Lattice-based Threshold Cryptography

Rikke Bendlin and Ivan Damgård
Department of Computer Science, Aarhus University
{rikkeb, ivan}@cs.au.dk

11. august 2009

Abstract

We present a variant of Regev's system first presented in [Reg05], but with a new choice of parameters. By a recent classical reduction by Peikert we prove the scheme semantically secure based on the worst-case lattice problem GapSVP. From this we construct a threshold cryptosystem which has a very efficient and non-interactive decryption protocol. We prove the threshold cryptosystem secure against passive adversaries corrupting all but one of the players, and againts active adversaries corrupting less than one third of the players. Finally we sketch how one can build a distributed key generation protocol.

1 Introduction

Cryptography based on lattice problems is one of the most important examples of techniques holding promise for public-key cryptography that is secure even under quantum attacks. Recently, these techniques have become much more efficient after it has been realized that one can base the actual cryptosystem on the learning with error problem (LWE), and then argue that the (variant of the) LWE problem used is as hard as some lattice related problem, typically computing the shortest vector in a lattice. In the LWE problem, the adversary must compute a secret vector s with entries in some field or ring, given only the inner product of s with some public vectors where, however, some noise has been added to the products. As mentioned, basing a cryptosystem on LWE can lead to quite efficent cryptosystems, see, e.g., [Reg05],[PVW08],[MR08],[Pei09].

As lattice-based cryptography moves close to practice, it becomes an important research question to investigate whether these cryptosystems can provide the same "extra" functionality we have come to expect from well-known public-key cryptosystems based on factoring or discrete logarithms. For instance, can we have threshold versions of these systems? In other words, we want to share the private key among a set of servers and efficently decrypt a ciphertext while revealing nothing but the plaintext to the adversary.

In this paper we construct such a threshold cryptosystem, based on a variant of Regev's system [Reg05]. We show our scheme is semantically secure based on a worst-case lattice problem using a recent reduction of Peikert[PVW08]. To the best of our knowledge, it is the first lattice-based threshold cryptosystem. We need to use a larger modulus than Regev, thus making ciphertexts larger, on the other hand we get a very efficient and non-interactive decryption protocol: each player needs only to do local computation and announce a single element from the underlying ring. The basic version of the protocol is secure against a passive adversary corrupting all but one of the players. For a small number of players, we show an equally efficent version secure against a malicious adversary corrupting less than a third of the players. Towards the end of the paper, we sketch a distributed protocol for generating keys.

Various improvements of Regev's original cryptosystem have been made since its first appearence, e.g. in [PVW08] and [MR08]. Our threshold cryptosystem can be generalized in the same way, but we stick to Regev's original approach here for simplicity.

2 Preliminaries

When writing $x \in_R S$ we mean that x is chosen uniformly at random from the set S. Equivalently $x \in_\chi S$ means choosing x from the set S according to the distribution χ.

Given a probability distribution χ on $\mathbb{Z}_q$, let n be some integer and $\mathbf{s} \in \mathbb{Z}_q^n$. We define $A_{\mathbf{s},\chi}$ as the distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing $\mathbf{a} \in_R \mathbb{Z}_q^n$, $e \in_\chi \mathbb{Z}_q$ and outputting $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s}\rangle + e)$. We define the decisional Learning With Errors (LWE) problem as being able to distinguish between a sample from $A_{s,\chi}$ and the uniform distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ with non-negligible probability. We define the search LWE problem as given a sample from $A_{s,\chi}$ finding s with non-negligible probability.

By $\overline{\Psi}_\alpha$ we denote a discrete Gaussian distribution on $\mathbb{Z}_q$ with mean 0 and standard deviation $\frac{q\alpha}{\sqrt{2\pi}}$. Likewise Ψ_α is a continuous Gaussian distribution on $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ with mean 0 and standard deviation $\frac{\alpha}{\sqrt{2\pi}}$. By χ^{*k} we denote the distribution given by summing k independent samples from χ. Note in particular that when $\chi = \overline{\Psi}_\alpha$ we have that $\chi^{*k} = \overline{\Psi}_{\sqrt{k}\cdot\alpha}$. This follows immediately from $\overline{\Psi}_\alpha$ behaving as an ordinary normal distribution.

3 Cryptosystem

We first present the underlying cryptosystem which was proposed first in [Reg05], but with a new choice of parameters better suited for the distributed decryption protocol given later.

3.1 Description

Let n be the security parameter of the cryptosystem. Then the main parameter is an integer q which is chosen as $q = 2^{O(n)}$. More specifically q will not be a prime but a B-smooth number where B is of polynomial size. That is $q = \prod p_i$ is a product of prime numbers $p_1, \dots, p_k$, where $p_i < B$ and also $p_i > u$, the number of players in the distributed decryption protocol. The latter requirement on the primes is necessary in order to do secret sharing over the the ring $\mathbb{Z}_q$, more on this later. We also need an integer m which will be chosen to be $O(n^3)$. Finally, we need a distribution χ on $\mathbb{Z}_q$ which will be taken to be the discrete Gaussian distribution $\overline{\Psi}_\alpha$, where $\alpha = q^\beta$ for some $\beta < 1/4$.

The cryptosystem is now defined as follows:

- **Secret key:** Choose $\mathbf{s} \in_R \mathbb{Z}_q^n$. The secret key is then $\mathbf{s}$.
- **Public key:** Choose m vectors $\mathbf{a}_1, \dots, \mathbf{a}_m \in_R \mathbb{Z}_q^n$, m elements $e_1, \dots, e_m \in_\chi \mathbb{Z}_q$. The public key is then given by $(\mathbf{a}_i, b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i)_{i=1}^m$.
- **Encryption:** Choose a random set S among all the subsets of $[m]$. Given a bit γ, the encryption of γ is given by $(\sum_{i \in S} \mathbf{a}_i, \gamma \cdot \lfloor \frac{q}{2} \rfloor + \sum_{i \in S} b_i)$.
- **Decryption:** Given a ciphertext $(\mathbf{a}, b)$, calculate $b - \langle \mathbf{a}, \mathbf{s} \rangle$ and determine whether it is closer to 0, the encrypted bit being 0, or closer to $\frac{q}{2}$, the encrypted bit being 1.

3.2 Correctness

The correctness of the decryption protocol is given in the following theorem.

Theorem 3.1 (Correctness)**.** *If for any $k \in \{0, 1, \dots, m\}$ it holds that*

$$\Pr_{e \sim \chi^{*k}}(|e| \geq \sqrt[3]{q}) \leq 2^{-O(n)}$$

then the decryption protocol will give correct output except with negligible probability.

A similar theorem is proved in [Reg05] for Regev's original choice of parameters. The intuition is clear, if the noise added is not too big, we will be able to decrypt to the right bit. The correctness with the new parameters follows from the following claim.

Claim 3.2 (Correctness)**.** *For the choice of parameters made, for any $k \in \{0, 1, \dots, m\}$ and $e \sim \chi^{*k}$ it holds that*

$$\Pr_{e \sim \chi^{*k}}(|e| \geq \sqrt[3]{q}) \leq 2^{-O(n)}$$

Proof. We will prove this using the Chebyshev inequality, but first we will reduce the problem from $\overline{\Psi}_\alpha$ to Ψ_α. Given $e \sim \overline{\Psi}_\alpha^{*k}$ we have that $e = \sum_{i=1}^k \lfloor qx_i \rceil \pmod q$, where $x_i \sim \Psi_\alpha$. The value of e is at most $k < m < \sqrt[3]{q}/2$ away from $\sum_{i=1}^k qx_i \pmod q$, so it is sufficient to prove that $|\sum_{i=1}^k qx_i \pmod q| < \sqrt[3]{q}/2$ unless with negligible probability. Since $\sum_{i=1}^k qx_i \pmod q$ comes from a distribution with standard deviation approximately $\sqrt{k} \cdot q^\beta$ and mean 0 we get the following result from Chebyshev's inequality, with $m = n^3$ and $t = \frac{\sqrt[3]{q}}{2\sqrt{m}\sqrt[4]{q}} \geq \frac{\sqrt[3]{q}}{\sqrt[20]{q}\sqrt[4]{q}} = \sqrt[30]{q}$.

$$\Pr\left(|e| \geq \sqrt[3]{q}/2\right) \leq \Pr\left(|e| \geq t \cdot \sqrt{k}\sqrt[4]{q}\right) \leq \frac{1}{t^2}$$

We see that $1/t^2 \leq 1/\sqrt[15]{q}$ is in fact negligibly small. □

Note that the inequalities used above are not very tight, especially the Chebyshev inequality. Therefore in practice one would expect to be able to choose much better parameters, for instance a bigger standard deviation on the distribution used. This would in turn give us security reductions to the hardness of somewhat bigger lattice problem instances. Furthermore the claim is actually stronger than what is needed for the original decryption protocol to be correct, but we will need this stronger result in the proofs of the distributed decryption protocols described below.

3.3 Security

The security of the cryptosystem is given by the following theorem.

Theorem 3.3 (Security). *The cryptosystem is semantically secure under the assumption that* GapSVP *is hard in the worst case.*

Below we will sketch the ideas of the proof. It boils down to showing how the proofs given in [Reg05] can be adjusted to the new choice of parameters.

Proof. The proof of security given in [Reg05] is based on the property that distinguishing between encryptions of 0 and 1 is at least as hard as distinguishing public keys from randomly chosen elements in $\mathbb{Z}_q^n \times \mathbb{Z}_q$. The latter problem being the decision LWE problem. The proof of the reduction does not depend heavily on the values of the parameters, and is therefore still valid with the new choice of parameters.

The decision LWE is then further reduced to search LWE. This reduction in [Reg05] heavily relies on the fact that q is chosen to be polynomial in that it does exhaustive search over all elements in Z_q. But in fact the same idea can be used when q is exponential in size, but B-smooth with B polynomial. The idea being to do the reduction modulo each of the primes p_i in q, and recombine the solutions to a full solution modulo q using the Chinese Remainder Theorem.

The last step is to reduce search LWE to standard lattice problems. Since q is chosen to be exponentially large we can use the reduction to GapSVP made in [PVW08]. □

This is another advantage of choosing an exponentially large q: With the original choice of a polynomial q the reductions to lattice problems are either a quantum reduction as in [Reg05] or a reduction to a special variant of GapSVP, the hardness of which is not completely understood.

4 Distributed Decryption (Passive Adversaries)

In this section we present a distributed decryption protocol for the above cryptosystem involving u players which is secure against a static, passive adversary corrupting up to $t = u - 1$ players. That is, we assume the adversary is able to see all messages and internal data of a corrupted player, but the player still follows the protocol. The adversary must choose which players to corrupt at the start of the protocol.

We assume that communication is synchronous and that the client has access to a broadcast channel to all players. Private channels between players are not necessary since there is no interaction between players in the protocol. We assume the adversary sees all communication between the client and the players.

We use Shamir secret sharing over $\mathbb{Z}_q$ as described in [Sha79] to make secret sharings of various values in the protocol. Normally Shamir secret sharing is done over a field, but since q is not a prime $\mathbb{Z}_q$ is only a ring. This turns out not to be a problem with the choice made of the prime factors in q. The only thing that is needed is that one can do Lagrange interpolation over the points $1, \ldots, u$ which in turn boils down to being able to invert elements in this range. We chose $q = \prod p_i$, where $p_i > u$, so obviously invertion of the points needed is possible.

We furthermore make use of the concept of pseudorandom secret sharing (PRSS) described in [CDI05]. PRSS will enable the players to non-interactively share a common random value from some interval. The idea is as follows. For each subset A of size t of the players we associate a key $K_A \in_R \mathbb{Z}_q$. Such a key is given to player i exactly if $i \notin A$. Assume we are given a pseudorandom function ϕ that given a key and a ciphertext as input, will output values in the interval $[-\sqrt{q}, \sqrt{q}]$. A player can now compute $\phi_{K_A}(c)$ for all K_A he has been given, and afterwards take an appropriate linear combination of the results. This will result in all players having a Shamir share of the common random value $x = \sum_A \phi_{K_A}(c)$. Since $|A| = t$ there are $\binom{u}{t}$ possibilities for A, so x will be in the interval $\left[-\binom{u}{t}\sqrt{q}, \binom{u}{t}\sqrt{q}\right]$. We note that $\binom{u}{t} = u$ for our choice of t (but we will consider other choices later).

The protocol and proofs will be given in the setting of the Universal Composability (UC) framework proposed by Canetti. For details of this see [Can01].

4.1 Key Generation and Distribution

We assume for now that generation and distribution of keys and key-shares to players are handled by some trusted party. This is described by the functionality F_{KeyGen}.

Functionality F_{KeyGen}

1. When receiving "start" from all honest players, choose the secret key $\mathbf{s}$ and construct the public key $(\mathbf{a}_i, b_i)_{i=1}^{m}$ as described in section 3. Furthermore for each subset A of size t of the players, choose key $K_A \in_R \mathbb{Z}_q$.

2. For each entry j in the secret key make a share $s_{i,j}$ for each player i. We write $[\mathbf{s}]$ as short for the set of shares in $\mathbf{s}$. To each player i privately send to him his shares from $[\mathbf{s}]$ and all keys K_A where $i \notin A$.

3. Finally send the public key to all players and the adversary.

4.2 Decryption Protocol

We now describe the decryption protocol. To make things more easily describable we introduce a client, who is the party receiving the ciphertext in the first place, and who wants to decrypt with help from the players.

Protocol *Decrypt*

1. Each player sends "start" to F_{KeyGen} and stores the public key, the share of the secret key and the keys K_A received.

2. When receiving a ciphertext $c = (\mathbf{a}, b)$, the client broadcasts c to all players.

3. The players compute $[e'] = [b - \langle \mathbf{a}, \mathbf{s} \rangle] = [e + \lfloor \frac{q}{2} \rfloor \cdot \gamma]$. Since $(\mathbf{a}, b)$ is public this is a linear operation on $\mathbf{s}$ and only requires the players to locally do the same linear operation on their shares. Then $\phi_{K_A}(c)$ is computed for all the keys K_A the player received and the player takes an appropriate linear combination of the result to obtain a sharing $[x] = [\sum_A \phi_{K_A}(c)]$. Finally the players compute $[x + e']$, and send all these shares to the client.

4. Having received all the shares of $[x + e']$ the client reconstructs $x + e'$, checks whether it is closer to 0 or to $q/2$, and outputs 0 or 1 accordingly.

4.3 Security

To prove security we wish to be able to implement the following functionality.

Functionality $F_{KeyGen-and-Decrypt}$

1. Upon receiving "start" from all honest players, choose the secret key and construct the public key to be used. Send the public key to all players, the client and the adversary.

2. Hereafter on receiving "decrypt $(\mathbf{a}, b)$" from the client, send "decrypt $(\mathbf{a}, b)$" to all players and the adversary.

3. In the next round, send "result γ" to the client and the adversary, where γ is the bit corresponding to the given ciphertext.

The security is now given by the following theorem.

Theorem 4.1 (Security). *When given access to the functionality F_{KeyGen} and assuming that ϕ is a pseudo-random function, the protocol Decrypt securely implements $F_{KeyGen-and-Decrypt}$. The adversary is assumed to be passive and static, corrupting up to $t = u - 1$ of the players.*

Proof. We abbreviate $F_{KeyGen-and-Decrypt}$ by F_{KG-D} in the following. To prove security we must construct a simulator to work on top of the ideal functionality F_{KG-D}, such that an adversary playing with either the simulator and ideal functionality or the real world decryption protocol cannot tell in which case he is. We denote by Adv the adversary communicating with the real decryption protocol and must show that we can simulate everything Adv sees. The simulation proceeds as follows.

1. Let B denote the set of players corrupted by Adv. When receiving "start" to F_{KeyGen} send "start" to F_{KG-D}. Upon receiving the public key, compute a sharing of $\mathbf{0}$, the zero-vector in $\mathbb{Z}_q^n$, to simulate sharing the secret key. Also choose the necessary keys K_A. Then send to the adversary the public key, the shares of the secret key corresponding to B, and the keys K_A that should be send to players in B.

2. When receiving "decrypt $(\mathbf{a}, b)$" from F_{KG-D}, the ciphertext is sent to Adv for each player in B. When "result γ" is received in the next round, we have to simulate the shares of $x + e'$ that honest players would send. To play the role of x, we form a value y as the sum of those $\phi_{K_A}(c)$ where the adversary knows K_A, and one uniformly random value from $[-\sqrt{q}, \sqrt{q}]$ for each K_A that adversary does not know. The idea is to let $y + \lfloor \frac{q}{2} \rfloor \cdot \gamma$ play the role of the value $x + e + \lfloor \frac{q}{2} \rfloor \cdot \gamma$ that would be revealed in the real protocol. Note that from the shares and keys given to the adversary, we can compute the shares corrupted players would send to the client. Using Lagrange interpolation, we can compute a polynomial f of degree at most t that is consistent with these shares and has $f(0) = y + \lfloor \frac{q}{2} \rfloor \cdot \gamma$. We use this polynomial to compute shares for the honest players and give these to the adversary.

The final thing is to prove that no environment is able to distinguish between the real decryption protocol and the simulation presented above. This basically comes down to proving that the decryption protocol is able to recover the bit encrypted and that the distributions of the shares sent to the adversary in both cases are computationally indistinguishable.

The shares of the secret key in step 1 are distributed the same in both cases beacuse of the security of the underlying secret sharing scheme used. The keys K_A are also obviously distributed identically in the two cases.

Next, note that in both simulation and real protocol, the shares revealed in the decryption step follow deterministically from the information sent in step 1 and the values $y + \lfloor \frac{q}{2} \rfloor \cdot \gamma$, $x + e + \lfloor \frac{q}{2} \rfloor \cdot \gamma$ used in simulation, respectively real protocol. It is therefore enough to show that these values are computationally indistinguishable in the view of the adversary. For this, note that in the real protocol the adversary is not given all keys K_A, and so, by pseudorandomness of ϕ and construction of y, $y + e + \lfloor \frac{q}{2} \rfloor \cdot \gamma$ is computationally indistinguishable from the $x + e + \lfloor \frac{q}{2} \rfloor \cdot \gamma$ in the view of the adversary. Second since y is a sum including at least one value that is uniform in an interval of size $2\sqrt{q}$, which is exponentially larger than the interval $[-\sqrt[3]{q}, \sqrt[3]{q}]$ in which e is distributed, we find that $y + \lfloor \frac{q}{2} \rfloor \cdot \gamma$ is statistically indistinguishable from $y + e + \lfloor \frac{q}{2} \rfloor \cdot \gamma$.

Finally in both the simulated and the real run the client will output the correctly decrypted value. This is obvious in the simulated case and in the real world it follows from Lemma 1 below. □

Lemma 4.2 (Correctness)**.** *Let* $\binom{u}{t} < \frac{1}{4}\sqrt{q} - 1$. *Assume that for any* $k \in \{0, 1, ...m\}$, χ^{*k} *satisfies that*

$$\Pr_{e \sim \chi^{*k}} [|e| \geq \lfloor \sqrt[3]{q} \rfloor] \leq 2^{-O(n)}.$$

Then the error probability when decrypting is negligible.

Proof. Given an encryption of 0 the result which is reconstructed is given by $b - \langle \mathbf{a}, \mathbf{s} \rangle = e + x = \sum_{i \in S} e_i + x$. The distribution of e is exactly given by $\chi^{*|S|}$, therefore according to our assumption $|e| < \lfloor \sqrt[3]{q} \rfloor$ with probability at least $1 - 2^{-O(n)}$. Since $\binom{u}{t} < \frac{1}{4}\sqrt{q} - 1$ according to our assumption, we have that $|x| < \frac{q}{4} - \sqrt[3]{q}$. Combined we get that $|e + x| < \frac{q}{4}$ with probability at least $1 - 2^{-O(n)}$. In this case the result is closer to 0 than $\frac{q}{2}$ and the decryption is correct. A similar proof can be done for an ecryption of 1. □

The assumptions in the lemma are fulfilled according to the claim in section 3. We note that the correctness puts an upper bound on the possible number of players, which is also to be expected, since there is a limit to how much random noise can be added before an encryption of 0 turns into an encryption of 1. Note though that when $t = u - 1$, as is the case in the passive case, we have $\binom{u}{t} = u$. So here the number of players is bounded by approximately $\sqrt{q}$ which is still quite a big number.

5 Distributed Decryption for Stronger Adversaries

The protocol for doing distributed decryption against a passive adversary corrupting less than $t = u - 1$ players, can easily be turned into a protocol secure against a stronger adversary. First, if the adversary is semi-honest, i.e. corrupted players follow the protocol but may stop at any point, exactly the same protocol will be secure, if $t < u/2$. The proof is the same, one just notes that at least $t + 1$ players will always complete the protocol.

If the adversary is active, again almost the same protocol and proof applies, if we assume $t < u/3$. The only significant difference to the protocol is that the client must use standard methods for error correction to reconstruct $x + e'$ at the end of the decryption since some players may lie about their shares. This is possible exactly when $t < u/3$.

It should be noted that both variants of the protocol are only feasible to execute for a small number of players, since the number of keys K_A we must give to each player increases exponentially with u whenever t is a constant fraction of u. However, in most realistic applications of threshold cryptography, one indeed expects the number of players to be small.

6 Distributed Key Generation

In this section we will briefly sketch how to implement the functionality F_{KeyGen} against an active adversary. In some of the parts involving interaction between the players, we will have to assume private communication channels between players.

The coordinates of the secret key can be generated as follows. For each coordinate each player chooses uniformly random element in $\mathbb{Z}_q$ and secret shares it among all the players. The secret key will now be given by the sum of the values chosen by the players, and the players can locally compute their share of the secret key.

The next issue is how to compute the public key which in turn boils down to securely generating secret shares of the error terms e_i distributed according to the distribution $\chi = \overline{\Psi}_\alpha$. The problem especially being that corrupted players might not construct shares in the right way. For this step we can use the idea of non-interactive verifiable secret sharing (NIVSS) described in [CDI05], which builds on top of PRSS described earlier[1]. In brief NIVSS enables us to securely construct sharings of a specific secret chosen by one of the players. We then form the e_i's as sums of secret values chosen by each of the players, where we note that we can enforce a limitation on the size of the contribution of each player to the sum.

Assuming that m public vectors $\mathbf{a}_1, \ldots, \mathbf{a}_m$ has been chosen randomly, each player can now compute their shares $[\langle \mathbf{a}_i, s\rangle + e_i]$ and open these to get the public key. By choosing the parameters appropriately, we can argue that we can decrypt correctly because the size of each e_i is bounded, and furthermore that the system is semantically secure since this would be the case if we only added the contributions of honest players to the $\langle \mathbf{a}_i, s\rangle$'s.

References

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS '01: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 136, Washington, DC, USA, 2001. IEEE Computer Society.

[1]The keys K_A for doing the pseudorandom secret sharing can be generated by first generating sufficiently many shared random values as described above and then sending the shares of these to the players that are supposed to know a given key.

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *TCC*, pages 342–362, 2005.

[MR08] Daniele Micciancio and Oded Regev. Lattice-based cryptography. In D. J. Bernstein and J. Buchmann, editors, *Post-quantum Cryprography*. Springer, 2008.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 333–342, New York, NY, USA, 2009. ACM.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO 2008: Proceedings of the 28th Annual conference on Cryptology*, pages 554–571, Berlin, Heidelberg, 2008. Springer-Verlag.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC '05: Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 84–93, New York, NY, USA, 2005. ACM.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.