On Constrained Implementation of Lattice-based Cryptographic Primitives and Schemes on Smart Cards [⋆]

Ahmad Boorghany
boorghany@ce.sharif.edu

Siavash Bayat Sarmadi
sbayat@sharif.edu

Rasool Jalili
jalili@sharif.edu

Data and Network Security Lab, Computer Engineering Department,
Sharif University of Technology, Tehran, Iran

July 1, 2014

Abstract. Most lattice-based cryptographic schemes with a security proof suffer from large key sizes and heavy computations. This is also true for the simpler case of authentication protocols which are used on smart cards, as a very-constrained computing environment. Recent progress on ideal lattices has significantly improved the efficiency, and made it possible to implement practical lattice-based cryptography on constrained devices. However, to the best of our knowledge, no previous attempts were made to implement lattice-based schemes on smart cards. In this paper, we provide the results of our implementation of several state-of-the-art lattice-based authentication protocols on smart cards and a microcontroller widely used in smart cards. Our results show that only a few of the proposed lattice-based authentication protocols can be implemented using limited resources of such constrained devices, however, cutting-edge ones are suitably-efficient to be used practically on smart cards. Moreover, we have implemented fast Fourier transform (FFT) and discrete Gaussian sampling with different typical parameters sets, as well as versatile lattice-based public-key encryptions. These results have noticeable points which help to design or optimize lattice-based schemes for constrained devices.

Keywords: Authentication protocol, constrained device, constrained implementation, lattice-based cryptography, post-quantum cryptography

1 Introduction

Since the seminal work of Ajtai [Ajt96], lattice-based cryptography has attracted much attention from cryptography community. Provably-secure lattice-based schemes have a remarkable feature that their securities are proved assuming a basic lattice problem is hard in the worst-case. This type of assumption is more reliable than average-case hardness assumptions in number-theoretic schemes supported by a security proof. Moreover, the lattice-based cryptography is one of the main candidates for post-quantum cryptography. No efficient quantum algorithm has been found yet to break lattice-based schemes. In contrast, widely used schemes in practice, such as RSA or elliptic curve (EC) based constructions

[⋆] A preliminary version of this paper has been posted on IACR Cryptology ePrint Archive, Report 2014/078. Available at https://eprint.iacr.org/2014/078.pdf.

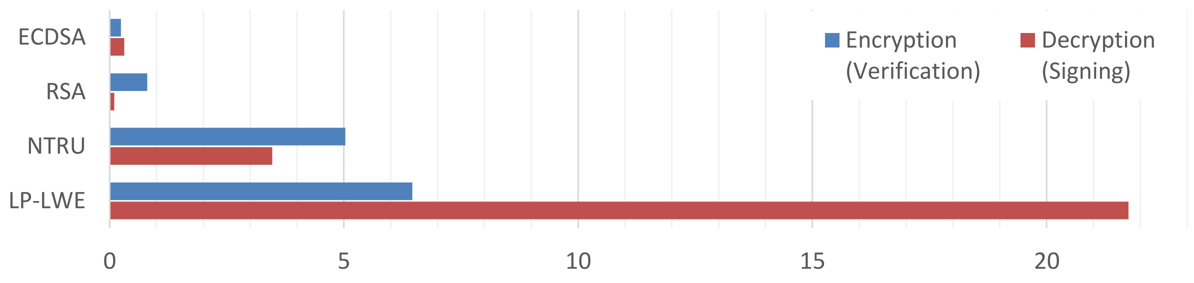

Fig. 1. Encryptions and decryptions per second for lattice-based (NTRU and LP-LWE) and non-lattice-based (ECDSA and RSA) schemes, implemented on the ARM7TDMI smart card; For ECDSA, the speed of verification and signing is reported, instead.

which are based on the hardness of factoring or discrete logarithm, will be broken upon appearance of large-scale quantum computers [Sho94].

Since the appearance of lattice-based cryptography, it is known that lattice-based schemes (e.g., encryptions) are *asymptotically* more efficient than their competitors, RSA and EC-based ones. However, traditional provably-secure lattice-based schemes suffer from heavy computation and large key sizes in practice. A major event in the development of lattice-based cryptography is the introduction of ideal lattices. An ideal lattice has extra algebraic structure which reduces the key size and computation time to a sub-quadratic order. Moreover, cryptographic schemes based on ideal lattices enjoy a security proof assuming worst-case hardness of basic problems on ideal lattices. It is in contrast to NTRU encryption [HPS98], which is a well-known and efficient lattice-based scheme. The security of NTRU is not proven and is preserved heuristically. Furthermore, NTRU is patented by its designers till 2020 [HS06].

Recent noticeable improvements on the efficiency of provably-secure lattice-based constructions have made it possible to port these schemes to constrained devices such as smart cards and microcontrollers. In this paper, we have implemented a number of important lattice-based cryptographic primitives, as well as a number of most efficient lattice-based encryptions and authentication protocols on a typical smart card and a microcontroller widely used in other smart cards. This smart card is a proprietary product which contains a 32-bit ARM7TDMI processor, and the microcontroller is from 8-bit AVR MEGA family. Other than smart cards, AVR MEGA microcontrollers have also been used in various low-cost electronic devices. Due to the development of the Internet of Things, these devices are now more likely to require cryptographic schemes. Our results give a measure for the efficiency of lattice-based schemes in such environments.

To the best of our knowledge, it is the first time that a lattice-based cryptographic scheme is implemented on a smart card, and its efficiency and practicality is measured. From lattice-based cryptographic primitives, we have implemented fast Fourier transform (FFT) with different degrees and typical parameter sets, as well as two of the best discrete Gaussian sampling algorithms. Both FFT and discrete Gaussian sampling are of main ingredients in provably-secure lattice-based cryptography. In the case of implementing lattice-based encryptions, we have chosen NTRU and a state-of-the-art provably-secure encryption by Lindner and Peikert [LP11] (referred to as LP-LWE); both configured to have a security equivalence to 128-bit symmetric encryptions. We have also implemented 3072-bit RSA and 256-bit ECDSA (using NIST elliptic curve P-256) on the same hardware, in order to compare against traditional encryption schemes. Figure 1 shows the number of encryptions/decryptions per second (correspondingly, verifications/signings per second for ECDSA), performed on the smart card. Note that RSA and ECDSA are executed without the help of smart card's cryptographic coprocessor.

Our performance results on the smart card show that the encryption and decryption of LP-LWE, as a provably-secure and patent-free lattice-based encryption, are 28% and 600% faster than NTRU encryption and decryption, respectively. Moreover, the key generation of NTRU is much slower. Results in Section 5.3 indicate that LP-LWE decryption is also much faster than a simplified CPA-secure version of NTRU. This shows that recent progress on improving efficiency of provably-secure lattice-based cryptography has made it comparable to and even better than the famous NTRU encryption.

We have also implemented four lattice-based authentication protocols on our smart card and microcontroller. Authentication protocols do not include secure key-exchange. Thus, their main application is in physical authentication, e.g., on a personal identification card. We have focused on efficient zero-knowledge-like protocols (technically including zero-knowledge proofs, witness-hiding protocols, and those which transfer messages that are statistically independent of the secret key). To this end, we have converted two lattice-based signature schemes by Güneysu et al. [GLP12] and Ducas et al. [DDLL13] to authentication protocols. These transformations are more efficient than previous original lattice-based authentications. One drawback of constructions is that the resulting authentication protocols will have a proof of security only against a passive attacker. However, due to a technique used in both protocols, called rejection sampling [Lyu09], an active attacker cannot learn anything about the secret key. In addition to the protocols derived from lattice-based signature schemes, we have chosen a recently-proposed authentication protocol from Dousti and Jalili [DJ13] which utilizes a commitment scheme and an encryption algorithm as building blocks. By instantiating this protocol with efficient lattice-based constructions, and performing some optimizations after the integration, we have reached two authentication protocols with competitive efficiency.

1.1 Related Work

Regarding implementation of provably-secure lattice-based schemes on constrained devices, Yao et al. [YHK$^+$11] first proposed a power efficient and compact authentication protocol based on integer LWE problem (not based on ideal lattices) that was designed for an RFID tag. Their protocol consists of only one round and invokes one LWE-based encryption inside the tag. Sinha Roy et al. [SRVV13] proposed an algorithm for high-precision discrete Gaussian sampling and a compact-area FPGA implementation. Very recently, Oder et al. [OPG14] have implemented the signature scheme of [DDLL13] and a few proposed Gaussian sampling algorithms (Bernoulli sampler [DDLL13], Knuth-Yao [SRVV13], and Ziggurat [BCG$^+$13]) on an ARM Cortex-M4F microcontroller. Their implementations are not directly comparable to ours because ARM Cortex-M4F is a relatively powerful microcontroller with a clock of 168 MHz, and 192 KB of RAM. Moreover, Cortex-M4F has instructions for signed and unsigned division which is missing in our target devices. Additionally, computing reminders occurs too many times in lattice-based implementations, and this can influence the performance significantly.

For NTRU encryption, there are more experimental results on constrained devices. The first work was [BCE$^+$01] which implemented NTRU encryption on some hand-held devices containing Motorola and ARM processors. Moreover, they provided an implementation on an FPGA from Xilinx Virtex 1000 family. Kaps [Kap06] introduced NTRU implementation for ultra-low power devices such as RFIDs and wireless sensor networks. Atici et al. [ABGV08] also implemented NTRU on RFIDs and performed power analysis attack on it. Atici et al. [ABF$^+$08] proposed more compact encryption-only and encryption/decryption NTRU implementations for RFID tags and sensor networks. Finally, Monteverde [Mon08] implemented NTRU on two microcontrollers from AVR MEGA family, and Lee et al. [LSCH10] provided the result of NTRU implementation on Tmote Sky hardware, which is widely used in practical wireless sensor networks.

There are many lattice-based authentication protocols consisting of a base zero-knowledge proof with significant soundness error. This soundness error can become negligible by several sequential repetitions [MV03, XT09, SCJD11]. This repetition makes the overall protocol inefficient in terms of computation, communication, and round complexities. Various protocols are also secure when the

Table 1. Overview of lattice-based authentication protocols; Check marks ✓ indicate implemented protocols in this paper. ZK and WH are zero-knowledge and witness hiding protocols, respectively.

Scheme	Rounds	Security	ZK/WH	Orig. Scheme
[MV03]	$3 \times N$	Active	ZK	Authentication
[Lyu08]	3	Concurrent	WH	Authentication
[KTX08]	3	Concurrent	WH	Authentication
[XT09]	$3 \times N$	Active	ZK	Authentication
[Lyu09]	3	Concurrent	WH	Authentication
[CLRS10]	5	Concurrent	WH	Authentication
[SCJD11]	$3 \times N$	Active	ZK	Authentication
	$5 \times N$	Active	ZK	Authentication
[SCL11]	5	Concurrent	WH	Authentication
[Lyu12]	3	Concurrent	WH	Signature
	3	Passive	–	Signature
✓ [GLP12]	3	Passive	–	Signature
✓ [DDLL13]	3	Passive	–	Signature
✓ [DJ13]	5	Active	ZK	Authentication

repetition is done in parallel [KTX08, CLRS10, SCL11], leading to fewer number of rounds, but still suffer from large-size messages and heavy computations. Lyubashevsky [Lyu08] proposed a more efficient authentication protocol based on ideal lattices. In a subsequent work [Lyu09], proposed another protocol with a breakthrough in the efficiency of lattice-based authentication protocols. This scheme has 0.63 completeness error due to a technique called rejection sampling, where the prover may reject to answer verifier's challenges to prevent secret key leakage.

Lyubashevsky [Lyu09] used Fiat-Shamir transformation to convert the proposed authentication scheme to an efficient lattice-based signature. Briefly, Fiat-Shamir transformation [FS87] is a provably-secure technique which replaces verifier in an authentication protocol with a random oracle (practically, a hash function). This oracle (hash function) gets the message to be signed and produces random (pseudo-random) challenges on behalf of the verifier. The transcript of this simulated protocol can be published as a signature. Therefore, everyone who get the message and its signature can verify the correctness of the challenge in the transcript, and then accept the signature if and only if the authentication is verified successfully. Later, Lyubashevsky [Lyu12], Güneysu et al. [GLP12], and Ducas et al. [DDLL13] improved the running time and signature size of [Lyu09].

Recently, in [DJ13], a 5-round zero-knowledge authentication protocol has been proposed which does not need any repetition. Their protocol makes use of a commitment scheme and a trapdoor function as black-boxes. Unfortunately, the most efficient lattice-based trapdoor function [MP12] does not fit into the smart card and microcontroller which we have used in our experiments (both in terms of computation time and the size of keys). However, as noted by the authors, this protocol can be modified slightly to use an encryption instead of trapdoor function. In this protocol, the verifier asks the prover to decrypt a challenge ciphertext, but first convinces her that he already knows the plain-text.

1.2 Orgranization

Notation and a brief mathematical background are provided in Section 2. Implementation details of fast Fourier transform and discrete Gaussian sampling, as well as a simplified CPA-secure version of NTRU are described in Section 3. Section 4 investigates the implemented authentication protocols and specific optimization notes. Section 5 presents the results of the measurements on time, memory, and communication complexity of implemented lattice-based schemes. Finally, Section 6 concludes our results.

2 Preliminaries

2.1 Notation

We define $\mathbb{Z}_q$ to be the set of integers in the interval $[-q/2, q/2)$. $\mathbb{Z}_q[x]$ denotes the set of polynomials with coefficients in $\mathbb{Z}_q$. The polynomial ring $\mathbb{Z}_q[x]/\langle x^n+1\rangle$ is specified by $\mathcal{R}_q$, where n is a power of 2 and $q \equiv 1 \pmod{2n}$. $\mathcal{R}_q$ contains all polynomials of degree less than n with coefficients in $\mathbb{Z}_q$, and its ring operations are polynomial addition and multiplication modulo q and x^n+1. The ring $\mathbb{Z}_q[x]/\langle x^N-1\rangle$ is denoted by $\mathcal{R}'_q$. Polynomials in $\mathcal{R}_q$ and vectors in $\mathbb{Z}_q^n$ are directly mapped to each other, thus, they are used interchangeably through the text. Norm 2 and infinity norm of a vector x is denoted by $\|x\| = \sqrt{\sum x_i^2}$ and $\|x\|_\infty = \max\{x_i\}$, respectively, where x_i is the i-th component of x. Operators $\ll$ and $\gg$ are bit-wise left and right shifts.

2.2 Lattices

A lattice Λ is defined as linear combinations of vectors $\boldsymbol{v_1}, ..., \boldsymbol{v_n} \in \mathbb{Z}^m$ with integer coefficients. These vectors are linearly independent and called lattice base vectors. Putting base vectors as columns in a matrix $\mathbf{B} \in \mathbb{Z}^{m\times n}$, a lattice generated by $\mathbf{B}$ is denoted by $\mathcal{L}(\mathbf{B}) = \{\mathbf{B}\boldsymbol{x} \mid \boldsymbol{x} \in \mathbb{Z}^n\}$. A fundamental hard lattice problem is the shortest vector problem (SVP). In SVP, given a base matrix $\mathbf{B}$, it is required to find the shortest non-zero $\boldsymbol{u} \in \mathcal{L}(\mathbf{B})$. It is believed that there is no polynomial-time algorithm even to approximate SVP to within a polynomial factor of n.

Using general lattices to construct cryptographic functions usually ends to inefficient schemes with high computation and storage complexities. This is because operations on lattices deal with quadratic-size matrices. To overcome this issue, special lattices with extra algebraic structure are developed. Consider the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n+1\rangle$. Each polynomial in $\mathcal{R}_q$ has n coefficients in $\mathbb{Z}_q$, meaning that there is a bijection between $\mathcal{R}_q$ and $\mathbb{Z}_q^n$. It can be shown that an ideal in $\mathcal{R}_q$ is mapped to a lattice in $\mathbb{Z}_q^n$, which is called an ideal lattice.

Ring-based short integer solution (Ring-SIS) and ring-based learning with errors (Ring-LWE) are two problems based on ideal lattices, where are frequently used as the underlying problem in building efficient cryptographic constructions. When configured with suitable parameters, it is proved that these problems are hard-on-average, assuming that approximate SVP is hard in the worst-case on ideal lattices [LM06, PR06, LPR10].

*Problem 1 (*Ring-SIS$_{\mathcal{D},m,n,q,\beta}$*).* Given m random polynomials $\boldsymbol{a} \in \mathcal{R}_q^m$ from the distribution $\mathcal{D}$, and a threshold $\beta \in \mathbb{Q}$: find m short polynomials $\boldsymbol{x} \in \mathcal{R}_q^m$ where $\boldsymbol{x} \neq 0$, $\boldsymbol{a} \cdot \boldsymbol{x} = 0$ (in $\mathcal{R}_q$), and $\|\boldsymbol{x}\| \leq \beta$.

*Problem 2 (*Ring-LWE$_{n,q}$*).* Given polynomially-many pairs $(\boldsymbol{a_i}, \boldsymbol{b_i}) \in \mathcal{R}_q \times \mathcal{R}_q$ where $\boldsymbol{a_i}$'s are uniformly random: decide either $\boldsymbol{b_i}$'s are also uniformly random, or there exist an $\boldsymbol{s} \in \mathcal{R}_q$ such that $\forall i\ \boldsymbol{b_i} = \boldsymbol{a_i} \cdot \boldsymbol{s} + \boldsymbol{e_i}$ and $\boldsymbol{e_i}$'s are short polynomials from a discrete Gaussian distribution.

Algorithm 1: $\mathsf{FFT}\,(x \in \mathbb{Z}_q^n)$

```
1  for $i \leftarrow 1$ to $n$ do
2      $x[i] \leftarrow \psi^i \cdot x[i]$
3  $x \leftarrow \mathsf{BitReverse}(x)$
4  for $s \leftarrow 1$ to $\log_2(n)$ do
5      for $j \leftarrow 1$ to $2^{s-1}$ do
6          for $k \leftarrow j+1$ to $n$ step $2^s$ do
7              $u \leftarrow x[k]$
8              $t \leftarrow \omega^{jn/2^s} \cdot x[k+2^{s-1}] \bmod q$
9              $x[k] \leftarrow u + t \bmod q$
10             $x[k+2^{s-1}] \leftarrow u - t \bmod q$
11 return $x$
```

2.3 Fast Fourier Transform

Almost all efficient lattice-based schemes with a security proof are based on ideal lattices. In these schemes, the most time-consuming operation is multiplying two polynomials in the polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$. School-book multiplication, which runs in $\mathcal{O}(n^2)$, is too slow for constrained devices. Another approach is to apply fast Fourier transform (FFT) to both polynomials, multiply them coordinate-wise, and then compute FFT inverse on the result. This method needs only $\mathcal{O}(n \log n)$ operations. FFT can be generalized to finite ring $\mathcal{R}_q$[1] where there is no need to any floating-point arithmetic, and all operations are additions and multiplications modulo q.

Algorithm 1 shows the iterated FFT algorithm. In order to multiply two polynomials $\boldsymbol{a}, \boldsymbol{b}$ in $\mathcal{R}_q$,, one should compute $\mathsf{FFT}^{-1}(\mathsf{FFT}(\boldsymbol{a}) \odot \mathsf{FFT}(\boldsymbol{b}))$, where $\odot$ is entry-wise multiplication. The procedure $\mathsf{BitReverse}$ is a simple routine which permutes an array by reversing the bit-string of each index. ψ is the primitive $2n$-th root of unity in $\mathbb{Z}_q$. In other words, ψ is the smallest integer for which $\psi^{2n} \equiv 1 \pmod q$. Thus, $\omega = \psi^2 \bmod q$ is the primitive n-th root of unity.

3 Implemented Primitives and Encryptions

In this section, we explain implementation notes of FFT and discrete Gaussian sampling as two primitives of lattice-based cryptography. Then, we describe a simplified CPA-secure form of NTRU encryption which is later used in an instantiation of Dousti-Jalili protocol (see Section 4.4). The complete form of NTRU encryption and the ring-LWE based encryption of Linder and Peikert (LP-LWE) are implemented without any changes. The reader is referred to IEEE P1363.1 standard and [LP11], respectively, for the description of these schemes.

3.1 Optimizing FFT

FFT pseudo-code is shown in Algorithm 1. By storing relatively small lookup-tables, the $\mathsf{BitReverse}$ routine and the powers of ω and ψ can be evaluated quickly. In our FFT implementations, the majority of the processor's clock cycles are spent on computing mod operations. The direct method of evaluating mod is to perform an integer division. However, constrained devices often do not have any dedicated hardware to divide. Thus, compilers generate a bulk of code containing loops, multiplications, and additions. A simple technique to reduce the number of mod operations is to remove them

[1] This conversion is usually called the Number Theoretic Transform (NTT). However, we continue to use FFT as an umbrella term for both types of transformation.

Algorithm 2: mod (a, q)

```
1  if $q = 7681$ then
2      $a \leftarrow a - q * \left[(a \gg 13) + (a \gg 17) + (a \gg 21)\right]$
3  else if $q = 12289$ then
4      $a \leftarrow a - q * \left[(a \gg 14) + (a \gg 16) + (a \gg 18) + (a \gg 20) + (a \gg 22) + (a \gg 24)\right]$
5  else if $q = 8383489$ then
6      $a \leftarrow a - q * \left[(a \gg 23) + (a \gg 34) + (a \gg 36) + (a \gg 45) + (a \gg 49)\right]$
7  else
8      return NotSupported
9  while $a \geq q$ do
10     $a \leftarrow a - q$
11 return $a$
```

for some intermediate values. Therefore, a modular reduction is applied once after a few consecutive arithmetic operations. However, caution must be taken to avoid any variable overflow.

There are a few proposed methods to speedup mod operation in the implementation of FFT. Güneysu et al. [GLP12] considered the smallest power of two, say 2^p, larger than the divisor q. Thus, $k = 2^p \bmod q$ is relatively small. Then, $a \bmod q$ is equal to $(a^{(1)} \times 2^p + a^{(0)}) \bmod q = (a^{(1)} \times k + a^{(0)}) \bmod q$, where $a^{(0)}$ is formed of p least significant bits of a, and $a^{(1)}$ is consists of other most significant bits. Ducas et al. [DDLL13] used the method introduced in [War03] to efficiently compute divisions with a constant divisor. The main idea is to compute a/q by multiplying a with the integer part of $2^w \times 1/q$, and then, shifting the result by w bits to right (to cancel the effect of 2^w). Here, w is usually set to the bit-size of the input. This method requires double-size multiplication. For example, to perform a 32-bit mod operation, one needs to multiply two 32-bit integers, a and $2^w \times 1/q$, thus, requires 64-bit multiplication.

We have modified the method of double-size multiplication to achieve more efficient implementation of mod operation. We avoid using double-size operation by the expense of approximating the result. This leads to a significant improvement in the running time of FFT on constrained devices, because their processors have short word sizes (8-bit to 32-bit). The idea is that if the position of 1's in $(2^w \times 1/q)$ are $p_1, ..., p_\ell$, we have $\lfloor a/q \rfloor = (\sum_{i=1}^{\ell} a \ll p_i) \gg w$. By combining the two opposite shifts we have the approximation $\lfloor a/q \rfloor \cong \sum_{i=1}^{\ell} (a \gg (w - p_i))$ which requires same-size additions. However, its result may be a few numbers less than the real $\lfloor a/q \rfloor$ and the result which is $a \bmod q = a - q \times \lfloor a/q \rfloor$ may become larger than q. This can be fixed by subtracting the result by q for a few times. A further optimization is to remove $(a \gg (w - p_i))$ terms with large $(w - p_i)$ from the sum. This way we only drop small amounts from the sum. Therefore, with an acceptable approximation, we omit nearly half of the terms in the sum. Algorithm 2 shows the implementation of mod in FFT implementations, using parameters of Section 5.2.

Lyubashevsky et al. [LMPR08] introduced a special parameter set for the lattice-based hash function SWIFFT, where computing mod can be done without any multiplications or loops. In order to reduce a signed 16-bit value x modulo 257, it is sufficient to compute $(x \wedge 255) - (x \gg 8)$, and the result will be in $\{-127, ..., 383\}$. Finally, one can subtract the result by 257 if it is larger than 128, however, this is not needed for intermediate values during FFT computation. Additionally, by keeping all values in the range $[-128, 128]$ (modulo 257) multiplications can be done on 16-bit words and no 32-bit arithmetic is needed. Implemented commitment in the instantiations of Dousti-Jalili protocol (Sections 4.3 and 4.4) enjoys such optimization.

3.2 Discrete Gaussian Sampling

The problem of discrete Gaussian sampling is to sample an integer $x \in \mathbb{Z}_q$ with probability proportional to $\exp(-x^2/2\sigma^2)$, where σ is the standard deviation. Available methods to sample from discrete Gaussian distribution requires relatively-large computation. Moreover, lattice-based schemes usually require a lot of such samples. As a result, efficient implementation of this primitive is a crucial and challenging task for constrained devices. An optimally-fast method to this end is to use a cumulative distribution table. Consider a lookup-table T where is indexed by $-\tau\sigma, ..., \tau\sigma$. Here, τ is the tail-cut factor which determines where we drop the negligible probability of far samples. The value of i-th cell is the scaled probability that a sample is chosen from the range $[-\tau\sigma, i]$. By choosing a suitable scale factor, all values in the lookup-table can be stored as integers. Finally, to obtain a discrete Gaussian sample, one can generate a uniformly random integer y, and perform a binary-search on the lookup-table to find the index x such that $T[x] \leq y < T[x+1]$, and then output x.

Although discrete Gaussian sampling via cumulative lookup-table is the fastest method, its table size is huge for wide distributions (i.e., large standard deviations). Another fast method, called Bernoulli sampler, is introduced recently by Ducas et al. [DDLL13]. In this algorithm, the size of the lookup-table is reduced significantly but some of the samples are rejected with Bernoulli distributions. The rejection introduces an overhead in computation time, and a fraction of generated random bits are not being used and wasted. The running time and table size of cumulative lookup-table method and Bernouli sampler are compared in Section 5.2. Other proposed algorithms, such as original rejection sampling [GPV08], Knuth-Yao [SRVV13], and Ziggurat [BCG$^+$13] algorithms are shown to be less efficient than the above methods [OPG14].

3.3 CPA-Secure NTRU Encryption

In this section, we explain a simplified CPA-secure variant of NTRU encryption where lacks some features of the latest NTRU scheme, standardized in IEEE P1363.1. The provably-secure padding of NTRU (called NAEP) is omitted. This padding is designed to obtain CCA security, as well as adding an amount of randomness to ensure that the message polynomial has secure number of different coefficients. This simplified encryption can be used securely in Dousti-Jalili protocol, intuitively because the protocol is zero-knowledge. Furthermore, the encrypted messages are random challenge strings where the message polynomial has required safe situations with high probability. Even if it is not the case, another random challenge can be generated. The security proof of Dousti-Jalili protocol integrated with this simplified NTRU encryption is provided in Section 4.4.

Polynomial operations in NTRU encryption are done in the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^N - 1\rangle$. Let $\mathcal{L}_{(i,j)}$ be the the subset of $\mathcal{R}_q$ in which i coefficients are 1, j coefficients are -1, and all others are zero. $\mathcal{L}_i$ is defined as $\mathcal{L}_{(i,i)}$. $\mathcal{R}_{[1]}$ is the subset of $\mathcal{R}_q$ with coefficients in $\{-1, 0, 1\}$.

- **KeyGen**: Generate a random polynomial $\boldsymbol{g} \leftarrow \mathcal{L}_{(N/3,N/3-1)}$ invertible in $\mathcal{R}_q$. Choose three random sparse polynomials $\boldsymbol{f_1} \leftarrow \mathcal{L}_{d_1}, \boldsymbol{f_2} \leftarrow \mathcal{L}_{d_2}, \boldsymbol{f_3} \leftarrow \mathcal{L}_{d_3}$. Now form $\boldsymbol{F} = \boldsymbol{f_1} \cdot \boldsymbol{f_2} + \boldsymbol{f_3}$ and $\boldsymbol{f} = 1 + 3\boldsymbol{F}$. Check that $\boldsymbol{f}$ is invertible in $\mathcal{R}_q$, otherwise, generate another $\boldsymbol{f}$. Now there is an $\boldsymbol{f_q}$ such that $\boldsymbol{f} \cdot \boldsymbol{f_q} = 1$. Finally, the public key is $\boldsymbol{pk} = 3\boldsymbol{g} \cdot \boldsymbol{f_q}$, and the secret key is $\boldsymbol{sk} = (\boldsymbol{F}, \boldsymbol{f}, \boldsymbol{f_q})$.
- **Encrypt** $(\boldsymbol{pk}, x)$: Encode $x \in \{0,1\}^\ell$ arbitrarily to a ternary polynomial $\bar{\boldsymbol{x}} \in \mathcal{R}_{[1]}$, but leave its constant coefficient as zero in the encoding. It is required that the sum of coefficients be smaller than or equal to T_s, while each coefficient (0's, 1's, and -1's) is repeated at least T_c times. Otherwise, fail and request another x. Finally generate random $\boldsymbol{r_1} \leftarrow \mathcal{L}_{d_1}, \boldsymbol{r_2} \leftarrow \mathcal{L}_{d_2}, \boldsymbol{r_3} \leftarrow \mathcal{L}_{d_3}$ to form a blinding polynomial $\boldsymbol{r} = \boldsymbol{r_1} \cdot \boldsymbol{r_2} + \boldsymbol{r_3}$. Now the ciphertext is $\boldsymbol{c} = \boldsymbol{r} \cdot \boldsymbol{pk} + \bar{\boldsymbol{x}}$.
- **Decrypt** $(\boldsymbol{sk}, \boldsymbol{c})$: Compute $\boldsymbol{a} = \boldsymbol{f} \cdot \boldsymbol{c}$. Considering the fact that the coefficients of $\boldsymbol{a}$ are in the range $[-\frac{q}{2}, \frac{q}{2})$, $\bar{\boldsymbol{x}} = \boldsymbol{c} \pmod 3$ will be a ternary polynomial. Decode $\bar{\boldsymbol{x}}$ to obtain the message.

NTRU encryption does not make use of FFT for polynomial multiplications. Polynomials are either ternary/sparse, or formed by three ternary/sparse polynomials. Thus, polynomial multiplications can be done using fast direct algorithms. Moreover, the parameter q is a power of two, which means that modular reduction is easily done by dropping most-significant bits. The above simplified NTRU is derived from "Fast Fp" variant of NTRU, where $\boldsymbol{f}$ is formed by $\boldsymbol{f} = 1 + 3\boldsymbol{F}$. As a result, $\boldsymbol{f}$ is always invertible in $\mathcal{R}_3 = \mathbb{Z}_3[x]/\langle x^N - 1\rangle$ which is a requirement in NTRU encryption.

4 Implemented Authentication Protocols

Proposed lattice-based authentication protocols were reviewed in Section 1.1. Most of them have a zero-knowledge base protocol with noticeable soundness error. As a result, it should be repeated several times to obtain a secure authentication protocol. Even parallel repetition makes them very inefficient for constrained devices, in terms of computation and communication complexities. Instead, we have chosen GLP [GLP12] and BLISS [DDLL13] signature schemes, and transformed them to lattice-based authentication Protocols 1 and 2. The other implemented protocols are two instantiations of Dousti and Jalili [DJ13] authentication scheme. Dousti-Jalili protocol is an efficient zero-knowledge proof which uses a statistically-hiding bit string commitment and a semantically-secure encryption algorithm, in a black-box manner. We have utilized a ring-based variant of Kawachi et al.'s commitment scheme [KTX08] in the instantiations. Additionally, LP-LWE encryption [LP11] and the simplified NTRU of Section 3.3 are integrated to form Protocols 3 and 4. Some significant improvement in reducing computation time, required RAM space, and communication overhead can be observed by integrating LP-LWE encryption into Dousti-Jalili Protocol, compared to a black-box usage of this encryption.

4.1 GLP Protocol

Protocol 1 explains the authentication protocol derived from GLP signature scheme. Random polynomial $\boldsymbol{a} \in \mathcal{R}_p$ is fixed by a trusted authority as a system parameter. In the absence of such authority, parties can run a multi-party random-generation protocol to produce $\boldsymbol{a}$. $\mathcal{R}_{[\alpha]}$ is a subset of $\mathcal{R}_p$ with coefficients between $-\alpha$ and α. $\boldsymbol{x}^{(1)}$ denotes a polynomial obtained by dividing $\boldsymbol{x}$'s coefficients by $2(\kappa - 32) - 1$, and keeping their integral parts.

Protocol 1 (GLP) The secret key $\boldsymbol{sk}$ is consist of random $\boldsymbol{s_1}, \boldsymbol{s_2} \in \mathcal{R}_{[1]}$. Corresponding public key is $\boldsymbol{pk} = \boldsymbol{a} \cdot \boldsymbol{s_1} + \boldsymbol{s_2}$. The protocol rounds are as follows.

1. **Prover:** *Generate two polynomials $\boldsymbol{y_1}, \boldsymbol{y_2} \in \mathcal{R}_{[\kappa]}$ at random. Then, send $\boldsymbol{u} = (\boldsymbol{a} \cdot \boldsymbol{y_1} + \boldsymbol{y_2})^{(1)}$ to the verifier.*
2. **Verifier:** *Send a random polynomial **c** in which 32 coefficients are either $+1$ or -1, and all others are zero.*
3. **Prover:** *Compute $\boldsymbol{z_1} = \boldsymbol{s_1} \cdot \boldsymbol{c} + \boldsymbol{y_1}$ and $\boldsymbol{z_2} = \boldsymbol{s_2} \cdot \boldsymbol{c} + \boldsymbol{y_2}$. If $\boldsymbol{z_1}$ or $\boldsymbol{z_2}$ is outside $\mathcal{R}_{[\kappa-32]}$, or $\boldsymbol{z_2}' = \mathtt{Compress}(\boldsymbol{a} \cdot \boldsymbol{z_1} - \boldsymbol{pk} \cdot \boldsymbol{c}, \boldsymbol{z_2})$ is equal to $\perp$, terminate the protocol. Otherwise, send $(\boldsymbol{z_1}, \boldsymbol{z_2}')$.*

Verification step: *Verify that $\boldsymbol{z_1}, \boldsymbol{z_2}'$ are both in $\mathcal{R}_{[\kappa-32]}$. Then, accept iff $\boldsymbol{u} = (\boldsymbol{a} \cdot \boldsymbol{z_1} + \boldsymbol{z_2} - \boldsymbol{pk} \cdot \boldsymbol{c})^{(1)}$.*

In the first round, sending $(\boldsymbol{a} \cdot \boldsymbol{y_1} + \boldsymbol{y_2})^{(1)}$ is like dropping least-significant bits of the message in order to reduce its size. The Compress function is a linear time compression algorithm which drops some bits from $\boldsymbol{z_2}$, again to reduce its size and leverage the communication complexity. The internal function of Compress is described in [GLP12]. Ducas et al. [DDLL13] showed that the use of Compress function introduces an existential forgery attack on GLP signature schemes. However, this attack is not applicable when this signature scheme is converted to an authentication protocol.

In round 3, prover may reject to respond in order to prevent information leakage about the secret key. In this case, the verifier should re-run the protocol from the beginning until a valid response is received. A possible technique to decrease total number of rounds (due to Lyubashevsky [Lyu09]) is to send multiple independent $\boldsymbol{y_1}$'s and $\boldsymbol{y_2}$'s by the prover in round 1. Then, in round 3, he can try challenges sequentially and respond to the first one which is not rejected. Unfortunately, this method will increase the average time because computation of each pair $\boldsymbol{y_1}, \boldsymbol{y_2}$ is time consuming and dominates the savings in communication time.

Note that Protocol 1 avoids using any Gaussian sampling and all random values are from uniform distribution, which is significantly more efficient. Moreover, computation of $\boldsymbol{a} \cdot \boldsymbol{z_1} - \boldsymbol{pk} \cdot \boldsymbol{c}$ as an input of `Compress` function do not need more polynomial multiplications because it is equal to $\boldsymbol{a} \cdot \boldsymbol{y_1} + \boldsymbol{y_2} - \boldsymbol{z_2}$.

4.2 DDLL Protocol

Protocol 2 explains the authentication protocol derived from BLISS signature scheme. $\mathcal{R}_{2q}$ denotes the polynomial ring $\mathbb{Z}_{2q}[x]/\langle x^n + 1\rangle$, where q is a prime. ζ is a scalar such that $\zeta(q-2) \equiv 1 \pmod{2q}$. By $\lfloor x \rceil_d$, we mean the value obtained by dropping d least-significant bits from x. $\mathcal{D}_{\mathbb{Z}^n,\sigma}$ is n-dimensional discrete Gaussian distribution with standard deviation σ, and $\langle a, b\rangle$ is the inner-product of a and b. All polynomial operations are performed in $\mathcal{R}_{2q}$.

Protocol 2 (DDLL) Random polynomials $\boldsymbol{f}, \boldsymbol{g}$ are sampled during key generation; both have exactly $\delta_1 n$ coefficients in $\{-1, +1\}$. Polynomial $\boldsymbol{f}$ should also be invertible in $\mathcal{R}_{2q}$. Then, $(\boldsymbol{s_1}, \boldsymbol{s_2}) = (\boldsymbol{f}, 2\boldsymbol{g}+1)$ is the secret key and $\boldsymbol{pk} = 2(2\boldsymbol{g}+1)/\boldsymbol{f}$ is the public key. The parameter M is a constant. Protocol rounds are as follows.

1. **Prover:** *Sample two polynomials* $\boldsymbol{y_1}, \boldsymbol{y_2}$ *from* $\mathcal{D}_{\mathbb{Z}^n,\sigma}$. *Then, compute* $\boldsymbol{u} = \zeta \boldsymbol{pk} \cdot \boldsymbol{y_1} + \boldsymbol{y_2}$ *and send* $\lfloor \boldsymbol{u} \rceil_d$ *to the verifier.*
2. **Verifier:** *Send a random polynomial* $\boldsymbol{c} \in \mathcal{R}_{2q}$ *which has exactly* κ *coefficients equal to 1 and all others equal to 0.*
3. **Prover:** *Choose a random bit* $b \in \{0, 1\}$. *Compute* $\boldsymbol{z_1} = \boldsymbol{y_1} + (-1)^b \boldsymbol{s_1} \cdot \boldsymbol{c}$, $\boldsymbol{z_2} = \boldsymbol{y_2} + (-1)^b \boldsymbol{s_2} \cdot \boldsymbol{c}$, *and* $\boldsymbol{z_2^\dagger} = (\lfloor \boldsymbol{u} \rceil_d - \lfloor \boldsymbol{u} - \boldsymbol{z_2} \rceil_d) \bmod p$. *Consider vectors* $\mathbf{S} = \begin{bmatrix} \boldsymbol{s_1} & \boldsymbol{s_2} \end{bmatrix}^t$ *and* $\mathbf{Z} = \begin{bmatrix} \boldsymbol{z_1} & \boldsymbol{z_2} \end{bmatrix}$. *With probability* $1 \Big/ \left(M \exp(-\frac{\|\mathbf{S} \cdot \boldsymbol{c}\|^2}{2\sigma^2}) \cosh(\frac{\langle \mathbf{Z}, \mathbf{S} \cdot \boldsymbol{c} \rangle}{2\sigma^2}) \right)$, *send* $(\boldsymbol{z_1}, \boldsymbol{z_2^\dagger})$ *to the verifier. Otherwise, terminate the protocol.*

Verification step: *Reject if either* $\left\| [\boldsymbol{z_1} | 2^d \cdot \boldsymbol{z_2^\dagger}] \right\| > \beta_2$ *or* $\left\| [\boldsymbol{z_1} | 2^d \cdot \boldsymbol{z_2^\dagger}] \right\|_\infty > \beta_\infty$. *Then, accept iff* $\lfloor \zeta \boldsymbol{pk} \cdot \boldsymbol{z_1} - \zeta q \boldsymbol{c} \rceil_d + \boldsymbol{z_2^\dagger} = \lfloor \boldsymbol{u} \rceil_d$.

Similar to Protocol 1, the verifier should restart the protocol if the prover fails in round 3, and in order to reduce the average running time of the protocol, it is better to repeat it sequentially. Several optimizations are proposed in the design of BLISS which can be directly applied here. In the key generation process, the public key is computed as $\boldsymbol{pk} = 2\boldsymbol{pk'}$ for some $\boldsymbol{pk'} \in \mathcal{R}_q$. Multiplication of $\boldsymbol{pk} \cdot \boldsymbol{y}$ (in $\mathcal{R}_{2q}$) in round 1 can be done more efficiently by first multiplying $\boldsymbol{pk'} \cdot \boldsymbol{y}$ (in $\mathcal{R}_q$) using FFT method, and then doubling the result. The use of a precomputed FFT form of $\boldsymbol{pk'}$ increases efficiency further. Multiplication of $\boldsymbol{s_1} \cdot \boldsymbol{c}$ and $\boldsymbol{s_2} \cdot \boldsymbol{c}$ in round 3 is more efficient to be done directly, rather than using FFT conversion; because $\boldsymbol{c}$ has only κ non-zero coefficients and the coefficients of $\boldsymbol{s_1}, \boldsymbol{s_2}$ are all small.

The acceptance probability in round 3 can be separated into two independent Bernoulli probabilities, one proportional to $1/\exp(-\|\mathbf{S} \cdot \boldsymbol{c}\|^2 / 2\sigma^2)$ and the other proportional to $1/\cosh(\langle \mathbf{Z}, \mathbf{S} \cdot \boldsymbol{c} \rangle / 2\sigma^2)$. The former does not depend on $\mathbf{Z}$ and can be evaluated before computing $\boldsymbol{z_1}$ and $\boldsymbol{z_2}$. In this case, the protocol can be repeated earlier. Because the distribution of coefficients in $\boldsymbol{z_1}, \boldsymbol{z_2^\dagger}$ is exactly known (i.e., a Gaussian distribution with known standard deviation), one can send them coded by Huffman codes in order to reduce the message size and communication time. To this end, an encoding lookup-table is used on the prover's side. To produce discrete Gaussian samples, we have implemented Bernoulli sampler which has a 0.5 KB precomputed table. One can implement the cumulative distribution table method instead, however, it consumes more than 40 KB of the flash memory. The performance comparison of these two sampling methods is provided in Section 5.2.

4.3 DJ-LWE Protocol

Protocol 3 is an instantiation of Dousti-Jalili protocol using a ring-SIS based commitment scheme and LP-LWE encryption. The dimension of LP-LWE encryption is denoted by N to be distinct from the dimension of the commitment. $D_{\mathbb{Z}^N,\sigma}$ is a discrete Gaussian distribution with standard deviation σ. The function `Encode` converts $x \in \{0,1\}^N$ to a polynomial in $\mathcal{R}_q$ by mapping bit 0 to a 0 coefficient, and bit 1 to a $\lfloor \frac{q}{2} \rfloor$ coefficient. On the other hand, `Decode` receives a polynomial and converts each coefficient to 0 if it is between $-\lfloor \frac{q}{4} \rfloor$ and $\lfloor \frac{q}{4} \rfloor$, and to 1 otherwise.

Protocol 3 (DJ-LWE) $\boldsymbol{a_1}, ..., \boldsymbol{a_m} \in \mathcal{R}_p$ and $\boldsymbol{a} \in \mathcal{R}_q$ are random polynomials generated by a trusted authority as system parameters, or achieved by running a multiparty random-generation protocol. The key-pair $(\boldsymbol{sk}, \boldsymbol{pk})$ is a key pair of LP-LWE. Commitment operations are performed in $\mathcal{R}_p$ while LP-LWE ones are done in $\mathcal{R}_q$.

1. **Prover:** *Generate two $(mn/2)$-bit random strings u and ρ, and build polynomials $\boldsymbol{u_1}, ..., \boldsymbol{u_{m/2}}, \boldsymbol{\rho_1}, ..., \boldsymbol{\rho_{m/2}}$ where the i-th coefficient of $\boldsymbol{u_j}$ (resp., $\boldsymbol{\rho_j}$) is 0 or 1 according to $(i \times j)$-th bit of u (resp., ρ). Send $\bar{\mathbf{c}} = \textsc{Com}(u; \rho) = \sum_{i=1}^{m/2} \boldsymbol{a_i} \cdot \boldsymbol{u_i} + \sum_{i=1}^{m/2} \boldsymbol{a_{m/2+i}} \cdot \boldsymbol{\rho_i}$.*
2. **Verifier:** *Choose a random challenge $x \in \{0,1\}^N$. Let $\bar{\boldsymbol{x}} = \texttt{Encode}(x)$. Sample $\boldsymbol{e_1}, \boldsymbol{e_2}, \boldsymbol{e_3}$ from $D_{\mathbb{Z}^N,\sigma}$, and compute $\boldsymbol{c_1} = \boldsymbol{a} \cdot \boldsymbol{e_1} + \boldsymbol{e_2}$ and $\boldsymbol{c_2} = \boldsymbol{pk} \cdot \boldsymbol{e_1} + \boldsymbol{e_3} + \bar{\boldsymbol{x}}$. Finally, send $(\boldsymbol{c_1}, \boldsymbol{c_2})$ to the prover.*
3. **Prover:** *Compute $x' = \texttt{Decode}(\boldsymbol{sk} \cdot \boldsymbol{c_1} + \boldsymbol{c_2})$, and send $x' \oplus u$.*
4. **Verifier:** *Send $\boldsymbol{e_1}$ as the randomness used during the encryption.*
5. **Prover:** *Compute $\boldsymbol{e_2} = \boldsymbol{c_1} - \boldsymbol{a} \cdot \boldsymbol{e_1}$ and $\boldsymbol{e_3} = \boldsymbol{c_2} - \boldsymbol{pk} \cdot \boldsymbol{e_1} - \bar{\boldsymbol{x}}'$, where $\bar{\boldsymbol{x}}' = \texttt{Encode}(x')$. All $\boldsymbol{e_1}, \boldsymbol{e_2}, \boldsymbol{e_3}$ should be short (i.e., in the expected range of the Gaussian distribution). Otherwise, terminate the protocol. Finally, send ρ to decommit.*

Verification step: *Compute $u' = (x' \oplus u) \oplus x$, and accept iff $\bar{\mathbf{c}} = \textsc{Com}(u'; \rho)$.*

If the prover is honest, x' will be equal to x and the verifier convinces in the verification step. In the general Dousti-Jalili protocol, the verifier sends all the randomness used for the encryption (i.e., all $\boldsymbol{e_1}$, $\boldsymbol{e_2}$, and $\boldsymbol{e_3}$) in round 4. However, it suffices to send only $\boldsymbol{e_1}$ because both $\boldsymbol{e_2}, \boldsymbol{e_3}$ can be uniquely determined. This also leads to less memory usage on the prover. Moreover, a precomputed FFT of $\boldsymbol{a}$ and $\boldsymbol{a_1}, ..., \boldsymbol{a_m}$ can be given to the prover (practically, loaded into the smart card or microcontroller at the manufacturing time). This applies to $\boldsymbol{sk}$ and $\boldsymbol{pk}$ as well. Although storing $\boldsymbol{sk}$ in FFT form may cause a little memory overhead (because $\boldsymbol{sk}$ coefficients are short and require less memory space, while the values in FFT representation are spread in $\mathbb{Z}_q$), but this still can improve the computation time as one FFT calculation is eliminated.

Applying FFT on $\boldsymbol{u_1}, ..., \boldsymbol{u_{m/2}}$ and $\boldsymbol{\rho_1}, ..., \boldsymbol{\rho_{m/2}}$ in round 1 can be performed efficiently using the special modulus $p = 257$. As described in Section 3.1, reducing modulo 257 can be done very quickly without any multiplications or loops. Moreover, these FFTs consists of only 16-bit multiplications and additions and no 32-bit arithmetic is needed. Considering the fact that 8-bit architectures are versatile in constrained devices, this leads to a noticeable reduction in computation time. A comparison between the performance of FFT implementation using this parameter set and other general ones is provided in Section 5.2.

The resulting commitment value $\bar{\boldsymbol{c}}$ in round 1 can be sent without applying FFT inverse. This does not violate the security properties of the commitment because FFT and FFT inverse are two publicly-known conversions. Polynomials $\boldsymbol{c_1}$ and $\boldsymbol{c_2}$, which are sent by the verifier in round 2, can be already in FFT representation, eliminating more FFT computations on the prover (i.e., the constrained device). Note that $\boldsymbol{e_1}$ should be sent in coefficient representation because the prover should verify its short length, which is defined on coefficient values. There are also two inevitable FFT inverses when computing $\boldsymbol{e_2}, \boldsymbol{e_3}$ in round 5, again to verify their short length. An advantage of this protocol over Protocol 2 is that there is no need to generate discrete Gaussians on the prover's side. All Gaussian samplings are done by the verifier which is very powerful compared to our constrained devices.

4.4 DJ-NTRU Protocol

Protocol 4 is another instantiation of Dousti-Jalili protocol using the simplified NTRU encryption, described in Section 3.3.

Protocol 4 (DJ-NTRU) The key-pair $(\boldsymbol{sk}, \boldsymbol{pk})$ is a key pair of NTRU. The protocol round are as follows:

1. **Prover:** *Generate two $(mn/2)$-bit random strings u and ρ, and build polynomials $\boldsymbol{u_1}, ..., \boldsymbol{u_{m/2}}, \boldsymbol{\rho_1}, ..., \boldsymbol{\rho_{m/2}}$ where the i-th coefficient of $\boldsymbol{u_j}$ (resp., $\boldsymbol{\rho_j}$) is 0 or 1 according to $(i \times j)$-th bit of u (resp., ρ). Send $\bar{\boldsymbol{c}} = \textsc{Com}(u; \rho) = \sum_{i=1}^{m/2} \boldsymbol{a_i} \cdot \boldsymbol{u_i} + \sum_{i=1}^{m/2} \boldsymbol{a_{m/2+i}} \cdot \boldsymbol{\rho_i}$.*
2. **Verifier:** *Choose a random challenge $x \in \{0,1\}^\ell$. Encrypt x by CPA-secure NTRU encryption, and send the ciphertext $\boldsymbol{c}$. Keep blinding polynomial $\boldsymbol{r}$ for round 4.*
3. **Prover:** *Decrypt $\boldsymbol{c}$ to recover the message $x' \in \{0,1\}^\ell$, and send $x' \oplus u$.*
4. **Verifier:** *Send $\boldsymbol{r}$ as the randomness used during the encryption.*
5. **Prover:** *Check that $\boldsymbol{c} = \boldsymbol{r} \cdot \boldsymbol{pk} + \bar{\boldsymbol{x}}'$, where $\bar{\boldsymbol{x}}'$ is the polynomial encoding of x'. Otherwise, terminate the protocol. Finally, send ρ to decommit.*

Verification step: *Compute $u' = (x' \oplus u) \oplus x$, and accept iff $\bar{\boldsymbol{c}} = \textsc{Com}(u'; \rho)$.*

If the prover is honest, x' will be equal to x and the verifier convinces in the verification step. In the following, we claim that a CPA-secure encryption is enough for the security of the Dousti-Jalili protocol. Smart-card model, explained in [DJ13], is an attacker model which allows the attacker to interact arbitrarily with an honest prover to gather information. Then, she is disconnected from the prover and tries to impersonate him to an honest verifier.

Lemma 1. *Assuming CPA-security of the simplified NTRU, Protocol 4 is a secure authentication protocol under active attacks in the smart-card model.*

Proof Assume that an adversary $\mathcal{A}$ is able to break Protocol 4. We will use $\mathcal{A}$ to build another adversary $\mathcal{B}$ that wins in an IND-CPA game for NTRU. When the adversary $\mathcal{B}$ receive the ciphertext challenge $\boldsymbol{c}$ from the challenger, she runs $\mathcal{A}$ in a block-box manner. In the first phase, $\mathcal{B}$ communicates to $\mathcal{A}$ as a prover. Note that $\mathcal{B}$ cannot perform all prover jobs because she does not know the secret key. Similar to the proof of zero-knowledgeness in Dousti-Jalili protocol, $\mathcal{B}$ sends a completely random message in round 3. Then, when she receives x in round 4, she rewinds $\mathcal{A}$ to round 3, and answers the challenge correctly. In the second phase, $\mathcal{A}$ tries to impersonate the prover. Now, $\mathcal{B}$ is acting as a verifier and sends $\boldsymbol{c}$ (the game challenge) to $\mathcal{A}$ in round 2, without knowing its blinding polynomial. If the adversary $\mathcal{A}$ is successful to generate correct $x' \oplus u$ in round 3, $\mathcal{B}$ can obtain u by rewinding, and then achieve x' and win the IND-CPA game. This contradicts with the assumption of the lemma.

5 Experimental Results

5.1 Measurement Settings

Constrained implementations are done on a proprietary smart card and an AVR MEGA microcontroller. The smart card consists of a 32-bit ARM7TDMI processor running at 4.92 MHz. It has an ISO/IEC 7816 (contact) interface, and supports transport protocol T=0. ARM source codes are written in C++ and compiled using ARM/Thumb Compiler of RVDS 2.2 and optimized for maximum speed.

Regarding microcontroller implementation, we have chosen ATmega64 from AVR MEGA family which has an 8-bit data bus. Although ATmega64 supports a higher clock speed, we have configured a clock of 8 MHz, because this is reasonable for the one embedded in a smart card. ATmega64 has 64 KB of flash memory and 4 KB of SRAM. Flash memory is programmable at manufacturing time but cannot be modified in runtime. Thus, it contains instruction codes, imported keys, any fixed protocol parameters, and lookup tables. Moreover, ATmega64 has 2 KB of internal EEPROM which is writable during operation, but has a very high writing delay in comparison to SRAM. Occasionally, when the implementations require more than 4 KB of SRAM, we have used EEPROM as a temporary storage. AVR source codes are written in C language, and compiled by AVR-GCC 4.7.2 and optimized for maximum speed.

The verifier's side of authentication protocols is implemented on a typical personal computer (PC), which has 4 GB of RAM and a 3.3 GHz Intel Core i3 CPU. The PC is equipped with an ACS ACR1281U-C1 card reader which has dual interfaces (contact and contactless). The source code for the verifiers are written in C# using WinSCard API to communicate with the smart card.

5.2 Performance of the Primitives

This section contains the performance results of FFT and discrete Gaussian sampling. We have chosen four parameter sets for FFT implementations, which are FFT–128–257, FFT–256–7681, FFT–512–12289, and FFT-512-8383489. The first number indicates the degree n, and the second one indicates the modulus q. The mentioned technique in Section 3.1 has been used in the parameter set FFT–128–257. FFT–256–7681 is a recommended parameter set [LP11, GFS$^+$12] for 128-bit secure LP-LWE encryption based on Ring-LWE. FFT–512–12289 and FFT–512–8383489 are from parameter sets of BLISS-I and GLP signature schemes. Each of these FFTs are implemented using four methods of computing mod operation. The first one is the compiler generated code for the mod operator of C/C++. Two other methods are smallest power of two [GLP12] and double-size multiplication [DDLL13]. The last one is our method of same-size multiplication which is optimized for constrained processors. All these methods have been explained in Section 3.1.

Figure 2 shows the running time of FFT implementations on the ARM7TDMI smart card and ATmega64 microcontroller. FFT–128–257 is exceptionally fast, specially on the 8-bit ATmega64. For the other parameter sets, our method of computing mod is faster than other methods on the 32-bit ARM7TDMI. It outperforms the double-size multiplication method, mainly because its 64-bit multiplication is translated into several 32-bit instructions. Moreover, the method of smallest power of two does not perform well on the ARM7TDMI, because reducing a few bits from operands does not have a significant improvement on the running time when multiplication of 16×16-bit values are performed by the hardware in a very short period of time. On the ATmega64, the results are different. Except FFT–256–7681 for which our method is the best, the compiler generated code has the best running time on other parameters. The overhead of breaking long-word multiplications into hardware-supported 4×4-bit multiplications is large enough to cancel out the optimizations of the methods.

In order to generate random samples from discrete Gaussian distribution, one needs a uniform pseudo-random number generator first. We have implemented an algorithm similar to ANSI X9.17

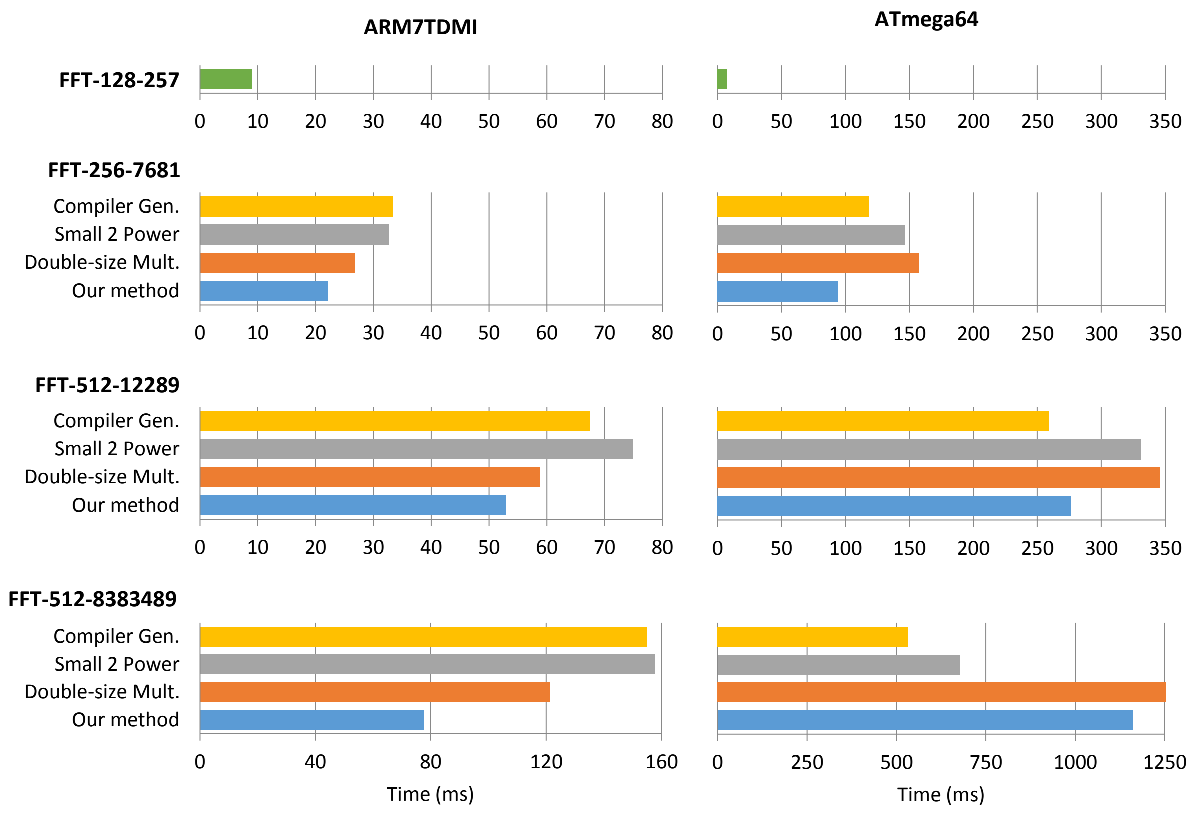

Fig. 2. The running time of fast Fourier transform with different parameter sets, where the *mod* operation is implemented using a number of methods. Numerical results are provided in Table 4.

standard to generate uniform random numbers. This standard exploits a symmetric encryption Enc. At the manufacturing stage, a random key k and an initial seed value s are embedded into the hardware. Whenever new random data is required, the current time d is obtained with maximum accuracy (e.g., by reading the counter of a fine-grained timer), and $t \leftarrow \mathsf{Enc}_k(d)$ is computed. Then, $x \leftarrow \mathsf{Enc}_k(s \oplus t)$ is outputted as the new random data, and the seed is updated as $s \leftarrow \mathsf{Enc}_k(x \oplus t)$. On our smart card, we have utilized a coprocessor which evaluates DES encryption for the Enc function. However, ATmega64 has no cryptographic hardware, and we have implemented DES encryption in software for random data generation.

Discrete Gaussian samples are obtained by running two algorithms, cumulative distribution table method and Bernoulli sampler. The target distributions are on $\mathbb{Z}$, centered at the origin, and with standard deviations 4.51 and 215. The standard deviation 4.51 is used in the 128-bit secure ring-based LP-LWE encryption [LP11, GFS$^+$12]. The other standard deviation (215) is used in BLISS-I signature scheme. Figure 3 shows the running time and lookup-table size of the sampling algorithms. For the standard deviation 4.51, the method of cumulative distribution table is significantly faster than Bernoulli sampler, at the expense of a slightly larger lookup table (nearly 150 bytes). This means that for *narrow* Gaussian distributions, cumulative distribution table method is better even for constrained devices. However, for the standard deviation 215, the cumulative distribution table is very large for our constrained devices.

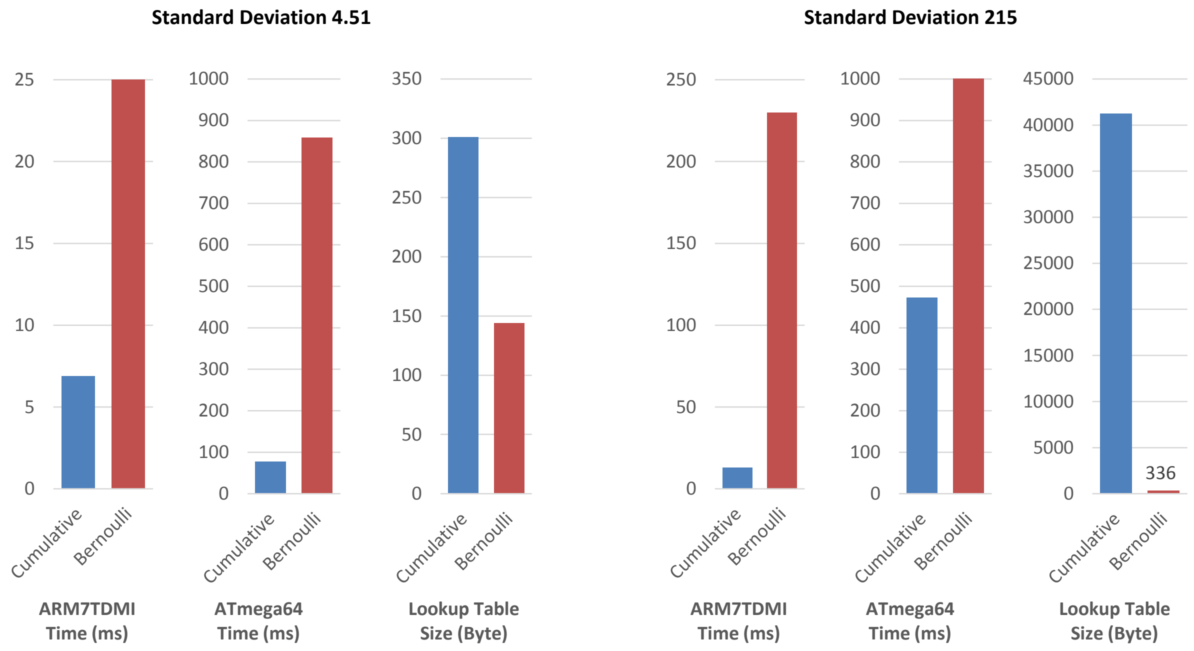

Fig. 3. Running time and lookup table size of cumulative distribution table and Bernoulli sampler algorithms, where 512 samples are generated from discrete Gaussian distribution with standard deviations 4.51 and 215. Numerical results are provided in Table 5.

Table 2. Parameters of implemented LP-LWE, NTRU, and CPA-secure NTRU encryptions

LP-LWE	N	q	σ					
	256	7681	4.51					
(CPA-) NTRU	N	q	d_1	d_2	d_3	ℓ	T_s	T_c
	439	2048	9	8	5	65	126	112

5.3 Performance of the Encryption Schemes

We have implemented NTRU encryption, standardized in IEEE P1363.1, as well as a simplified CPA-secure NTRU encryption described in Section 3.3, and provably-secure LP-LWE encryption by Lindner and Peikert [LP11]. The used parameters for LP-LWE and both NTRU encryptions are specified in Table 2. Parameters for LP-LWE are proposed by Lindner and Peikert [LP11] and Göttert et al. [GFS$^+$12]. NTRU parameters are from EES439EP1 parameter set in IEEE P1363.1. The security of these configurations are estimated equivalent to 128-bit symmetric encryption. In the implementation of LP-LWE, we have considered that the polynomial a, which is a global parameter, is already converted to the FFT form. Moreover, key-pair (sk, pk) are outputted in their FFT representation by the key generation algorithm. Thus, encryption and decryption are performed faster while key generation has a small overhead.

The running time of lattice-based encryptions are provided in Figure 4. The results are separately presented for key generation time, encryption time, and decryption time. Except the key-generation time of NTRU and CPA-secure NTRU encryptions, all other operations are performed in less than 400 ms. Utilizing a cryptography co-processor for these computations, which is common in the design of smart cards, one can achieve very fast lattice-based encryptions in practice. It is also worth-mentioning

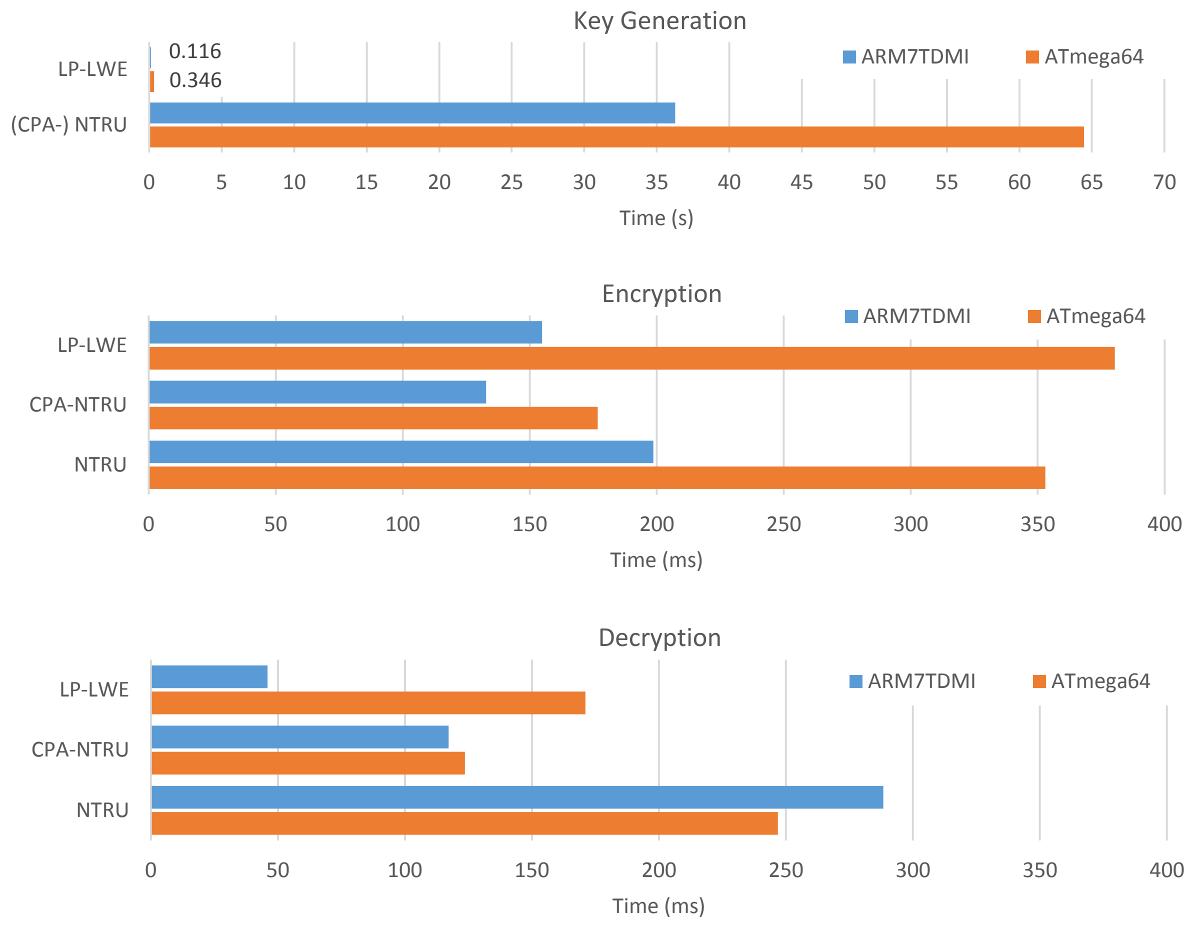

Fig. 4. Encryption, decryption, and key generation time of lattice-based encryption schemes; Numerical results are provided in Table 6.

that LP-LWE is 6.4 times faster than the standard NTRU in decryption, and 1.37 times faster in encryption. Additionally, LP-LWE decryption is even 2.6 times faster than the simplified CPA-secure NTRU. Considering the fact that LP-LWE is provably-secure and patent-free, the use of this scheme as an alternative to the NTRU ones appears to be advantageous.

5.4 Protocols Performance

Parameters for the implemented authentication protocols are specified in Table 3. The parameters for GLP protocol are the same as "Set I" parameters proposed in [GLP12], where provide 100 bits of security. DDLL protocol's parameters are obtained from 128-bit secure BLISS-I [DDLL13]. The parameters for the commitment part of DJ-LWE and DJ-NTRU protocols are chosen in such a way that they take advantage of FFT–128–257 optimization techniques. Moreover, setting $m = 20$ leads to a statistically-hiding commitment scheme, needed by the security proof of these protocols. Other parameters of DJ-LWE and DJ-NTRU are same as the parameters of LP-LWE and NTRU encryptions in Table 2, respectively. According to the results of Section 5.2, on the ARM7TDMI, all protocols are implemented using the proposed technique for computing mod operation. However, on the ATmega64, GLP and DDLL implementations use the mod operator of C/C++, and only DJ-LWE uses

Table 3. Parameters of implemented lattice-based authentication protocols

GLP Protocol	n	p	κ						
	512	8383489	2^{14}						
DDLL Protocol	n	q	d	p	δ_1	σ	κ	β_2	β_∞
	512	12289	10	24	0.3	215	23	12872	2100
DJ-LWE / DJ-NTRU	n	m	p						
	128	20	257						

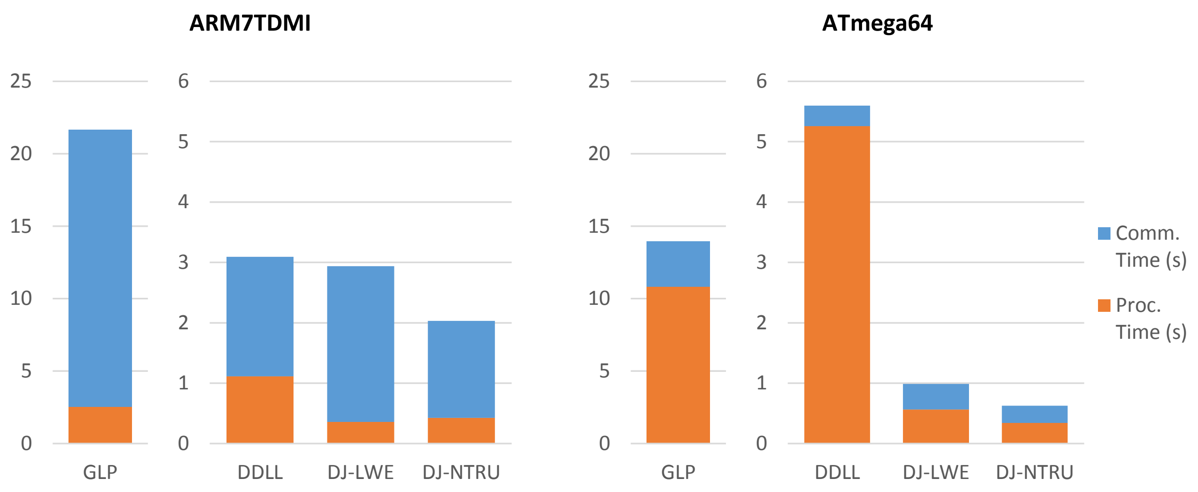

Fig. 5. The overall running time of implemented authentication protocols, separated into processing and communication time; Note that the communication times of ATmega64 are estimated. Numerical results are provided in Table 8.

our method. The CPA-secure NTRU encryption in DJ-NTRU protocol has $q = 2048$. Thus, computing mod operations are easily performed by right shifts.

Figure 5 illustrates the running time of a complete authentication process using each of the implemented protocols. The processing times of the verifiers were all less than 35 ms and ignored. Communication times are measured using a dummy prover who performs no computations and just sends and receives real-size messages. Note that the communication times for ATmega64 are estimated. The communication times of ARM7TDMI smart card are quite large. That is because this smart card only supports T=0 transport protocol, and the messages are in the form of traditional application protocol data units (APDUs), limited by a maximum length of 256 bytes. We have estimated communication times of the ATmega64 microcontroller for the case of T=1 transport protocol and extended APDUs (without the limit of traditional APDUs). Moreover, note that the running times of GLP and DDLL protocols are different in each run due to the protocol repetition, and the results in Figure 5 are in average. Figure 6 also shows the portion of important computation parts in the processing time of the protocols.

Figure 7 illustrates flash memory and RAM usage of the lattice-based authentication protocols on our constrained devices. Required RAM for the GLP and DDLL protocols are a little larger than available RAM on the ATmega64. In order to solve this issue, some intermediate polynomials are temporarily swapped out into the EEPROM.

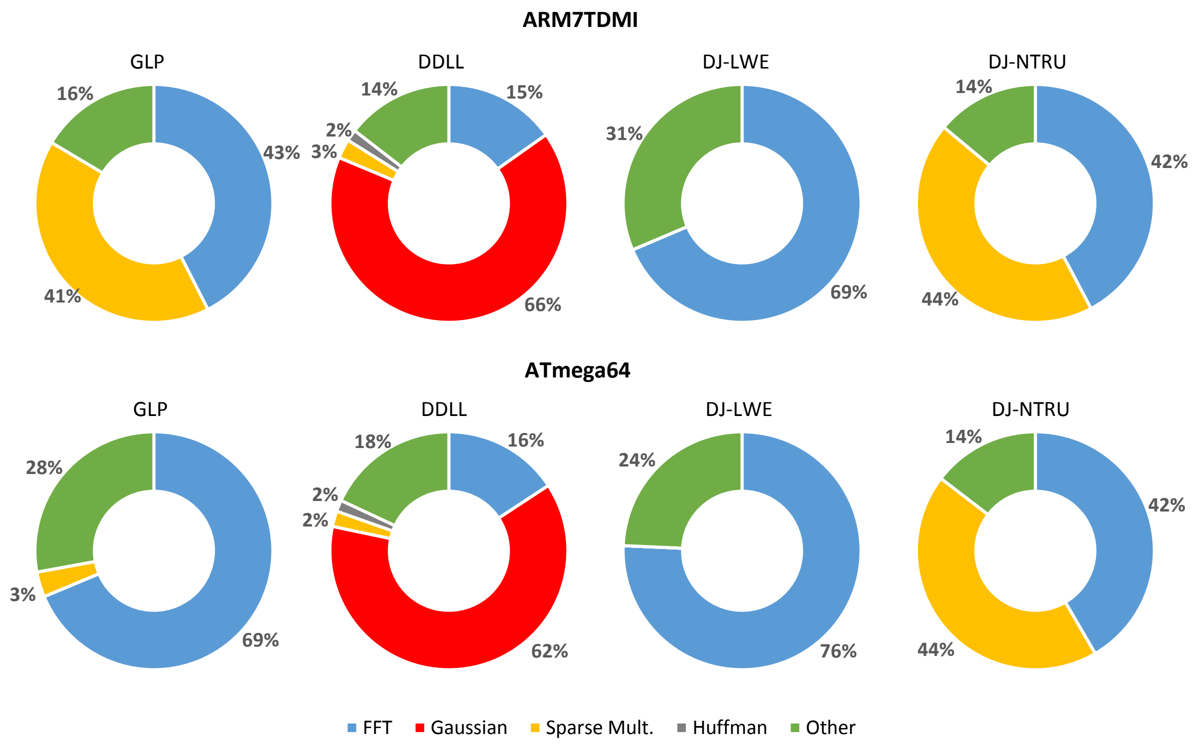

Fig. 6. The portions of important computation parts in the processing time of the protocols; Numerical results are provided in Table 9.

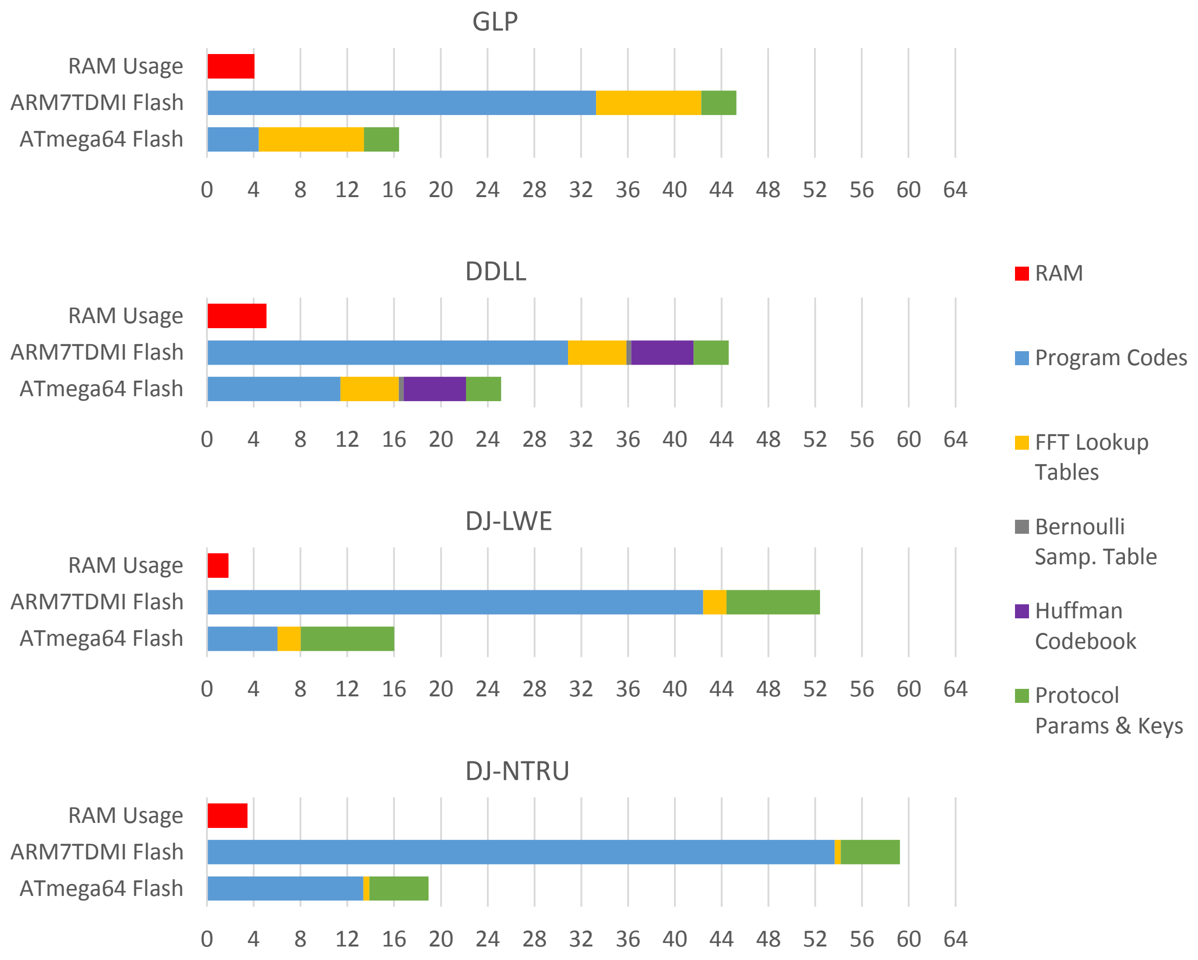

Fig. 7. RAM and flash memory usage of the authentication protocols; Numerical results are provided in Table 7.

6 Conclusion

In this paper, we have implemented efficient lattice-based authentication protocols on a typical smart card and a microcontroller. In addition, we have implemented fast Fourier transform under various configurations, and discrete Gaussian sampling using different algorithms. These implementations enjoy special optimizations regarding our constrained devices. Our results show that lattice-based schemes are efficient enough to be used practically on such constrained environments. Moreover, our analysis on the performance of different methods to perform primitives of lattice-based cryptography may be applied to design new lattice-based schemes for similar platforms. We have also measured the performance of a provably-secure lattice-based encryption in comparison to the well-known NTRU encryption. These results indicate that provably-secure lattice-based encryption is comparable to and even better than NTRU in terms of the running time, while NTRU's security is heuristic and it is patent-encumbered.

Bibliography

[ABF⁺08] Ali Can Atici, Lejla Batina, Junfeng Fan, Ingrid Verbauwhede, and S Berna Ors Yalcin. Low-cost implementations of NTRU for pervasive security. In *Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008. International Conference on*, pages 79–84, 2008.

[ABGV08] Ali Atici, Lejla Batina, Benedikt Gierlichs, and Ingrid Verbauwhede. Power analysis on NTRU implementations for RFIDs: First results. 2008.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 99–108, 1996.

[BCE⁺01] Daniel V. Bailey, Daniel Coffin, Adam Elbirt, Joseph H. Silverman, and Adam D. Woodbury. NTRU in Constrained Devices. In Çetin K. Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2001*, number 2162 in Lecture Notes in Computer Science, pages 262–272. Springer, 2001.

[BCG⁺13] Johannes Buchmann, Daniel Cabarcas, Florian Göpfert, Andreas Hülsing, and Patrick Weiden. Discrete Ziggurat: A Time-Memory Trade-off for Sampling from a Gaussian Distribution over the Integers. Technical Report 510, 2013.

[CLRS10] Pierre-Louis Cayrel, Richard Lindner, Markus Rückert, and Rosemberg Silva. Improved Zero-Knowledge Identification with Lattices. In Swee-Huay Heng and Kaoru Kurosawa, editors, *Provable Security*, number 6402 in Lecture Notes in Computer Science, pages 1–17. Springer, 2010.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice Signatures and Bimodal Gaussians. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, number 8042 in Lecture Notes in Computer Science, pages 40–56. Springer, 2013.

[DJ13] Mohammad Sadeq Dousti and Rasool Jalili. Efficient Statistical Zero-Knowledge Authentication Protocols for Smart Cards Secure Against Active & Concurrent Quantum Attacks. *Cryptology ePrint Archive, Report 2013/709*, 2013.

[FS87] Amos Fiat and Adi Shamir. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, number 263 in Lecture Notes in Computer Science, pages 186–194. Springer, 1987.

[GFS⁺12] Norman Göttert, Thomas Feller, Michael Schneider, Johannes Buchmann, and Sorin Huss. On the Design of Hardware Building Blocks for Modern Lattice-Based Encryption Schemes. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, number 7428 in Lecture Notes in Computer Science, pages 512–529. Springer, 2012.

[GLP12] Tim Güneysu, Vadim Lyubashevsky, and Thomas Pöppelmann. Practical Lattice-Based Cryptography: A Signature Scheme for Embedded Systems. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, number 7428 in Lecture Notes in Computer Science, pages 530–547. Springer, 2012.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for Hard Lattices and New Cryptographic Constructions. page 197. ACM Press, 2008.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In Joe P. Buhler, editor, *Algorithmic Number Theory*, number 1423 in Lecture Notes in Computer Science, pages 267–288. Springer, 1998.

[HS06] Jeffrey Hoffstein and Joseph H. Silverman. Speed enhanced cryptographic method and apparatus, April 2006. U.S. Classification 380/28, 708/255, 380/44; International Classification G09C1/00, G06F1/02, G06F7/72, H04L9/30, H04L9/00; Cooperative Classification G06F7/725, G06F7/723, H04L9/30; European Classification G06F7/72F1, G06F7/72E, H04L9/30.

[Kap06] Jens-Peter Kaps. *Cryptography for ultra-low power devices.* PhD thesis, WORCESTER POLYTECHNIC INSTITUTE, 2006.

[KTX08] Akinori Kawachi, Keisuke Tanaka, and Keita Xagawa. Concurrently Secure Identification Schemes Based on the Worst-Case Hardness of Lattice Problems. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, number 5350 in Lecture Notes in Computer Science, pages 372–389. Springer, 2008.

[LM06] Vadim Lyubashevsky and Daniele Micciancio. Generalized Compact Knapsacks Are Collision Resistant. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, number 4052 in Lecture Notes in Computer Science, pages 144–155. Springer, 2006.

[LMPR08] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. SWIFFT: A Modest Proposal for FFT Hashing. In Kaisa Nyberg, editor, *Fast Software Encryption*, number 5086 in Lecture Notes in Computer Science, pages 54–72. Springer, 2008.

[LP11] Richard Lindner and Chris Peikert. Better Key Sizes (and Attacks) for LWE-Based Encryption. In Aggelos Kiayias, editor, *Topics in Cryptology – CT-RSA 2011*, number 6558 in Lecture Notes in Computer Science, pages 319–339. Springer, 2011.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, number 6110 in Lecture Notes in Computer Science, pages 1–23. Springer, 2010.

[LSCH10] Mun-Kyu Lee, Jeong Eun Song, Dooho Choi, and Dong-Guk Han. Countermeasures against power analysis attacks for the NTRU public key cryptosystem. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 93(1):153–163, 2010.

[Lyu08] Vadim Lyubashevsky. Lattice-Based Identification Schemes Secure Under Active Attacks. In Ronald Cramer, editor, *Public Key Cryptography – PKC 2008*, number 4939 in Lecture Notes in Computer Science, pages 162–179. Springer, 2008.

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures. In Mitsuru Matsui, editor, *Advances in Cryptology – ASIACRYPT 2009*, number 5912 in Lecture Notes in Computer Science, pages 598–616. Springer, 2009.

[Lyu12] Vadim Lyubashevsky. Lattice Signatures without Trapdoors. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, number 7237 in Lecture Notes in Computer Science, pages 738–755. Springer, 2012.

[Mon08] Mariano Monteverde. *NTRU software implementation for constrained devices.* PhD thesis, Katholieke Universiteit Leuven, Belgija, 2008.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, number 7237 in Lecture Notes in Computer Science, pages 700–718. Springer, 2012.

[MV03] Daniele Micciancio and Salil P. Vadhan. Statistical Zero-Knowledge Proofs with Efficient Provers: Lattice Problems and More. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, number 2729 in Lecture Notes in Computer Science, pages 282–298. Springer, 2003.

[OPG14] Tobias Oder, Thomas Pöppelmann, and Tim Güneysu. Beyond ECDSA and RSA: Lattice-based Digital Signatures on Constrained Devices. In *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference*, DAC '14, page 110:1–110:6, New York, NY, USA, 2014. ACM.

[PR06] Chris Peikert and Alon Rosen. Efficient Collision-Resistant Hashing from Worst-Case Assumptions on Cyclic Lattices. In Shai Halevi and Tal Rabin, editors, *Theory of Cryp-*

tography, number 3876 in Lecture Notes in Computer Science, pages 145–166. Springer, 2006.

[SCJD11] Rosemberg Silva, Antonio C. de A. Campello Jr., and Ricardo Dahab. LWE-based identification schemes. In *Information Theory Workshop (ITW), 2011 IEEE*, pages 292–296, 2011.

[SCL11] Rosemberg Silva, Pierre-Louis Cayrel, and Richard Lindner. Zero-knowledge identification based on lattices with low communication costs. *XI Simpósio Brasileiro de Segurançada Informação e de Sistemas Computacionais*, 8:95–107, 2011.

[Sho94] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. IEEE, 1994.

[SRVV13] Sujoy Sinha Roy, Frederik Vercauteren, and Ingrid Verbauwhede. High Precision Discrete Gaussian Sampling on FPGAs. In *Selected Areas in Cryptography*. 2013.

[War03] Henry S. Warren. *Hacker's delight*. Addison-Wesley Professional, 2003.

[XT09] Keita Xagawa and Keisuke Tanaka. Zero-Knowledge Protocols for NTRU: Application to Identification and Proof of Plaintext Knowledge. In Josef Pieprzyk and Fangguo Zhang, editors, *Provable Security*, number 5848 in Lecture Notes in Computer Science, pages 198–213. Springer, 2009.

[YHK$^+$11] Yu Yao, Jiawei Huang, Sudhanshu Khanna, Abhi Shelat, Benton Highsmith Calhoun, John Lach, and David Evans. A Sub-0.5V Lattice-Based Public-Key Encryption Scheme for RFID Platforms in 130nm CMOS, 2011.

APPENDIX

A Java Card Implementation

In this section, we report the results of implementing lattice-based authentication protocols on Java Card. Java Card is an specification, defined and maintained by Oracle Corporation, which enables smart card issuers or the end user to program a smart card, rather than using a fixed functionality programmed at the manufacturing time. Java Cards have a cryptography library to perform a wide range of encryption algorithms, as well as some primitives such as modular exponentiation and big-number operations. However, our implementations could not get advantage of these API because lattice-based schemes rely only on modular addition and multiplication on small numbers. Thus, the running times of lattice-based authentication protocols are very large, due to numerous arithmetic operations which are done using high-level JVM instructions. Nevertheless, this shows that lattice-based scheme can also be implemented on widely-used Java Cards. According to the detailed running times of Figure 6, an approach to achieve desired efficiency is that Java Card operating systems provide APIs for heavy primitives of lattice-based schemes, such as FFT or discrete Gaussian sampling.

Lattice-based authentication protocols are implemented as Java Card applets, and executed on a Feitian FT-Java/H10CR smart card. Feitian FT-Java/ H10CR supports Java Card 2.2.2 and Global Platform 2.1.1 specifications. It has 3 KB of transient RAM and 160 KB of EEPROM. This card has dual interfaces of ISO/IEC 7816 (contact) and ISO/IEC 14443 (contactless). We have measured the running times when the card is used via both interfaces. Figure 8 shows the running time of Java Card executions. The processing time using the contactless interface is larger because the clock rate is lower in this case.

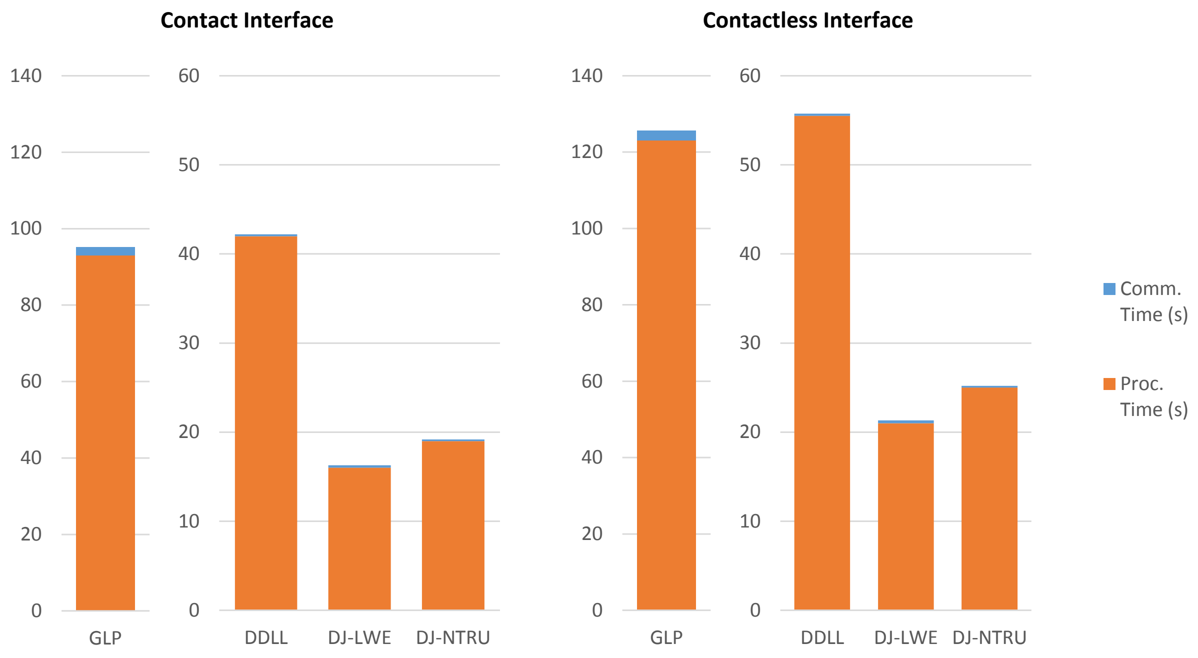

Fig. 8. The overall running time of the authentication protocols on Java Card, separated into processing and communication time

B Numerical Results

In this section, we have provided our detailed time and space measurements on the ARM7TDMI smart card and ATmega64 microcontroller. The processing times are both in clock cycles and milliseconds for better comparison. Tables 4 and 5 contain the results of running fast Fourier transform and discrete Gaussian sampling, respectively. Table 6 shows the performance of lattice-based encryption schemes. RAM and flash memory usages of implemented authentication protocols are provided in Table 7. Lastly, Tables 8 and 9 contain the timing results of these protocols. Table 9 has also a column “Num.” which shows the number of invocations for each part, as well as the number of repetitions for the whole protocol.

Table 4. Running time of FFT with various typical parameters using different algorithms to compute *mod* operation (times are in millisecond.)

		Compiler Gen.		**Small 2 Power**		**Double-size Mult.**		**Our Method**	
Device	**Parameter**	**Cycle**	**Time**	**Cycle**	**Time**	**Cycle**	**Time**	**Cycle**	**Time**
ARM7TDMI	128-257	44,053	8.95						
	256-7681	164,070	33.3	161,012	32.7	132,023	26.8	109,306	22.2
	512-12289	332,300	67.5	368,228	74.8	287,227	58.8	260,521	53.0
	512-8383489	762,489	155	775,575	158	597,413	121	381,951	77.6
ATmega64	128-257	56,916	7.11						
	256-7681	949,552	118	1,171,989	146	1,259,799	157	754,668	94.3
	512-12289	2,072,459	259	2,650,317	331	2,765,274	346	2,207,787	276
	512-8383489	4,252,710	532	5,422,161	678	10,233,177	1,279	9,289,380	1,161

Table 5. Average running time of sampling 512 values from discrete Gaussian distributions using two algorithms of cumulative distribution table and Bernoulli sampling

Device	**Algorithm**	**Dev.**	**Cycle**	**Time**
ARM7TDMI	Cumulative	4.51	33,843	6.88 ms
	Table	215	63,917	13.0 ms
	Bernoulli	4.51	709,776	144 ms
	Sampler	215	1,131,305	230 ms
ATmega64	Cumulative.	4.51	617,600	77,2 ms
	Table	215	3,780,894	473 ms
	Bernoulli	4.51	6,869,236	859 ms
	Sampler	215	8,219,955	1,027 ms

Table 6. Running time of the lattice-based encryption schemes

		Key Generation		**Encryption**		**Decryption**	
Device	**Algorithm**	**Cycle**	**Time**	**Cycle**	**Time**	**Cycle**	**Time**
ARM7TDMI	LP-LWE	575,047	0.117 s	761,718	155 ms	226,235	46.0 ms
	NTRU	178,461,520	36.3 s	977,467	199 ms	1,418,946	288 ms
	CPA-NTRU	178,461,520	36.3 s	653,817	133 ms	576,729	117 ms
ATmega64	LP-LWE	2,770,592	0.346 s	3,042,675	380 ms	1,368,969	171 ms
	NTRU	515,652,250	64.5 s	2,824,324	353 ms	1,974,601	247 ms
	CPA-NTRU	515,652,250	64.5 s	1,414,047	177 ms	988,672	124 ms

Table 7. Flash memory and RAM usage of implemented lattice-based authentication protocols

Device	Memory	GLP	DDLL	DJ-LWE	DJ-NTRU
Both	RAM	4166 B	5202 B	1888 B	3510 B
	FFT Lookup Tables	9.00 KB	5.00 KB	2.0 KB	0.50 KB
	Bernoulli Samp. Table	–	0.40 KB	–	–
	Huffman Codebook	–	5.33 KB	–	–
	Protocol Params & Keys	3.00 KB	3.00 KB	8.00 KB	5.08 KB
ARMT7DMI	Program Codes	33.3 KB	30.9 KB	42.4 KB	53.7 KB
	Total Flash Memory	45.3 KB	44.6 KB	52.4 KB	59.3 KB
ATmega64	Program Codes	4.43 KB	11.4 KB	6.02 KB	13.4 KB
	Total Flash Memory	16.4 KB	25.1 KB	16.0 KB	18.9 KB

Table 8. Total running time of implemented lattice-based authentication protocols

Device	Time Part	GLP	DDLL	DJ-LWE	DJ-NTRU
ARM7TDMI	Proc.	2.56 s	1.11 s	0.358 s	0.424 s
	Comm.	19.1 s	1.98 s	2.58 s	1.61 s
	Overall	21.7 s	3.09 s	2.94 s	2.03 s
ATmega64	Proc.	10.8 s	5.26 s	0.562 s	0.342 s
	Comm. (est.)	3.12 s	0.338 s	0.425 s	0.283 s
	Overall	13.9 s	5.60 s	0.986 s	0.625 s

Table 9. Detailed running time of implemented lattice-based authentication protocols

			ARM7TDMI		ATmega64	
Protocol	**Part**	**Num.**	**Cycle**	**Time**	**Cycle**	**Time**
GLP	FFT	2	5,347,314	1,087 ms	59,537,940	7,442 ms
	Sparse Mult.	2	5,160,778	1,049 ms	2,963,086	370 ms
	Total	7 rep.	12,579,070	2,557 ms	86,656,602	10,832 ms
DDLL	FFT	2	833,667	169 ms	6,631,868	829 ms
	Gaussian	1024	3,620,176	736 ms	26,303,858	3,288 ms
	Sparse Mult.	2	143,360	29.1 ms	886,798	111 ms
	Huff. Enc.	1024	85,303	17.3 ms	651,755	81.5 ms
	Total	1.6 rep.	5,478,943	1,113 ms	42,069,682	5,259 ms
DJ-LWE	FFT	23	1,208,978	246 ms	3,402,324	425 ms
	Total	1 rep.	1,761,831	358 ms	4,495,186	562 ms
DJ-NTRU	FFT	20	881,060	179 ms	1,138,320	142 ms
	Sparse Mult.	2	913,016	186 ms	1,197,910	150 ms
	Total	1 rep.	2,084,943	424 ms	2,735,521	342 ms