A Lattice-Based Batch Identification Scheme

Rosemberg Silva, Pierre-Louis Cayrel, Richard Lindner

A Lattice-Based Batch Identification Scheme

Rosemberg Silva* Pierre-Louis Cayrel † Richard Lindner ‡

July 22, 2011

Abstract

This paper presents a batch version of the lattice-based identification scheme known as CLRS. Our version consists of a method for allowing a user to authenticate himself with different levels of clearance upon the choice of a subset of keys in his possession. It bears similarity with the Schnorr batch scheme, in the sense that the communication costs are kept constant, regardless of the number of keys involved. We use the hardness of a lattice problem, namely the Inhomogeneous Small Integer Solution problem (I-SIS), as security assumption.

1 Introduction

Access control consists of a set of rules for allowing or denying access to some resource. It defines precisely who can interact with what, and also the actions that such subject may perform during that interaction. Central to its implementation is the concept of the granting of rights to a subject with regarding to objects. Thus, an essential step for assuring the correct operation of such system is the existence of a reliable identification scheme, which is meant to establish trust in the identity of the subject. Authentication schemes can be built on top of something that the subject knows, as, for example, the secret key corresponding to a public key.

As the nature of the activities that an individual is allowed to perform may vary, it may be a good strategy to associate with him a set of keys that can be used in different scenarios in order to well adjust to his actual needs. As a solution to this problem, a batch version of the Schorr scheme was proposed by Gennaro *et al.* [GLSY04], exhibiting a cost slightly superior to that of a single execution. The security assumption is the hardness of computing discrete logarithms.

The quantum algorithm devised by Shor [Sho94], however, has stressed the need of diversity in hard problems used as security assumptions, by exposing the fragility of number theoretical constructions in the eventuality of quantum computers become mature. Lattices has been successfully provided options that, besides being resilient to Shor's algorithm, also exhibit worst-case to average-case reductions, increasing the confidence that random instances of schemes built upon some lattice problems are hard to break.

In this paper we present an adaptation of the CLRS [CLRS10] identification scheme to the batch setting. Our construction exhibits computational and communication costs close to those from single CLRS execution.

2 Preliminaries

In this section we give a number of definitions that are employed throughout this paper.

2.1 Lattices

Our main algorithms are built on top of lattice problems, which are described below.

Definition 2.1 (Lattice)**.** A lattice $\mathcal{L}$ is an additive subgroup of $\mathbb{R}^m$ consisting of integral linear combinations of a set of linearly independent vectors corresponding to one of its basis. The number of vectors comprising a basis is called rank of the lattice.

*University of Campinas, Brazil. E-Mail: rasilva@ic.unicamp.br.
†Center for Advanced Security Research Darmstadt, Germany. E-Mail: pierre-louis.cayrel@cased.de.
‡Technische Universität Darmstadt, Germany. E-Mail: rlindner@cs.tu-darmstadt.de.

Definition 2.2 (Shortest Vector Problem - SVP)**.** Given a lattice $\mathcal{L}$, the Shortest Vector Problem consists in finding a non-zero lattice vector with the minimum norm.

Definition 2.3 (Small Integer Solution - SIS)**.** Given an integer q, a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a real number β, the Small Integer Solution problem consists in finding a non-null vector $\mathbf{x} \in \mathbb{Z}^m$ such that $\mathbf{Ax} = 0 \bmod q$, with $\|\mathbf{x}\| \leq \beta$.

The SIS also has approximate formulations. There are several reductions from worst-case instances of hard lattice problems to average-case instances of SIS, provided that the parameters (n, m, q, β) obey certain restrictions. For a thorough discussion on this subject, please refer to [MR07].

Definition 2.4 (Inhomogeneous SIS - I-SIS)**.** Given an integer q, a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, a vector $y \in \mathbb{Z}_q^n$ and a real number β, the Small Integer Solution problem consists in finding a non-null vector $\mathbf{x} \in \mathbb{Z}^m$ such that $\mathbf{Ax} = y \bmod q$, with $\|\mathbf{x}\| \leq \beta$.

Several cryptographic constructions based on I-SIS can be found in the literature, like compression functions, identification schemes and pseudo-random number generators.

2.2 Codes

When presenting an adaptation of our batch construction to codes, the concepts below were also applied.

Definition 2.5 (Code)**.** Let $\mathbb{F}_q^n$ denote the vector space of all n-tuples over the finite field $\mathbb{F}_q$. An (n, M) code $\mathcal{C}$ over $\mathbb{F}_q$ is a subset of $\mathbb{F}_q^n$ with size M. The elements of $\mathcal{C}$ are called codewords.

Definition 2.6 (Linear Code)**.** $\mathcal{C}$ is a $[n, k]$ linear code over $\mathbb{F}_q$ when it is a k-dimensional subspace of $\mathbb{F}_q^n$. A generator matrix $\mathbf{G}$ for $\mathcal{C}$ is any $k \times n$ matrix whose rows form a basis for $\mathcal{C}$. A parity check matrix $\mathbf{H}$ for $\mathcal{C}$ is a matrix defining a linear transformation that has $\mathcal{C}$ as its kernel.

Definition 2.7 (Syndrome Decoding Problem)**.** Let $\mathbb{F}_q$ be a finite field, $\mathbf{H} \in \mathbb{F}_q^{r \times n}$ be a parity check matrix of a linear code $\mathcal{C}$, an integer $w < n$ and a vector $\mathbf{s} \in \mathbb{F}_q^n$. The Syndrome Decoding Problem consists in finding a vector $\mathbf{e} \in \mathbb{F}_q^n$, with Hamming weight $\mathrm{wt}(\mathbf{e}) \leq w$, such that $\mathbf{He}^T = s$.

2.3 Zero-Knowledge Interactive Proofs

In the security proofs along this text we use the concept of zero-knowledge interactive proof of knowledge system. In such context, an entity called prover P has as goal to convince a probabilistic polynomial-time (PPT) verifier V that a given string x belongs to a language L, without revealing any other information.

This kind of proof satisfies three properties:

- Completeness: any true theorem can be proven. That is, $\forall x \in L$ $\mathrm{Prob}\left[(P, V)\left[x\right] = \mathrm{YES}\right] \geq 1 -$ negligible(k). Where, (P, V) denotes the protocol describing the interaction between prover and verifier, and negligible(k) is a negligible function on some security parameter κ.
- Soundness: no false theorem can be proven. That is, $\forall x \notin L$ $\forall P'$ $\mathrm{Prob}\left[(P', V)\left[x\right] = \mathrm{YES}\right] \leq 1/2$, where P' denotes any entity playing the role of prover.
- Zero-Knowledge: anything one could learn by listening to P, one could also have simulated by oneself. That is, $\forall V'_{PPT}$ $\exists S_{PPT}$ $\forall x \in L$ $\mathrm{VIEW}_{P,V'}(x)$ close to $S(x)$. Where, VIEW represents the distribution of the transcript of the communication between prover and verifier, and $S(x)$ represents the distribution of the simulation of such interaction. Depending on the proximity of $\mathrm{VIEW}_{P,V'}(x)$ and $S(x)$, as defined in [GMR85], one can have the following flavors: perfect, statistical and computational.

3 Our Scheme

3.1 Key Generation Algorithm

One of the major modifications that we propose in the batch version of CLRS is to associate with a given user a set of private keys that are binary vector with disjoint support sets, i.e., the positions where the

coordinates are 1 are different considering all pairs of private keys. With such choice, when adding the keys belonging to a subset, the max-norm of the result does not change. The parameters d and γ are chosen in such a way that even when adding all the private keys linked to a user the outcome would still be a vector with norm small enough to prevent it from being efficiently found with currently known algorithms, such as LLL, BKZ and their variations [Ber09].

Algorithm 1 Key Generation

procedure KEYGEN(n, m, q, d, γ)
  $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$.
  for $i \in \{1, \ldots, d\}$ **do**
    $\mathbf{x}_i \xleftarrow{\$} \{0,1\}^m$, s.t. $\mathrm{wt}(\mathbf{x}_i) = \lfloor \gamma/d \rfloor$
    $\triangleright$ $\mathbf{x}_i$ and $\mathbf{x}_j$ have disjoint supports for $i \neq j$
    $\mathbf{y}_i \longleftarrow \mathbf{A}\mathbf{x}_i \bmod q$
  end for
  $\triangleright$ S is the set of d private keys $\mathbf{x}_i$
  COM $\xleftarrow{\$} \mathcal{F}$, suitable family of commitment functions
  Output $(\mathrm{sk}, \mathrm{pk}) = (\mathbf{x}, (\mathbf{y}, \mathbf{A}, \mathrm{COM}))$
end procedure

3.2 Identification Algorithm

The Algorithm 2 is meant to authenticate the entity named as Prover with regard to a set of keys S' that indicates the intended privilege level needed to grant access to some resource. We assume that the Prover and the Verifier have agreed beforehand in assigning those levels to a number of different $S' \subseteq S$.

3.3 ZK Properties

The batch-CLRS protocol constitutes an interactive proof that the entity named as Prover knows a set $S' = \{\mathbf{x}'_1, \ldots, \mathbf{x}'_{d'}\} \subseteq S$ of binary vectors with Hamming weight $\left\lfloor \frac{\gamma}{d} \right\rfloor$ that constitutes solutions to I-SIS instances defined as $\mathbf{y}'_i \longleftarrow \mathbf{A}\mathbf{x}'_i \bmod q$, where $\mathbf{A}$ and $\mathbf{y}'_i$ are public and S is private.

3.3.1 Completeness

Any honest Prover, knowing the private keys that are solution to the I-SIS instances defined above, is capable of correctly computing the commitments as described in Algorithm 2. Besides, he is able to respond to any challenge posed by any honest Verifier in the form of $\{\alpha, b\}$, because of his knowledge of all elements used to compute the commitment values: either randomly chosen variables, or subsets composed of private keys. Hence, all honest executions of the protocol terminate with the Prover being accepted by the Verifier, which characterizes the property of completeness.

3.3.2 Zero-Knowledge

We give a demonstration of the zero-knowledge property for the identification protocol shown in Algorithm 2. Here, we require the commitment function COM to be statistically hiding, i.e., $\mathrm{COM}(\mathbf{x}; \mathbf{r})$ is indistinguishable from uniform for a uniform $\mathbf{r} \in \{0,1\}^n$.

Theorem 3.1. *Let q be prime. The protocol described in in Algorithm 2 is a statistically zero-knowledge proof of knowledge that the prover knows a set $S' = \{\mathbf{x}'_1, \ldots, \mathbf{x}'_{d'}\}$ of secret binary vectors $\mathbf{x}'_i$ with Hamming weight $\left\lfloor \frac{\gamma}{d} \right\rfloor$ that satisfy $\mathbf{A}\mathbf{x}'_i = \mathbf{y}'_i \bmod q$, for $i \in \{1, \ldots, d'\}$, if the employed commitment scheme* COM *is statistically-hiding.*

Proof. To prove the zero-knowledge property of our protocol, we construct a simulator S that, given oracle access to a verifier V' (either honest or not), produces a communication tape that is statistically indistinguishable from a tape generated by the interaction between an honest prover P and the given verifier V'. The construction of such simulated tape is described below. The simulator prepares to answer

Algorithm 2 Batch-CLRS Identification

procedure IDENTIFICATION($\text{sk}, \text{pk}, S, d, \gamma$)
  ▷ S : subset of key pairs desired
  ▷ **Prover**
  $\mathbf{u} \xleftarrow{\$} \mathbb{Z}_q^m, \sigma \xleftarrow{\$} S_m$
  $\overline{\mathbf{x}} = \sum_i \mathbf{x}_i$, with $\mathbf{x}_i \in S$
  $\overline{\mathbf{y}} = \mathbf{A}\overline{\mathbf{x}}$
  $\mathbf{z} \longleftarrow \mathbf{P}_\sigma(\overline{\mathbf{x}})$
  $\mathbf{r}_0 \xleftarrow{\$} \{0,1\}^n, \mathbf{r}_1 \xleftarrow{\$} \{0,1\}^n$
  $c_0 \longleftarrow \text{COM}(\sigma \,||\, \mathbf{Au}; \mathbf{r}_0)$
  $c_1 \longleftarrow \text{COM}(\mathbf{z} \,||\, \mathbf{P}_\sigma\mathbf{u}; \mathbf{r}_1)$
  Send c_0, c_1 to the Verifier
  ▷ **Verifier**
  $\alpha \xleftarrow{\$} \mathbb{Z}_q$
  Send α to the Prover.
  ▷ **Prover**
  $\beta \longleftarrow \mathbf{P}_\sigma(\mathbf{u} + \alpha\overline{\mathbf{x}})$
  Send β to the Verifier.
  ▷ **Verifier**
  $b \xleftarrow{\$} \{0,1\}$
  Send b to the Prover.
  ▷ **Prover**
  if $b = 0$ **then**
    Send $\sigma, \mathbf{r}_0$
  else
    Send $\mathbf{z}, \mathbf{r}_1$
  end if
  ▷ **Verifier**
  if $b = 0$ **then**
    $c_0 \stackrel{?}{=} \text{COM}(\sigma \,||\, \mathbf{AP}_\sigma^{-1}\beta - \alpha\overline{\mathbf{y}}; \mathbf{r}_0)$
    ▷ With $\mathbf{y}_i = \mathbf{Ax}_i$, with $\mathbf{x}_i \in S$
  else
    $c_1 \stackrel{?}{=} \text{COM}(\mathbf{z} \,||\, \beta - \alpha\mathbf{z}; \mathbf{r}_1)$
    $\mathbf{z} \stackrel{?}{\in} \{0,1\}^m, \text{wt}(\mathbf{z}) \stackrel{?}{=} \lfloor \gamma/d \rfloor \, |S|$
  end if
  Return "success" when all checks work, or "failure" otherwise.
end procedure

the second challenge b by guessing its value $\tilde{b}$ taken uniformly at random from $\{0,1\}$, and produces the corresponding commitments $\mathbf{c}_1$ and $\mathbf{c}_2$. It accesses the verifier, as an oracle, passing the commitments $\mathbf{c}_1$ and $\mathbf{c}_2$, obtaining as response the first challenge α. Then, it computes the commitment β and passes it to the verifier, obtaining the final challenge b. If b and $\tilde{b}$ match, the simulator records the interaction in the communication tape. Otherwise, it repeats the process. The number of rounds recorded r corresponds to what would be expected from a real pair (P, V') in order to reach a specified value for the overall soundness error L.

The computations of each commitment are executed as follows.

If $\tilde{b} = 0$, the simulator selects $\mathbf{u}$, σ and the nonces $\{\mathbf{r}_0, \mathbf{r}_1\}$ as per protocol, computes the commitments $\{c_0, c_1\}$ and passes them to the verifier V', which responds with the first challenge α. The simulator solves the equation $\mathbf{A}\overline{\mathbf{x}} = \overline{\mathbf{y}} \pmod q$ for $\overline{\mathbf{x}}$, without any restriction regarding the Hamming weight of the solution. Such step is not computationally hard, and can be done in polynomial time. With this pseudo sum of secret keys $\overline{\mathbf{x}}$, the simulator computes β according to the protocol and sends it to the verifier. The deviation in β is not noticed because of the blinding sum with the uniform $\mathbf{u}$ and the subsequent application of the random permutation σ. Upon receipt of β the verifier responds with the final challenge b. If it matches the value with which the simulator had prepared to answer, namely $\tilde{b}$, then it reveals

the values needed by the Verifier in order to reproduce the commitment $\mathbf{c}_0$. Such values are composed of a permutation seed and a random nonce, both uniformly chosen at random. The whole set of data exchanged between the simulator and the verifier is recorded. If there is not a match between b and $\tilde{b}$, however, the simulator does not record anything and prepares for a new round by picking another $\tilde{b}$ uniformly at random and restarts the process by rewinding the Verifier.

If $\tilde{b} = 1$, the simulator needs to play against the second verification branch. It selects $\mathbf{u}$, σ and the nonces $\{\mathbf{r}_0, \mathbf{r}_1\}$ according to the protocol, uniformly at random. It computes the commitments $\{\mathbf{c}_0, \mathbf{c}_1\}$, sends them to the verifier, obtaining the answer α as the first challenge. Then, the simulator picks $\overline{\mathbf{x}}$ uniformly at random as a vector of dimension m and Hamming weight given by $\lfloor \gamma/d \rfloor |S|$, without having to satisfy the restriction $\mathbf{A}\overline{\mathbf{x}} = \overline{\mathbf{y}} \pmod{q}$, given that this will not be used to check the validity of the commitments, in case the guessed challenge is correct. It suffices that $\mathbf{c}_1$ can be reproduced by the verifier, and that the sum of the private keys $\overline{\mathbf{x}}$ has the expected Hamming weight. The simulator computes commitment β, sends it to the Verifier, and gets the second challenge as response. Again, such deviation is hidden by the blinding sum and the permutation, as in the previous case. In case the final challenge matches the guessed value, the whole interaction of this round is recorded. Otherwise, the simulator picks another $\tilde{b}$ and restarts the process, by rewinding the Verifier. As a result, after $2r$ rounds in average, the simulator produces a communication tape that is statistically indistinguishable from a real one, provided that Com is statistically hiding. □

3.3.3 Soundness

Theorem 3.2. *If after r rounds of protocol execution a cheating prover is accepted with probability at least $(\frac{q+1}{2q})^r + \epsilon$, for any $\epsilon > 0$, either there is a knowledge extractor, which is a polynomial time probabilistic Turing machine that computes with overwhelming probability some private key $\mathbf{x}_i$, with $i \in \{1, \ldots, |S|\}$, which are solutions to instances of the inhomogeneous SIS, or the binding property of the commitment scheme* Com *is broken.*

Proof. Let us suppose that a cheating prover can be accepted with such probability as $(\frac{q+1}{2q})^r + \epsilon$, or higher. Then, by rewinding this prover a number of times given by $1/\epsilon$, we can find with probability $(1-(1-\epsilon))^{1/\epsilon}$ a node with two sons in the execution tree associated with the protocol between the cheating prover and the verifier, corresponding the reception of the two possible values for the challenge b. This means that such cheating prover will be able to answer all challenges for a fixed set of commitments $\mathbf{c}_0, \mathbf{c}_1$.

One possibility is to have the arguments from which the commitments assuming different values for the two challenges, but with similar images through the application of the function Com. This means that the binding property of the commitment function was violated, given that a collision was found.

The second possibility is that the arguments to the function Com are exactly the same for the two challenge values. Let us call σ_0 and $\mathbf{r}_0$ the values revealed by the cheating prover upon receipt of the challenge $b = 0$. Let us call $\mathbf{z}_1$ and $\mathbf{r}_1$ the answer to the challenge $b = 1$. Then, we can obtain the sum of private keys $\overline{\mathbf{x}}$ as $\sigma_0^{-1}(\mathbf{z}_1))$. This also means that we solved an arbitrary inhomogeneous SIS instance in probabilistic polynomial time, violating our assumption that such problem is hard.

□

3.3.4 Performance

The several private keys associated with a given Prover have disjoint support. Individually, these keys have the same Hamming weight. By a proper choice of parameters, similarly to what was observed with CLRS [CLRS10], a given public key has several private keys associated with which, all equally hard to find given their small norm. With such fact, the witness indistinguishability property is satisfied. Such property is kept under parallelization, enabling the scheme to have better performance, if a parallel implementation is used.

Ideal lattices can still be applied with the batch design, enabling compact lattice basis to be employed, with subsequent efficient use of memory in their storage. Besides, one can use FFT to carry out multiplications between matrices and vectors, with consequent gains in speed.

The use of simultaneous key pairs does not impact severely the performance of the scheme, when compared with the pure CLRS. The extra cost in the new identification scheme is reflected mostly in

additions among vectors corresponding to the public and private keys. Such operations usually cost less than FFT, permutations or pseudo-random generation, which take place during the protocol execution.

3.3.5 Worst-case connection

Our private and public keys are related via an I-SIS instantiation: $\mathbf{y} = \mathbf{A}\mathbf{x} \bmod q$, where $\mathbf{x} \in \{0,1\}^m$, $\mathbf{y} \in \mathbb{Z}^n$ and $A \in \mathbb{Z}^{n \times m}$. The hardness to invert this mapping has been used in several cryptographic schemes, such as Ajtai's [Ajt96] and SWIFFT [LMPR08], where the ability to find collisions was reduced to the ability of finding short vectors in a lattice.

Our I-SIS instantiation can be re-phrased to take the form of particular SIS instances $\mathbf{A}_*\mathbf{x}_* = 0 \bmod q$, with $\mathbf{A}_* = [\mathbf{A}\ ; -\mathbf{y}]$ and $\mathbf{x}_* = [\mathbf{x}\ ; 1]^T$. An adversary able to obtain the private keys in our construction can, thus, be used to find short vectors in the lattice defined by $\mathbf{A}_*$ with the particular format $\mathbf{x}_* = [\mathbf{x}\ ; 1]^T$. Considering that in such solution $\mathbf{x}$ is binary, this implies in some of the worst-cases in finding short vectors in lattices.

This observation is valid for CLRS, both in its plain and batch version. In both cases, the private keys correspond to binary vectors. The batch construction makes use of keys with disjoint supports. Therefore, when adding them with modular reduction by q, the outcome still is binary, and with small norm. The parameter is γ is chosen in such a way to assure this.

4 Applicability to Codes

Given that CLRS was derived from a code-based scheme, a natural question would be whether the design proposed in this paper can also be applied to codes, by exploring linearity. As it is, it turns out that a direct adaptation is not entirely possible. The reason is as follows. In the lattice domain, we can add several short vectors with disjoint supports and still have as result a vector that is hard to find via the state of the art algorithms meant to solve approximations of SVP (or approximations thereof). For codes, however, relative small variations in the Hamming weight of error words in the syndrome decoding problem may turn an instance into something easily solved polynomial time. It suffices that the Gilbert-Varshamov bound (GV) and the new Hamming weight are slightly different to incur in this situation.

One rather obvious way to address this is to choose the keys in such a manner that their supports are no longer disjoint. Let us consider a toy example of two private keys $\mathbf{v}_1$ and $\mathbf{v}_2$ corresponding to error-words, having exactly the same Hamming weight. If we make them overlap in exactly half of their support set, when those two keys are added they will result in a third vector $\mathbf{v}_0 = \mathbf{v}_1 + \mathbf{v}_2$ with the same Hamming weight as the other two. We can consider these three vectors as a binary tree of height 1. This can naturally be applied to q-ary codes. It suffices to have $\mathbf{v}_1[i] + \mathbf{v}_2[i] = 0 \bmod q$ for the coordinates where corresponding to the intersection of $\mathbf{v}_1$ and $\mathbf{v}_2$. The outcome of the sum will have the same Hamming weight as $\mathbf{v}_1$ and $\mathbf{v}_2$.

Naturally, we can generalize this approach by means of a complete binary tree, with height parameterized by h, and apply a top-down approach in which first the root of the tree (corresponding to an error-word) is chosen uniformly at random, with a Hamming weight given by the parameter w, taken as the closest even integer to the GV bound. Such node is then split in two sons, having Hamming weight weight equal to w and overlapping in half of their support sets, as explained in the previous paragraph. We recursively apply this procedure to each node until we have a tree with the specified height.

Next, we apply to each node of the tree some clearance level or privilege rights. When applying the batch identification scheme, it suffices to tell which node of the tree is needed. The rest of the protocol would be kept unchanged. Trivially, the cost of the modified identification scheme (just for the sake of argument, let us consider that it Stern's [Ste93] or Cayrel's [CVA10]) is approximately the same of the original one. The extra-cost resides almost entirely in the key generation algorithm (which is usually some off line operation), that is the occasion where the tree containing the keys is actually built.

The generation of keys is shown in algorithms 3 and 4. They cover both binary and q-ary cases, depending on the value with which the q input parameter is set. The "augment" operation used in algorithm 4 consists in adding elements to a set, whereas the operation "support of" consists in returning the coordinates of a vector that correspond to non-null elements of the field over which the vector is defined.

As the identification protocol was kept the same (which varies is an indication of which key to use), the proofs of security and zero-knowledge properties are inherited (from both Stern's and Cayrel's schemes).

The lattice construction shown in this paper has a higher degree of freedom, as far as which individual keys are chosen to constitute the set S that the Prover will use in this protocol. Any non-empty subset can be used. In the code construction described in this section, however, only entire sub-trees (or the entire tree) can be used. Isolated leaves cannot be added in this design: it is easy to see that the addition of two leaves that have different fathers will in general result in a vector having Hamming weight differing those of the individual leaves. Such vector, then, would correspond to an easy instance for the syndrome decoding problem.

The tree-based construction can also be applied to lattices.

Algorithm 3 Key Generation

procedure KeyGen(q, n, m, H, w, h)
  ▷ $\mathbf{H} \in \mathbb{F}_q^{n \times m}$
  ▷ Fields of node are {L, R, $\mathbf{k_{sec}}$, $\mathbf{k_{sec}}$}
  ▷ Where L and R are the left and right sons.
  ▷ And $\mathbf{k_{sec}}$, $\mathbf{k_{sec}}$ are secret and public keys.
  Root ← new node()
  Root.$\mathbf{k_{sec}} \xleftarrow{\$} \mathbb{F}_2^m$, with wt(Root.$\mathbf{k_{sec}}$) $= w$
  Root.$\mathbf{k_{pub}} \longleftarrow \mathbf{H}\mathbf{k_{sec}}^T$
  KeyTree($q, n, m, H, w,$ Root, h)
end procedure

Algorithm 4 TreeOfKeys Generation

procedure KeyTree($q, n, m, H, w, Root, h$)
  $s_{\text{root}} =$ support of(Root.$\mathbf{k_{sec}}$)
  $s_{\text{common}} \xleftarrow{\$} s_{\text{root}}$, such that $|s_{\text{common}}| = w/2$
  $s_1 \leftarrow$ augment(s_{common}) from $\mathbb{F}_q$ until $|s_1| = w$
  Generate $\mathbf{v}_1$ from the support set s_1
  $s_2 \leftarrow$ augment(s_{common}) from $\mathbb{F}_q$ until $|s_2| = w$
  Generate $\mathbf{v}_2$ from the support set s_2
  ▷ Constraint 1: $s_1 \cap s_2 = s_{\text{common}}$
  ▷ Constraint 2: $\mathbf{v}_1[i] + \mathbf{v}_2[i] = 0 \bmod q$ if $i \in s_{\text{common}}$
  Root.L ← new node()
  Root.R ← new node()
  Root.L.$\mathbf{k_{sec}} \leftarrow \mathbf{v}_1$
  Root.L.$\mathbf{k_{pub}} \leftarrow \mathbf{H}\mathbf{v}_1$
  Root.R.$\mathbf{k_{sec}} \leftarrow \mathbf{v}_2$
  Root.R.$\mathbf{k_{pub}} \leftarrow \mathbf{H}\mathbf{v}_2$
  if h=1 **then**
    return
  else
    ▷ Recursive call with the sons
    KeyTree($q, n, m, H, w,$ Root.L, $h-1$)
    KeyTree($q, n, m, H, w,$ Root.R, $h-1$)
  end if
end procedure

5 Conclusion

We have shown a batch identification scheme based on lattices. The zero-knowledge construction allowed that several key-pairs be combined into groups assigning the owner different levels of clearance regarding

access control. As the approach relied heavily on the linearity of the relation associating a private key to its public counterpart, we believe that it can be applied to other schemes than CLRS.

We have also shown a tree-based construction that is applicable to the code-based identification schemes from Stern and Cayrel. The extra cost is almost entirely located in the key generation algorithm. The identification protocol is kept nearly unchanged.

References

[Ajt96] M. Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*, pages 99–108. 1996. On p. 6.

[Ber09] D. Bernstein. *Post-quantum cryptography*. Springer Verlag, 2009. On p. 3.

[CLRS10] P.-L. Cayrel, R. Lindner, M. Rückert, and R. Silva. Improved zero-knowledge identification with lattices. In S.-H. Heng and K. Kurosawa, editors, *ProvSec*, volume 6402 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2010. ISBN 978-3-642-16279-4. On pp. 1 and 5.

[CVA10] P.-L. Cayrel, P. Véron, and S. M. E. Y. Alaoui. A zero-knowledge identification scheme based on the q-ary syndrome decoding problem. In A. Biryukov, G. Gong, and D. R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 171–186. Springer, 2010. ISBN 978-3-642-19573-0. On p. 6.

[GLSY04] R. Gennaro, D. Leigh, R. Sundaram, and W. Yerazunis. Batching schnorr identification scheme with applications to privacy-preserving authorization and low-bandwidth communication devices. *Advances in Cryptology-ASIACRYPT 2004*, pages 187–198, 2004. On p. 1.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, page 304. ACM, 1985. On p. 2.

[LMPR08] V. Lyubashevsky, D. Micciancio, C. Peikert, and A. Rosen. Swifft: A modest proposal for fft hashing. In K. Nyberg, editor, *FSE*, volume 5086 of *Lecture Notes in Computer Science*, pages 54–72. Springer, 2008. ISBN 978-3-540-71038-7. On p. 6.

[MR07] D. Micciancio and O. Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007. On p. 2.

[Sho94] P. W. Shor. Polynominal time algorithms for discrete logarithms and factoring on a quantum computer. In L. M. Adleman and M.-D. A. Huang, editors, *ANTS*, volume 877 of *Lecture Notes in Computer Science*, page 289. Springer, 1994. ISBN 3-540-58691-1. On p. 1.

[Ste93] J. Stern. A new identification scheme based on syndrome decoding. In D. R. Stinson, editor, *CRYPTO*, volume 773 of *Lecture Notes in Computer Science*, pages 13–21. Springer, 1993. ISBN 3-540-57766-1. On p. 6.