Threshold Decryption and Zero-Knowledge Proofs for Lattice-Based Cryptosystems

Rikke Bendlin and Ivan Damgård
March 29, 2010

Department of Computer Science, Aarhus University
{rikkeb, ivan}@cs.au.dk

Abstract. We present a variant of Regev's cryptosystem 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. We also describe how one can build a distributed key generation protocol. In the final part of the paper we show how one can, in zero-knowledge - prove knowledge of the plaintext contained in a given ciphertext from Regev's original cryptosystem or our variant. The proof is of size only a constant times the size of the public key.

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 and are also interesting in that they can be based on worst-case complexity assumptions. 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 products 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 closer 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. And furthermore, can one prove, in zero-knowledge and efficiently, knowledge of the plaintext contained in a given ciphertext?

In this paper we construct such a threshold cryptosystem, based on a variant of Regev's system [Reg05]. We show our scheme semantically secure based on a worst-case lattice problem using a recent reduction of Peikert [Pei09]. 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. We also describe 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.

In the final part of the paper we present a zero-knowledge protocol for proving knowledge of the plaintext contained in a given ciphertext, for Regev's original cryptosystem as well as our variant. The proof is much more efficient than what generic methods would give us: it has size only a constant times the size of the public key, and the computation required is comparable to what is required to generate keys. The protocol is based on the construction from [IKOS07] of zero-knowledge from multiparty computation protocols. Whereas this paradigm has perhaps been perceived primarily as a theoretical tool, we show here that it can also be potentially relevant in practice.

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. For some distribution χ writing $x \sim \chi$ means that x is distributed according to 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 $A_{\mathbf{s},\chi}$ 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 $A_{\mathbf{s},\chi}$ finding $\mathbf{s}$ 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.

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.

Let n be the security parameter of the cryptosystem. Then the main parameter is an integer q which is chosen as $q = 2^{\Theta(n)}$. 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 $p_1, \ldots, p_k$, 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 $\overline{\Psi}_\alpha$, with $\alpha = \sqrt{2\pi} q^{\beta-1}$ for $\beta = 1/4$. This means that the standard deviation is q^β.

Note that we have chosen a particular value of β for concreteness and simplicity. The choice is somewhat arbitrary, however, and in fact any value between 0 and 1 could be used. One would just need to adjust the other parameters accordingly to still have all our protocols work correctly, this essentially follows from Lemma 1 below. For security of the cryptosystem itself, however, one needs to be more careful. We will say more about this below in the remarks after Theorem 1.

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 $\mathbf{a}_1, \ldots, \mathbf{a}_m \in_R \mathbb{Z}_q^n$, m elements $e_1, \ldots, e_m \in_\chi \mathbb{Z}_q$. 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 vector $\mathbf{r} = (r_1, \ldots, r_m) \in_R \{0,1\}^m$. Given a bit γ, the encryption of γ is given by $(\sum_{i=1}^m r_i \mathbf{a}_i, \gamma \cdot \lfloor \frac{q}{2} \rfloor + \sum_{i=1}^m r_i b_i)$.
- **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.

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

Theorem 1 (Correctness). *If for any $k \in \{0, 1, \ldots, m\}$ 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.

Lemma 1 (Correctness). *For the choice of parameters made, for any $k \in \{0, 1, \ldots, m\}$, a constant $c \in (0, 4)$ and $e \sim \chi^{*k}$ it holds that*

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

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 $k < m < \sqrt[c]{q}/2$ away from $e' = \sum_{i=1}^{k} qx_i \pmod q$, so it is sufficient to prove that $|e'| < \sqrt[c]{q}/2$ except with negligible probability. Since e' comes from a distribution with standard deviation $\sqrt{k} \cdot \sqrt[4]{q}$ and mean 0 we get the following result from Chebyshev's inequality,

$$\Pr\left(|e'| \geq \sqrt[3]{q}/2\right) \leq \Pr\left(|e'| \geq t \cdot \sqrt{k}\sqrt[4]{q}\right) \leq \frac{1}{t^2}$$

where $m = n^3$ and $t = \frac{\sqrt[c]{q}}{2\sqrt{m}\sqrt[4]{q}} \geq \frac{\sqrt[c]{q}}{\sqrt[d]{q}\sqrt[4]{q}} = q^{1/c} \cdot q^{-1/d} \cdot q^{-1/4}$, for some constant d. Now considering $1/t^2$ we see that this will be negligible if $d > -\frac{4c}{c-4}$. But we can always choose such a d since $c < 4$. □

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.

As for security of the cryptosystem we have an advantage in choosing an exponentially large q: with the original choice of a polynomial q the reductions from lattice problems are either a quantum reduction as in [Reg05] or a reduction from a special variant of GapSVP, the hardness of which is not completely understood. With a large q we can use Peikert's reduction [Pei09]. We get the following theorem which we state informally and then give some remarks on the details below.

Theorem 2 (Security). *The cryptosystem is semantically secure under the assumption that* GapSVP *is hard in the worst case.*

Proof. We sketch the ideas of the 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 search LWE is then further reduced to decision 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 $\mathbb{Z}_q$. 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. This was already observed in [Pei09].

The last step is to reduce standard lattice problems to search LWE. Since q is chosen to be exponentially large we can use the reduction to GapSVP made in [Pei09]. □

Looking more into the details, the approximation to the shortest vector that Peikert's reduction produces is off by a factor (roughly) n/α. Of course, if this factor is so large that known lattice algorithms can beat it in polynomial time, then the theorem is not of any use, although on the other hand, this does not mean that the cryptosystem is insecure. According to Micciancio and Regev [MR08], state of the art of lattice algorithms suggests that an approximation factor of $(1.01)^n$ is infeasible to achieve for large enough n.

So suppose we are willing to assume that there exists some constant $\delta > 1$ such that an approximation factor δ^n cannot be achieved in polynomial time. Assume also that we have $q = 2^{cn}$ for a constant c. Now, to make Peikert's reduction from our LWE version produce an algorithm contradicting the assumption, we just need to have

$$n/\alpha = \delta^n$$

Plugging in $\alpha = \sqrt{2\pi}q^{\beta-1}$ and $q = 2^{cn}$, one easily sees that we need to satisfy

$$-(\beta - 1)c = \log \delta.$$

up to terms that vanish as n grows. In particular, for any $0 < \beta < 1$, there is a $c > 0$ that makes this be true.

It is more standard to assume that a *polynomial* approximation factor is infeasible, but this assumption cannot be used with our parameter settings. On the other hand, our assumption could be weakened by taking α not to be exponentially small but, say, $n^{-\Theta(\log n)}$. Our protocols could still be made to work, with negligible but slower converging error probabilities.

Finally, we warn the reader that from a practical point of view, the asymptotic analysis should be taken with a large grain of salt. To choose parameters for practical use one needs a much more careful analysis of the attacks actually known on LWE.

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 inversion 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 P_j exactly if $P_j \notin A$. 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 the functionality F_{KeyGen}.

Functionality F_{KeyGen}

1. When receiving "start" from all honest players, choose the secret key $\mathbf{s} = (s_1, ..., s_n)$ 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. Receive from the adversary a set of shares $s_{i,j}, i = 1, \ldots, n$ for each corrupted player P_j. Then construct using Lagrange interpolation a complete set of shares $s_{i,j}, i = 1, \ldots, n, j = 1, \ldots, u$ consistent with the shares received from the adversary, and such that $s_{i,1}, \ldots, s_{i,u}$ form a set of shares of s_i. We write $[\mathbf{s}]$ as short for the set of all shares. Send privately to each player P_j his shares from $[\mathbf{s}]$ and all keys K_A where $P_j \notin A$.
3. Finally send the public key to all players and the adversary.

It may seem strange that this functionality allows the adversary to decide which shares he wants to get of the secret key – why not let the functionality do the sharing on its own? However, we need to define the functionality this way to make sure it can be implemented. The problem is that a simulator trying to simulate a given protocol will have to make sure that the view of the protocol it generates for the adversary is consistent with what the functionality says to the honest players. This is not possible if the functionality decides on all shares on its own. One could say that what we model here is that we don't care which shares the adversary gets, as long as the secret is safe.

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.

Theorem 3 (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 the 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 in the same way 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 2 (Correctness). *Let* $\binom{u}{t} < \frac{1}{4}\sqrt{q} - 1$*. Assume that the reconstructed value in the distributed descryption protocol is given by* $e + x$*, and furthermore that the following is satisfied*

$$\Pr\left[|e| \geq \lfloor\sqrt{q}\rfloor\right] \leq 2^{-O(n)}.$$

Then the error probability when decrypting is negligible.

Proof. Given an encryption of 0 the result which is reconstructed is given by $b - \langle \mathbf{a}, \mathbf{s} \rangle = e + x = \sum_{i=1}^{m} r_i e_i + x$. Since $\binom{u}{t} < \frac{1}{4}\sqrt{q} - 1$ according to our assumption, we have that $|x| < \frac{q}{4} - \sqrt{q}$. Combined with the assumption on $|e|$ 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 encryption of 1. □

The distribution of e is exactly given by $\chi^{*\sum r_i}$, when F_{KeyGen} has been used to produce the keys, therefore according to the claim of section 3 we know that $|e| < \lfloor\sqrt[3]{q}\rfloor$ with probability at least $1 - 2^{-O(n)}$. And so the assumptions in the lemma is fulfilled.

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.

5 Distributed Decryption for Stronger Adversaries

The protocol for doing distributed decryption against a passive adversary corrupting up to $t = u - 1$ 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 describe how to do key generation and distribution. In some of the parts involving interaction between the players, we will have to assume private communication channels between players. For a passive adversary, key generation is quite straightforward, so we focus on the more interesting case of an active, static adversary corrupting less than $t = u/3$ players.

We will need the following functionality for generating random (shared) values. It offers a number of commands, and a command is executed if all honest players input the same command.

Functionality F_{Rand}

- On input "Random value to B" for a set of players B, choose s at random in $\mathbb{Z}_q$ and send s to all players in B.
- On input "Random shared value", ask the adversary for a set of shares $S = \{s_j | \ P_j \text{ is corrupt}\}$. Choose s at random in $\mathbb{Z}_q$ and use Lagrange interpolation to construct a set of shares $[s]$ consistent with S, i.e., each corrupt P_j gets share s_j. Send shares from $[s]$ to all honest players.
- On input "Shared value from D", do the same as for "Random shared value", but get s from player D instead of choosing it at random.
- On input "Constrained value from D", do the same as for "Shared value from D", but check that s received from D is in the interval $[-2\binom{u}{t}\sqrt[3]{q}, 2\binom{u}{t}\sqrt[3]{q}]$. If not, send "fail" to all players. Furthermore, if D is honest, he is assumed to choose s in the interval $[-\sqrt[3.5]{q}, \sqrt[3.5]{q}]$. The seemingly strange choice of intervals is dictated by the implementations that are available, see more details below.

"Shared value from D" can be implemented using any protocol for verifiable secret sharing. A simulator would simply run the protocol with the adversary while following the protocol for the honest players (and using a dummy value for s if D is honest), and then send the shares obtained for corrupt players to the functionality. "Random shared value" can be done by calling "Shared value from P_i" for each P_i, asking P_i to supply a random value s_i, and then locally adding the resulting shares, thus obtaining $[\sum_i s_i]$ which we use as $[s]$. "Random value to B" is implemented by calling "Random shared value" and then have players send their shares to all players in B.

Finally, "Constrained value from D" can be implemented using the technique of non-interactive verifiable secret sharing (NIVSS) described in [CDI05] which builds on top of PRSS described earlier. In the protocol for doing NIVSS, a set of keys $\{K_A^D\}$ is assumed to be set up similar as for PRSS, i.e., K_A^D is known to all players not in A. But furthermore player D holds all keys and the value s to be shared. The keys can be set up by calling "Random value to B". The pseudorandom function involved is chosen such that it outputs random values from the interval $[-\sqrt[3]{q}, \sqrt[3]{q}]$. To generate the shared value, each player locally computes random shares of a value r as in PRSS, and D can compute r since he knows all keys. D then broadcasts $s - r$, and each player adds this value to their share in r, thus obtaining $[s]$. D is disqualified if the value broadcast is not in the interval $[-\binom{u}{t}\sqrt[3]{q}, \binom{u}{t}\sqrt[3]{q}]$. This guarantees that s is in the required interval even if D is corrupt, since r is in the interval $[-\binom{u}{t}\sqrt[3]{q}, \binom{u}{t}\sqrt[3]{q}]$. If D is honest, the distribution of $s-r$ is statistically close to uniform in $[-\binom{u}{t}\sqrt[3]{q}, \binom{u}{t}\sqrt[3]{q}]$ since s is smaller than r by an exponentially large factor.

Given F_{Rand}, key generation is for the most part straightforward. The tricky part, however, is that to generate the noise to be added to the public key, shares of non-uniformly distributed values are to be generated and distributed. For this we will invoke "Constrained value from P_j" for each P_j, since we can rely on honest P_j's using the correct distribution, while corrupt P_j's cannot choose values that are large enough to do any damage, as we shall see.

Protocol *KeyGeneration*
The protocol assumes F_{Rand} is available.

1. To generate and distribute the secret key, invoke "Random shared value" n times to form $[\mathbf{s}]$.
2. To generate and distribute the keys K_A for PRSS, invoke for each set A of t players "Random value to B", where B is the complement of A (we assume for simplicity that a random value from $\mathbb{Z}_q$ is sufficient to form a key K_A).
3. To generate the public key, invoke "Random value to P" nm times, where P is the set of all players, and use the output as entries in the vectors $\mathbf{a}_i$.
4. Each player P_j chooses noise contributions $e_{i,j}, i = 1, \ldots, m$ according to the distribution $\overline{\Psi}_\alpha$ and uses these as input to invocations of "Constrained value from P_j". Note that a correctly chosen $e_{i,j}$ will be in the correct interval $[-\sqrt[3.5]{q}, \sqrt[3.5]{q}]$ except with negligible probability. Thus, we obtain $[e_{i,j}]$ for $i = 1 \ldots u, j = 1, \ldots, m$, and players compute by local operations $[e_i] = [\sum_j e_{i,j}]$.
5. Finally the players can compute by local operations $[b_i] = [\mathbf{a}_i \cdot \mathbf{s} + e_i]$, and reconstruct the b_i's by broadcasting the shares.

6.1 Security

For proving security one could show that the protocol *KeyGeneration* securely implements the functionality F_{KeyGen} defined in section 4.1. This functionality however does not reflect the influence an active adversary will have on the public key when using the protocol above. We therefore define a slightly different functionality $F_{KeyGen'}$ and use this in the security proof instead. In the end of this section we will then show that the differences in the two functionalities does not matter in terms of correctness and security.

The main difference from F_{KeyGen} is that we will have the adversary supply additional inputs before constructing and distributing keys. More specifically the adversary will supply the functionality with noise contributions used in generating the public key.

Functionality $F_{KeyGen'}$

1. When receiving "start" from all honest players, also receive from the adversary, for each corrupted player P_j shares $s_{i,j}, i = 1, \dots, n$ of the secret key to assign to P_j.
2. Choose the secret key $\mathbf{s}$ and for each subset A of size t of the players choose keys $K_A \in_R \mathbb{Z}_q$.
3. For each entry i in the secret key make a complete set of shares $s_{i,j}, i = 1, \dots, n, j = 1, \dots, u$ for each player consistent with the shares already received from the adversary. This is done by Lagrange interpolation. To each player P_j privately send his shares from $[\mathbf{s}]$ and all keys K_A where $P_j \notin A$.
4. For each corrupted player P_j receive noise contributions $e_{i,j}, i = 1, \dots, m$ for generating the public key.
5. To generate the public key choose the m vectors $\mathbf{a}_1, \dots, \mathbf{a}_m \in_R \mathbb{Z}_q^n$. For each non-corrupted player P_j choose noise contributions $e_{i,j}$ according to the distribution $\overline{\Psi}_\alpha$, the noise elements e_i are now given by $e_i = \sum_{j=1}^{u} e_{i,j}$. The public key is then given by $(\mathbf{a}_i, b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i)_{i=1}^m$.
6. Finally send the public key to all players and the adversary.

Theorem 4. *Given access to F_{Rand}, the protocol KeyGeneration securely implements the functionality $F_{KeyGen'}$. The adversary is assumed to be active and static, corrupting less than $t = u/3$ of the players.*

Proof. We must construct a simulator to work on top of $F_{KeyGen'}$, such that an adversary playing with either the simulator and $F_{KeyGen'}$ or the real world key generation protocol cannot tell the difference. By *Adv* we denote the adversary communicating with the real world and must show that we can simulate everything *Adv* sees. The simulation proceeds as follows.

1. When receiving the set of shares S from *Adv* in order to invoke "Random shared value", send "start" to $F_{KeyGen'}$ and the shares received from *Adv*.
2. To simulate the generation of the public key, first choose nm random values and send them to all players to simulate running "Random value to P".
3. When receiving the noise contributions $e_{i,j}$ from *Adv*, also give these to $F_{KeyGen'}$. Now we must simulate sharing all the noise contributions in the real protocol from the invocation of "Constrained value from P_j". Again receive the shares that corrupted players will be given from *Adv*, and for the rest simply choose random shares.
4. Finally when given the public key $(\mathbf{a}_i, b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i)_{i=1}^m$ from F_{KeyGen} we must simulate the broadcast of shares of the b_i's in the real protocol. First compute the shares the corrupted players are holding, based on the shares provided by the adversary during the simulation. Then broadcast shares of the b_i's consistent with the shares of the corrupted players. These can be computed using Lagrange interpolation.

It should be fairly clear from the above steps, that an adversary will not be able to distinguish communicating with the real protocol and the functionality with simulator. Everything that is send back and forth, the secret key shares, public key shares, K_A keys and intermediate shares are distributed exactly the same and in the same order. □

We must also prove that security is still maintained in the original cryptosystem, and furthermore that correctness and security is maintained in the distributed decryption protocol "*Decrypt*". We abbreviate $F_{KeyGen'-and-Decrypt}$ by $F_{KG'-D}$ in the following. By the functionality $F_{KG'-D}$ we denote the functionality F_{KG-D} using $F_{KeyGen'}$ instead of F_{KeyGen}.

Theorem 5. *Assume we use $F_{KeyGen'}$ to generate a key pair pk, sk and the number of players u satisfies $u\binom{u}{t} < \sqrt[10]{q}/(2m)$. If* GapSVP *is hard in the worst case, encryption under pk is semantically secure against any polynomial time adversary who gets to interact with $F_{KeyGen'}$ during key generation. Moreover, the protocol Decrypt securely implements the slightly modified functionality $F_{KG'-D}$ when given access to $F_{KeyGen'}$, in particular, decryption under sk produces the correct plaintext except with negligible probability.*

Proof. For semantic security note that by previous arguments solving decision LWE is at least as hard as solving GapSVP. First note that a ciphertext is made out of the public key $(\mathbf{a}_i, b_i = \langle \mathbf{a}_i, \mathbf{s} \rangle + e_i)_{i=1}^m$, especially if the b_i's are random, ciphertexts contain no information on the plaintext. What we then show is, that if an adversary is able to distinguish a public key generated by $F_{KeyGen'}$ from a sample from the uniform distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$, then we could use such an adversary to solve decision LWE. Now given an instance I of LWE pretend to run the $F_{KeyGen'}$ functionality with the adversary. Get the noise contributions from the adversary and add them to the LWE instance I. Return the instance as the public key to the adversary and output exactly what he outputs. If I contains uniform values so will the "public key" given to the adversary, if I is taken from $A_{\mathbf{s},\chi}$ then our output given to the adversary exactly matches the output of the real $F_{KeyGen'}$.

We will now argue that decryption is still correct except with negligible probability. Let $e + x$ be the reconstructed value after running the decryption protocol, we will then look at e. First note that the noise contributed by honest players is much smaller than that by corrupted players. We will look at the worst case where the public key is made entirely by corrupted players. We have $e = \sum_{i=1}^m r_i e_i = \sum_{i=1}^m \sum_{j=1}^u r_i e_{i,j}$, where each $e_{i,j}$ has potential size $2\binom{u}{t}\sqrt[3]{q}$. This leads to a worst case with $|e| = 2um\binom{u}{t}\sqrt[3]{q}$. According to Lemma 1 decryption will be correct if the probability that $|e| \geq \sqrt{q}$ is negligible. Therefore we get that decryption is correct if the equality $u\binom{u}{t} < \sqrt[6]{q}/(2m)$ is fulfilled.

Finally we argue that we can still simulate the execution of the protocol *Decrypt* now using the slightly modified $F_{KG'-D}$. The proof is essentially the same as the proof of Theorem 3, the only difference is that we should argue that the interval from which e is taken is still exponentially much smaller than the interval $[-\sqrt{q}, \sqrt{q}]$ from which x is taken. Following the argument from above we see that if we further limit the number of players, this can still be satisfied. Assume for instance that we limit e to the interval $[-\sqrt[2.5]{q}, \sqrt[2.5]{q}]$, this gives the requirement that the inequality $u\binom{u}{t} < \sqrt[10]{q}/(2m)$ should be fulfilled. □

7 Zero-Knowledge Proof of Plaintext Knowledge

In this section, we consider Regev's original cryptosystem, where the random choices and plaintext are binary and q is a prime. All arithmetic in this section is modulo q. In the appendix we describe a slightly more complicated scheme that works for our variant

We define the relation R_{Regev} as the set of pairs $\{x, w\}$ such that $x = ((\mathbf{a}_i, b_i)_{i=1}^m, (\mathbf{a}, b))$, and $w = (r_1, \ldots, r_m, \gamma)$ such that $(\mathbf{a}, b) = (\sum_{i=1}^m r_i \mathbf{a}_i, \gamma \cdot \lfloor \frac{q}{2} \rfloor + \sum_{i=1}^m r_i b_i)$. The language L_{Regev} will be the set of x for which there exist w with $(x, w) \in R_{Regev}$. Our goal is to build a zero-knowledge interactive proof for L_{Regev} which is also a proof of knowledge for R_{Regev}. In other words, the prover demonstrates that the ciphertext is well-formed and that he knows the plaintext and random coins that were used to form it.

We will use the technique from [IKOS07] where it was shown how to construct zero-knowledge proofs from multiparty computation protocols. We briefly sketch the idea: Assume we have a multiparty computation protocol π for input client I, players $P_1, \ldots, P_u$ and output client O, where I gets the prover's secret witness as input, shares it among the players, who then carry out a secure computation that verifies whether the witness is valid with respect to the public common input. The players send their results to O who outputs 1 or 0 accordingly. The protocol must be secure against a malicious adversary corrupting the clients and/or up to t of the other players. The prover now emulates π "in his head" and commits to the views of all players. Here, a view consists of the inputs and random coins of the player, and all received messages. The verifier selects a random subset of players among those that π can tolerate as corrupted sets[1]. The prover must open the corresponding commitments and the verifier checks that these views are consistent with each other and with the protocol and accepts or rejects accordingly.

The intuition is that the protocol is zero-knowlegde since π is secure even if the set chosen by the verifier is corrupted, and hence no information on the secret witness is released. The protocol is sound since if the witness is invalid, the prover must introduce some inconsistency to make it seem that π accepts the witness.

Indeed, it is shown in [IKOS07] that if π implements the function that checks the witness with perfect sercurity and if both u and t are $\theta(n)$, then the resulting two-party protocol has soundness error $2^{-\Omega(n)}$. It is honest verifier zero-knowledge, and can be made zero-knowledge in general, e.g., by generating the verifier's choice of subset to corrupt via a suitable coinflip protocol.

We make a couple of observations that are helpful in constructing a protocol π for our purposes: first, while broadcast is usually considered an expensive resource, it is virtually for free in this setting - any information π would broadcast can just be sent to the verifier immediately, as he would see it anyway no matter what subset is chosen. This was already noted in [IKOS07]. Second, π does

[1] The protocol must be secure against a corrupt I, but the verifier is of course not allowed to "open" I.

not have to guarantee termination, in the following sense: suppose all players broadcast some message in some round of π, and then all honest players decide (using the same procedure) whether to abort or continue. Suppose further that if all players have behaved honestly so far, we will never abort, and that further π has perfect correctness and privacy conditioned on the event that we do not abort. In this case, we can simply ask the verifier to reject if the prover sends a set of broadcast messages that would cause an abort. This will not hurt the honest prover, but will force a cheating prover to claim that he lets the virtual players behave such that π terminates.

In view of the above, all we have to do is to build an efficient protocol π that checks $r_1, \ldots, r_m, \gamma$ against $(\mathbf{a}_i, b_i)_{i=1}^m$ and $(\mathbf{a}, b)$. In order to do this, we need to borrow two tools from the design of efficient multiparty protocols, namely Packed Secret-Sharing[FY92] and Hyper-Invertible Matrices[BTH08], which we describe below.

7.1 Packed Secret Sharing

Packed Secret-Sharing is a generalization of standard Shamir sharing where secret values are assigned to more than one interpolation point. In other words, the secret to share is in fact a vector $(x_1, \ldots, x_\ell) \in \mathbb{Z}_q^l$. To do the sharing, we construct a random polynomial f of degree at most d, such that $f(0) = x_1, f(-1) = x_2, \ldots, f(-\ell+1) = x_\ell$. The shares are, as usual, $f(1), \ldots, f(u)$. To make this possible, and to guarantee privacy against t corrupted players, d must be at least $t+\ell-1$. In our case, we will choose $\ell = n+1$, and t to be $\theta(n)$. Furthermore, we will need that there are sufficiently many honest players such that their shares alone can determine a polynomial of degree $2d$, i.e., $u - t \geq 2(t+n+1)$. This shows that we can indeed choose u to be $\theta(n)$, as promised above.

Note that to ensure that we have enough distinct evaluation points, we need that if q is a prime, it must be larger than $\ell + u = n + 1 + u$ which is $\theta(n)$ or, in our construction of q for the threshold scheme, the smallest prime factor must be larger than $\ell + u$. This is already satisfied by the schemes as they stand.

We will write $[\mathbf{z}]_d$ for a set of shares determining a packed sharing of the block $\mathbf{z}$ using a polynomial of degree d.

Note that if players locally add respectively multiply their shares of blocks $\mathbf{z}, \mathbf{z'}$, this results in shares in the coordinate-wise sum respectively product, i.e., we have $[\mathbf{z}]_d + [\mathbf{z'}]_d = [\mathbf{z}+\mathbf{z'}]_d$, and $[\mathbf{z}]_d * [\mathbf{z'}]_d = [\mathbf{z} * \mathbf{z'}]_{2d}$, where $*$ denotes the coordinate-wise product.

7.2 Hyper-Invertible Matrices

A hyper-invertible matrix M (with entries in $\mathbb{Z}_q$) has the property that any square submatrix of M is invertible. Such matrices can be constructed from Van der Monde matrices and were used in [BTH08] to check consistency of secret sharings with zero error probability. We briefly explain how this works:

Suppose M is a matrix with u rows and $u - t$ columns. Suppose the players hold $u - t$ sets of shares $[\mathbf{z}_1], \ldots, [\mathbf{z}_{u-t}]$, and we want to check that each set of shares is consistent with a polynomial of degree at most e. The players can locally compute u new sets of shares,

$$[M(\mathbf{z}_1,\ldots,\mathbf{z}_{u-t})_1],\ldots,[M(\mathbf{z}_1,\ldots,\mathbf{z}_{u-t})_u] := [\mathbf{y}_1],\ldots,[\mathbf{y}_u],$$

simply by multiplying M on the vector of $u-t$ shares that they hold (thinking of the shares as a column vector). Assume now that for $i = 1,\ldots,u$, each player sends his share in $[\mathbf{y}_i]$ to P_i. This allows P_i to check that the shares he receives are e-consistent, i.e., on a polynomial of degree at most e. P_i can now broadcast whether his check was OK or not.

We can see that if all players are happy, it means in particular that all honest players are happy, and that they therefore agree with all honest players on the set of $u-t$ e-consistent shares that they checked. I.e., $\{[\mathbf{y}_j]\}_{j\in H}$, where H is the set of honest players, are all e-consistent. Let M_H be the matrix we get from M by only taking the rows corresponding to players in H. This matrix is invertible by assumption on M, so we can obtain $[\mathbf{z}_1],\ldots,[\mathbf{z}_{u-t}]$ as a linear function defined by M_H^{-1} of the the shares in $\{[\mathbf{y}_j]\}_{j\in H}$, and hence the $[\mathbf{z}_i]$'s are all e-consistent as well.

Furthermore, if it is important that the shared information is kept secret, one can arrange the input shares such that only $[\mathbf{z}_1],\ldots,[\mathbf{z}_{u-2t}]$ contains information we want to protect, while $[\mathbf{z}_{u-2t+1}],\ldots,[\mathbf{z}_{u-t}]$ are chosen randomly using polynomials of degree at most e. These t random sets of shares will randomize the t sets of shares seen by corrupt players, again by hyper-invertibility of M. This also means that we do not need, for instance, $[\mathbf{z}_1]$ to be a *random* sharing of $\mathbf{z}_1$ to be able to hide it.

Note also that this method can be used to also check if $\mathbf{z}_1,\ldots,\mathbf{z}_{u-t}$ all satisfy some fixed condition, as long as what the condition asks is that each $\mathbf{z}_i$ satisfies some linear equation. For instance, we might want to check that $\mathbf{z}_i = (0,\ldots,0)$ for all i. This is done by having players verify that all $\mathbf{y}_i$ satisfy the same condition.

Regarding the complexity, it is easy to see that a set of shares of total size T bits can be verified while keeping the shared information perfectly private by sending $O(T)$ bits and creating random shares of size $O(T)$ bits.

7.3 The Multiparty Protocol

Recall that the secret witness to be checked consists of binary values $r_1,\ldots,r_m,\gamma$ where $(\mathbf{a},b) = (\sum_{i=1}^m r_i\mathbf{a}_i, \gamma\cdot\lfloor\frac{q}{2}\rfloor+\sum_{i=1}^m r_i b_i)$, and where the public information is public key $(\mathbf{a}_i,b_i)_{i=1}^m$ and ciphertext $(\mathbf{a},b)$. For any $z\in\mathbb{Z}_q$, we set $\overline{z} = (z,z,...,z)$, a vector of length $n+1$. The protocol works as follows:

Protocol VerifyCiphertext

1. The input client I sends shares $[\overline{r_i}]_d$, $i = 1,\ldots,m$ and $[\overline{\gamma}]_d$ to the players. In addition, it also sends random shares as required for the verifications below using the hyper-invertible matrix M.

2. Verify that $[\overline{r_1}]_d, \ldots, [\overline{r_m}]_d, [\overline{\gamma}]_d$ are d-consistent and that in each block shared, all $n+1$ entries are equal. If any player broadcasts "not OK", the protocol aborts.
3. Compute, using local multiplications, $[\overline{r_i(1-r_i)}]_{2d}$ for $i = 1, \ldots, m$ and $[\overline{\gamma(1-\gamma)}]_{2d}$.
4. Form sharings of the public vectors: $[(\mathbf{a}_i, b_i)]_d$, $i = 1, \ldots, m$, $[(0, \ldots, 0, \lfloor \frac{q}{2} \rfloor)]_d$, and $[(\mathbf{a}, b)]_d$ (using some default choice of polynomial of degree at most d). We then emulate the encryption on the shared values: compute, using local computation,

$$[\sum_{i=1}^{m} \overline{r_i} * (\mathbf{a}_i, b_i)]_{2d} + [(0, \ldots, 0, \lfloor \frac{q}{2} \rfloor) * \overline{\gamma}]_{2d} = [(\sum_{i=1}^{m} r_i \mathbf{a}_i, \sum_{i=1}^{m} r_i b_i + \gamma \lfloor \frac{q}{2} \rfloor)]_{2d}$$

From this, we locally subtract shares of the ciphertext $[(\mathbf{a}, b)]_d$, so we get

$$[(\sum_{i=1}^{m} r_i \mathbf{a}_i - \mathbf{a}, \sum_{i=1}^{m} r_i b_i + \gamma \lfloor \frac{q}{2} \rfloor) - b]_{2d} := [(\mathbf{z}, v)]_{2d}$$

5. Verify that $[\overline{r_1(1-r_1)}]_{2d}, \ldots, [\overline{r_m(1-r_m)}]_{2d}$, $[\overline{\gamma(1-\gamma)}]_{2d}$ and $[(\mathbf{z}, v)]_{2d}$ are indeed $2d$-consistent sharings of all-zero blocks. If any player broadcasts "not OK", the protocol aborts. This ensures that the r_i's and γ are binary, and that encryption results in the claimed ciphertext.

Since the verifications of shares work with zero error probability, it is clear that if the protocol terminates successfully, we are guaranteed that the shared values determine the correct ciphertext. No information on the secret is released, since the only communication is what is required for the verification of sharings, and we already argued above that these release no information on the shared values that we verify.

Regarding complexity, it is clear from inspection of the protocol that it is completely determined by the total size T of the sharings $[\overline{r_i}]_d$, $i = 1, \ldots, m$ and $[\overline{\gamma}]_d$, in particular, the total size of communication is $O(T)$. We have that T is $O(mu \log q)$ which is $O(mn \log q)$. Note that the size of the public key is also $O(mn \log q)$.

It is described in [IKOS07] how to transform this protocol into a zero-knowledge proof using an unconditionally binding commitment scheme. If this scheme allows us to commit to strings with an additive length increase that is independent of the string length, we can preserve the efficiency of the multiparty protocol. An unconditionally hiding commitment scheme is also needed, for the verifier to commit to his challenge. This gives us:

Theorem 6. *Given an unconditionally binding and an unconditionally hiding commitment scheme with constant additive overhead, using protocol VerifyCiphertext in the construction from [IKOS07] produces a two-party zero-knowledge proof for L_{Regev}. The protocol has communication complexity $O(mn \log q)$ bits and error probability $2^{-\Omega(n)}$.*

We can base the commitment schemes needed on lattice problems, thus using assumptions we would need anyway. An efficient unconditionally binding scheme follows from the cryptosystem in[PVW08], while an unconditionally hiding scheme can be based on any collision intractable hash function [DPP98], which in turn can be based on lattice assumptions.

In [IKOS07], it was not shown that their construction is a proof of knowledge for R_{Regev}. However, for the honest verifier zero-knowledge version of the protocol, one can do a rewinding argument to show that it is indeed a proof of knowledge with negligible knowledge error. If we go to the version that is zero-knowledge in general, things are different, since the construction from [IKOS07] has the verifier commit to his challenges, which means rewinding the prover is not possible unless the extractor can equivocate these commitments.

However, in the common reference string model, we can easily make the protocol be a proof of knowledge for R_{Regev}, by having a public key for a commitment scheme placed in the reference string, e.g., a public key for the cryptosystem from [PVW08], and the prover uses these for committing to the views. If the extractor knows the corresponding secret key, it can extract all committed views without rewinding and easily compute the secret.

References

[BTH08] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure mpc with linear communication complexity. In *TCC*, pages 213–230, 2008.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

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

[DPP98] Ivan Damgård, Torben P. Pedersen, and Birgit Pfitzmann. Statistical secrecy and multibit commitments. *IEEE Transactions on Information Theory*, 44(3):1143–1151, 1998.

[FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *STOC*, pages 699–710, 1992.

[IKOS07] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In *STOC*, pages 21–30, 2007.

[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*, pages 333–342, 2009.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, pages 554–571, 2008.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.

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

A Zero-Knowledge Proof when q is not prime

The only part of the multiparty protocol underlying our zero-knowledge proof that does not work when q is not a prime is the step where it is verified that the r_i are binary, essentially by verifying that $r_i(1-r_i) \bmod q = 0$. Of course, this check is not good if q is not prime. We sketch a procedure that can be used instead, but has only statistical security:

The input client I supplies $[\overline{r_i}]_d$ and it is checked as in the original protocol that a block has been shared where all entries are equal. Note that if the sharing was correctly formed, it would be the case that $\mathbf{r}' = 2\overline{r_i} - (1, \ldots, 1)$ would be $(1, \ldots, 1)$ or $(-1, \ldots, -1)$. I also supplies a sharing $[\mathbf{z}]_d = [(z_1, \ldots, z_{n+1}]_d$ such that all z_i are randomly chosen to be 1 or -1. Finally, a public random challenge is generated: $\mathbf{v} = (v_1, \ldots, v_{n+1})$, where each v_i is 0 or 1. (When transforming this to a 2-party protocol, we let the verifier generate the challenge). We compute (locally)

$$[\mathbf{r}' * \mathbf{z} * \mathbf{v} + \mathbf{z} * (\overline{1} - \mathbf{v})]_{3d}.$$

Finally we add a random degree $3d$-sharing of the all-zero block and open the result. The opened block must contain only 1's and -1's. Put another way, the opening shows us, in each coordinate position, an entry from $\mathbf{z}$ or from $\mathbf{r}' * \mathbf{z}$ and they must all be ± 1.

For privacy, the intuition is that by random choice of $\mathbf{z}$, $\mathbf{r}' * \mathbf{z}$ has no information on $\mathbf{r}'$ and neither does $\mathbf{z}$, so seeing, for each index i, the i'th entry of $\mathbf{r}' * \mathbf{z}$ or $\mathbf{z}$ reveals nothing on $\mathbf{r}'$.

For correctness, if there is just a single position in which both $\mathbf{z}$ and $\mathbf{r}' * \mathbf{z}$ are ± 1, $\mathbf{r}'$ will be ± 1 in that position too, and this implies that the original r_i was 0 or 1. On the other hand, if no such position exists, the honest players will accept $[\overline{r_i}]$ with probability only 2^{-n-1}, by the assumed randomness of $\mathbf{v}$.