Implementation and Comparison of Lattice-based Identification Protocols on Smart Cards and Microcontrollers

Ahmad Boorghany
boorghany@ce.sharif.edu

Rasool Jalili
jalili@sharif.edu

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

February 4, 2014

Abstract. Most lattice-based cryptographic schemes which enjoy a security proof suffer from huge key sizes and heavy computations. This is also true for the simpler case of identification protocols. Recent progress on ideal lattices has significantly improved the efficiency, and made it possible to implement practical lattice-based cryptography on constrained devices like FPGAs and smart phones. However, to the best of our knowledge, no previous attempts were made to implement lattice-based schemes on smart cards. In this paper, we report the results of our implementation of several state-of-the-art and highly-secure lattice-based identification protocols on smart cards and microcontrollers. Our results show that only a few of such protocols fit into the limitations of these devices. We also discuss the implementation challenges and techniques to perform lattice-based cryptography on constrained devices, which may be of independent interest.

Keywords: Smart Card Implementation, Lattice-based Cryptography, Post-quantum Cryptography, Identification Protocol, Constrained Devices.

1 Introduction

Since the seminal work of Ajtai [Ajt96] who was the first to prove security of some lattice-based cryptography scheme, the research in this direction is quickly growing. It is one of the main candidates for post-quantum cryptography. No efficient quantum algorithm has been found yet to break such schemes. In contrast, widely used schemes such as RSA, ElGamal, or ECC-based constructions will be defenseless [Sho94] upon probable appearance of quantum computers.

Furthermore, lattice-based schemes are asymptotically more efficient than the competing number theoretic ones. For example, to achieve 128-bit security in RSA encryption, a 3072-bit modulus should be used [BBB+11]. Notice that RSA moduli above 2048 bits are uncommon because they are a bit slow. Almost all smart cards support RSA-2048 encryption as maximum security. In an implementation to estimate the running time of RSA-3072 on a native smart card (see Section 6.1 for the platform setting), it took more

than 2.4 seconds to decrypt a single block[1]. The same experiment for 128-bit secure LP-LWE encryption, which is a very efficient lattice-based scheme, showed a running time of 77 ms. This motivates the use of lattice-based schemes for high-security requirements.

Originally, most lattice-based cryptographic schemes were too inefficient in practice. That involved multiplication of large matrices and a few hundred kilo-bytes were needed to store a single key. When using general lattices, storage and computation requirements have quadratic order ($\mathcal{O}(n^2)$ in lattice dimension n). A major event in the development of lattice-based cryptography is the introduction of ideal lattices. An ideal lattice has some extra algebraic structure, which is used to reduce the key size and computation time to quasi-linear order. These schemes enjoy a security proof which assumes worst-case hardness of basic problems on ideal lattices.

Recent noticeable improvements to the efficiency of lattice-based cryptographic constructions have made it possible to bring these schemes to smart card. Smart cards have limited resources, typically consist of an 8-bit processor running at 30 MHz and have 4 KB of RAM. To the best of our knowledge, it is the first time that a lattice-based cryptographic scheme is implemented on a smart card. We have implemented three identification protocols among many proposed ones in the literature. The main selection criterion was to achieve high level of security. Thus, the focus is on zero-knowledge like protocols[2], i.e. protocols, which have a guarantee not to leak the secret key. Implemented protocols are also provably secure, which means that their security is theoretically based on an underlying more-studied problem. Most concrete parameters are chosen in such a way that provide 128-bit security, which is believed to be immune beyond 2030 [BBB+11].

The implementation of identification protocols are done in three environment settings. The first two are on a Java smart card which is accessed through contact and contactless interfaces. Java Cards have an operating system which interprets user-written programs in Java language. The results show that the interpretation overhead has a great impact upon the efficiency of implemented identification protocols. The most efficient one takes about 16 seconds overall when using a Java Card. A better solution is to use a native smart card in which there is no OS or platform overhead, and the protocol executes in native processor instructions. Unfortunately, we could not obtain such a smart card because native cards are only provided to companies developing smart card OS. This also require signing a restrictive non-disclosure agreement, and paying often more than thirty thousand dollars. Instead, we have chosen a microcontroller which estimates the performance of an average-level native smart card. The third setting in our implementations, is on AVR ATxmega64A3 microcontroller which contains an 8-bit processor and run at 32 MHz clock speed. AVR ATxmega64A3 has flash memory to store the program which is read-only when the microcontroller is running. Besides, there is a small read/write EEPROM. This difference to smart cards is not so important for our implementation because in the case of identification protocols, there is no need to write anything to non-volatile memories. To justify the use of AVR ATxmega64A3 on behalf of a native smart card, let us note that some advanced smart cards, which are being used in production, run at above 50 MHz, have flash memory instead of ROM and EEPROM, and have 32-bit processors (like Infineon SLE 70 and 88 families, or NXP smart cards with ARM Cortex-M0 chips).

[1] To implement an identification protocol, the decryption procedure should be run on the smart card which is heavier than encryption in the case of RSA. Note that CRT method is also used to improve RSA decryption performance.

[2] Technically, including zero-knowledge proofs [GMR89], witness-hiding protocols [FS90], and even those which transfer messages that are statistically independent from the secret key [DDLL13].

1.1 Our Contribution

The main contribution of this paper is the implementation of state-of-the-art lattice-based identification schemes on smart cards[3]. This experiment was a bit different from lattice-based implementations on other constrained devices (see Section 1.2 for a short review). For example, the computation speed in smart cards is much slower than typical field-programmable field arrays (FPGA). Instead, there is relatively larger read-only memory in smart cards, which can be used to store lookup tables and accelerate the computation. There are also differences between implementations on smart cards versus on ARMv7 processor, which is utilized in smart phones and tablets. The single-instruction multiple-data (SIMD) technology in this processor is much stronger than tiny processors in smart cards. As a consequence, various techniques may be used for design and implementation of cryptographic schemes, when targeting different types of constrained devices. The used implementation techniques for smart cards are described in Sections 4 and 5, which may be of independent interest.

The other outcome of the paper is the set of performance results, presented in Section 6. The performance results of the implemented identification protocols show that most lattice-based identification protocols does not fit into the computational limitation of smart cards. However, there are some highly efficient candidates that are ready to be used in practice. There is also a separate report on the performance of lattice-based cryptography primitives on smart cards. That includes LP-LWE encryption and decryption, and fast Fourier transforms with different degrees, which are important building blocks in various lattice-based schemes.

1.2 Related Work

The first lattice-based identification protocol was proposed by Micciancio and Vadhan [MV03]. Their protocol was a zero-knowledge proof based on the hardness of the approximate closest vector problem (CVP) on general lattices. Each run of this protocol has $1/2$ soundness error, meaning that it should be repeated several times to achieve negligible error. Unfortunately, this repetition should be sequential, otherwise the zero-knowledge property would be lost. Kawachi et al. [KTX08] then created a lattice-based commitment scheme which was a building block for another zero-knowledge identification protocol based on Stern protocol [Ste96]. Their ID scheme had $2/3$ soundness error, but they proved that the repetition could be done in parallel. However, even parallel repetition requires a lot of computation resources and leads to transferring big messages. After that, Lyubashevsky [Lyu08] proposed a more efficient ID protocol based on the shortest vector problem (SVP) on ideal lattices. Later, he claimed that the source of inefficiency in lattice-based ID schemes is that they send separate response blocks for each bit of the challenge. So he proposed another identification scheme [Lyu09] solving this problem by using the challenge string as a whole. Each run of this protocol had 0.63 completeness error, which was solved by repeating some parts in parallel.

Lyubashevsky used Fiat-Shamir transformation in [Lyu09] to convert the proposed ID scheme to an efficient lattice-based signature. Briefly, Fiat-Shamir transformation [FS87] is a technique to replace verifier in an identification protocol with a hash function which produces pseudo-random challenges. Therefore, the transcript of this simulated protocol can be verified publicly, making a digital signature scheme. This signature scheme is

[3] Implementation source codes can be found here: `http://ce.sharif.edu/~boorghany/latticeid`

provably-secure under random oracle model. The same technique followed in [Lyu12, GLP12, DDLL13] to reach more efficient lattice-based signatures, but the authors did not point explicitly to the underlying identification protocol. By converting above signature schemes back to identification protocols, we reach highly efficient lattice-based ID schemes. In terms of identification schemes, [Lyu12] reduced the communication complexity from $\tilde{\mathcal{O}}(n^{1.5})$ in [Lyu09] to $\tilde{\mathcal{O}}(n)$. Güneysu et al. [GLP12] added a compression function to the scheme of [Lyu12] to remove some less important bits of the messages, while keeping the security proof correct. They also implemented their (signature) scheme on FPGAs. Finally, Ducas et al. [DDLL13] proposed the most efficient scheme in this series, which uses Gaussian distribution to shorten the message (signature) size even more.

There are some other lattice-based identification schemes [XT09, CLRS10, SCL11, SCJD11] where all have major soundness errors in a single run. So they lose much efficiency when repeated several times. Recently, Dousti and Jalili [DJ13] designed a 5-round zero-knowledge identification protocol based on lattices, which does not need repetition. In this protocol, the verifier asks the prover to decrypt a challenge ciphertext, but first convinces her that he already knows the plain-text.

There are many results for implementing provably-secure lattice-based schemes on constrained devices. Lots of effort devoted to implement fast Fourier transform (FFT) on FPGAs [PG12, GCB13, APS13, RVM+13]. FFT is an important building block for efficient lattice-based cryptosystems. Sinha Roy et al. [SRVV13] proposed an FPGA implementation of high-precision discrete Gaussian sampling. Göttert et al. [GFS+12], and Pöppelmann and Güneysu [PG13] both implemented the LP-LWE encryption (an efficient lattice-based encryption proposed by Lindner and Peikert [LP11]) on FPGAs. Towards this end, they also implemented efficient FFT and discrete Gaussian sampling. As mentioned before, Güneysu et al. [GLP12] introduced an FPGA implementation of their lattice-based signature scheme.

1.3 Smart Cards

Smart card is some kind of constrained devices, containing an integrated circuit with limited resources. A simple type of smart card is a memory card. It has non-volatile writable memory and a tiny chip which maintains the memory structure and may do elementary authentication. A more advanced type is microprocessor card. Actually, it has a more powerful processor which is used to run complicated protocols and various cryptographic algorithms. A coprocessor is likely to be provided in order to accelerate cryptography operations, such as AES or RSA encryption. Three memory kinds are available in a typical smart card: RAM, ROM, and a non-volatile read/write memory, which is often EEPROM. The size of RAM is usually small (2-16 KB) and its content loses when the card is removed. However, read/write on RAM is roughly 20 times faster than other types of memory. ROM is a write-once memory which is used to load the operating system (OS) while manufacturing. Its read-time is typically 100-300 ns, and it varies in size from 64 KB to a few hundred kilo-bytes. EEPROM which is usually smaller in size, stores data and programs of the user (see below). It is capable of reading and writing, but the write-time is considerably long (5-10 ms/page) while the read-time is roughly as fast as ROM. A processor, which usually has an 8-bit architecture, executes instructions from ROM with a clock speed of 10-60 MHz.

There are generally two types of OS for smart card. The first type runs a fixed set of functionalities for the end user. For example, Mifare and PKCS11 cards do some predetermined and standardized operations such as storing user data, authentication, encryption,

etc. The second type is programmable, which allows issuers to load a program to EEPROM and run their desired functionality. This idea is also extended to load multiple applications by different issuers, in order to form a multi-purpose smart card. The variety of smart card manufacturers, issuers, and application designers have led to generation of standard specifications and platforms in which the OS implements interfaces and interprets user applications in a virtual running environment. The most wide-spread platform is Oracle's Java Card where the application is written in a simplified Java language and compiled to byte-codes, to prepare an encapsulated Java Card applet. Then the smart card OS runs a minimal Java virtual machine to execute loaded applets for different issuers.

The high-level method of communication between applications on the smart card and the reader, is through sending application protocol data units (APDU), which is a 256-byte packet, described in ISO/IEC 7816 standard. In general, the reader sends an APDU request to the card, which is processed by loaded application (e.g., a Java Card applet), and then replied by an APDU response. 256 bytes are not enough for most applications so a new version, called extended APDU, is provided supporting up to 65 KB. Extended APDUs are used to implement the identification protocols in Section 5, as a means to reduce the communication time.

Outline. Notation and a brief mathematical background are explained in Section 2. Lattice-based cryptography primitives which are building blocks in the implemented identification protocol are introduced in Section 3. Section 4 is devoted to put the applied implementation techniques into words. The details of implemented protocols are specified in section 5 which also contains protocol-specific implementation techniques. Section 6 is a report of performance results. After indicating our platform settings, the timing and storage results are outlined for each protocol. Additionally, there are separate performance results for lattice-based cryptography primitives.

2 Preliminaries

2.1 Notation

$\mathbb{Z}_q$ is the set of integers from $\lfloor -q/2 \rfloor$ to $\lfloor q/2 \rfloor$. $\mathbb{Z}_q[x]$ denote the set of polynomials with coefficients in $\mathbb{Z}_q$. The polynomial ring $\mathbb{Z}_q[x]/\langle x^n + 1 \rangle$ is denoted by $\mathcal{R}_q$. It contains all polynomials of degree less than n with coefficients in $\mathbb{Z}_q$, as well as two ring operations, which are polynomial addition and multiplication modulo $x^n + 1$. In some cases, $\mathcal{R}_q$ denotes $\mathbb{Z}_q[x]/\langle x^N - 1 \rangle$ which is noted explicitly. Polynomials in $\mathcal{R}_q$ and vectors in $\mathbb{Z}_q^n$ are simply mapped to each other, so they may be used interchangeably through the text. Vectors or polynomials are written in little bold letters while matrices are determined by big bold letters.

2.2 Integer Lattices

An integer lattice (shortly, a lattice) is a set of discrete points in $\mathbb{Z}^n$ which form an additive subgroup of $\mathbb{Z}^n$. Alternatively, a lattice Λ can be defined as linear combination of n linearly-independent vectors $\boldsymbol{v_1}, ..., \boldsymbol{v_n} \in \mathbb{Z}^n$ with integer coefficients, i.e., $\Lambda = \{\sum_{i=1}^{n} x_i \boldsymbol{v_i} \mid x_1, ..., x_n \in \mathbb{Z}\}$. Vectors $\boldsymbol{v_i}$'s are called lattice base vectors. They are usually put as columns in a matrix $\mathbf{B} \in \mathbb{Z}^{n \times n}$. The lattice generated by $\mathbf{B}$ is denoted by $\mathcal{L}(\mathbf{B}) = \Lambda = \{\mathbf{B}\boldsymbol{x} \mid \boldsymbol{x} \in \mathbb{Z}^n\}$.

There are many hard problems defined on lattices. Two fundamental ones are shortest vector problem (SVP) and closest vector problem (CVP). In SVP, given a base matrix $\mathbf{B}$, it is required to find the shortest non-zero $\boldsymbol{u} \in \mathcal{L}(\mathbf{B})$ In other words, you should find the best integer coefficients of base vectors to get near to the origin [4]. $\lambda_1(\Lambda)$ represents the size of the shortest non-zero vector in Λ. When base vectors are long and highly non-orthogonal, even approximating the shortest vector is very hard. Technically, reaching to a vector whose size is smaller than $n^{c/\log\log n} \cdot \lambda_1(\Lambda)$ is NP-Hard [HR07] for any constant c. It is conjectured that there is no efficient algorithm for approximating SVP to a polynomial factor. Similar conjecture is proposed for CVP where it is required to find the closest $\boldsymbol{u} \in \mathcal{L}(\mathbf{B})$ to a target point $\boldsymbol{t}$ outside the lattice.

Using general lattices to construct cryptographic functions, usually ends to inefficient schemes with high computation and storage complexities. This is because operations are done on quadratic size matrices (e.g., the base matrix $\mathbf{B} \in \mathbb{Z}^{n\times n}$). To overcome these obstacles in application, special lattices with extra algebraic structures, are used in application. 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 $\mathbb{Z}_q$, so there is a bijection between $\mathcal{R}_q$ and $\mathbb{Z}_q^n$. It can be shown that ideals in $\mathcal{R}_q$ are mapped to a lattice in $\mathbb{Z}_q^n$, which is called an ideal lattice [LM06, PR06]. Ring-based shortest integer solution (Ring-SIS) and ring-based learning with errors (Ring-LWE) are two primary problems based on ideal lattices (defined below) that have extensive applications in building cryptography constructions. It is proved [LM06, PR06, LPR10] that these problems are hard-on-average, i.e., cannot be solved efficiently on random inputs, assuming that approximate SVP is hard in worst-case on ideal lattices.

Problem 1 (Ring-SIS) Given m random polynomials $\boldsymbol{a_1}, ..., \boldsymbol{a_m} \in \mathcal{R}_q$ and a threshold $\beta \in \mathbb{Z}$: find short polynomials $\boldsymbol{x_1}, ..., \boldsymbol{x_m} \in \mathcal{R}_q$ in which the absolute value of all coefficients is below β, and $\sum_{i=0}^{m} \boldsymbol{a_i} \cdot \boldsymbol{x_i} = 0$ (in $\mathcal{R}_q$).

Problem 2 (Ring-LWE) Given m pairs of $(\boldsymbol{a_i}, \boldsymbol{b_i}) \in \mathcal{R}_q \times \mathcal{R}_q$ where $\boldsymbol{a_i}$'s are uniformly random: decide whether $\boldsymbol{b_i}$'s are also uniformly random, or there exist an $\boldsymbol{s} \in \mathcal{R}_q$ such that $\forall i\ \boldsymbol{b_i} = \boldsymbol{a_i} \cdot \boldsymbol{s} + \boldsymbol{e_i}$. Note that $\boldsymbol{e_i}$'s are chosen independently from discrete Gaussian distribution.

The security of lattice-based schemes is often depended to the hardness of finding a relatively short vector in the lattice. The best algorithm to find short vectors in an n-dimensional lattice is BKZ 2.0 [CN11]. When trying to find a vector of length $1.006^n \cdot \lambda_1(\Lambda)$, this algorithm is estimated to take 2^{128} processing cycles [CN11] (1.006 is called a Hermite factor). A common step to set concrete parameters for a lattice-based scheme is to tune its critical Hermite factor to 1.006 in order to achieve 128-bit security.

2.3 Discrete Gaussian Distribution

The Gaussian or normal distribution is a continuous probability distribution which is defined on $x \in \mathbb{R}$: $f_{\mu,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp(\frac{-(x-\mu)^2}{2\sigma^2})$. μ determines the distribution center and σ is the standard deviation. Its discrete version is simply obtained by limiting the domain to $x \in \mathbb{Z}$, and rescaling so that the total probability equals to 1. Assuming that $\mu = 0$, each integer $x \in \mathbb{Z}$ is sampled with a probability proportional to $\exp(-x^2/2\sigma^2)$.

Note that sampling a (high-precision) continuous Gaussian and then rounding it off to the nearest integer is a completely different distribution, and substantially deviates

[4] In this paper, all length metrics are Euclidean distance. Although, most lattice results have been extended to other ℓ_p norms [Pei08].

from a discrete Gaussian distribution. It is common to ignore long-enough tails of a discrete Gaussian, i.e., sampling only from $\{-\tau\sigma, ..., \tau\sigma\}$. By choosing a suitable tail-cut factor τ, the resulting distribution has negligible difference from the ideal distribution. For example, when $\tau = 12$, the sum of ignored probabilities is less than 2^{-100}. Sampling a discrete Gaussian efficiently is a challenging job which is discussed in Section 4.1.

3 Cryptographic Primitives

3.1 Lattice-based Encryption

Provably-secure lattice-based encryptions are all based on LWE and Ring-LWE problems. The most efficient one is proposed by Lindner and Peikert [LP11], which is referred to as LP-LWE. The following procedures define LP-LWE encryption. Random polynomial $\boldsymbol{a} \in \mathcal{R}_q$ is used as a system parameter, which can be chosen by a trusted third party or generated in a multiparty random-generation protocol. $D_{\mathbb{Z}^n,\sigma}$ denotes a discrete Gaussian distribution on $\mathcal{R}_q$ where each coefficient is sampled independently with standard deviation σ. Notice that all operations are inside $\mathcal{R}_q$.

- **KeyGen**: Sample $\boldsymbol{sk}, \boldsymbol{r} \leftarrow D_{\mathbb{Z}^n,\sigma}$ which are respectively the secret key and a temporary random value. The public key is $\boldsymbol{pk} = \boldsymbol{r} - \boldsymbol{a} \cdot \boldsymbol{sk}$.
- **Encrypt** $(\boldsymbol{pk}, x)$: First, encode $x \in \{0,1\}^n$ to a polynomial $\bar{\boldsymbol{x}} \in \mathcal{R}_q$ in such a way that coefficients of $\bar{\boldsymbol{x}}$ are either 0 or $\lfloor \frac{q}{2} \rfloor$ according to whether the corresponding bit in x is 0 or 1. Moreover, generate $\boldsymbol{e_1}, \boldsymbol{e_2}, \boldsymbol{e_3} \leftarrow D_{\mathbb{Z}^n,\sigma}$. Output the pair $(\boldsymbol{c_1}, \boldsymbol{c_2}) = (\boldsymbol{a} \cdot \boldsymbol{e_1} + \boldsymbol{e_2}, \boldsymbol{pk} \cdot \boldsymbol{e_1} + \boldsymbol{e_3} + \bar{\boldsymbol{x}})$ as ciphertext.
- **Decrypt** $(\boldsymbol{sk}, (\boldsymbol{c_1}, \boldsymbol{c_2}))$: Compute $\tilde{\boldsymbol{x}} = \boldsymbol{sk} \cdot \boldsymbol{c_1} + \boldsymbol{c_2}$. Decode each coefficient to 0 if it is closer to 0 than $\lfloor \frac{q}{2} \rfloor$, and decode it to 1 otherwise.

3.2 Lattice-based Commitment

A commitment scheme is a two-phase protocol between a sender and a receiver. In the first phase, the sender commits to a value x. After that, she use x in some other computations and interactions. Finally in the second phase, she reveals x to the receiver. Any commitment scheme should have *hiding* which means the receiver cannot learn x before phase 2. It should also have *binding* meaning that the sender cannot decommit a different value x' unless the receiver being aware. By the following, we explain a ring-based version of the (non-interactive) lattice-based commitment introduced in [KTX08]. Binding property is implied from collision-resistance feature of Ring-SIS [LM06, PR06]. Hiding property is proven by the same argument in [KTX08]. Notice that $\boldsymbol{a_1}, ..., \boldsymbol{a_m}$ are public random polynomials, and all operations are performed in $\mathcal{R}_p = \mathbb{Z}_p[x]/\langle x^n + 1 \rangle$.

- **Commit**: Given a message u and a random value ρ, both in $\{0,1\}^{ml/2}$, build polynomials $\boldsymbol{u_1}, ..., \boldsymbol{u_{\frac{m}{2}}}, \boldsymbol{\rho_1}, ..., \boldsymbol{\rho_{\frac{m}{2}}}$ where the i-th coefficient of $\boldsymbol{u_j}$ (resp., $\boldsymbol{\rho_j}$) is either 0 or 1 according to the corresponding (ij)-th bit of u (resp., ρ). Send $\textsc{Com}(u;\rho) = \sum_{i=1}^{m/2} \boldsymbol{a_i} \cdot \boldsymbol{u_i} + \sum_{i=m/2+1}^{m} \boldsymbol{a_i} \cdot \boldsymbol{\rho_i}$.
- **Decommit**: Send u, ρ to the receiver. He can now compute $\textsc{Com}(u;\rho)$ and compare it with the commitment message.

3.3 Fast Fourier Transform

All identification protocols in Section 5 are based on ideal lattices, where the heaviest operation is multiplying two polynomials. School-book multiplication, which runs in $\mathcal{O}(n^2)$, is too slow for a smart card. A better solution is to apply Fast Fourier Transform (FFT) to both polynomials, multiply them coordinate-wise, and then compute FFT inverse on the result [CT65]. This method needs only $\mathcal{O}(n \log n)$ operations. There are special variants of FFT for multiplication in $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$, where n is a power of 2. That neither require any floating-point arithmetic nor need reducing the final product modulo $(x^n + 1)$[5].

Algorithm 1 shows the iterated FFT algorithm used, with some optimizations, in the implemented identification protocols. BitReverse in line 3 is a simple routine which permutes an array as follows: The new index of each element is calculated by reversing the bit-string of its old index. This can be done quickly in one pass by using a lookup table. ψ is the primitive $2n$-th root of unity in $\mathbb{Z}_q$. In other words, ψ is the smallest integer for which $\psi^{2n} \equiv 1 \pmod{q}$, so $\omega = \psi^2 \bmod q$ is the primitive n-th root of unity.

Multiplying all coefficients by ψ^i in line 2 is not included in the traditional FFT algorithm [CLRS01]. Using the general form of FFT multiplication, one should add extra zero terms to obtain an equivalent polynomial with $2n$ coefficients, apply an FFT of double order $2n$, and reduce the final product modulo $(x^n + 1)$. Algorithm 1 eases this process substantially. To multiply two polynomials $\boldsymbol{a}, \boldsymbol{b}$ in $\mathcal{R}_q$, it is only required to compute $\boldsymbol{c} = \mathsf{FFT}^{-1}(\mathsf{FFT}(\boldsymbol{a}) \odot \mathsf{FFT}(\boldsymbol{b}))$, where $\odot$ is entry-wise multiplication. Notice that powers of ψ, ω are obtained quickly using lookup tables.

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

```
1  for i ← 1 to n do
2      x[i] ← ψ^i · x[i]
3  x ← BitReverse(x)
4  for s ← 1 to log_2(n) do
5      for j ← 1 to 2^(s−1) do
6          for k ← j + 1 to n step 2^s do
7              u ← x[k]
8              t ← ω^(jn/2^s) · x[k + 2^(s−1)] mod q
9              x[k] ← u + t mod q
10             x[k + 2^(s−1)] ← u − t mod q
11 return x
```

4 Implementation Techniques

The main bottleneck while implementing lattice-based schemes on smart card, is low computational power. Smart card processors mostly run at a clock speed below 30 MHz. They have also 8-bit architectures. That means, to do operations on long words the compiler should produce a relatively-large set of instructions, which take several cycles.

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

For example, multiplication and modulo operations on 32-bit arguments take 74 and 592 cycles respectively, on AVR ATxmega64A3 microcontroller. However, smart cards have quite big ROM and EEPROM (usually more than 200 KB in total), which are very fast for reading. That enables developers to enhance computation speed by using relatively-large lookup tables. For the implemented identification protocols in Section 5, lookup tables has been used for various situations, e.g., to generate Gaussian distribution, computing FFTs, Huffman coding, etc.

Computing modulo operation, which is very common in the implemented protocols, involves a time-consuming division. It is possible to cumulate a few consecutive operations and perform one modulo on the result. Minimizing modulo operations should be done carefully to prevent overflowing the result. You can also add some conditions to detect when it is necessary to reduce. These conditions have often less overhead on native smart cards. However, this technique is limited on Java Cards due to noticeable interpretation overhead. Adding two comparisons to detect the necessity of modulo operation leads to more overall time. There is also a trade-off between fewer modulo operations and using wider words (e.g., 32-bit). Though for the case of 8-bit processors, most of the time it is better to increase the number of modulo operations instead of running all operations (including additions and multiplications) on larger words.

Special modulo operations can be interestingly computed by using simple bit operations [LMPR08]. For example, in order to reduce a signed 16-bit value x modulo 257, we can compute $(x \wedge 255) - (x \gg 8)$ which has an equivalent result. Note that its result lay in the range $\{-127, ..., 383\}$ which is well for intermediate results. By keeping all values between -128 to 128 (modulo 257), multiplications can be fully done with 16-bit arithmetic without any overflows and there is no need to 32-bit arithmetic. Considering 8-bit architecture of target processors, this leads to a noticeable reduction in computation time. However, these techniques are unsuitable for Java Cards and increase the total computation time. Unfortunately, Java specification forces Java virtual machine to perform 32-bit operations while evaluating an expression, regardless of involved variables size.

There are other performance limitations on smart cards. Tiny RAM capacity leads to using EEPROM to store some temporary values, which in turn affect running time. Communication time is another performance bottleneck specially on native cards. It is always better to use extended APDU (if supported by the hardware), instead of transferring multiple simple APDUs. Excluding Java Card (where the computational overhead dominates savings on communication time), it is a good idea to shorten APDU size by concatenating values to form a bit stream, instead of sending values in separate bytes, which usually wastes some most-significant bits. In Protocol 2), noticeable amount of Gaussian values are transmitted during the protocol. Because the distribution of these values is known, using Huffman coding substantially reduces the communication time [DDLL13]. However, huge lookup tables are needed for this encoding. There are more protocol-specific techniques in our implementations, which are explained in Section 5.

4.1 Uniform/Gaussian Random Generation

Generating secure random numbers is a critical task in cryptographic protocols. We have used two different methods according to provided platform facilities in the implementation of identification protocols. Java Card specification provides an API to obtain uniform random values via the `RandomData` package. When passing appropriate parameters, Java Card OS is required to produce cryptographically-secure random data. By experiment, it was observed that this method is nicely fast and suitable for the identification protocols. AVR microcontroller has also APIs to generate random data, but those are not promised

to be secure-enough for cryptographic usage. Instead, we have implemented an algorithm similar to ANSI X9.17 standard to generate uniform random data. This method uses AVR's 128-bit AES encryption accelerator: A random key k and an initial seed value s are loaded into the microcontroller. Each time new random data is needed, current time is acquired as d and a temporary value $t \leftarrow \mathsf{Enc}_k(d)$ is generated. Enc is AES encryption in this place. Then, $x \leftarrow \mathsf{Enc}_k(s \oplus t)$ is outputted as random data and the seed is updated as $s \leftarrow \mathsf{Enc}_k(x \oplus t)$. Cryptographic coprocessor can run in parallel to the main processor and fill randomness pool, so random number generation is done with almost no overhead.

Generating discrete Gaussian numbers (see Section 2.3) is a more complex and resource-consuming operation. To this end, there are two approaches providing contrasting time/ memory tradeoffs. In the first approach [GPV08], one chooses a uniform integer from $\{-\tau\sigma, ..., \tau\sigma\}$. The selected integer is then accepted with probability proportional to $\exp(-x^2/2\sigma^2)$, or rejected otherwise. In the latter case, a new integer is sampled, and the process continues until one integer is accepted. This approach is slow, but does not require much memory. The second approach [Pei10] is quite fast, but requires relatively large memory. Assume that p_i is the probability that $i \in \{-\tau\sigma, ..., \tau\sigma\}$ is sampled. Consider a sufficiently large integer L. It is possible to partition $\{0, .., L\}$ into $2\tau\sigma + 1$ ranges denoted by R_i, such that the size of R_i is proportional to p_i. Now if a random number generator produces uniform $x \in \{0, ..., L\}$, do binary search to find the i for which $x \in R_i$. O i at the moment, produces desired Gaussian distribution. Although there are other methods [DDLL13, DN12, GD12] that lay between these two extreme approaches, the second approach which is the fastest but consumes large memory is the best choice for smart cards. This is because smart cards are slow but have relatively large read-only memory.

5 Lattice-based Identification Protocols

Lattice-based identification protocols were reviewed in Section 1.2. Most of them [MV03, KTX08, XT09, CLRS10, SCL11, SCJD11] have a zero-knowledge base protocol with noticeable soundness error (e.g., $1/2$), and should be repeated several times (e.g., 128 times or more) to obtain a secure identification protocol. Even parallel repetition, which is secure only for [KTX08, CLRS10, SCL11], makes them very inefficient in terms of computation and communication complexities. We have chosen two most efficient protocols in [Lyu09, Lyu12, GLP12, DDLL13], which are a series of improving schemes. These two protocols were originally proposed as signature schemes. As discussed in Section 1.2, these signatures use an underlying identification protocol. By converting these schemes back to identification, we achieve the following Güneysu et al. and Ducas et al. protocols. The efficient protocol of Dousti and Jalili [DJ13] is also chosen for the implementation using LP-LWE encryption. In order to achieve better efficiency on smart cards, some improvements are applied to design and implementation of these protocols. In the following, we describe final implemented protocols and explain modifications done to the original scheme.

5.1 Güneysu et al. Protocol

Güneysu et al. identification protocol is derived from lattice-based signature scheme in [GLP12]. This identification protocol can be proven to be secure against active attacks, using same proof technique of [Lyu09]. Protocol 1 explains this identification 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.

Protocol 1 (Güneysu et al. [GLP12]) The secret key $\boldsymbol{sk}$ is consist of $\boldsymbol{s_1}, \boldsymbol{s_2} \in \mathcal{R}_{[1]}$. Corresponding public key is $\boldsymbol{pk} = \boldsymbol{a} \cdot \boldsymbol{s_1} + \boldsymbol{s_2}$. The protocol rounds are as follows. S $\rightarrow$ R denotes that the smart card sends a message to the reader.

1. **(S $\rightarrow$ R)** Generate two polynomials $\boldsymbol{y_1}, \boldsymbol{y_2} \in \mathcal{R}_{[\kappa]}$ at random. Then send $\boldsymbol{u} = \boldsymbol{a} \cdot \boldsymbol{y_1} + \boldsymbol{y_2}$ to the reader.
2. **(R $\rightarrow$ S)** Send a special random polynomial $\boldsymbol{c}$ in which 32 coefficients are either $+1$ or -1 and all others are zero.
3. **(S $\rightarrow$ R)** 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 $\boldsymbol{z_2}$ fell outside $\mathcal{R}_{[\kappa-32]}$, 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 if $\boldsymbol{u} = \boldsymbol{a} \cdot \boldsymbol{z_1} + \boldsymbol{z_2} - \boldsymbol{pk} \cdot \boldsymbol{c}$.

As it is seen in round 3, smart card may reject to respond in order to prevent information leakage about the secret key. In this case, the reader should re-run the protocol from the beginning until a valid response is received. One may suggest to make this repetition parallel (like [Lyu09]), but note that round 1 takes long time to be computed and parallel runs increase the average time of the protocol. Moreover, saved communication time is negligible in comparison to computation time for this protocol.

In the original scheme [GLP12], authors used a compression function to reduce the size of $\boldsymbol{z_2}$. They removed most of the least-significant bits in $\boldsymbol{z_2}$ without disturbing the security proof. Later, [DDLL13] showed that this kind of compression has security weaknesses. Although compression of $\boldsymbol{z_2}$ significantly reduces the communication, it is eliminated in protocol 1 because it forces large computation overhead. To implement the compression function in the smart cards, at least two more FFT conversions are needed, which alone take a few seconds. Notice that Güneysu et al. protocol does not need any Gaussian sampling at all.

As reported in Section 6.3, Protocol 1 is less efficient than other implemented protocols. Two main reasons can be mentioned for this inefficiency. First, the repetition rate of the protocol is above seven times on average. Second, this protocol works in relatively large modulus, so multiplication of numbers should be done in 64-bit words. Considering our interest on 8-bit processors, it has a great impact on the efficiency of the protocol.

5.2 Ducas et al. Protocol

Ducas et al. identification protocol is actually derived from BLISS signature scheme [DDLL13]. This scheme is the most efficient lattice-based signature after a series of work [Lyu09, Lyu12, GLP12, DDLL13]. Unfortunately, when the underlying identification protocol is extracted (see the discussion in Section 1.2), the proof technique of [Lyu09] cannot be used. Thus, only passive security is provable. However, this is just a lack of proof, and the identification protocol is still worth to be implemented.

Protocol 2 explains the Ducas et al. identification 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 $\zeta(q-2) = 1 \pmod{2q}$. Ducas et al. [DDLL13] used a compression technique which drops least-significant bits of some messages and reducing them modulo p, which is a small integer. $\lfloor x \rceil_d$ denotes the value obtained by dropping d least-significant bits from x.

Protocol 2 (Ducas et al. [DDLL13]) Polynomials $\boldsymbol{f}, \boldsymbol{g}$ are made during key generation, and both have exactly $\delta_1 n$ coefficients in $\{\pm 1\}$. $\boldsymbol{f}$ 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. $\mathcal{D}_{\mathbb{Z}^n,\sigma}$ is n-dimensional discrete Gaussian distribution with standard deviation σ. The protocol rounds are as follows. Note that M is a constant and $\langle ., . \rangle$ is the inner-product operation. All operations are in $\mathcal{R}_{2q}$.

1. **(S → R)** 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 $\lfloor u \rceil_d$ to the reader.
2. **(R → S)** 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. **(S → R)** 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 $\mathbf{S} = \begin{bmatrix} \boldsymbol{s_1} & \boldsymbol{s_2} \end{bmatrix}^t$, $\mathbf{Z} = \begin{bmatrix} \boldsymbol{z_1} & \boldsymbol{z_2} \end{bmatrix}$. With probability $1 \Big/ \Big(M \exp(-\frac{\|\mathbf{S} \cdot \boldsymbol{c}\|^2}{2\sigma^2}) \cosh(\frac{\langle \mathbf{Z}, \mathbf{S} \cdot \boldsymbol{c} \rangle}{2\sigma^2}) \Big)$, send $(\boldsymbol{z_1}, \boldsymbol{z_2^\dagger})$ to the reader. Otherwise, terminate the protocol.

Verification step: Reject if either $\Big|\Big| [\boldsymbol{z_1} | 2^d \cdot \boldsymbol{z_2^\dagger}] \Big|\Big| > \beta_2$, or the absolute value of any coefficient in $[\boldsymbol{z_1} | 2^d \cdot \boldsymbol{z_2^\dagger}]$ is bigger than β_∞. Then, accept if $\lfloor \zeta \boldsymbol{pk} \cdot \boldsymbol{z_1} - \zeta q \boldsymbol{c} \rceil_d + \boldsymbol{z_2^\dagger} = \lfloor \boldsymbol{u} \rceil_d$.

Like previous protocol, reader should re-run the protocol if smart card fails in round 3. To reduce the average running time of the protocol, it is better to repeat it sequentially. That is because the computation time in round 1 is relatively large, and the success probability in the first run is high.

Several optimizations are proposed in [DDLL13] which can be directly used here. In the key generation process, the public key is computed as $\boldsymbol{pk} = 2\boldsymbol{pk'}$ for some $\boldsymbol{pk'} \in \mathcal{R}_q$. Multiplying $\boldsymbol{pk} \cdot \boldsymbol{y}$ in $\mathcal{R}_{2q}$ in step 1 can be made easier by first multiplying $\boldsymbol{pk'} \cdot \boldsymbol{y}$ in $\mathcal{R}_q$ and then doubling the result. Precomputing the FFT of $\boldsymbol{pk'}$ before loading to the smart card increases efficiency. The computation time of round 3 is much less than round 1. Although, the smart card should multiply $\mathbf{S}$ by $\boldsymbol{c}$ in round 3, no FFT conversion is not necessary here. Multiplying these polynomials directly is more efficient, because $\boldsymbol{c}$ has only κ non-zero coefficients and the coefficients of $\mathbf{S}$ are all small. The acceptance probability in round 3 can be broken into two independent Bernoulli probabilities, one proportional to $1/\exp(-\frac{\|\mathbf{S} \cdot \boldsymbol{c}\|^2}{2\sigma^2})$ and the other proportional to $1/\cosh(\frac{\langle \mathbf{Z}, \mathbf{S} \cdot \boldsymbol{c} \rangle}{2\sigma^2})$. The former does not depend on $\mathbf{Z}$ and can be evaluated before computing $\boldsymbol{z_1}, \boldsymbol{z_2}$. In this case, the protocol is repeated earlier.

The coefficients of $\boldsymbol{z_1}, \boldsymbol{z_2^\dagger}$ have Gaussian distribution. To reduce the message size even further, one can send them in Huffman codes [DDLL13]. To this end, an encoding/decoding table (called code book) is used. To reduce the storage requirement, the authors have proposed intelligent procedures to efficiently produce Gaussian values using small lookup tables, with cost of a little more computation. But we decided to use more naive method of cumulative distribution table (see Section 4.1), because it has optimal running time and there is enough read-only memory to store the large table.

5.3 Dousti-Jalili Protocol

Dousti and Jalili [DJ13] have proposed a fast zero-knowledge identification scheme, using a lattice-based commitment and trapdoor function. Unfortunately, the most efficient lattice-based trapdoor function [MP12] does not fit into smart card constraints (both in terms of computation and key size). However, as noted by the authors, this protocol

can be modified slightly to use a lattice-based encryption instead of trapdoor function. In a high-level, reader encrypts a message and asks the smart card to decrypt it. By using a commitment scheme, the smart card ensures first that the reader already knows the response. Dousti-Jalili protocol is instantiated using LP-LWE [LP11] encryption (see section 3.1). Protocol 3 describes this identification scheme.

Protocol 3 (Dousti-Jalili [DJ13]) $\boldsymbol{a_1}, ..., \boldsymbol{a_m} \in \mathcal{R}_p$ and $\boldsymbol{a} \in \mathcal{R}_q$ are random polynomials determined as system parameters by a trusted authority, or by using a multiparty random-generation protocol. The key-pair $(\boldsymbol{sk}, \boldsymbol{pk})$ is actually a key pair of LP-LWE. $D_{\mathbb{Z}^n,\sigma}$ is a discrete Gaussian distribution with standard deviation σ. Commitment operations are computed in $\mathcal{R}_p$ while LP-LWE ones are in $\mathcal{R}_q$.

1. **(S $\rightarrow$ R)** Generate two $(\frac{mn}{2})$-bit random strings u, ρ. Using the commitment scheme in Section 3.2, send $\boldsymbol{c} = \textsc{Com}(u; \rho)$ to commit to u.
2. **(R $\rightarrow$ S)** Choose a random challenge $x \in \{0,1\}^n$ and encrypt it with $\boldsymbol{pk}$. That means to sample $\boldsymbol{e_1}, \boldsymbol{e_2}, \boldsymbol{e_3}$ from $D_{\mathbb{Z}^n,\sigma}$ and compute $(\boldsymbol{c_1} = \boldsymbol{a} \cdot \boldsymbol{e_1} + \boldsymbol{e_2}, \boldsymbol{c_2} = \boldsymbol{pk} \cdot \boldsymbol{e_1} + \boldsymbol{e_3} + \bar{\boldsymbol{x}})$, according to Section 3.1. Then send $(\boldsymbol{c_1}, \boldsymbol{c_2})$.
3. **(S $\rightarrow$ R)** Decrypt $(\boldsymbol{c_1}, \boldsymbol{c_2})$ to obtain $x' \in \{0,1\}^n$ and send $x' \oplus u$.
4. **(R $\rightarrow$ S)** Send $(x, \boldsymbol{e_1})$ to show that you already know the plain-text.
5. **(S $\rightarrow$ R)** Compute $\boldsymbol{e_2} = \boldsymbol{c_1} - \boldsymbol{a} \cdot \boldsymbol{e_1}$ and $\boldsymbol{e_3} = \boldsymbol{c_2} - \boldsymbol{pk} \cdot \boldsymbol{e_1} - \bar{\boldsymbol{x}}$. All $\boldsymbol{e_1}, \boldsymbol{e_2}, \boldsymbol{e_3}$ should be short (i.e., in the most expected range of Gaussian distribution). Otherwise, terminate the protocol. Finally, send ρ to decommit from u.

Verification step: If the smart card is honest, x' will be equal to x. So the reader can extract u from $x' \oplus u$. So accept if $\boldsymbol{c} = \textsc{Com}(u; \rho)$.

In the original protocol [DJ13], the reader sends all the randomness used for the encryption (i.e., $\boldsymbol{e_1}, \boldsymbol{e_2}, \boldsymbol{e_3}$) in step 4. However, it suffices to send only $\boldsymbol{e_1}$ because both $\boldsymbol{e_2}, \boldsymbol{e_3}$ can be uniquely determined. It also needs less memory on the smart card. Moreover, a precomputed FFT of $\boldsymbol{a}, \boldsymbol{a_1}, ..., \boldsymbol{a_m}$ can be loaded to the smart card. This applies to $\boldsymbol{sk}, \boldsymbol{pk}$ as well. Storing sk in FFT form may cause a little memory overhead (because its coefficients are shorter in the coefficient representation), but saves much computation time.

Applying FFT on $\boldsymbol{u_1}, ..., \boldsymbol{u_{m/2}}, \boldsymbol{\rho_{m/2+1}}, ..., \boldsymbol{\rho_m}$ in round 1 (see the commitment details in Section 3.2) seems to be a heavy task. However, these computations are modulo 257 for which there exist excellent implementation techniques to avoid modulo operation. Actually, each FFT in round 1 takes only 2 ms on a native smart card. Moreover, the resulting commitment value $\boldsymbol{c}$ can be sent without running FFT inverse. There is no security risk here, because removing a public conversion does not violate the security properties of the commitment. $\boldsymbol{c_1}, \boldsymbol{c_2}$ which are sent by the reader in step 2, can be already in FFT format. This eliminates some extra FFT computations on the smart card. Notice that $\boldsymbol{e_1}$ is sent in polynomial representation, because the smart card should verify its short length in coefficient representation. There are also two unavoidable FFT inverses when computing $\boldsymbol{e_2}, \boldsymbol{e_3}$ in step 5 to check their lengths. An advantage of this protocol over Protocol 2 is that it does not need to generate discrete Gaussians on the smart card.

6 Implementation Results

6.1 Platform Settings

Implementations were done on two sides of smart card and reader. On the reader's side, a personal computer performs computation. It had 4 GB of RAM and a 3.3 GHz Intel Core

Table 1. The running times of lattice-based cryptographic primitives on three smart card settings.
* The performance of a typical native card is estimated by implementing on AVR ATxmega64A3.

Card Type	**LP-LWE Enc.**	**LP-LWE Dec.**	**FFT-128**	**FFT-256**	**FFT-512**
Java Card (contact)	5910 ms	1245 ms	570 ms	1200 ms	2610 ms
Java Card (contactless)	6935 ms	1505 ms	730 ms	1450 ms	3375 ms
Native Card *	157 ms	77 ms	17 ms	38 ms	86 ms

i3 CPU. The PC is equipped with an ACS ACR1281U-C1 card reader which is accessed through a PC/SC driver. The reader communicates with the smart card through either ICC (contact) or PICC (contactless) interfaces, using ISO/IEC 7816 and ISO/IEC 14443 standards respectively. On the side of smart card, the protocols are implemented on both Feitian FT-Java/H10CR Java Card and AVR ATxmega64A3 microcontroller. FT-Java/H10CR supports Java Card 2.2.2 and GlobalPlatform 2.1.1 specifications. It has 3 KB of transient RAM and 160 KB of EEPROM. The dual interface enables us to use it in both contact (T=1 protocol) or contactless (TypeA/B) modes. It supports extended APDU but only has 261 bytes of APDU buffer, which means that receiving and processing data should be simultaneous.

As discussed in the introduction, the identification protocols are also implemented on AVR ATxmega64A3 microcontroller to estimate the performance on a native smart card. That is a smart card which can be programmed in native processor instructions, without the overhead of OS and interpreter. AVR ATxmega64A3 is very similar to typical native smart cards respecting resources and specification. It has an 8-bit architecture which runs at (max.) 32 MHz. It also has 64 KB of flash memory and 4 KB of SRAM. Flash memory is much like EEPROM except that flash is faster in writing. It is not a problem for our estimation because in the implementations, flash memory is only used to store the program and static data. Moreover, there are major smart card processors (e.g., Infineon SLE 70 and 88 families) which use flash memory instead of EEPROM. AVR ATxmega64A3 has also an AES encryption engine which is used to produce pseudo-random numbers (see Section 4.1). Although the running time of implementations on AVR ATxmega64A3 is close to typical native smart cards, we are not interested in the communication times in AVR ATxmega64A3. In order to estimate an overall duration of a protocol on native cards, we used the communication times of Java Card when accessed through the contact interface. In microcontroller implementation, AVR-GCC compiler with the optimization level O3 is used to build the source codes.

6.2 Primitives Performance

In this section, three primitive constructions in lattice-based cryptographic schemes are implemented separately on smart cards, and the performance results are reported. Those are LP-LWE encryption and decryption with parameters for 128-bit security, in addition to implementation of FFT on polynomials of degrees 128, 256, and 512. LP-LWE parameters are selected from [LP11, GFS$^+$12]: $(n, q, \sigma) = (256, 7681, 4.51)$. Table 1 shows the processing time of LP-LWE and different-degree FFTs on three smart card settings. For encryption tasks, it is assumed that input polynomials and parameters are already converted to FFT format. FFT-128 running time on native cards can be substantially reduced when applied modulo 257. Using the techniques of Section 4, it can be decreased from 17 ms to less than 2 ms.

Table 2. The processing, communication, and overall run time of implemented lattice-based identification protocols. The overall row indicates the total running time of protocols.
* The performance of a typical native card is estimated by implementing on AVR ATxmega64A3.

Device / Time Slice		**Protocol 1** [GLP12]	**Protocol 2** [DDLL13]	**Protocol 3** [DJ13]
Reader	Process	9 ms	8 ms	35 ms
Java Card (contact)	Process	93 s	42 s	16 s
	Comm.	2.2 s	216 ms	273 ms
	Overall	95 s	42.2 s	16.3 s
Java Card (contactless)	Process	123 s	55.5 s	21 s
	Comm.	2.6 s	258 ms	315 ms
	Overall	125.6 s	55.7 s	21.3 s
Native Card *	Process	2.7 s	604 ms	206 ms
	Comm.	(estimated the same as contact Java Card)		
	Overall	4.9 s	828 ms	514 ms

6.3 Protocols Performance

Concrete parameters for each identification protocol are specified as follows. The parameter set of Güneysu et al. protocol (Protocol 1) is chosen to be the same as SET-I in [GLP12]: $(n, p, \kappa) = (512, 8383489, 2^{14})$. Although its security is below 128 bits, the performance results in Table 2 show that it is already much inefficient for smart cards. For Ducas et al. protocol (Protocol 2) concrete parameters are the same as BLISS-I parameter set [DDLL13] $(n, q, d, p, \delta_1, \sigma, \kappa.\beta_2, \beta_\infty) = (512, 12289, 10, 24, 0.3, 215, 23, 12872, 2100)$. BLISS-I is optimized for speed and claimed to reach 128-bit security. Two instantiations of Dousti-Jalili protocol (Protocol 3) is implemented. Concrete parameters for the commitment are $(n, m, p) = (128, 20, 257)$ according to [DJ13] analysis. Parameters of LP-LWE are the same as Section 6.2, i.e., $(n, q, \sigma) = (256, 7681, 4.51)$.

Table 2 shows the timing results separated as the reader process time, smart card process time, communication time, and overall running time. The performance of identification protocols are measured in three settings: on a Java Card accessed through contact and contactless interfaces, and on AVR ATxmega64A3 microcontroller on behalf of a native smart card. In all of these settings, a PC does the computations on the reader's side. Three protocols are implemented in total, which are explained in Section 5. Figure 1 compare these results on a diagram. Note that the running time of Guneysu and Ducas schemes are on average.

Güneysu et al. protocol does not seem to be suitable for smart cards, even though there are very efficient instantiations of this scheme on FPGAs [GLP12]. Although Güneysu et al. protocol has no discrete Gaussian sampling at all, its modulus q is large. Operations on 64-bit words are required to multiply such coefficients. This is a very time-consuming job on an 8-bit smart card processor. The noticeable completeness error of the base protocol in Güneysu et al. protocol is another reason for this observation. The remaining protocols are both practical in native card setting. Table 6.3 shows RAM and EEPROM usage of each protocol. RAM usage in Java Card implementations is actually a little bigger than native cards, which is neglected. Required RAM for Protocols 1 and 2 is a little larger than available RAM in the implementations. To overcome this problem, some intermediate data are temporarily stored to EEPROM. EEPROM contains program code, lookup tables, system parameters of the protocol, and public and private keys (public key is usually needed for computations on the smart card side). Memory usages are essentially independent of contact or contactless communication. Note that in our implementations,

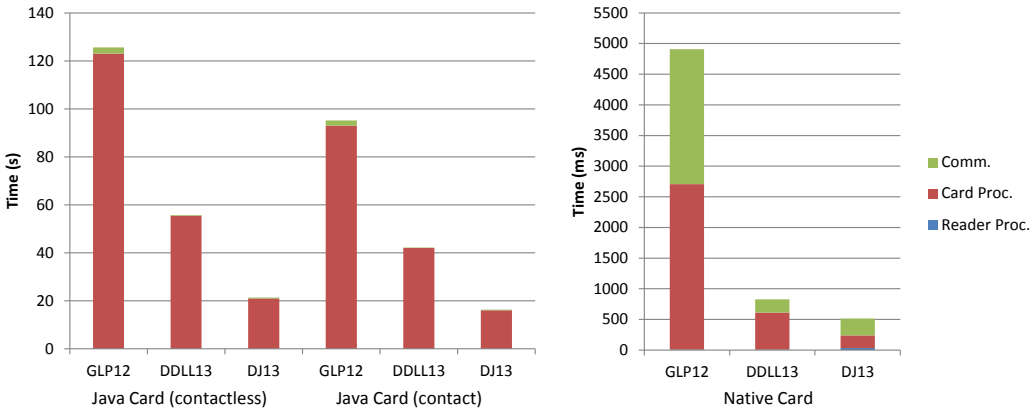

Fig. 1. The timing results of implemented lattice-based identification protocols, separated by reader process, smart card process, and communication time.

Table 3. Memory usages of implemented protocols.

Memory Usage	**Protocol 1** [GLP12]	**Protocol 2** [DDLL13]	**Protocol 3** [DJ13]
RAM	4166 B	5202 B	1888 B
EEPROM			
Lookup Tables	10.1 KB	53.5 KB	2.6 KB
System Parameters	2.7 KB	-	5.6 KB
Private Key	1024 B	2048 B	512 B
Public Key	-	1024 B	512 B
Java Card Total	15.6 KB	60.7 KB	14 KB
Native Card Total	18 KB	66.5 KB	15.8 KB

in order to decrease the computation time, we tried to use EEPROM as much as possible. Especially, the keys are stored in large representations.

References

[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.

[APS13] Aydin Aysu, Cameron Patterson, and Patrick Schaumont. Low-cost and area-efficient fpga implementations of lattice-based cryptography. 2013.

[BBB+11] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for key management–part 1: General (revision 3). *NIST special publication*, 800:57, 2011.

[CLRS01] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. The MIT press, 2001.

[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 Berlin Heidelberg, 2010.

[CN11] Yuanmi Chen and Phong Q. Nguyen. Bkz 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptol-*

ogy –ASIACRYPT 2011, number 7073 in Lecture Notes in Computer Science, pages 1–20. Springer Berlin Heidelberg, 2011.

[CT65] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[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 Berlin Heidelberg, 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.

[DN12] Léo Ducas and Phong Q. Nguyen. Faster gaussian lattice sampling using lazy floating-point arithmetic. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology –ASIACRYPT 2012*, number 7658 in Lecture Notes in Computer Science, pages 415–432. Springer Berlin Heidelberg, 2012.

[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 Berlin Heidelberg, 1987.

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 416–426, 1990.

[GCB13] Tamas Gyorfi, Octavian Cret, and Zalan Borsos. Implementing modular ffts in fpgas–a basic block for lattice-based cryptography. In *Digital System Design (DSD), 2013 Euromicro Conference on*, pages 305–308, 2013.

[GD12] Steven D. Galbraith and Nagarjun C. Dwarakanath. Efficient sampling from discrete gaussians for lattice-based cryptography on a constrained device, 2012.

[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 Berlin Heidelberg, 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 Berlin Heidelberg, 2012.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, February 1989.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. page 197. ACM Press, 2008.

[HR07] Ishay Haviv and Oded Regev. Tensor-based hardness of the shortest vector problem to within almost polynomial factors. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 469–477, 2007.

[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 Berlin Heidelberg, 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 Berlin Heidelberg, 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 Berlin Heidelberg, 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 Berlin Heidelberg, 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 Berlin Heidelberg, 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 Berlin Heidelberg, 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 Berlin Heidelberg, 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 Berlin Heidelberg, 2012.

[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 Berlin Heidelberg, 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 Berlin Heidelberg, 2003.

[Pei08] Chris Peikert. Limits on the hardness of lattice problems in p norms. *computational complexity*, 17(2):300–351, May 2008.

[Pei10] Chris Peikert. An efficient and parallel gaussian sampler for lattices. In Tal Rabin, editor, *Advances in Cryptology –CRYPTO 2010*, number 6223 in Lecture Notes in Computer Science, pages 80–97. Springer Berlin Heidelberg, 2010.

[PG12] Thomas Pöppelmann and Tim Güneysu. Towards efficient arithmetic for lattice-based cryptography on reconfigurable hardware. In Alejandro Hevia and Gregory Neven, editors, *Progress in Cryptology –LATINCRYPT 2012*,

number 7533 in Lecture Notes in Computer Science, pages 139–158. Springer Berlin Heidelberg, 2012.

[PG13] Thomas Pöppelmann and Tim Güneysu. Towards practical lattice-based public-key encryption on reconfigurable hardware. *Selected Areas in Cryptography SAC 2013*, 2013.

[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 Cryptography*, number 3876 in Lecture Notes in Computer Science, pages 145–166. Springer Berlin Heidelberg, 2006.

[RVM+13] Sujoy Sinha Roy, Frederik Vercauteren, Nele Mentens, Donald Donglong Chen, and Ingrid Verbauwhede. Compact hardware implementation of ring-lwe cryptosystems. *Cryptology ePrint Archive, Report 2013/866*, 2013.

[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 Seguranca da Informacão 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.

[Ste96] Jacques Stern. A new paradigm for public key identification. *Information Theory, IEEE Transactions on*, 42(6):1757–1768, 1996.

[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 Berlin Heidelberg, 2009.