejection sampling ($N_\kappa(\mathbf{S})$), and computation of an inverse. To sign a message, two masking polynomials $\mathbf{y}_1, \mathbf{y}_2 \leftarrow D_{\mathbb{Z}^n,\sigma}$ are sampled from a discrete Gaussian distribution using the $\text{SampleGauss}_\sigma$ function. The computation of $\mathbf{a}\mathbf{y}_1$ is performed using the NTT and the compressed $\mathbf{u}$ is then hashed together with the message μ by Hash. The binary string c' is used by GenerateC to generate a sparse polynomial $\mathbf{c}$. Polynomials $\mathbf{y}_1, \mathbf{y}_2$ then hide the secret key which is multiplied with the sparse polynomials using the SparseMul function. This function exploits that only κ coefficients in $\mathbf{c}$ are set and only $d_1 + d_2$ coefficients in $\mathbf{s}_1$ and $\mathbf{s}_2$. After a rejection sampling and compression step the signature $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$ is returned. The verification procedure $\text{BLISS}_{\text{VER}}$ in Algorithm 6 just checks norms of signature components and compares the hash output with $\mathbf{c}$ in the signature.

In this work we focus on the 128-bit pre-quantum secure BLISS-I parameter set which uses $n = 512$ and $q = 12289$ (same base parameters as RLWEenc-IIa). The density of the secret key is $\delta_1 = 0.3$ and $\delta_2 = 0$, the standard deviation of the coefficients of $\mathbf{y}_1$ and $\mathbf{y}_2$ is $\sigma = 215.73$ and the repetition rate is 1.6. The number of dropped bits in $\mathbf{z}_2$ is $d = 10$, $\kappa = 23$, and $p = \lfloor 2q/2^d \rfloor$. The final size of the signature is 5,600 bits with Huffman encoding and approx. 7,680 bits without Huffman encoding.

Algorithm 4 BLISS Key Generation

1: **function** $\text{BLISS}_{\text{GEN}}()$
2: Choose uniform polynomials $\mathbf{f}, \mathbf{g}$ with $\lceil \delta_1 n \rceil$ entries in $\{\pm 1\}$ and $\lceil \delta_2 n \rceil$ entries in $\{\pm 2\}$
3: $\mathbf{S} = (\mathbf{s}_1, \mathbf{s}_2)^t \leftarrow (\mathbf{f}, 2\mathbf{g} + 1)^t$
4: **if** $N_\kappa(\mathbf{S}) \geq 5 \cdot C^2 \cdot (\lceil \delta_1 n \rceil + 4 \lceil \delta_2 n \rceil) \cdot \kappa$ **then restart**
5: $\mathbf{a}_q = (2\mathbf{g} + 1)/\mathbf{f} \bmod q$ (**restart** if $\mathbf{f}$ non-invertible)
6: **Return**$(pk = \mathbf{A}, sk = \mathbf{S})$ where $\mathbf{A} = (\tilde{\mathbf{a}}_1 = \text{NTT}(\mathbf{a}_q), q - 2) \bmod 2q$
7: **end function**

3. FASTER NTTS FOR LATTICE-BASED CRYPTOGRAPHY

In this section we examine fast algorithms for the computation of the number theoretic transform (NTT) and show techniques to speed up polynomial multiplication for lattice-based cryptography[4]. The most straightforward implementation of the NTT is

[4]Most of the techniques discussed in this section have already been proposed in the context of the fast Fourier transform (FFT). However, they have not yet been considered to speed up ideal lattice-based cryptography (at least not in works like [Roy et al. 2014; de Clercq et al. 2014; Pöppelmann et al. 2014; Boorghany et al. 2014]). Moreover, some optimizations and techniques are mutually exclusive and a careful selection and balancing has to be made.

Algorithm 5 BLISS Signing

function $\text{BLISS}_{\text{SIGN}}(\mu \in \{0,1\}^*, pk=\mathbf{A}, sk=\mathbf{S})$
  $\mathbf{y}_1 \leftarrow \mathsf{SampleGauss}_\sigma()$
  $\mathbf{y}_2 \leftarrow \mathsf{SampleGauss}_\sigma()$
  $\mathbf{u} = 2\zeta \cdot \mathsf{INTT}(\tilde{\mathbf{a}}_1 \circ \mathsf{NTT}(\mathbf{y}_1)) + \mathbf{y}_2 \bmod 2q$
  $c' \leftarrow \mathsf{Hash}(\lfloor \mathbf{u} \rceil_d \bmod p, \mu)$
  $\mathbf{c} \leftarrow \mathsf{GenerateC}(c')$
  Choose a random bit b
  $\mathbf{z}_1 \leftarrow \mathbf{y}_1 + (-1)^b \mathsf{SparseMul}(\mathbf{s}_1, \mathbf{c})$
  $\mathbf{z}_2 \leftarrow \mathbf{y}_2 + (-1)^b \mathsf{SparseMul}(\mathbf{s}_2, \mathbf{c})$
  Continue with probability
    $1 \Big/ \left(M \exp\left(-\frac{\|\mathbf{Sc}\|^2}{2\sigma^2} \right) \cosh\left(\frac{\langle \mathbf{z}, \mathbf{Sc} \rangle}{\sigma^2} \right) \right)$
    otherwise **restart**
  $\mathbf{z}_2^\dagger \leftarrow (\lfloor \mathbf{u} \rceil_d - \lfloor \mathbf{u} - \mathbf{z}_2 \rceil_d) \bmod p$
  Return $(\mathbf{z}_1, \mathbf{z}_2^\dagger, \mathbf{c})$
end function

Algorithm 6 BLISS Verification

function $\text{BLISS}_{\text{VER}}(\mu \in \{0,1\}^*, pk=\mathbf{A}, sk=\mathbf{S})$
  if $\|(\mathbf{z}_1 | 2^d \cdot \mathbf{z}_2^\dagger)\|_2 > B_2$ **then** Reject
  if $\|(\mathbf{z}_1 | 2^d \cdot \mathbf{z}_2^\dagger)\|_\infty > B_\infty$ **then** Reject
  $\mathbf{r} \leftarrow \mathsf{INTT}(\tilde{\mathbf{a}}_1 \circ \mathsf{NTT}(\mathbf{z}_1))$
  $c' = \mathsf{Hash}\big(\lfloor 2\zeta \cdot \mathbf{r} + \zeta \cdot q \cdot \mathbf{c} \rceil_d + \mathbf{z}_2^\dagger \bmod p, \mu \big)$
  Accept iff $\mathbf{c} = \mathsf{GenerateC}(c')$
end function

a Cooley-Tukey radix-2 decimation-in-time (DIT) approach [Cooley and Tukey 1965] that requires a bit-reversal step as the algorithm takes bit-reversed input and produces naturally ordered output (from now on referred to as $\text{NTT}^{CT}_{bo \rightarrow no}$). To compute the NTT as defined in Section 2.1 the $\text{NTT}^{CT}_{bo \rightarrow no}$ algorithm applies the Cooley-Tukey (CT) butterfly, which computes $a' \leftarrow a + \omega b$ and $b' \leftarrow a - \omega b$ for some values of $\omega, a, b \in \mathbb{Z}_q$, overall $\frac{n \log_2(n)}{2}$ times. The biggest disadvantage of relying solely on the $\text{NTT}^{CT}_{bo \rightarrow no}$ algorithm is the need for bit-reversal, multiplication by constants, and that it is impossible to merge the final multiplication by powers of ψ^{-1} into the twiddle factors of the inverse NTT (see [Roy et al. 2014]). With the assumption that twiddle factors (powers of ω) are stored in a table and thus not computed on-the-fly it is possible to further simplify the computation and to remove bit-reversal and to merge certain steps. This assumption makes sense on constrained devices like the ATxmega, which have a rather large read-only flash.

3.1. Merging the Inverse NTT and Multiplication by Powers of ψ^{-1}

In [Roy et al. 2014] Roy et al. use the standard $\text{NTT}^{CT}_{bo \rightarrow no}$ algorithm for a hardware implementation and show how to merge the multiplication by powers of ψ (see Section 2.1) into the twiddle factors of the forward transformation. However, this approach does not work for the inverse transformation due to the way the computations are performed in the CT butterfly as the multiplication is carried out before the addition. In this section we show that it is possible to merge the multiplication by powers of ψ^{-1}

during the inverse transformation using a fast decimation-in-frequency (DIF) algorithm [Gentleman and Sande 1966]. The DIF NTT algorithm splits the computation into a sub-problem on the even outputs and a sub-problem on the odd outputs of the NTT and has the same complexity as the $\text{NTT}^{CT}_{bo\rightarrow no}$ algorithm. It requires usage of the so-called Gentlemen-Sande (GS) butterfly which computes $a' \leftarrow a+b$ and $b' \leftarrow (a-b)\omega$ for some values of $\omega, a, b \in \mathbb{Z}_q$. Following [Chu and George 2000, Section 3.2], where ω_n is an n-th primitive root of unity and by ignoring the multiplication by the scalar n^{-1}, the inverse NTT and application of PMul_ψ can be defined as

$$\mathbf{a}[r] = \psi^{-r} \sum_{\ell=0}^{n-1} \mathbf{A}[\ell]\omega_n^{-r\ell} \tag{1}$$

$$= \psi^{-r} \left(\sum_{\ell=0}^{\frac{n}{2}-1} \mathbf{A}[\ell]\omega_n^{-r\ell} + \sum_{\ell=0}^{\frac{n}{2}-1} \mathbf{A}[\ell+\frac{n}{2}]\omega_n^{-r(\ell+\frac{n}{2})} \right) \tag{2}$$

$$= \psi^{-r} \sum_{\ell=0}^{\frac{n}{2}-1} \left(\mathbf{A}[\ell] + \mathbf{A}[\ell+\frac{n}{2}]\omega_n^{-r\frac{n}{2}} \right) \omega_n^{-r\ell}, \tag{3}$$

$$r = 0, 1, \ldots, n-1. \tag{4}$$

When r is even this results in

$$\mathbf{a}[2k] = \sum_{\ell=0}^{\frac{n}{2}-1} \left(\mathbf{A}[\ell] + \mathbf{A}[\ell+\frac{n}{2}] \right) \omega_{\frac{n}{2}}^{-k\ell} (\psi^2)^{-k} \tag{5}$$

and for odd r in

$$\mathbf{a}[2k+1] \tag{6}$$

$$= \sum_{\ell=0}^{\frac{n}{2}-1} \left(\mathbf{A}[\ell] - \mathbf{A}[\ell+\frac{n}{2}]\omega_n^{-\ell} \right) \omega_{\frac{n}{2}}^{-k\ell} \psi^{-(2k+1)} \tag{7}$$

$$= \sum_{\ell=0}^{\frac{n}{2}-1} \left(\left(\mathbf{A}[\ell] - \mathbf{A}[\ell+\frac{n}{2}] \right) \psi^{-1}\omega_n^{-\ell} \right) \omega_{\frac{n}{2}}^{-k\ell} (\psi^2)^{-k}, \tag{8}$$

$$k \in [0, \frac{n}{2} - 1]. \tag{9}$$

The two new half-size sub-problems where ψ is exchanged by ψ^2 can now be again solved using the recursion. As a consequence, when using an in-place radix-2 DIF algorithm it is necessary to multiply all twiddle factors in the first stage by ψ^{-1}, all twiddle factors in the second stage by ψ^{-2} and in general by ψ^{-2^s} for stage $s \in \{0, 1, \ldots, \log_2(n) - 1\}$ to merge the multiplication by powers of ψ^{-1} into the inverse NTT (see Figure 1 for an illustration). In case the PMul_ψ or $\mathsf{PMul}_{\psi^{-1}}$ operation is merged into the NTT computation we denote this by an additional superscript ψ or ψ^{-1}, e.g., as $\text{NTT}^{CT,\psi}_{bo\rightarrow no}$.

3.2. Removing Bit-Reversal

For memory efficient and in-place computation a reordering or so-called bit-reversal step is usually applied before or after an NTT/FFT transformation due to the required reversed input ordering of the $\text{NTT}^{CT}_{bo\rightarrow no}$ algorithm used in works like [Roy et al. 2014; de Clercq et al. 2014; Pöppelmann et al. 2014; Boorghany et al. 2014]. However, by

manipulation of the standard iterative algorithms and independently of the used butterfly (CT or GS) it is possible to derive natural order to bit-reversed order ($no \rightarrow bo$) as well as bit-reversed to natural order ($bo \rightarrow no$) forward and inverse algorithms. The derivation of FFT algorithms with a desired ordering of inputs and outputs is described in [Chu and George 2000] and we followed this description to derive the NTT algorithms $\text{NTT}^{CT}_{bo\rightarrow no}$, $\text{NTT}^{CT}_{no\rightarrow bo}$, $\text{NTT}^{GS}_{no\rightarrow bo}$, and $\text{NTT}^{GS}_{bo\rightarrow no}$, as well es their respective inverse counterparts. It is also possible to construct self-sorting NTTs ($no \rightarrow no$) but in this case the structure becomes irregular and temporary memory is required (see [Chu and George 2000]).

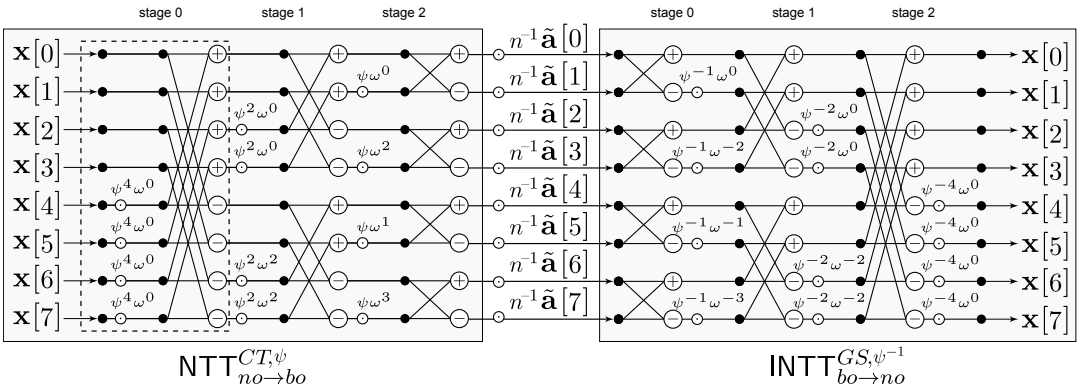

Fig. 1: Signal flow graph for multiplication of a polynomial $\mathbf{x}$ by a pre-transformed polynomial $\tilde{\mathbf{a}} = \text{NTT}^{CT,\psi}_{no\rightarrow bo}(\mathbf{a})$, using the $\text{NTT}^{CT,\psi}_{no\rightarrow bo}$ and $\text{INTT}^{GS,\psi^{-1}}_{bo\rightarrow no}$ algorithms.

3.3. Tuning for Lattice-Based Cryptography

The optimizations discussed in this section so far can be used to generically optimize polynomial multiplication in $\mathbb{Z}_q[\mathbf{x}]/\langle x^n + 1\rangle$. However, for lattice-based cryptography there are special conditions that hold for most practical algorithms; in the NTT-enabled algorithms of RLWEenc and BLISS every point-wise multiplication (denoted by $\circ$) is performed with a constant and a variable, usually a randomly sampled polynomial. Thus the most common operation in lattice-based cryptography is *not* simple polynomial multiplication but multiplication of a (usually random) polynomial by a constant polynomial (i.e., global constant or public key). Thus the scaling factor n^{-1} can be multiplied into the pre-computed and pre-transformed constant $\tilde{\mathbf{a}} = n^{-1}\text{NTT}^{CT}_{no\rightarrow bo}(\mathbf{a})$. Taking into account that we also want to remove the need for bit-reversal and want to merge the multiplication by powers of ψ into the forward and inverse transformation (as discussed in Section 3.1) we propose to use an $\text{NTT}^{CT,\psi}_{no\rightarrow bo}$ for the forward transformation and an $\text{INTT}^{GS,\psi^{-1}}_{bo\rightarrow no}$ for the inverse transformation. In this case a polynomial multiplication $\mathbf{c} = \mathbf{a} \cdot \mathbf{e}$ can be implemented without bit-reversal as $\mathbf{c} = \text{INTT}^{GS,\psi^{-1}}_{bo\rightarrow no}\left(\text{NTT}^{CT,\psi}_{no\rightarrow bo}(\mathbf{a}) \circ \text{NTT}^{CT,\psi}_{no\rightarrow bo}(\mathbf{e})\right)$. In Figure 2 we compare the necessary blocks for the straightforward approach and our proposal and provide an example flow diagram for $n = 8$ in Figure 1. For more details, pseudo-code of $\text{NTT}^{CT,\psi}_{no\rightarrow bo}$ is provided in Algorithm 8 and pseudo-code of $\text{INTT}^{GS,\psi^{-1}}_{bo\rightarrow no}$ is given in Algorithm 9. Longa and Naehrig [Longa and Naehrig 2016] present another technique to speed up the NTT that is related to ours. As their proposal focuses on 64-bit platforms and is expected

to have a considerable lower performance on 32-bit platforms (and thus even more on 8-bit ones) as stated in [Alkim et al. 2016b], we did not consider this approach for our implementation.

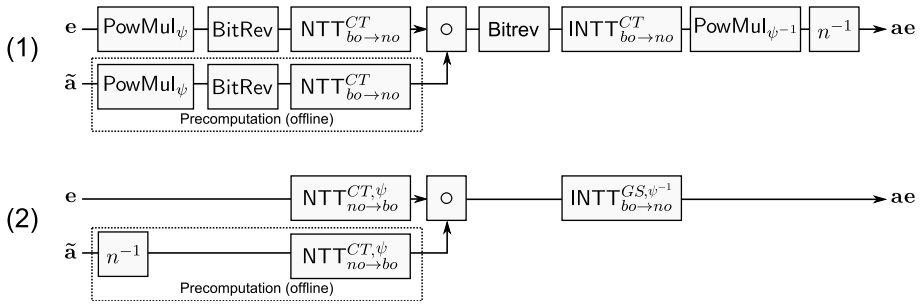

Fig. 2: Comparison of naive implementation of polynomial multiplication by a precomputed constant using only $\text{NTT}^{CT}_{bo\to no}$ and $\text{INTT}^{CT}_{bo\to no}$ (1) and our proposed approach (2) using $\text{NTT}^{CT,\psi}_{no\to bo}$ and $\text{INTT}^{GS,\psi^{-1}}_{bo\to no}$.

4. IMPLEMENTATION OF LATTICE-BASED CRYPTOGRAPHY ON ATXMEGA128

In this section we provide details on our implementation of the NTT as well as RLWEenc, and BLISS on the ATxmega128. Our implementation of RLWEenc has a constant execution time and is therefore protected against timing side-channels which is required for most practical and interactive applications. However, sampling in ideal lattice-based signature schemes, like BLISS, requires a much higher standard deviation and sampling precision. Unfortunately, it is still an open question how to efficiently perform constant-time sampling from these distributions and therefore, we solely focus on performance of BLISS and to a lesser extent on a small memory and code footprint in this work.

4.1. Implementation of the NTT

For the use in RLWEenc and BLISS we focus on the optimization of the $\text{NTT}^{CT,\psi}_{no\to bo}$ and $\text{INTT}^{GS,\psi^{-1}}_{bo\to no}$ transformations as described in Section 3. We implemented both algorithms in C and optimized the modular multiplication and butterfly operations using assembly language.

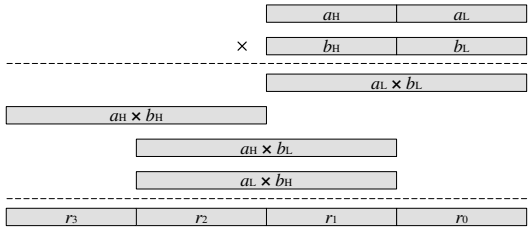

Fig. 3: The MOV-and-ADD coefficient multiplication.

MOV-and-ADD + SAMS2. We propose an optimized technique for performing coefficient multiplication called "MOV-and-ADD". The MOV-and-ADD multiplication technique resembles the traditional double-precision multiplication, but calculates the byte-products in a more efficient order to reduce the number of `adc` instructions. As shown in Figure 3, we split the computation process into four blocks and perform the byte multiplication in a sequence of $a_L \cdot b_L$, $a_H \cdot b_H$, then $a_H \cdot b_L$, and finally $a_L \cdot b_H$.

Inspired by the work in [Boorghany et al. 2014], we propose an optimized technique for reduction technique called "SAMS2", which only consists of four different basic operations, namely, shifting, addition, multiplication as well as subtractions. We will use Figure 4 to describe the process of SAMS2. The input of SAMS2 is the product of $t = a \cdot b$ obtained from the "MOV-and-ADD" method, where t is kept in four registers $(r3, r2, r1, r0)$ and have been marked by different colors. Each of the registers $r3, r2, r1, r0$ are 8 bits long. The reduction with 7681 using SAMS2 approach can be performed as follows: we first get the value of $t \gg 13$, $t \gg 17$ and $t \gg 21$ by *shifting*. We right shift $r3, r2, r1$ by one bit, and store the intermediate result of $r3, r2$ in $t0$ (i.e. the value of $t \gg 17$). After that, we right shift by 4 bits to get the results $t1$ (i.e. the value of $t \gg 13$) and $t2$ (i.e. the value of $t \gg 21$). Then, we can get the quotient h by performing an *addition* of $t0 + t1 + t2$. Apparently, the sum result is less than 16-bit, which can be kept in two registers. Thereafter, we calculate the values of $h \cdot q$ by *multiplication*. Since the lower byte of 7681 is `1`, a 16×8-bit multiplication and several addition operations are sufficient. Next, we *subtract* both the sum of h and the product obtained from step 3 from t. However, the result we get in step 4 may still be larger than $q = 7681$, thus, we *subtract* the modulus q at most two times. A constant-time modular coefficient multiplication using a combination of MOV-and-ADD and SAMS2 requires 63 and 94 cycles for $q = 7681$ and $q = 12289$, respectively.

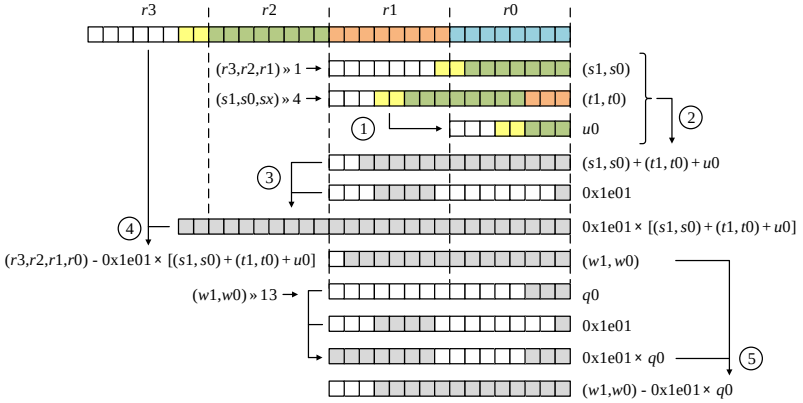

Fig. 4: Fast reduction operation with SAMS2 method for $q = 7681$. ①: shifting; ②: addition; ③: multiplication; ④: subtraction; ⑤: subtraction; The colorful parts mean that this bit has been occupied while the white part means the current bit is empty.

Tiny Montgomery Multiplication. We propose a constant-time tiny Montgomery multiplication for computing coefficient multiplication. As shown in Algorithm 7, the tiny Montgomery multiplication can be computed in four steps. The first step (line 2) is to compute the product of coefficients $a \cdot b$ by using proposed MOV-and-ADD method. Then, we only compute the lower 16-bit of $t \cdot q'$ and get the value of s by a masking operation (line 3). In line 4, we first compute the coefficient multiplication $s \cdot q$, and

Algorithm 7 Tiny Montgomery Multiplication for $q = 7681$

Precondition: 13-bit modulus $q = 7681$, Montgomery radix $R = 2^{13}$, (incomplete) coefficients $a, b \in [0, 2^{13} - 1]$, pre-computed constant $q' = -q^{-1} \bmod 2^{13} = 7679$

1: **function** z = $(a \cdot b \cdot R^{-1} \mod q)$
2: $t \leftarrow a \cdot b$
3: $s \leftarrow t \cdot q' \bmod R$
4: $z \leftarrow (t + s \cdot q)/R$
5: **if** $z \geq q$ **then** $z \leftarrow z - q$ **end if**
6: **if** $z \geq q$ **then** $z \leftarrow z - q$ **end if**
7: **return** z
8: **end function**

then add the result with t (kept in four registers), after that, we compute z by using several shifting operations. Finally, in line 5 and 6, two conditional subtractions are necessary to make sure the result is in $[0, q - 1]$. In summary, the processing of tiny Montgomery multiplication in Algorithm 7 requires 73 clock cycles to run in constant time. This achieved result outperforms the Barrett reduction [Barrett 1987] (roughly 600 cycles) and the shift-and-add (roughly 216 cycles) by a factor of roughly 8.2 and 3 respectively.

Since the Montgomery multiplication needs one transformation into the Montgomery domain at the beginning of the $\text{NTT}^{CT}_{no\rightarrow bo}$ and one backwards transformation at the end of $\text{NTT}^{GS}_{bo\rightarrow no}$, the MOV-and-ADD multiplication with subsequent SAMS2 reduction is still faster for $q = 7681$. For $q = 12289$, a coefficient multiplication can be computed within 70 cycles. Three cycles are saved by needing less shift operations and therefore the tiny Montgomery reduction gets outperformed even when the transformation to and from the Montgomery domain are taken into account.

Usage of Look-up Tables for Narrow Input Distributions. As discussed in Section 3.3 it is common in lattice-based cryptography to apply forward transformations mostly to values sampled from a narrow binomial error/noise distribution (other polynomials are usually constants and pre-computed). In this case only a limited number of possible inputs to the butterfly of the first stage of the $\text{NTT}^{CT,\psi}_{no\rightarrow bo}$ transformation exist and it is possible to pre-compute look-up tables to speed-up the modular multiplication. Especially for RLWEenc the range of possible input coefficients to the first stage butterfly is rather small, since they are Gaussian distributed with a low standard deviation.

4.2. Implementation of LP-RLWE

Our implementation of $\text{RLWEenc}_{\text{ENC}}$ and $\text{RLWEenc}_{\text{DEC}}$ of the RLWEenc scheme as described in Section 2.2 mainly consists of forward and inverse NTT transformations ($\text{NTT}^{CT,\psi}_{no\rightarrow bo}$ and $\text{INTT}^{GS,\psi^{-1}}_{bo\rightarrow no}$), Gaussian sampling (Sampler), and point-wise multiplication (PwMulFlash, also denoted as $\circ$). We assume that secret and public keys are stored in the read-only flash memory, but loading from RAM would also be possible, and probably even faster. Due to the usage of the NTT, the $\text{NTT}^{CT,\psi}_{no\rightarrow bo}$ transformation is only applied on the small noise polynomials $\mathbf{e}_1, \mathbf{e}_2$. Thus we can optimize the transformation for this input distribution and it is possible to substitute all multiplications in stage one of $\text{NTT}^{CT,\psi}_{no\rightarrow bo}$ by look-ups to a table of 41/45 entries that requires 82/90 bytes of flash memory.

4.2.1. $\text{Sampler}_{\text{CDT-high}}$*: Non-Constant Time High-Precision CDT.* For the sampling of the Gaussian distributed polynomials with high precision our first approach is to use a cumulative distribution table (CDT) [Devroye 1986; Peikert 2010]. We construct the

table M with entries $p_z = \Pr(x \leq z : x \leftarrow D_\sigma)$ for $z \in [0, \tau\sigma]$ with a precision of $\lambda = 96$ bits. The tail-cut factor τ determines the number of lines $|z_t| = \lceil \tau\sigma \rceil$ of the table and reduces the output to the range $x \in \{-\lceil \tau\sigma \rceil, \ldots, \lceil \tau\sigma \rceil\}$. To sample a value we choose a uniformly random y from the interval $[0, 1)$ and a bit b and return the integer $(-1)^b z \in \mathbb{Z}$ such that $y \in [p_{z-1}, p_z)$. Further we store only the positive half of the tables and then sample a sign bit. For this, the probability of sampling zero has been pre-halved when constructing the table. The constant CDF matrix M is stored in the read-only flash memory with $k = \lceil \sigma\tau \rceil$ rows and $l = \lceil \lambda/8 \rceil$ columns. Implementing the CDT approach with high precision to run in constant time is quite expensive as shown in [Bos et al. 2015] since the complete table has to be searched for every run of the sampler. Additionally, 12 bytes of randomness would be required for each constant-time run of the sampler.

4.2.2. Sampler$_{KY\text{-}high}$*: Non-Constant Time High-Precision Knuth-Yao Sampling.* Another approach is to use the Knuth-Yao sampler [Knuth and Yao 1976]. In order to achieve a precision of 2^{-90} for dimension $n = 256$, the Knuth-Yao algorithm requires a probability matrix P_{mat} of 55 rows and 109 columns [de Clercq et al. 2014]. Our implementation stores each column (i.e. 55-bit) in seven bytes and the probability matrix occupies 763 bytes in total. A straightforward implementation using bit-scanning method operation requires to check each bit and decreases the distance (d) whenever the bit is set. Instead of this, we perform the algorithm in a byte-scanning fashion which compares the eight concatenated bits of each byte and only perform the scanning operations for such non-zero bytes. We also apply the Look-Up Table (LUT)-based approach to our byte-wise scanning implementation [de Clercq et al. 2014]. If the most significant bit of the look-up result is cleared then the algorithm completed the look-up operation successfully. Otherwise, the most significant bit of the look-up result is set, which means a look-up failure has occurred and we proceed with the next level of the sampling. Similarly, a second LUT is used for level 9 to 13 in the same Gaussian distribution.

4.2.3. Sampler$_{Bio\text{-}low}$*: Constant Time Binomial Sampler.* In [Alkim et al. 2016a] it was shown that the high precision discrete Gaussian distribution used in a key exchange scheme similar to RING-LWEENCRYPT can be replaced by a centered binomial distribution without significantly hurting security. It is possible to sample from the centered binomial distribution by computing $\sum_{i=0}^{k} b_i - b'_i$, where the $b_i, b'_i \in \{0, 1\}$ are uniform independent bits. In this case the variance is $k/2$ and to achieve the required standard deviation of $\sigma = 4.51$ (RLWEenc-Ia) resp. $\sigma = 4.86$ (RLWEenc-IIa), k has to be set to a value of 41 resp. 48. The implementation of the binomial sampler is very straight-forward. For k=48 we need exactly 12 bytes of randomness to sample one coefficient. This huge randomness consumption unfortunately leads to a performance penalty. On the other hand, a constant execution time and no need for precomputed tables are advantages of the binomial sampler.

4.2.4. Sampler$_{CDT\text{-}low}$*: Constant Time Medium-Precision CDT.* A more efficient approach to sample in constant time is a cumulative distribution table with two bytes of precision. As elaborated in Section 2.2 a CDT sampler with less precision is not expected to lower the security of the scheme significantly. To avoid comparing the input with each entry in the CDT, we also make use of a guide table [Pöppelmann et al. 2014; Devroye 1986] with a size of 256 bytes. This table maps one input byte onto an entry in the CDT. Now we can reduce the number of comparisons to the maximum number of CDT entries that share the same first byte value. The result of the guide table look-up is then used as starting point for those comparisons. For $n = 256$, we can reduce the number of comparisons from 20 to 7 by this technique since 7 table entries have a first byte value of $0xFF$.

The sampler must also keep track of how many bytes of randomness were used. The sampler always needs one bit for the sign and one byte for the comparisons with the table entries. In case this input bytes matches one of the table entries, the sampler needs another byte. Therefore after the first comparison, we check whether the first input byte and the starting table entry were equal or not. Since we are using a guide table approach, it suffices to perform this check after the first comparison only because subsequent table entries can only be equal to the input if the first table entry already was.

Random numbers for the Gaussian sampling are obtained from a pseudo random number generator (PRNG) using the hardware AES-128 engine running in counter mode. The PRNG is seeded by noise from the LSB of the analogue digital converter. For the state (key, plaintext) 32 bytes of statically allocated memory are necessary. At the beginning of each encryption we fill a randomness buffer by running the PRNG multiple times in succession. Our Gaussian sampler needs one bit to determine the sign and with a probability of $\frac{242}{256}$ ($n = 256$) one byte of randomness for the table look-up and two bytes otherwise. Therefore we have to set the buffer size such that the probability of not providing enough randomness is negligible. In case every sampler run only requires one byte of randomness, the total amount of randomness used is $3n + \frac{3n}{8}$. It makes sense to choose a multiple of 16 as buffer size since the PRNG outputs 16 bytes in one run. For n=256, the probability that in 768 sampler runs at least 144 runs need two bytes of randomness instead of one is

$$1 - \sum_{i=0}^{144} \binom{768}{i} (\frac{242}{256})^{768-i} (\frac{14}{256})^{i} = 2^{-124.029}. \tag{10}$$

Therefore we set the buffer size to $3n + \frac{3n}{8} + 144 = 1008$. For $n = 512$, the buffer size is $3n + \frac{3n}{8} + 224 = 1952$. Even though in many cases we do not need a second byte, we always perform a two-byte comparison to make the sampler run in constant time. In case the output of the sampler is only determined by the first byte and the second byte has no influence on the result, we reuse the second byte in the next sampling operation.

4.2.5. Constant-time arithmetic. To create a constant-time and still efficient implementation, we implemented all core operations in assembly. Addition and subtractions are computed $\mod q$ and therefore usually contain a conditional subtraction or addition of q. To make this operation unconditional, we follow the standard approach of constant-time implementation by subtracting or adding q, checking whether a carry appeared, and creating a mask out of the carry. We then compute the AND-conjunction of the mask and q and attempt to reverse to addition/subtraction by subtracting/adding the masked value of q. By doing so, the executed operations will always be the same and the result will still be correctly reduced $\mod q$. For the multiplication in the NTT bufferfly operation, we used to SAMS2 algorithm for $q = 7681$ and the Montgomery multiplication for $q = 12289$. Beside the multiplication, the butterfly operations of the NTT always consist of one addition and one subtraction with subsequent reduction. For efficiency reasons we also created constant-time assembly functions for threshold encoding and decoding. For the encoding, we load one byte of the message, shift it to the right, and again use the carry to create a mask. We then proceed with the standard method as described above and repeat the procedure for each message bit. For decoding, we load one element of the ciphertext polynomial and subtract $\frac{q-1}{4}$. If a carry appears, we already know the decoded message bit has to be 0. If no carry appears, we proceed with subtracting $\frac{q-1}{4}$. This time a carry indicates that the decoded message bit has to be 1.

4.3. Implementation of BLISS

Besides polynomial arithmetic the most expensive operation for BLISS is the sampling of $\mathbf{y}_1$ and $\mathbf{y}_2$ from a discrete Gaussian distribution ($\mathsf{SampleGauss}_\sigma$). For the rather large standard deviation of $\sigma = 215.73$ (RLWEenc requires only $\sigma = 4.86$) a straightforward CDT sampling approach, even with binary search, would lead to a large table with roughly $\tau\sigma = 2798$ entries of approx. 30 to 40 kilobytes overall (see [Boorghany and Jalili 2014]). Another option for embedded devices would be the Bernoulli approach from [Ducas et al. 2013a] implemented in [Boorghany et al. 2014] but the reported performance of 13,151,929 cycles to sample one polynomial would cause a massive performance penalty. As a consequence, we implemented the hardware-optimized sampler from [Pöppelmann et al. 2014] on the ATxmega. It uses the convolution property of Gaussians combined with Kullback-Leibler divergence and mainly exploits that it is possible to sample a Gaussian distributed value with variance σ^2 by sampling x_1, x_2 from a Gaussian distribution with smaller standard deviation σ' such that $\sigma'^2 + k^2\sigma'^2 = \sigma^2$ and combining them as $x_1 + kx_2$ (for BLISS-I $k = 11$ and $\sigma' = 19.53$). Additionally, the performance of the σ'-sampler is improved by the use of short-cut intervals where each possible value of the first byte of the uniformly distributed input is assigned to an interval that specifies the range of the possible sampled values. This approach reduces the number of necessary comparisons and nearly compensates for the additional costs incurred by the requirement to sample two values (x_1, x_2 with $\sigma' = 19.53$) instead of one directly (with $\sigma = 215.73$). The sampling is again performed in a lazy manner and we use the same PRNG based on AES-128 as for RLWEenc.

To implement the $\mathrm{NTT}^{CT}_{no\rightarrow bo}$ we did not use a look-up table for the first stage as the input range $[-\sigma\tau, \sigma\tau]$ of $\mathbf{y}_1$ or $\mathbf{z}_1$ is too large. For the instantiation of the random oracle (Hash) that is required during signing and verification we have chosen the official AVR implementation of Keccak [Bertoni et al. 2012]. From the output of the hash function the sparse polynomial $\mathbf{c}$ with κ coefficients equal to one is generated by the GenerateC (see [Ducas et al. 2013b, Section 4.4]) routine. We store only κ indices where a coefficient of $\mathbf{c}$ is one. This reduces the dynamic RAM consumption and allows a more efficient implementation of the multiplication of $\mathbf{c}$ by $\mathbf{s}_1$ and $\mathbf{s}_2$ using the SparseMul routine. By using column-wise multiplication and by ignoring all zero coefficients, the multiplication can be performed more efficiently than with the NTT.

5. RESULTS AND COMPARISON

All implementations are measured on an 8-bit ATxmega128A1 microcontroller running at 32 MHz and featuring 128 Kbytes read-only flash, 8 Kbytes RAM and 2 Kbytes EEPROM. Cycle accurate performance measurement were obtained using two coupled 16-bit timer/counters and dynamic RAM consumption is measured using stack canaries (see [Pöppelmann et al. 2015] for more details). All public and private keys are assumed to be stored in the flash of the microcontroller and we consider the `.text` + `.data` + `.bootloader` sections to determine the flash memory utilization. For our implementation we used no calls to the standard library, the avr-gcc compiler in version 4.7.0, and the following compiler options (shortened): `-Os -fpack-struct -ffunction-sections -fdata-sections -flto`.

RLWEenc. Detailed cycle counts for the encryption and decryption as well as the most expensive operations are given in Table I. The costs of the encryption are dominated by the NTT (two calls of $\mathrm{NTT}^{CT,\psi}_{no\rightarrow bo}$ and one call of $\mathrm{INTT}^{GS,\psi^{-1}}_{bo\rightarrow no}$) which requires approx. 70% of the overall cycles for RLWEenc-Ia. The Gaussian sampling requires 20% of the overall cycles which is approx. 207 cycles per sample. The remaining amount of cycles (10%) is consumed by additions, point-wise multiplications by a key stored in the

Table I: Performance of our implementation of RLWEenc on an 8-bit ATxmega128 microcontroller.

Operation	(n=256, q=7681)	(n=512, q=12289)
	Cycle counts and stack usage	
$\text{RLWEenc}_{\text{ENC}}$	796,872 (93 bytes)	1,975,806 (86 bytes)
$\text{RLWEenc}_{\text{DEC}}$	215,031 (73 bytes)	553,536 (68 bytes)
$\text{NTT}^{CT,\psi}_{no\rightarrow bo}$	194,145	516,971
$\text{INTT}^{GS,\psi^{-1}}_{bo\rightarrow no}$	174,023	468,090
$\text{Sampler}_{\texttt{CDT-low}}$	53,023	105,153
PwMulFlash	25,148	53,821
AddEncode	7,631	15,217
Decode	4,825	9,595
	Static memory consumption in bytes	
Complete binary	6,524	9,322
RAM	1,088	2,144

Note that the stack usage is divided into a fixed amount of memory necessary for plaintext, ciphertext, and additional components (like random number generation) and the dynamic consumption of the encryption and decryption routine. We encrypt a message of n bits.

flash (PwMulFlash), and message encoding (Encode). Decryption is extremely simple, fast, and basically calls $\text{INTT}^{GS,\psi^{-1}}_{bo\rightarrow no}$, the decoding and an addition so that roughly 149 decryption operation could be performed per second on the ATxmega128.

The NTT can be performed in place so that no additional large temporary memory on the stack is needed. But storing the NTT twiddle factors for forward and inverse transforms in flash consumes $2n$ words = $4n$ bytes which is around 16% of the allocated flash memory for $q = 7681$ and around 22% for $q = 12289$.

BLISS. In Table II, we present detailed cycle counts for signing and verifying as well as for the most expensive operations in BLISS-I. Due to the rejection sampling and the chosen parameter set 1.6 signing attempts are required on average to create one signature. One attempt requires 6,341,763 cycles on average and only a small portion of the computation, i.e., the hashing of the message, does not have to be repeated in case of a rejection. During a signing attempt the most expensive operation is the sampling of two Gaussian distributed polynomials which takes $2\times1{,}141{,}007 = 2{,}282{,}014$ cycles (36% of the overall cycles). The calls to $\text{NTT}^{CT,\psi}_{no\rightarrow bo}$ and $\text{INTT}^{GS,\psi^{-1}}_{bo\rightarrow no}$ account for 15.2% of the overall cycles of one attempt. In contrast to the RLWEenc implementation we do not use the constant-time Montgomery approach and therefore the performance of the $\text{NTT}^{CT,\psi}_{no\rightarrow bo}$ is slightly different compared to RLWEenc. Hashing the compressed u and the message μ is time consuming and accounts for roughly 21% of the overall cycles during one attempt. Savings would be definitely possible by using a different hash function (see [Balasch et al. 2012] for an evaluation of different functions) but Keccak appears to be a conservative choice that matches the 128-bit security target very well. The sparse multiplication takes only 504,023 cycles for one multiplication what is preferable than computing the product with multiple NTT transformations. The flash memory consumption includes $2n$ words which equals $4n = 2{,}048$ bytes for the NTT twiddle factors and 3,374 bytes for look-up tables of the sampler.

Table II: Performance of our implementation of BLISS on an 8-bit ATxmega128 microcontroller.

Operation	(n=512, q=12289)
	Cycle counts and stack usage
$\text{BLISS}_{\text{SIGN}}$	10,156,247 (3,708 bytes)
$\text{BLISS}_{\text{VER}}$	2,760,244 (1,167 bytes)
$\text{NTT}^{CT,\psi}_{no\rightarrow bo}$	489,147
$\text{INTT}^{GS,\psi^{-1}}_{bo\rightarrow no}$	477,220
$\text{Sampler}_{\texttt{CDT-high}}$	1,141,007
SparseMul	504,023
Hash	1,335,040
GenerateC	4,411
DropBits	11,826
$\cosh\left(\frac{\langle \mathbf{z},\mathbf{Sc}\rangle}{\sigma^2}\right)$	75,499
$M\exp\left(-\frac{\|\mathbf{Sc}\|^2}{2\sigma^2}\right)$	37,102
	Static memory consumption in bytes
Complete binary	18,401
RAM	2,411

The stack usage is divided into a fixed amount of memory necessary for for message, signature, and additional components (like random number generation) and the dynamic consumption of the signing and verification routine. We sign a message of n bits.

Comparison. A detailed comparison of our implementation with related work that also targets the AVR platform[5] and provides 80 to 128-bits of security is given in Table III. Our implementation of RLWEenc-Ia encryption outperforms the software from [Boorghany et al. 2014] by a factor of 3.8 and results from [Boorghany and Jalili 2014] by a factor of 6.3 in terms of cycle counts. Decryption is 6.4 times and 11.5 times faster, respectively. For comparison, we also present the cycle counts for the previous versions of this work. We highlight that these implementations were not protected against timing side-channels and timing-constant implementations are usually expensive to achieve (especially for the Gaussian sampling operation). Due to our optimizations presented in this paper, we are still able to achieve similar or even better performance numbers.

A comparison between our implementation of BLISS-I and the implementations of [Boorghany et al. 2014] and [Boorghany and Jalili 2014] is difficult since the authors implemented the signature as authentication protocol. Therefore they only provide the runtime of a complete protocol run that corresponds to one signing operation and one verification in our results, but without the expensive hashing as the sparse polynomial $\mathbf{c}$ is not obtained from a random oracle but randomly generated by the other protocol party. However, our implementations of $\text{BLISS}_{\text{SIGN}}$ and $\text{BLISS}_{\text{VER}}$ still require less cycles than the implementation of BLISS-I. As our implementation of BLISS-I needs 18,401 bytes of flash memory, it is also smaller than the implementation of [Boorghany and Jalili 2014] that requires 66.5 kB of flash memory and the implementation of [Boorghany et al. 2014] that needs 25.1 kB of flash memory.

[5]While the ATxmega128 and ATxmega64 compared to the ATmega64 differ in their operation frequency and some architectural differences cycle counts are mostly comparable.

ACM Transactions on Embedded Computing Systems, Vol. ?, No. ?, Article ?, Publication date: February 2016.

Table III: Comparison of our implementations with related work. Cycle counts marked with (*) indicate the runtime of BLISS-I as authentication protocol instead of signature scheme. By AX128 we identify the ATxmega128A1 clocked with 32 MHz, by AX64 the ATxmega64A3 clocked with 32 MHz, by AT64 the ATmega64 clocked with 8 MHz, by AT128 the ATmega128 clocked with 8 MHz, and by AT2560 the ATmega2560 clocked with 16 MHz. Implementations marked with (†) are resistant against timing attacks. This work achieves similar results, even though our implementation is timing-constant, since the implementation of [Liu et al. 2015b] lacks the NTT optimizations discussed in Section 3 and the implementations of [Pöppelmann et al. 2015] use the relatively slow $\text{Sampler}_{\text{CDT-high}}$ sampler.

Scheme	Device	MHz	Operation	Cycles		OP/s	
RLWEenc-Ia ($n = 256$)† **(this work)**	AX128	32	Enc/Dec	796,872	215,031	40.16	148.82
RLWEenc-IIa ($n = 512$)† **(this work)**	AX128	32	Enc/Dec	1,975,806	553,536	16.20	57.81
RLWEenc-Ia ($n = 256$)[Liu et al. 2015b]	AX128	32	Enc/Dec	671,628	275,646	47.64	116.09
RLWEenc-IIa ($n = 512$)[Liu et al. 2015b]	AX128	32	Enc/Dec	2,617,459	686,367	12.23	46.62
RLWEenc-Ia ($n = 256$)[Pöppelmann et al. 2015]	AX128	32	Enc/Dec	874,347	215,863	36.60	148.24
RLWEenc-IIa ($n = 512$)[Pöppelmann et al. 2015]	AX128	32	Enc/Dec	2,196,945	600,351	14.57	53.30
RLWEenc-Ia ($n = 256$)[Boorghany et al. 2014]	AT64	8	Enc/Dec	3,042,675	1,368,969	2.63	5.84
RLWEenc-Ia ($n = 256$)[Boorghany and Jalili 2014]	AX64	32	Enc/Dec	5,024,000	2,464,000	6.37	12.98
BLISS-I, **(this work)**	AX128	32	Sign/Ver	10,156,247	2,760,244	3.15	11.59
BLISS-I[Pöppelmann et al. 2015]	AX128	32	Sign/Ver	10,537,981	2,814,118	3.04	11.37
BLISS-I (Bernoulli) [Boorghany et al. 2014]	AT64	8	Sign+Ver	42,069,682*		0.19	
BLISS-I (CDT) [Boorghany and Jalili 2014]	AX64	32	Sign+Ver	19,328,000*		1.65	
$\text{NTT}^{CT,\psi}_{no\rightarrow bo}$ ($n = 256$)† **(this work)**	AX128	32	NTT	194,145		164.83	
$\text{NTT}^{CT}_{bo\rightarrow no}$ ($n = 256$) [Liu et al. 2015b]	AX128	32	NTT	193,731		165.18	
$\text{NTT}^{CT}_{bo\rightarrow no}$ ($n = 256$) [Pöppelmann et al. 2015]	AX128	32	NTT	198,491		161.21	
$\text{NTT}^{CT}_{bo\rightarrow no}$ ($n = 256$) [Boorghany et al. 2014]	AT64	8	NTT	754,668		10.60	
$\text{NTT}^{CT}_{bo\rightarrow no}$ ($n = 256$) [Boorghany and Jalili 2014]	AX64	32	NTT	1,216,000		26.32	
$\text{NTT}^{CT,\psi}_{no\rightarrow bo}$ ($n = 512$)† **(this work)**	AX128	32	NTT	516,971		61.90	
$\text{NTT}^{CT}_{bo\rightarrow no}$ ($n = 512$) [Liu et al. 2015b]	AX128	32	NTT	441,572		72.46	
$\text{NTT}^{CT}_{bo\rightarrow no}$ ($n = 512$) [Pöppelmann et al. 2015]	AX128	32	NTT	521,872		61.31	
$\text{NTT}^{CT}_{bo\rightarrow no}$ ($n = 512$) [Boorghany et al. 2014]	AT64	8	NTT	2,207,787		3.62	
$\text{NTT}^{CT}_{bo\rightarrow no}$ ($n = 512$) [Boorghany and Jalili 2014]	AX64	32	NTT	2,752,000		11.63	
QC-MDPC [Heyse et al. 2013]	AX128	32	Enc/Dec	26,767,463	86,874,388	1.20	0.36
Ed25519† [Hutter and Schwabe 2013]	AT2560	16	Sign/Ver	23,211,611	32,619,197	0.67	0.49
RSA-1024 [Gura et al. 2004]	AT128	8	Enc/Dec	3,440,000	87,920,000	2.33	0.09
RSA-1024† [Liu et al. 2010]	AT128	8	priv. key	75,680,000		0.11	
Curve25519† [Düll et al. pear]	AT2560	16	Point mul.	13,900,397		1.15	
ECC-ecp160r1 [Gura et al. 2004]	AT128	8	Point mul.	6,480,000		1.23	

Table IV: Comparison of different samplers used in the RLWEenc scheme for the implementation of Sampler. Cycle counts are given for sampling a complete polynomial of $n = 256$ (resp. 512) coefficients. Implementations marked with (†) are resistant against timing attacks.

	RLWEenc-Ia ($n = 256$, $\sigma = 4.51$)		RLWEenc-IIa ($n = 512$, $\sigma = 4.86$)	
Sampler	Cycles	Table	Cycles	Table
$\text{Sampler}_{\text{CDT-low}}{}^\dagger$ **(this work)**	53,023	1.3 kbytes	105,153	2.0 kbytes
$\text{Sampler}_{\text{Bio-low}}{}^\dagger$ **(this work)**	139,470	0 kbytes	293,838	0 kbytes
$\text{Sampler}_{\text{KY-high}}$[Liu et al. 2015b]	26,763	1.2 kbytes	255,218	1.4 kbytes
$\text{Sampler}_{\text{CDT-high}}$[Pöppelmann et al. 2015]	84,001	0.6 kbytes	170,861	0.7 kbytes

Compared with different classes of schemes, like QC-MDPC McEliece, RSA, and ECC, or implementations are at least one order of magnitude faster. A comparison with NTRU implementations is currently not easily possible due to lack of published results for the AVR platform[6]. We also refer to [de Clercq et al. 2014] for an implementation of RLWEenc-Ia and RLWEenc-IIa on an ARM Cortex-M4 (32-bit, 168 MHz). However, as the Cortex-M4 is a 32-bit processor a comparison across architectures with different bit-widths is naturally hard.

Comparison of samplers. Table IV shows a comparison of the different sampling approaches for RLWEenc investigated in this work. Important to note is that only $\text{Sampler}_{\text{Bio-low}}$ and $\text{Sampler}_{\text{CDT-low}}$ run in constant time. At the same time, these samplers provide less precision than $\text{Sampler}_{\text{KY-high}}$ and $\text{Sampler}_{\text{CDT-high}}$. Among the high-precision samplers, $\text{Sampler}_{\text{KY-high}}$ is 3.1 times faster for $\sigma = 4.51$. But on the other hand $\text{Sampler}_{\text{CDT-high}}$ needs only half half as much memory. Guide tables could be used to trade memory space for cycle counts for $\text{Sampler}_{\text{CDT-high}}$. Since $\text{Sampler}_{\text{CDT-low}}$ already features guide tables, it needs more memory, but is still faster than $\text{Sampler}_{\text{CDT-high}}$. Indeed, $\text{Sampler}_{\text{Bio-low}}$ does not need any precomputed tables, but the cycle counts are 2.6/2.8 times higher than for $\text{Sampler}_{\text{CDT-low}}$. For constant-time implementations we therefore recommend to use $\text{Sampler}_{\text{CDT-low}}$.

REFERENCES

Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. 2016a. Post-quantum Key Exchange - A New Hope. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, Thorsten Holz and Stefan Savage (Eds.). USENIX Association, 327–343. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/alkim

Erdem Alkim, Philipp Jakubeit, and Peter Schwabe. 2016b. NewHope on ARM Cortex-M. In *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings (Lecture Notes in Computer Science)*, Claude Carlet, M. Anwar Hasan, and Vishal Saraswat (Eds.), Vol. 10076. Springer, 332–349. DOI:http://dx.doi.org/10.1007/978-3-319-49445-6_19

Shi Bai and Steven D. Galbraith. 2014. An Improved Compression Technique for Signatures Based on Learning with Errors. In *Topics in Cryptology - CT-RSA 2014 - The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings (LNCS)*, Josh Benaloh (Ed.), Vol. 8366. Springer, 28–47. DOI:http://dx.doi.org/10.1007/978-3-319-04852-9_2

Josep Balasch, Baris Ege, Thomas Eisenbarth, Benoît Gérard, Zheng Gong, Tim Güneysu, Stefan Heyse, Stéphanie Kerckhof, François Koeune, Thomas Plos, Thomas Pöppelmann, Francesco Regazzoni,

[6]One exception is a Master thesis by Monteverde [Monteverde 2008], but the implemented NTRU251:3 variant is not secure anymore according to recent recommendations in [Hirschhorn et al. 2009].

François-Xavier Standaert, Gilles Van Assche, Ronny Van Keer, Loïc van Oldeneel tot Oldenzeel, and Ingo von Maurich. 2012. Compact Implementation and Performance Evaluation of Hash Functions in ATtiny Devices. In *Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers (LNCS)*, Stefan Mangard (Ed.), Vol. 7771. Springer, 158–172. DOI:http://dx.doi.org/10.1007/978-3-642-37288-9_11

Paul Barrett. 1987. Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor. In *Proceedings on Advances in cryptology—CRYPTO '86*. Springer-Verlag, London, UK, 311–323. http://dl.acm.org/citation.cfm?id=36664.36688

Lejla Batina and Matthew Robshaw (Eds.). 2014. *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*. LNCS, Vol. 8731. Springer. DOI:http://dx.doi.org/10.1007/978-3-662-44709-3

Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche, and Ronny Van Keer. 2012. KECCAK implementation overview. (2012). version 3.2, see http://keccak.noekeon.org/Keccak-implementation-3.2.pdf.

Richard E. Blahut. 2010. *Fast Algorithms for Signal Processing*. Cambridge University Press. http://amazon.com/o/ASIN/0521190495/

Ahmad Boorghany and Rasool Jalili. 2014. Implementation and Comparison of Lattice-based Identification Protocols on Smart Cards and Microcontrollers. *IACR Cryptology ePrint Archive* 2014 (2014), 78. http://eprint.iacr.org/2014/078 preliminary version of [Boorghany et al. 2014].

Ahmad Boorghany, Siavash Bayat Sarmadi, and Rasool Jalili. 2014. On Constrained Implementation of Lattice-based Cryptographic Primitives and Schemes on Smart Cards. *IACR Cryptology ePrint Archive* 2014 (2014), 514. http://eprint.iacr.org/2014/514 successive version of [Boorghany and Jalili 2014].

Joppe W. Bos, Craig Costello, Michael Naehrig, and Douglas Stebila. 2015. Post-Quantum Key Exchange for the TLS Protocol from the Ring Learning with Errors Problem. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*. 553–570. DOI:http://dx.doi.org/10.1109/SP.2015.40

Daniel Cabarcas, Patrick Weiden, and Johannes Buchmann. 2014. On the Efficiency of Provably Secure NTRU, See Mosca [2014], 22–39. DOI:http://dx.doi.org/10.1007/978-3-319-11659-4_2

Donald Donglong Chen, Nele Mentens, Frederik Vercauteren, Sujoy Sinha Roy, Ray C. C. Cheung, Derek Pao, and Ingrid Verbauwhede. 2015. High-Speed Polynomial Multiplication Architecture for Ring-LWE and SHE Cryptosystems. *IEEE Trans. on Circuits and Systems* 62-I, 1 (2015), 157–166. DOI:http://dx.doi.org/10.1109/TCSI.2014.2350431

Eleanor Chu and Alan George. 2000. *Inside the FFT Black Box Serial and Parallel Fast Fourier Transform Algorithms*. CRC Press, Boca Raton, FL, USA.

James W. Cooley and John W. Tukey. 1965. An algorithm for the machine calculation of complex Fourier series. *Math. Comp.* 19 (1965), 297–301.

Richard Crandall and Barry Fagin. 1994. Discrete weighted transforms and large-integer arithmetic. *Math. Comp.* 62, 205 (1994), 305–324.

Richard Crandall and Carl Pomerance. 2001. *Prime Numbers: A Computational Perspective*. Springer-Verlag, Berlin, Germany / Heidelberg, Germany / London, UK / etc. xv + 545 pages. DOI:http://dx.doi.org/10.1007/978-1-4684-9316-0

Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. 2014. Efficient Software Implementation of Ring-LWE Encryption. *IACR Cryptology ePrint Archive* 2014 (2014), 725. http://eprint.iacr.org/2014/725

Luc Devroye. 1986. *Non-Uniform Random Variate Generation*. Springer-Verlag. http://luc.devroye.org/rnbookindex.html.

Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. 2013a. Lattice Signatures and Bimodal Gaussians. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I (LNCS)*, Ran Canetti and Juan A. Garay (Eds.), Vol. 8042. Springer, 40–56. DOI:http://dx.doi.org/10.1007/978-3-642-40041-4_3 conference version of [Ducas et al. 2013b].

Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. 2013b. Lattice Signatures and Bimodal Gaussians. *IACR Cryptology ePrint Archive* 2013 (2013), 383. http://eprint.iacr.org/2013/383 full version of [Ducas et al. 2013b].

Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. 2014. Efficient Identity-Based Encryption over NTRU Lattices. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II (LNCS)*, Palash Sarkar and Tetsu Iwata (Eds.), Vol. 8874. Springer, 22–41. DOI:http://dx.doi.org/10.1007/978-3-662-45608-8_2

Michael Düll, Björn Haase, Gesine Hinterwälder, Michael Hutter, Christof Paar, Ana Helena Sánchez, and Peter Schwabe. to appear. High-speed Curve25519 on 8-bit, 16-bit and 32-bit microcontrollers. *Design, Codes and Cryptography* (to appear). Document ID: bd41e6b96370dea91c5858f1b809b581, http://cryptojedi.org/papers/#mu25519.

W. Morven Gentleman and G. Sande. 1966. Fast Fourier Transforms: for fun and profit. In *American Federation of Information Processing Societies: Proceedings of the AFIPS '66 Fall Joint Computer Conference, November 7-10, 1966, San Francisco, California, USA (AFIPS Conference Proceedings)*, Vol. 29. AFIPS / ACM / Spartan Books, Washington D.C., 563–578. DOI:http://dx.doi.org/10.1145/1464291.1464352

Norman Göttert, Thomas Feller, Michael Schneider, Johannes Buchmann, and Sorin A. Huss. 2012. On the Design of Hardware Building Blocks for Modern Lattice-Based Encryption Schemes, See Prouff and Schaumont [2012], 512–529. DOI:http://dx.doi.org/10.1007/978-3-642-33027-8_30

Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. 2012. Practical Lattice-Based Cryptography: A Signature Scheme for Embedded Systems, See Prouff and Schaumont [2012], 530–547. DOI:http://dx.doi.org/10.1007/978-3-642-33027-8_31

Tim Güneysu, Tobias Oder, Thomas Pöppelmann, and Peter Schwabe. 2013. Software Speed Records for Lattice-Based Signatures. In *Post-Quantum Cryptography - 5th International Workshop, PQCrypto 2013, Limoges, France, June 4-7, 2013. Proceedings (LNCS)*, Philippe Gaborit (Ed.), Vol. 7932. Springer, 67–82. DOI:http://dx.doi.org/10.1007/978-3-642-38616-9_5

Nils Gura, Arun Patel, Arvinderpal Wander, Hans Eberle, and Sheueling Chang Shantz. 2004. Comparing Elliptic Curve Cryptography and RSA on 8-bit CPUs. In *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings (LNCS)*, Marc Joye and Jean-Jacques Quisquater (Eds.), Vol. 3156. Springer, 119–132. DOI:http://dx.doi.org/10.1007/978-3-540-28632-5_9

Stefan Heyse, Ingo von Maurich, and Tim Güneysu. 2013. Smaller Keys for Code-Based Cryptography: QC-MDPC McEliece Implementations on Embedded Devices. In *Cryptographic Hardware and Embedded Systems - CHES 2013 - 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings (LNCS)*, Guido Bertoni and Jean-Sébastien Coron (Eds.), Vol. 8086. Springer, 273–292. DOI:http://dx.doi.org/10.1007/978-3-642-40349-1_16

Philip S. Hirschhorn, Jeffrey Hoffstein, Nick Howgrave-Graham, and William Whyte. 2009. Choosing NTRUEncrypt Parameters in Light of Combined Lattice Reduction and MITM Approaches. In *Applied Cryptography and Network Security, 7th International Conference, ACNS 2009, Paris-Rocquencourt, France, June 2-5, 2009. Proceedings (LNCS)*, Michel Abdalla, David Pointcheval, Pierre-Alain Fouque, and Damien Vergnaud (Eds.), Vol. 5536. 437–455. DOI:http://dx.doi.org/10.1007/978-3-642-01957-9_27

Jeffrey Hoffstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, and William Whyte. 2014. Practical Signatures from the Partial Fourier Recovery Problem. In *Applied Cryptography and Network Security - 12th International Conference, ACNS 2014, Lausanne, Switzerland, June 10-13, 2014. Proceedings (LNCS)*, Ioana Boureanu, Philippe Owesarski, and Serge Vaudenay (Eds.), Vol. 8479. Springer, 476–493. DOI:http://dx.doi.org/10.1007/978-3-319-07536-5_28

Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. 1998. NTRU: A Ring-Based Public Key Cryptosystem. In *Algorithmic Number Theory, Third International Symposium, ANTS-III, Portland, Oregon, USA, June 21-25, 1998, Proceedings (LNCS)*, Joe Buhler (Ed.), Vol. 1423. Springer, 267–288. DOI:http://dx.doi.org/10.1007/BFb0054868

Michael Hutter and Peter Schwabe. 2013. NaCl on 8-Bit AVR Microcontrollers. In *Progress in Cryptology - AFRICACRYPT 2013, 6th International Conference on Cryptology in Africa, Cairo, Egypt, June 22-24, 2013. Proceedings (LNCS)*, Amr Youssef, Abderrahmane Nitaj, and Aboul Ella Hassanien (Eds.), Vol. 7918. Springer, 156–172. DOI:http://dx.doi.org/10.1007/978-3-642-38553-7_9

D. E. Knuth and A. C. Yao. 1976. The complexity of nonuniform random number generation. In *Algorithms and Complexity: New Directions and Recent Results*. 357–428.

Richard Lindner and Chris Peikert. 2011. Better Key Sizes (and Attacks) for LWE-Based Encryption. In *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings (LNCS)*, Aggelos Kiayias (Ed.), Vol. 6558. Springer, 319–339. DOI:http://dx.doi.org/10.1007/978-3-642-19074-2_21

Mingjie Liu and Phong Q. Nguyen. 2013. Solving BDD by Enumeration: An Update. In *Topics in Cryptology - CT-RSA 2013 - The Cryptographers' Track at the RSA Conference 2013, San Francisco,CA, USA, February 25-March 1, 2013. Proceedings (LNCS)*, Ed Dawson (Ed.), Vol. 7779. Springer, 293–309. DOI:http://dx.doi.org/10.1007/978-3-642-36095-4_19

Zhe Liu, Johann Großschädl, and Ilya Kizhvatov. 2010. Efficient and Side-Channel Resistant RSA Implementation for 8-bit AVR Microcontrollers. In *Proceedings of the 1st International Workshop on the Security of the Internet of Things (SECIOT 2010)*. IEEE Computer Society Press.

Zhe Liu, Johann Großschädl, and Duncan S.Wong. 2013. Low-Weight Primes for Lightweight Elliptic Curve Cryptography on 8-bit Processors. In *The 9th China International Conference on Information Security and Cryptology — INSCRYPT 2013 (Lecture Notes in Computer Science)*, Dongdai Lin, Shouhai Xu, and Moti Yung (Eds.). Springer Verlag.

Zhe Liu, Hwajeong Seo, Sujoy Sinha Roy, Johann Großschädl, Howon Kim, and Ingrid Verbauwhede. 2015a. Efficient Ring-LWE Encryption on 8-bit AVR Processors. *IACR Cryptology ePrint Archive* 2015 (2015), 410. http://eprint.iacr.org/2015/410 to appear in CHES'15.

Zhe Liu, Hwajeong Seo, Sujoy Sinha Roy, Johann Großschädl, Howon Kim, and Ingrid Verbauwhede. 2015b. Efficient Ring-LWE Encryption on 8-Bit AVR Processors. In *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings (LNCS)*, Tim Güneysu and Helena Handschuh (Eds.), Vol. 9293. Springer, 663–682. DOI:http://dx.doi.org/10.1007/978-3-662-48324-4_33

Patrick Longa and Michael Naehrig. 2016. Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography. In *Cryptology and Network Security - 15th International Conference, CANS 2016, Milan, Italy, November 14-16, 2016, Proceedings (Lecture Notes in Computer Science)*, Sara Foresti and Giuseppe Persiano (Eds.), Vol. 10052. 124–139. DOI:http://dx.doi.org/10.1007/978-3-319-48965-0_8

Vadim Lyubashevsky. 2012. Lattice Signatures without Trapdoors. In *Advances in Cryptology - EUROCRYPT 2012 - 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings (LNCS)*, David Pointcheval and Thomas Johansson (Eds.), Vol. 7237. Springer, 738–755. DOI:http://dx.doi.org/10.1007/978-3-642-29011-4_43

Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010a. On Ideal Lattices and Learning with Errors over Rings. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings (LNCS)*, Henri Gilbert (Ed.), Vol. 6110. Springer, 1–23. DOI:http://dx.doi.org/10.1007/978-3-642-13190-5_1

Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010b. On Ideal Lattices and Learning With Errors Over Rings. (2010). Presentation of [Lyubashevsky et al. 2010a] given by Chris Peikert at Eurocrypt'10. See http://www.cc.gatech.edu/~cpeikert/pubs/slides-ideal-lwe.pdf.

Carlos Aguilar Melchor, Xavier Boyen, Jean-Christophe Deneuville, and Philippe Gaborit. 2014. Sealing the Leak on Classical NTRU Signatures, See Mosca [2014], 1–21. DOI:http://dx.doi.org/10.1007/978-3-319-11659-4_1

Mariano Monteverde. 2008. *NTRU software implementation for constrained devices*. Master's thesis. Katholieke Universiteit Leuven.

Michele Mosca (Ed.). 2014. *Post-Quantum Cryptography - 6th International Workshop, PQCrypto 2014, Waterloo, ON, Canada, October 1-3, 2014. Proceedings*. LNCS, Vol. 8772. Springer. DOI:http://dx.doi.org/10.1007/978-3-319-11659-4

NIST. 2015. Workshop on Cybersecurity in a Post-Quantum World. (2015). http://www.nist.gov/itl/csd/ct/post-quantum-crypto-workshop-2015.cfm.

NSA. 2015. NSA Suite B Cryptography. (2015). https://www.nsa.gov/ia/programs/suiteb_cryptography/, Updated on August 19, 2015.

Henri J. Nussbaumer. 1982. *Fast Fourier Transform and Convolution Algorithms*. Springer Series in Information Sciences, Vol. 2. Springer, Berlin, DE.

Tobias Oder, Thomas Pöppelmann, and Tim Güneysu. 2014. Beyond ECDSA and RSA: Lattice-based Digital Signatures on Constrained Devices. In *The 51st Annual Design Automation Conference 2014, DAC '14, San Francisco, CA, USA, June 1-5, 2014*. ACM, 1–6. DOI:http://dx.doi.org/10.1145/2593069.2593098

Chris Peikert. 2010. An Efficient and Parallel Gaussian Sampler for Lattices. In *CRYPTO (LNCS)*, Tal Rabin (Ed.), Vol. 6223. Springer, 80–97.

Chris Peikert. 2014. Lattice Cryptography for the Internet, See Mosca [2014], 197–219. DOI:http://dx.doi.org/10.1007/978-3-319-11659-4_12

Thomas Pöppelmann, Léo Ducas, and Tim Güneysu. 2014. Enhanced Lattice-Based Signatures on Reconfigurable Hardware, See Batina and Robshaw [2014], 353–370. DOI:http://dx.doi.org/10.1007/978-3-662-44709-3_20

Thomas Pöppelmann and Tim Güneysu. 2013. Towards Practical Lattice-Based Public-Key Encryption on Reconfigurable Hardware. In *Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers (LNCS)*, Tanja Lange, Kristin E. Lauter, and Petr Lisonek (Eds.), Vol. 8282. Springer, 68–85. DOI:http://dx.doi.org/10.1007/978-3-662-43414-7_4

Thomas Pöppelmann and Tim Güneysu. 2014. Area optimization of lightweight lattice-based encryption on reconfigurable hardware. In *IEEE International Symposium on Circuits and*

Systemss, ISCAS 2014, Melbourne, Victoria, Australia, June 1-5, 2014. IEEE, 2796–2799. DOI:http://dx.doi.org/10.1109/ISCAS.2014.6865754

Thomas Pöppelmann, Tobias Oder, and Tim Güneysu. 2015. High-Performance Ideal Lattice-Based Cryptography on 8-Bit ATxmega Microcontrollers. In *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings (LNCS)*, Kristin E. Lauter and Francisco Rodríguez-Henríquez (Eds.), Vol. 9230. Springer, 346–365. DOI:http://dx.doi.org/10.1007/978-3-319-22174-8_19

Emmanuel Prouff and Patrick Schaumont (Eds.). 2012. *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*. LNCS, Vol. 7428. Springer.

Steven Rich and Barton Gellman. 2013. NSA seeks quantum computer that could crack most codes. *The Washington Post* (2013). http://wapo.st/19DycJT.

Sujoy Sinha Roy, Frederik Vercauteren, Nele Mentens, Donald Donglong Chen, and Ingrid Verbauwhede. 2014. Compact Ring-LWE Cryptoprocessor, See Batina and Robshaw [2014], 371–391. DOI:http://dx.doi.org/10.1007/978-3-662-44709-3_21

Markku-Juhani O. Saarinen. 2015. Gaussian Sampling Precision and Information Leakage in Lattice Cryptography. *IACR Cryptology ePrint Archive* 2015 (2015), 953. http://eprint.iacr.org/2015/953

P.W. Shor. 1994. Algorithms for quantum computation: discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*. IEEE, 124–134.

Damien Stehlé and Ron Steinfeld. 2011. Making NTRU as Secure as Worst-Case Problems over Ideal Lattices. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings (LNCS)*, Kenneth G. Paterson (Ed.), Vol. 6632. Springer, 27–47. DOI:http://dx.doi.org/10.1007/978-3-642-20465-4_4

Franz Winkler. 1996. *Polynomial Algorithms in Computer Algebra (Texts and Monographs in Symbolic Computation)* (1 ed.). Springer. http://amazon.com/o/ASIN/3211827595/

A. APPENDIX

A.1. NTT Algorithms

A description of $\text{NTT}^{CT,\psi}_{no\rightarrow bo}$ is given in Algorithm 8 and $\text{INTT}^{GS,\psi^{-1}}_{bo\rightarrow no}$ is described in Algorithm 9.

Algorithm 8 Optimized CT Forward NTT

Precondition: Store n powers of ψ in bit-reversed order in psi^*

function $\text{NTT}^{CT,\psi}_{no\rightarrow bo}$(a)
  $m \leftarrow 1$
  $k \leftarrow n/2$
  while $m < n$ **do**
    for $i = 0$ **to** $m - 1$ **do**
      $jFirst \leftarrow 2 \cdot i \cdot k$
      $jLast \leftarrow jFirst + k - 1$
      $\psi_i \leftarrow psi^*[m + i]$
      for $j = jFirst$ **to** $jLast$ **do**
        $l \leftarrow j + k$
        $t \leftarrow \mathbf{a}[j]$
        $u \leftarrow \mathbf{a}[l] \cdot \psi_i$
        $\mathbf{a}[j] \leftarrow t + u \mod q$
        $\mathbf{a}[l] \leftarrow t - u \mod q$
      end for
    end for
    $m \leftarrow m \cdot 2$
    $k \leftarrow k/2$
  end while
  Return a
end function

Algorithm 9 Optimized GS Inverse NTT

Precondition: Store n powers of ψ^{-1} in bit-reversed order in $invpsi^*$

function $\text{INTT}^{GS,\psi^{-1}}_{bo\rightarrow no}$ (a)
  $m \leftarrow n/2$
  $k \leftarrow 1$
  while $m \geq 1$ **do**
    for $i = 0$ **to** $m - 1$ **do**
      $jFirst \leftarrow 2 \cdot i \cdot k$
      $jLast \leftarrow jFirst + k - 1$
      $\psi_i \leftarrow invpsi^*[m + i]$
      for $j = jFirst$ **to** $jLast$ **do**
        $l \leftarrow j + k$
        $t \leftarrow \mathbf{a}[j]$
        $u \leftarrow \mathbf{a}[l]$
        $\mathbf{a}[j] \leftarrow t + u \mod q$
        $\mathbf{a}[l] \leftarrow (t - u) \cdot \psi_i \mod q$
      end for
    end for
    $m \leftarrow m/2$
    $k \leftarrow k \cdot 2$
  end while
  Return a
end function