Space-efficient variants of cryptosystems based on learning with errors

Steven D. Galbraith

Mathematics Department,
University of Auckland,
New Zealand.
S.Galbraith@math.auckland.ac.nz

Abstract. We consider public key encryption based on the learning with errors problem (LWE). There are several reasons why this encryption scheme is not considered to be practical, and one of the most commonly cited is the size of the public key. However, there is a simple way to greatly reduce the size of the public key. The aim of this paper is to investigate such ideas and to consider some ways to implement LWE encryption on devices with relatively small storage. We remark that a more natural approach for such settings would be to use NTRU or Ring-LWE; we also apply our ideas to these cases.

Some of the variants we propose no longer have the same rigorous security guarantees enjoyed by standard encryption based on LWE. Hence, the bulk of the paper consists of remarks about security issues. In particular, we study variants of LWE where the coefficients of the public key matrix are not chosen uniformly modulo q but are instead "small". We give evidence that binary matrices might be secure for LWE encryption. We remark that binary matrices are not secure for Ring-LWE encryption.

The aim of the paper is not to make a complete proposal for practical use, but to raise some questions and to introduce some computational problems that may be interesting targets for cryptanalysis. We hope that scrutiny of these problems will shed further light on the practical aspects of the standard LWE problem.

Keywords: learning with errors, short public keys, pseudorandom generators.

1 Introduction

Regev [11] proposed the learning with errors problem (LWE) and it has lead to an explosion in new research on lattice-based public key cryptography. Several features of these systems are notable:

1. The security of the cryptosystems relies on well-defined worst-case computational assumptions in lattices.
2. The cryptosystems are simple to implement, and fast.
3. The computational assumptions are not known to be vulnerable to quantum computers.

The public key for LWE encryption includes a "randomly chosen" $m \times n$ matrix $\mathbf{A}$ over $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$, and this leads to rather large storage requirements. The goal of this paper is to consider variants of LWE encryption that would be more suitable for devices with small storage.

A simple observation to obtain short public keys for LWE is that, rather than publishing $\mathbf{A}$, one could publish a seed such that the entries of $\mathbf{A}$ are obtained as the output of a pseudorandom generator initialized with the value of the seed (note that this process is exactly what would be done in the key generation algorithm anyway). We mention that Coron et al [3] have used a similar idea to compress public keys for a cryptosystem whose security relies on computational problems in lattices. The main contribution of our paper is to argue that it is possible to implement LWE cryptosystems on constrained devices with relatively little storage and without seriously damaging the speed of the systems.

We remark that, for constrained devices, it would be more natural to use NTRU [4], or encryption based on Ring-LWE [7], or the Stehlé-Steinfeld variant of NTRU [14]. We also discuss the use of our idea in these situations. Section 7.6 gives a remark, due to Steinfeld, that shows that one cannot restrict to matrices with small coefficients when using Ring-LWE.

Regarding our technical contributions, we show that LWE is well-defined for binary matrices (and for other matrices with entries from certain subsets of $\mathbb{Z}_q$) and, in Section 7.2, we give a new algorithm for finding $\mathbf{u}$ given $(\mathbf{A}, \mathbf{u}^T\mathbf{A})$ over $\mathbb{Z}$. We also make a number of remarks about LWE that are possibly in the folklore, but that may be of interest to some readers.

The paper is organised as follows. Section 2 recalls the LWE problem and the basic encryption scheme based on it. Section 3 discusses the simple idea for short keys and discusses related remarks in the literature. Section 4 describes some approaches for implementing LWE cryptosystems efficiently on devices with very low storage. Section 5 recalls the basic security issues surrounding the choice of the public key matrix $\mathbf{A}$ in LWE (more details of attacks are recalled in Section A of the Appendices). Section 6 discusses pseudorandom generators that may be appropriate for our application. On suggestion in this section is to choose the matrix $\mathbf{A}$ to have entries chosen from $\{0, 1\}$. Section 7 discusses the potential security implications of using binary matrices in LWE. Several original observations are made. Section 8 is concerned with an "intermediate" case between binary matrices $\mathbf{A}$ and matrices with entries chosen uniformly from $\mathbb{Z}_q$. Section 9 summarises our results and gives some specific targets for cryptanalysis.

2 The LWE cryptosystem

We recall the basic LWE encryption scheme, as invented by Regev [11, 12]. The private key is a length n column vector $\mathbf{s}$ over $\mathbb{Z}_q$, where q is prime. (All vectors in the paper will be column vectors and we write $\mathbf{s}^T$ for the transpose.) The public key is a pair $(\mathbf{A}, \mathbf{b})$ where $\mathbf{A}$ is a randomly chosen $m \times n$ matrix over $\mathbb{Z}_q$ with rows $\mathbf{a}_i^T$, and $\mathbf{b}$ is a length m column vector with entries $b_i = \mathbf{a}_i^T\mathbf{s} + e \pmod q$ such that e is sampled from a discrete normal distribution on $\{-(q-1)/2, \ldots, -1, 0, 1, \ldots, (q-1)/2\}$ (with mean 0 and standard deviation σ).

To encrypt a message $x \in \{0, 1\}$ to a user one chooses a row vector $\mathbf{u}^T \in \{0, 1\}^m$ (i.e., the entries of $\mathbf{u}$ are chosen independently and uniformly in $\{0, 1\}$) and computes

$$(C_1, C_2) = (\mathbf{u}^T\mathbf{A} \pmod q, \mathbf{u}^T\mathbf{b} + x\lfloor q/2 \rfloor \pmod q).$$

Decryption is to compute $y = C_2 - C_1\mathbf{s} \pmod q$ and determine x by seeing if y is "closer" to 0 or $\lfloor q/2 \rfloor$. Note that decryption fails with some probability, which may be made arbitrarily small by taking q sufficiently large compared with m and σ (see Lemma 5.1 of Regev [12]).

One can encrypt many bits by either repeating the above process for each bit or by using a single $\mathbf{A}$ and different values for $\mathbf{s}$ (e.g., see Lindner and Peikert [6]). There are also many other cryptosystems based on LWE and ideas could be applied to any of them.

Both key generation and encryption require generating randomness. Note that key generation requires sampling from the Gaussian error distribution, which is not an entirely trivial computation (the best current method in the literature is Peikert [10]). There is a dual version of LWE: in this case the public key is $\mathbf{A}$ and $\mathbf{u}^T\mathbf{A} \pmod q$, and the ciphertext involves computing $C_1 = \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \pmod q$ and $C_2 = \mathbf{u}^T\mathbf{b} + x\lfloor q/2 \rfloor \pmod q$. The disadvantage, in practice, of this method is that the encryptor now has to perform the Gaussian sampling. Hence, for constrained devices, we believe the original encryption scheme will be more practical than the dual scheme.

3 Short public keys

To encrypt to a user one must store their public key, which in this case is $(\mathbf{A}, \mathbf{b})$, consisting of $m(n+1)$ elements of $\mathbb{Z}_q$. Lindner and Peikert [6] suggest

$$(n, m, q) = (256, 640, 4093)$$

so the public key requires $640 \cdot 257 \cdot \log_2(4093) = 1973586$ bits, which is about 247 kilobytes – way too much for a constrained device (such as a smartcard, embedded device, or mobile phone).

In contrast, an RSA public key is around 2048 bits and a public key for elliptic curve cryptography is usually fewer than 1000 bits, both significantly less than one kilobyte.

It was observed by Regev (page 32 of [12]) that the matrix $\mathbf{A}$ could be fixed for all users and "hard-wired" into the encryption software. Precisely he says:

> In fact, it is possible to reduce the size of the public key . . . by the following idea of Ajtai. Assume all users of the cryptosystem share some fixed (and trusted) random choice of $\mathbf{a}_1, \ldots, \mathbf{a}_m$. This can be achieved by, say, distributing these vectors as part of the encryption and decryption software. Then the public key need only consist of $(b_1, \ldots, b_m)$. This modification does not affect the security of the cryptosystem.

Note that it is still the case that the matrix $\mathbf{A}$ must be stored somewhere on the device. The above suggestion does not solve the problem of implementing LWE encryption on low-storage devices. A similar remark is made by Lindner and Peikert [6] (they split the public key as $\mathbf{A}$ and $\mathbf{P}$ where $\mathbf{A}$ is fixed for all users).

How is the matrix $\mathbf{A}$ chosen in the first place? The natural way to implement the LWE key generation algorithm is to generate the matrix $\mathbf{A}$ using a pseudorandom number generator. One therefore starts with a short "seed" (typically 128 or 256 bits in length) and then uses the generator to produce mn elements of $\mathbb{Z}_q$. These are the entries of the matrix $\mathbf{A}$.

A simple observation to obtain short public keys for LWE is that one could publish the seed, rather than the matrix $\mathbf{A}$. A user can then generate the matrix $\mathbf{A}$, whenever it is needed, using the pseudorandom generator. The public key is therefore the seed and the value $\mathbf{b}$. The public key requires $256 + m\log_2(q) = 256 + 640 \cdot \log_2(4093) = 7935$ bits, which is about one kilobyte. This is now quite practical for a smartcard. However, one must also consider the implications on encryption times of re-constructing the public key.

Ajtai [1] in fact uses a pseudorandom generator as part of his protocol, but not to create a shorter public key. As noted already, Coron et al [3] also use a pseudorandom generator to shorten public keys for a different lattice-based cryptosystem.

The aim of the paper is to consider some consequences of this design decision. In particular, we consider how to implement LWE encryption in this setting, suggest some choices for the pseudorandom generator, and consider security implications.

Note that a similar idea can be used to compress the public keys for systems based on Ring-LWE. In this situation the public key "matrix" requires $n\log(q)$ bits of storage, and this can be reduced to a single seed. The improvement is not so significant in this case, but could still have some relevance to very constrained devices.

4 Encryption process

As described above, the memory-intensive part of encryption is computing $C_1 = \mathbf{u}^T\mathbf{A} \pmod q$ where $\mathbf{u}^T \in \{0,1\}^m$ is a randomly chosen row vector. (One also has to compute $\mathbf{u}^T\mathbf{b} \pmod q$, but this is independent of $\mathbf{A}$ and does not require large amounts of memory to compute.) We give three approaches to computing $\mathbf{u}^T\mathbf{A} \pmod q$ that do not require large storage. In particular, it is not necessary to generate and store the whole of $\mathbf{A}$ on the device at any one time.

Vector-wise Initialise the vector C_1 (of length n) to zero. Run the pseudorandom generator to compute the rows of $\mathbf{A}$ one-by-one. Each time the corresponding bit of $\mathbf{u}$ is one (it is not necessary to store $\mathbf{u}$, one simply generates a fresh pseudorandom bit each time a new row is generated, and also updates the current value of C_2 accordingly) then that row of $\mathbf{A}$ is added to the row vector C_1. Hence, the total memory required to compute C_1 is just the n elements of $\mathbb{Z}_q$ (to store the current version of C_1), another n elements to store the current row of $\mathbf{A}$, and any other storage needed for the pseudorandom generator (only a few more bits). Hence, the additional storage needed to compute the ciphertext is roughly $2n\log_2(q)$ bits (which is around $512 \cdot \log_2(4093) \approx 6143$ bits, less than one kilobyte).

Entry-wise Initialise the vector C_1 to zero. Run the pseudorandom generator to compute the entries of each row of $\mathbf{A}$ one-by-one. Each time the corresponding bit of $\mathbf{u}$ is one (again, the whole of $\mathbf{u}$ is never stored), then that entry of $\mathbf{A}$ is added to the corresponding entry of the vector C_1. It follows that only one entry of $\mathbf{A}$ need be stored at any given time. Hence, the total memory required to compute C_1 is just the n elements of $\mathbb{Z}_q$ (to store the current version of C_1), the current entry of $\mathbf{A}$, and any other storage needed for the pseudorandom generator (only a few more bits). Hence, the additional storage needed to compute the ciphertext is roughly $n\log_2(q)$ bits, less than 400 bytes for our sample parameters.

Very small We now consider a scenario where the device does not even have enough storage for the whole vector C_1 (of length n with entries in $\mathbb{Z}_q$). Instead, we will "stream" the entries of C_1 from the device to the reader. We also do not want to store $\mathbf{u}$ in local memory (it is m bits, and m is a similar size to $n\log(q)$, the number of bits needed to store C_1). So assume a pseudorandom generator outputs the bits of $\mathbf{u}$, and another pseudorandom generator outputs the entries of $\mathbf{A}$ column by column. Denote by $\mathbf{A}_j$ the j-th column of $\mathbf{A}$. Running both generators together one can compute, for $1 \le j \le n$, the entry $C_{1,j} = \mathbf{u}^T\mathbf{A}_j \pmod q$, using just two elements of $\mathbb{Z}_q$ (one for the current entry of $\mathbf{A}_j$ and the other accumulating the value of $C_{1,j}$).

Of course, the device must have enough storage for the public key $\mathbf{b}$, which is $n \log(q)$ bits. So this scenario is possibly not likely. But it is notable that the additional memory requirement, over and above the storage of $\mathbf{b}$, can essentially be made as small as two seeds and a couple of elements of $\mathbb{Z}_q$, say $2 \cdot 256 + 2 \cdot \log_2(4093) \approx 536$ bits (67 bytes).

5 Security

For a theoretical analysis one could require that the pseudorandom generator generates a matrix $\mathbf{A}$ that is indistinguishable from a random $m \times n$ matrix with entries in $\mathbb{Z}_q$. Under this assumption, the scheme would enjoy precisely the same security guarantee as standard LWE.

For example, suppose one has a random oracle $H : \mathbb{Z}_q^n \to \mathbb{Z}_q^n$ and builds a pseudorandom generator from H with seed $x_0 \in \mathbb{Z}_q^n$ and generating rule $x_{n+1} = H(x_n)$. Then the first m outputs of the generator provide the rows of the matrix $\mathbf{A}$. The random oracle can then be "programmed" to consist of any desired matrix for Regev's security reduction.

Hence, we have an encryption scheme that requires low storage and that enjoys (in the random oracle model) the full security assurance of standard LWE encryption. One would then conjecture that if H is replaced by a "good enough" hash function then the scheme would still be secure. However, there would potentially be a significant performance overhead compared with standard LWE encryption.

In reality it should not be necessary to consider extremely strong pseudorandom generators. We now consider what properties of $\mathbf{A}$ are required to resist the standard attacks on LWE (these attacks are summarised in Section A). The main properties required for the matrix $\mathbf{A}$ are:

1. It should be hard to find a "nice" basis for the lattice
$$L = \{\mathbf{v} \in \mathbb{Z}^m : \mathbf{v} \equiv \mathbf{A}\mathbf{u} \pmod{q} \text{ for some } \mathbf{u} \in \mathbb{Z}^n\}.$$
This security requirement is needed to prevent an attack on the encryption scheme based on solving the closest vector problem; see Section A.1.
2. It should be hard to solve the equation $C_1 = \mathbf{u}^T\mathbf{A} \pmod{q}$ for $\mathbf{u} \in \{0,1\}^m$. Otherwise, one can break the encryption by computing $C_2 - \mathbf{u}^T\mathbf{b} \pmod{q}$ (under the reasonable assumption that the vector $\mathbf{u}$ is unique); see Section A.3.
3. It should be hard to find short vectors $\mathbf{w}$ such that $\mathbf{w}^T\mathbf{A} \equiv 0 \pmod{q}$. This is needed to ensure that the decisional-LWE problem is hard (see Section A.2). In particular, we do not want $\mathbf{A}$ to have repeated rows, or simple linear relationships between neighbouring rows.
The precise relevance of this property to the encryption scheme seems to be more subtle and is briefly discussed at the end of Section A.2. If one restricts $\mathbf{A}$ to a certain class of matrices then it seems that short vectors in the left kernel of $\mathbf{A}$ do not directly lead to an attack on an associated ciphertext. Hence, in practice one might be able to relax this requirement on the matrix $\mathbf{A}$.

When the matrix $\mathbf{A}$ is randomly chosen then the above properties are likely to hold, but if $\mathbf{A}$ is constructed by some non-random process then it is possible that some of these properties may not hold. Indeed, later in the paper we indicate situations where one or more of these properties may not hold.

However, it is quite reasonable to suppose that families of matrices could be considered that are far from uniformly distributed, but nevertheless have these properties. As a result, it seems appropriate to consider pseudorandom generators that are very efficient and that perhaps do not have uniform output. Nevertheless, the security will now require some assumption about the pseudorandom generator.

6 Suitable pseudorandom generators

We now consider the choice of pseudorandom generator. We need this to be very fast, so that reconstructing $\mathbf{A}$ (as in Section 4) does not slow down encryption too much. Certain operations need to be implemented on any device that performs LWE encryption. In particular, one needs addition and multiplication in $\mathbb{F}_q$. There might also be specific

optimised code for computing inner products of length n vectors over $\mathbb{F}_q$. It is natural to try to design a pseudorandom generator that is also based on these functions.

The literature on pseudorandom generators is, understandably, large. We present several well-known generators. Note that the generators we study are not suitable for most cryptographic purposes (e.g., generating random nonces in protocols) as they do not have a large seed and hence can be easily predicted. Such attacks are irrelevant in our setting, as **A** is a public key. The only goal is to ensure that **A** satisfies the properties listed in Section 5, and this does not seem to require a cryptographically strong pseudorandom generator.

1. The *Linear Congruential Generator* (see Section 3.2.1 of Knuth [5]) is given by the recurrence $x_{i+1} = ax_i + b \pmod q$. This is fast and simple, and seems appropriate for use on a device that already contains arithmetic modulo q. However, its period is at most $q-1$, which is typically much shorter than nm (the number of entries needed for the martrix). Our example parameters have $nm = 163840 \approx 40q$. The fact that the sequence repeats does not immediately lead to short linear relations among the rows of the matrix (unless the period is a multiple of n), but one would naturally be nervous about it. We refer to Appendix **??** for further discussion.
   To extend the period one could use the *Generalised Linear Congruential Generator* (see Section 3.2.2 of Knuth [5]) $x_{i+1} = \sum_{j=0}^{k-1} a_j x_{i-j} \pmod q$. This can have period $q^k - 1$, so even taking $k = 2$ is enough to ensure that the period is larger than nm with the above parameters. It is quite fast, though performing k multiplications modulo q for each entry of A is a significant overhead.
   The existence of linear relations among the entries does not automatically lead to problems with the scheme. The scheme is vulnerable if there are "short" linear relations among the rows of A. It is unclear that short linear relations between the entries of the matrix would lead to short linear relations among the rows. We some examples in Appendix **??**.
2. Generic pseudorandom generators obtained from block ciphers or hash functions. These are stronger than we need, and would lead to significant overhead in terms of running time and code space (though, depending on the context, the device may necessarily already contain an implementation of a hash function or block cipher). There is also the question of turning bits into integers modulo q. One approach would be to form an integer modulo q from $k = \lfloor \log_2(q) \rfloor$ consecutive bits of the output (so that not every integer modulo q arises); this might be suitable when $q = 2^k + \epsilon$ where ϵ is a small integer. When $q = 2^k - \epsilon$ (such as $4093 = 2^{12} - 3$) then it would also be natural to generate a random k bit integer (it is not worth reducing modulo q at this stage, it will be handled during the computation of C_1).
   A more audacious suggestion is to choose **A** to be a binary matrix. In other words, an n-bit binary output provides n entries of the matrix **A**. In Section 7 we discuss some security implications of using binary matrices **A**; as far as we can tell, such systems may indeed be secure for LWE cryptosystems. Section 8 considers the case where the entries of **A** are chosen from a larger set than $\{0, 1\}$ but a smaller set than $\mathbb{Z}_q$ (e.g., in the range $[0, 2^k - 1]$ where $2^k \approx 8q/m$ as discussed in Section 8); it seems that this case could be a secure intermediate case that still allows fast generation using binary strings.
3. There are much simpler pseudorandom generators producing binary strings. One general approach is to generate a sequence of length w binary (row) vectors $\mathbf{x}$ by the rule $\mathbf{x}_{i+n} = \mathbf{x}_{i+m} \oplus (\mathbf{x}_i R)$ where $0 < m < n$ are integers and R is a $w \times w$ binary matrix.
   One well-known variant of this framework is the *Mersenne Twister* generator due to Matsumoto and Nishimura [8]. The parameters (w, n, m) for this variant are $(32, 624, 397)$. In other words, one must store the previous 624 states of the algorithm, requiring $624 \cdot 32 = 19968$ bits (approximately 2.4 kilobytes). This is not disasterous, but it would be preferable to use less storage when implementing on constrained devices.
4. We expect that much more lightweight pseudorandom word generators could be used in our application without loss of security. The desired properties of such a pseudorandom word generator are: just a few clock cycles needed for each new word; low storage requirements (unlike the Mersenne twister); no simple relations in the matrix obtained from the output of the generator. Using such a generator, the performance of the low-storage LWE encryption scheme would be almost the same as the performance of the standard method.
   For example, a simple way to generate a pseudorandom binary sequence is a linear feedback shift register (LFSR). These can be implemented using shift operations and XOR, so only need a couple of clock cycles at each iteration. However, these are usually considered as a method to generate sequences of bits, rather than sequences of words. Their linearity is not necessarily a problem for our application.

7 Binary matrices

As mentioned above, since most previous work on pseudorandom generators involves generating binary strings it is natural to wonder whether the matrix $\mathbf{A}$ can be chosen to have entries in $\{0,1\}$. This would enable very fast generation of $\mathbf{A}$. This is an audacious suggestion, and we now discuss some potential risks associated with it.

Consider the basic attacks as summarised in Section A. Several of these attacks required finding a nice basis for the either of the lattices $\{\mathbf{v} \in \mathbb{Z}^m : \mathbf{v} \equiv \mathbf{A}\mathbf{u} \pmod q \text{ for some } \mathbf{u} \in \mathbb{Z}^n\}$ or $\{\mathbf{v} \in \mathbb{Z}^m : \mathbf{v}^T\mathbf{A} \equiv 0 \pmod q\}$. The first observation is that this problem still appears to be hard for binary matrices. We are not aware of any literature on this problem. However, there are several other security issues that arise when using binary matrices. We discuss them now.

7.1 Binary-LWE is well-defined

The first issue is whether LWE is well-defined for binary matrices $\mathbf{A}$. This is a serious question, as the general approach to showing that LWE is well-defined is to consider a guess $\mathbf{s}'$ for the secret $\mathbf{s}$ and to consider the corresponding "error vector" $\mathbf{e}' = \mathbf{b} - \mathbf{A}\mathbf{s}' \pmod q$; the proof then uses the fact that the entries of $\mathbf{A}$ are chosen uniformly at random to argue that if $\mathbf{s} \neq \mathbf{s}'$ then the entries of $\mathbf{e}'$ are uniformly distributed. But if the entries of $\mathbf{A}$ are binary then this argument breaks down. Indeed, if $\mathbf{s}' = \mathbf{s} + (1, 0, \ldots, 0)^T$ then $\mathbf{e}' = \mathbf{e} + (a_{1,1}, a_{2,1}, \ldots, a_{m,1})^T$, which is still a short vector (though its entries no longer have mean close to zero, as they would be if the error vector had been sampled correctly).

Theorem 1 in Appendix C shows that LWE is well-defined in this case. The interpretation of this result in practice is a little less clear. One interpretation would be that a much larger value for m should be used. Another interpretation might be that moderate sized m can still be used, but that the computational assumption be weakened to allow for the fact that there might be a somewhat large and fuzzy set of possible "solutions" $\mathbf{s}$ to the LWE instance. For the rest of this paper we take the latter approach: We assume that m can be taken to be the same size as used by other authors, and that finding a vector $\mathbf{s}$ such that $\mathbf{b} - \mathbf{A}\mathbf{s}' \pmod q$ is "small" is still a hard computational problem (this assumption is just CVP restricted to integer lattices given by binary matrices).

7.2 An attack on the nonce used in encryption

One natural line of attack is to try to determine the binary vector $\mathbf{u}$ from the first component $C_1 = \mathbf{u}^T\mathbf{A} \pmod q$ of the ciphertext. Note that, if the entries of $\mathbf{u}$ and $\mathbf{A}$ are chosen uniformly at random in $\{0,1\}$ then the expected value of an entry of C_1 is $m/4$ (since one is computing a sum of m terms $u_i A_{i,j}$, and each is 0 with probability $3/4$) and the maximal value is m, which is less than q for our example parameters $(m, q) = (640, 4093)$. Hence, there is no modular reduction taking place and we are solving a "vector subset-sum" problem over $\mathbb{Z}$. Note first that the solution space over $\mathbb{Q}$ of the system $C_1 = \mathbf{u}^T\mathbf{A}$ typically has dimension $m - n$ and that solutions can be computed easily. The problem is to find a solution with binary entries.

There is the obvious lattice attack: Find some solution $\mathbf{w} \in \mathbb{Z}^m$ such that $C_1 = \mathbf{w}^T\mathbf{A} \pmod q$, then solve CVP to $\mathbf{w}$ in the lattice $L = \{\mathbf{v} \in \mathbb{Z}^m : \mathbf{v}^T\mathbf{A} \equiv 0 \pmod q\}$. The hope is that $\mathbf{w} - \mathbf{v}$ is the vector $\mathbf{u}$ we are looking for.

This attack can be implemented as follows: Consider the lattice $L = \{(\mathbf{v}, v_{m+1}) \in \mathbb{Z}^{m+1} : \mathbf{v}^T\mathbf{A} - v_{m+1}C_1 = 0\}$. A basis for this lattice can be found by taking the Hermite normal form of the $(m+1) \times n$ matrix formed by adding the row $-C_1$ to $\mathbf{A}$. One can then use LLL to find a short vector $\mathbf{v}$ in L; one hopes that $\mathbf{v}$ is of the form $(\mathbf{u}, 1)$ where $\mathbf{u}$ is a binary vector. This process works well in practice as long as m is not too large. Experiments using Magma with $n = 256$ take only a few seconds and seem to always output the correct binary vector when $260 \leq m \leq 340$. However, for $350 \leq m$ then this trivial version of the attack does not work. Obviously one could extend this further by using more sophisticated methods for lattice reduction and the CVP, but this gives a rough guideline and suggests that taking $m = 640$ is a safe choice.

However, we now suggest a new method that tries to "learn" a number of entries of $\mathbf{u}$. The goal is to find enough entries to reduce the problem to the size that can be handled using the basic lattice attack.

The idea is to make use of entries in C_1 that are either significantly larger or smaller than the mean value of $m/4$. We will then exploit the fact that, for $1 \leq j \leq n$,

$$C_{1,j} = \sum_{\substack{i=1, \\ \mathbf{A}_{i,j}=1}}^{m} \mathbf{u}_i.$$

For example, if an entry $C_{1,j}$ is rather small and the hamming weight of the j-th column of $\mathbf{A}$ is not especially low, then it follows that $\mathbf{u}_i$ is zero for relatively many of the values i such that $\mathbf{A}_{i,j} = 1$. Suppose we have a set $J = \{j_1, \ldots, j_l\}$ corresponding to very small values of $C_{1,j}$ and we consider those values for i such that $\mathbf{A}_{i,j} = 1$ for all $j \in J$; it is natural to guess that, for each such i, we have $u_i = 0$ (at the very least, it is more "likely" that $u_i = 0$ than $u_i = 1$). Similarly, by considering columns of $\mathbf{A}$ corresponding to large values of $C_{1,j}$ one can guess that certain entries of $\mathbf{u}$ are likely to be 1. Once a guess for u_i has been made one can reduce the problem to a problem with $m - 1$ unknowns by setting $C_2' = C_2 - u_i \mathbf{A}_i$ and by removing the i-th row from the matrix $\mathbf{A}$. One then repeats the process of "learning" entries of $\mathbf{u}$ until we are in the zone for the lattice attack, at which point one finishes the attack using that method.

We have experimented with this idea and it does not seem to be effective. A more detailed description and some experimental results are given in Appendix B. We conjecture that the ideas mentioned in this section can be extended to handle values for m up to 380 or 390 (taking $n = 256$) in reasonable time, but it would be impressive to solve cases with $m > 400$ without exploiting weeks or months of computing resources.

7.3 Variable reduction attack

An LWE instance consists of a pair $(\mathbf{A}, \mathbf{b})$ such that

$$\mathbf{b} \equiv \mathbf{A}\mathbf{s} + \mathbf{e} \pmod{q}$$

where $\mathbf{A}$ is an $m \times n$ matrix, $\mathbf{s}$ is a secret length n vector and $\mathbf{e}$ is a length m vector over $\mathbb{Z}$ with entries chosen independently from a discrete Gaussian distribution. A simple observation is that if the first column of $\mathbf{A}$ is all zero then the system can be written as

$$\mathbf{b} \equiv \begin{pmatrix} \mathbf{0} & \mathbf{A}' \end{pmatrix} \begin{pmatrix} s_1 \\ \mathbf{s}' \end{pmatrix} + \mathbf{e}$$

where $\mathbf{0}$ is a zero vector. In other words, the value s_1 does not make any contribution to the system of equations and we can instead consider the LWE instance $\mathbf{b} \equiv \mathbf{A}\mathbf{s}' + \mathbf{e}$ in $(n-1)$ variables.

The basic idea of the attack (this is similar to the idea as the Blum, Kalai and Wasserman [2] attack) is therefore to try to arrange that a large number of zero entries appear in the first column of $\mathbf{A}$. In other words, we perform row and column operations (multiplication by matrices $\mathbf{R}$ and $\mathbf{C}$) to write

$$\mathbf{RAC} = \begin{pmatrix} \mathbf{0} & \mathbf{A}' \\ \mathbf{a}'' & \mathbf{A}'' \end{pmatrix}.$$

The LWE instance is also transformed as

$$\mathbf{Rb} \equiv (\mathbf{RAC})(\mathbf{C}^{-1}\mathbf{s}) + \mathbf{Re}.$$

If $\mathbf{A}'$ is an $m' \times (n-1)$ matrix then we reduce the original $m \times n$ instance of LWE with a length n secret to an $m' \times (n-1)$ instance with a length $(n-1)$ secret.

In the general case one may try to use linear algebra to achieve this, but the problem is that we still need the vector $\mathbf{Re}$ to be small. Hence, we are constrained to take $\mathbf{R}$ to be a sparse matrix with small entries.

In the case of binary-LWE a natural approach is to choose $\mathbf{R}$ and $\mathbf{C}$ to be permutations. We simply find the columns with the most naturally occuring zero entries and move them to the top left of the matrix $\mathbf{A}$. This is easily done.

The expected number of zeroes in any given column of $\mathbf{A}$ is $n/2$, with half the columns having at least this many zeroes. The process is to find the column with the largest number of zero entries and move that to the top left and discard all rows for which this column features a non-zero entry. One then repeats: Find the column in the subsequent matrix with the next-largest number of zeroes and move that to the second column and discard rows. To make k columns all zero one expects to reduce the number of rows from the original m to around $m/2^k$. Hence, we expect to be able to eliminant at most $\log_2(m)$ entries from the secret by this attack. In other words, we reduce an $m \times n$ binary-LWE instance to an instance with no fewer than $n - \log_2(m)$ secret values. This final system is then attacked using lattice techniques.

When m is very large compared with n then this attack might give a significant reduction in the cost of the lattice attack. However, it is easily prevented by increasing n by adding $\log_2(m)$.

7.4 General LWE can be phrased as LWE with a binary matrix

Let $(\mathbf{a}^T, b = \mathbf{a}^T\mathbf{s} + e \pmod q)$ be an LWE instance. For $1 \leq i \leq n$ write

$$a_i = \sum_{j=0}^{\lfloor \log_2(q) \rfloor} a_{i,j} 2^j$$

where $a_{i,j} \in \{0, 1\}$. Then

$$\sum_{i=1}^{n} a_i s_i \; = \; \sum_{i=1}^{n} \sum_{j=0}^{\lfloor \log_2(q) \rfloor} a_{i,j}(2^j s_i) \pmod q.$$

Hence, writing

$$\mathbf{s}' = (s_1, 2s_1, \ldots, 2^{\lfloor \log_2(q) \rfloor} s_1, s_2, \ldots, 2^{\lfloor \log_2(q) \rfloor} s_n)^T$$

we have $b = \mathbf{a}'\mathbf{s}' + e \pmod q$ where $\mathbf{a}' = (a_{1,0}, a_{1,1}, \ldots)^T$ is a binary vector.

In other words, general instances of LWE become binary instances of LWE, but with a "structured" secret.

7.5 Secrets must be large

It is known that, in standard LWE, without loss of generality the secrets may be chosen from the same distribution as the errors. The argument that this is valid is as follows:

Lemma 1. *One can reduce LWE to LWE where the secret is chosen from the error distribution.*

Proof. Given an arbitrary LWE instance $(\mathbf{A}, \mathbf{b})$, re-order the rows of A so that

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}_1 \\ \mathbf{A}_2 \end{pmatrix}$$

where $\mathbf{A}_1$ is an invertible $n \times n$ matrix over $\mathbb{Z}_q$. and $\mathbf{A}_2$ is $(m-n) \times n$ matrix. Similarly, write $\mathbf{e} = \binom{\mathbf{e}_1}{\mathbf{e}_2}$ and $\mathbf{b} = \binom{\mathbf{b}_1}{\mathbf{b}_2}$ where $\mathbf{b}_i = \mathbf{A}_i\mathbf{s} + \mathbf{e}_i$ for $i \in \{1, 2\}$. Set $\mathbf{A}' = -\mathbf{A}_2\mathbf{A}_1^{-1} \pmod q$ and $\mathbf{b}' = \mathbf{A}'\mathbf{b}_1 + \mathbf{b}_2 \pmod q$. Note that $\mathbf{A}'$ is an $(m-n) \times n$ matrix and $\mathbf{b}'$ is a vector of length $m - n$. Then one can verify that

$$\mathbf{b}' = \mathbf{A}'\mathbf{e}_1 + \mathbf{e}_2 \pmod q.$$

In other words $(\mathbf{A}', \mathbf{b}')$ is an LWE instance where the "secret" is drawn from the error distribution. If such instances were easy to solve, then one could compute $\mathbf{e}_1$ and then solve the original LWE instance using

$$\mathbf{s} = \mathbf{A}_1^{-1}(\mathbf{b}_1 - \mathbf{e}_1).$$

Note that this proof is not applicable to the case when the matrix $\mathbf{A}$ is binary. This is because if $\mathbf{A}_1$ is binary then $\mathbf{A}_1^{-1} \pmod q$ and $\mathbf{A}' = -\mathbf{A}_2\mathbf{A}_1^{-1} \pmod q$ are unlikely to have entries in $\{0, 1\}$. Hence, an instance of LWE with a binary matrix is reduced to an instance of LWE with a general matrix over $\mathbb{Z}_q$ but with secrets from the error distribution.

Indeed, the failure of this reduction is symptomatic of genuine security issue. Suppose that $\mathbf{A}$ were a binary matrix and $\mathbf{s}$ and $\mathbf{e}$ were chosen from an error distribution with small standard deviation σ. Then, computing over $\mathbb{Z}$,

$$\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}$$

is a "short" vector. Indeed, each entry of $\mathbf{b}$ is a sum of, on average, $m/2 + 1$ samples from the normal distribution of mean 0 and standard deviation σ. It follows from the central limit theorem that the entries of $\mathbf{b}$ are also of mean 0 and standard deviation $\sqrt{m}\sigma$. For the parameters mentioned above (i.e., $(n, m, q) = (256, 640, 4093)$ and $\sigma \approx 3$) we have $\sqrt{m}\sigma \approx 75 \ll q$. In other words, in this case $\mathbf{A}\mathbf{s} + \mathbf{e} \pmod q$ actually gives us $\mathbf{A}\mathbf{s} + \mathbf{e}$ over $\mathbb{Z}$! The problem of determining $\mathbf{s}$ is then the classical problem of linear regression, which can be solved using least squares approximation

(note that the system of computing $\mathbf{s}$ is over-determined). In particular, given $\mathbf{A}$ and $\mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e}$ one computes (using numerical analysis)

$$\mathbf{s}' = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b}.$$

Hence, for sure, we need to take the entries of $\mathbf{s}$ to be large in this setting. Many applications of LWE benefit from taking the entries of $\mathbf{s}$ from the error distribution. Hence, it is clear that binary matrices are completely unsuitable for some applications of LWE.

7.6 The case of Ring-LWE

One could ask whether it is secure to use binary matrices with Ring-LWE. Ron Steinfeld has pointed out that this is not the case.

A Ring-LWE instance is a list of k pairs $(\mathbf{a}_i, \mathbf{y}_i = \mathbf{a}_i * \mathbf{s}_i + \mathbf{e}_i)$ where $\mathbf{a}_i, \mathbf{s}_i, \mathbf{e}_i \in R_q$, where R_q is typically the ring $\mathbb{Z}_q[x]/(x^n + 1)$ with n being a power of 2, and where $*$ denotes multiplication in R_q. Here the entries of $\mathbf{e}_i$ are small and we now impose the non-standard condition that the entries of $\mathbf{a}_i$ are small (e.g., binary).

The observation is that, for $2 \leq j \leq k$

$$\mathbf{w}_j = \mathbf{a}_j * \mathbf{y}_1 - \mathbf{a}_1 * \mathbf{y}_j \equiv \mathbf{a}_j * \mathbf{e}_1 - \mathbf{a}_1 * \mathbf{e}_j \pmod q$$

is an element having small entries, and so is known over $\mathbb{Z}$ (not modulo q). Treating $\mathbf{e}_1$ as the unknown and the terms $-\mathbf{a}_1 * \mathbf{e}_j$ as the "errors", this is a linear regression problem solvable using least squares. Our experimental results confirm this.

8 Matrices with bounded entries

We now consider the attacks from Section 7 in the case where $\mathbf{A}$ is not a binary matrix, but has entries chosen uniformly and independently in the range $[0, 2^k - 1]$. It is natural to choose $2^k \approx 8q/m$ so that the average size of integers in the vector $\mathbf{u}^T\mathbf{A}$ is $(m/2)(4q/m) = 2q$, and so some reduction modulo q is likely to occur. For our example parameters this leads to generating entries of $\mathbf{A}$ using 6 bits rather than 12; potentially still making the cost of constructing $\mathbf{A}$ about twice as fast as the general method.

We briefly go over the results of the previous section. First, that LWE is well-defined follows from Theorem 1. Also note that this reduction is tighter when $\mathbf{A}$ has larger entries, and so the requirement that m be very large is less severe in this case. The attacks in Section 7.2 will not occur, since C_1 is no longer equal to $\mathbf{u}^T\mathbf{A}$, but is really $\mathbf{u}^T\mathbf{A} \pmod q$. The facts that LWE can be phrased as having a matrix of this form (but with structured secrets), and the fact that the secret cannot be chosen from the error distribution, go over verbatim. Similarly, the remark about Ring-LWE is still valid as long as the entries of $\mathbf{A}$ are significantly smaller than q.

Overall, it seems that generating the matrix $\mathbf{A}$ by taking each entry to be a word of 6 bits (rather than 12) will be completely secure.

9 Conclusion

We have explained that short public keys for LWE cryptosystems can be obtained by publishing a short seed, rather than the matrix $\mathbf{A}$. We have explained that the pseudorandom generator used to build $\mathbf{A}$ need not have strong cryptographic properties. We have argued that it should be secure to use a very fast pseudorandom word generator and to construct $\mathbf{A}$ as a binary matrix, in which case the computational overhead from having to reconstruct the matrix $\mathbf{A}$ should be small. Hence, contrary to what seems to be believed, it should be possible to implement LWE cryptosystems on constrained devices with relatively little storage without seriously damaging the speed of the systems.

Regarding our technical contributions, we have shown that LWE is well-defined for such matrices and have sketched a new algorithm for finding $\mathbf{u}$ given $(\mathbf{A}, \mathbf{u}^T\mathbf{A})$ over $\mathbb{Z}$.

Finally, recalling that a scientific hypothesis should make precise statements that are disprovable, we consider that any cryptographic proposal should make precise statements that are cryptanalysable. In this spirit we give two precise challenges for cryptanalysis. Solving the first should be interpreted as causing embarrassment to the author(s), while solving the second should be considered as a "total break" of the paper.

– Let $(n, m) = (256, 400)$. Let $\mathbf{A}$ be a random $m \times n$ binary matrix and $\mathbf{u}$ a random length m binary vector. Set $C_1 = \mathbf{u}^T\mathbf{A}$. Give an algorithm to compute $\mathbf{u}$, given $(\mathbf{A}, C_1)$, that runs in "reasonable" time (e.g., less than a day on a relatively ordinary PC or laptop).
– Let $(n, m, q, \sigma) = (256, 640, 4093, 3.33)$ and suppose the Mersenne twister is used to generate binary matrices $\mathbf{A}$ for the LWE cryptosystem. Give an OWE-CPA attack on the cryptosystem in this setting that could be mounted using current computing facilities and that would take less than one year.

Acknowledgements

I thank Mark Holmes, Ron Steinfeld, Damien Stehlé and Frederik Vercauteren for helpful suggestions and I thank Timothy Vogel for doing some computer experiments.

References

1. M. Ajtai, Representing hard lattices with $O(n \log n)$ bits, in H. N. Gabow and R. Fagin (eds.), STOC 2005, ACM (2005)
2. A. Blum, A. Kalai and H. Wasserman, Noise-tolerant learning, the parity problem, and the statistical query model, Journal of ACM, **50**, no. 4 (2003) 506–519.
3. J.-S. Coron, A. Mandal, D. Naccache and M. Tibouchi, Fully Homomorphic Encryption over the Integers with Shorter Public Keys, eprint 2011/441 (2011).
4. J. Hoffstein, J. Pipher and J. H. Silverman, NTRU: A Ring-Based Public Key Cryptosystem, in J. Buhler (ed.), ANTS-III, Springer LNCS 1423 (1998) 267–288.
5. D. Knuth, The Art of Computer Programming, Volume 2: Seminumerical Algorithms (3rd edition), Addison-Wesley, 1997.
6. R. Lindner and C. Peikert, Better key sizes (and attacks) for LWE-based encryption, in A. Kiayias (ed.), CT-RSA, Springer LNCS 6558 (2011) 319–339.
7. V. Lyubashevsky, C. Peikert and O. Regev, On Ideal Lattices and Learning with Errors over Rings, in H. Gilbert (ed.) EUROCRYPT 2010, Springer LNCS 6110 (2010) 1–23.
8. M. Matsumoto, and T. Nishimura, Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator, ACM Trans. Model. Comput. Simul., **8**, No. 1, pp. 3–30 (1998)
9. D. Micciancio and O. Regev, Lattice-based cryptography, in D. J. Bernstein, J. Buchmann, and E. Dahmen (eds.), Post Quantum Cryptography, Springer (2009) 147–191.
10. C. Peikert, An Efficient and Parallel Gaussian Sampler for Lattices, in T. Rabin (ed.), CRYPTO 2010, Springer LNCS 6223 (2010) 80–97.
11. O. Regev, On lattices, learning with errors, random linear codes, and cryptography, in H. N. Gabow and R. Fagin (eds.), STOC 2005, ACM (2005) 84–93.
12. O. Regev, On lattices, learning with errors, random linear codes, and cryptography, Journal of the ACM 56(6), article 34, 2009.
13. V. Shoup, A computational introduction to number theory and algebra, Cambridge, 2005.
14. D. Stehlé and R. Steinfeld, Making NTRU as secure as worst-case problems over ideal lattices, in K. G. Paterson (ed.), EUROCRYPT 2011, Springer LNCS 6632 (2011) 27–47.

A Practical attacks on LWE

It is not the goal of this paper to present a survey of lattice attacks on LWE, but to make the paper self-contained we list a few basic attacks.

A.1 Reducing to CVP

One basic lattice attack is to reduce LWE to a closest vector problem in a lattice. Given $(\mathbf{A}, \mathbf{b})$ one constructs a basis for the lattice

$$L = \{\mathbf{v} \in \mathbb{Z}^m : \mathbf{v} \equiv \mathbf{Au} \pmod{q} \text{ for some } \mathbf{u} \in \mathbb{Z}^n\}$$

of dimension m. One then tries to find an element $\mathbf{v}$ of L which is close to $\mathbf{b}$. Hopefully, $\mathbf{v} = \mathbf{b} - \mathbf{e} \equiv \mathbf{As} \pmod{q}$. Once $\mathbf{As}$ has been computed one can compute $\mathbf{s}$ efficiently with overwhelming probability.

In other words, the LWE problem is reduced to the closest vector problem in the lattice. Such problems are solved using lattice reduction and the Babai nearest plane method. To be effective one wants a basis for the lattice consisting of vectors that are "close to orthogonal" so that the parallelepiped formed by them is "fat".

Note that one does not typically use the same value for m as is provided by the LWE instance (e.g., the size of the public key). Instead one selects an optimal number of rows for the attack. The value used by Lindner and Peikert [6] (also see equation (10) of [9]) is $m \approx \sqrt{n \log(q)/\log(\delta)}$ where δ is the "root Hermite factor" in the lattice reduction algorithm.

Lindner and Peikert [6] have presented experimental results using this attack.

A.2 Solving the decisional-LWE problem

The decisional-LWE problem is to distinguish $(\mathbf{A}, \mathbf{b} = \mathbf{As} + \mathbf{e} \pmod q)$ from $(\mathbf{A}, \mathbf{b}')$ where $\mathbf{b}'$ is uniformly chosen with entries in $\mathbb{Z}_q$. This problem arises in the security analysis of the LWE cryptosystem: A successful adversary against the cryptosystem leads to an algorithm to solve the decisional-LWE problem, and so if decisional-LWE is hard then the cryptosystem is secure. A lattice attack on this problem is to find a short vector (or several short vectors) $\mathbf{w}^T$ such that $\mathbf{w}^T\mathbf{A} \equiv 0 \pmod q$. Then, given $(\mathbf{A}, \mathbf{b})$ one can compute $\mathbf{w}^T\mathbf{b} \pmod q$ and see if the result is "small". The point is that if $\mathbf{b} = \mathbf{As} + \mathbf{e} \pmod q$ then $\mathbf{w}^T\mathbf{b} = \mathbf{w}^T\mathbf{As} + \mathbf{w}^T\mathbf{e} \equiv \mathbf{w}^T\mathbf{e} \pmod q)$, which should be "small".

Hence, the difficulty of decisional-LWE depends on the problem of finding short vectors in the kernel lattice modulo q corresponding to $\mathbf{A}$.

Structured matrices will not necessarily resist this attack. For example, if there is a repeated row in $\mathbf{A}$ then there is a vector of the form $(0, \ldots, 0, 1, 0, \ldots, 0, -1, 0 \ldots, 0)$ in the left kernel and this is perfect for the above attack.

We also remark that the ideas in this attack have the potential to lead to improved algorithms for determining the secret vector $\mathbf{s}$. To give the basic idea, note that short vectors $\mathbf{w}$ such that $\mathbf{w}^T\mathbf{A} \equiv 0 \pmod q$ will, assuming there is an entry in $\mathbf{w}$ that is ± 1, allow one to express some row of $\mathbf{A}$ as a sum of several other rows. So let us assume for the moment that among the rows of $\mathbf{A}$ are a number of sets of many identical rows. Looking at the corresponding entries of $\mathbf{b}$ gives many samples of the form $\mathbf{a}^T\mathbf{s} + e \pmod q$, from which one can potentially learn $\mathbf{a}^T\mathbf{s}$ with somewhat high confidence. Collecting a number of relations of this form at least confines the secret vector $\mathbf{s}$ to lie in some subspace of $\mathbb{Z}_q^n$, which might lead to improved attacks on the system.

It is important to note that short vectors in the left kernel do not seem to directly lead to an attack on the cryptosystem. The ciphertext is $(C_1 = \mathbf{u}^T\mathbf{A} \pmod q, C_2 = \mathbf{u}^T\mathbf{b} + m\lfloor q/2 \rfloor \pmod q)$ and one cannot directly multiply C_2 by $\mathbf{w}^T$. This does not contradict the logic of the security proof: an attacker against the cryptosystem leads to an algorithm for decisional-LWE; short vectors in the left kernel lead to an algorithm for decisional-LWE. Hence, there is the possibility that the cryptosystem could be secure even if there are short vectors in the left kernel of $\mathbf{A}$. A clarification is needed to this final claim: there is a reduction from decisional-LWE to LWE, but it modifies the matrices $\mathbf{A}$; hence it is possible for decisional-LWE to be easy for a class of matrices $\mathbf{A}$ but LWE still be hard for that class of matrices.

A.3 Determining u

The ciphertext features $C_1 = \mathbf{u}^T\mathbf{A} \pmod q$ and if one can determine $\mathbf{u}$ then one can break the scheme (using $C_2 - \mathbf{u}^T\mathbf{b} \pmod q$). Of course, there is a large space of vectors $\mathbf{w}$ such that $C_1 = \mathbf{w}^T\mathbf{A} \pmod q$, but one expects a unique solution $\mathbf{u}$ having entries in $\{0, 1\}$.

This problem also boils down to CVP: Find any solution $\mathbf{w}$ to $C_1 = \mathbf{w}^T\mathbf{A} \pmod q$ and then, in the lattice $L = \{\mathbf{v} \in \mathbb{Z}^m : \mathbf{v}^T\mathbf{A} \equiv 0 \pmod q\}$ find a close vector $\mathbf{v}$ to $\mathbf{w}$. The hope is that $\mathbf{w} - \mathbf{v}$ is the binary vector $\mathbf{u}$ we seek.

We investigate in Section B some statistical approaches to learning some entries of $\mathbf{u}$.

A.4 Blum-Kalai-Wasserman attack

The idea of Blum, Kalai and Wasserman [2] is to find subsets of rows of $\mathbf{A}$ that add to simple vectors (such as unit vectors). This attack requires an exponentially large number of rows, so is irrelevant to the matrix formulation of LWE considered in this paper.

B Statistical attack on vector-subset-sum

As already noted, an LWE ciphertext features $C_1 = \mathbf{u}^T\mathbf{A} \pmod q$ and if one can determine $\mathbf{u}$ then one can break the scheme. Hence one can try to solve this problem by deducing entries in $\mathbf{u}$ until the number of unknowns left is able to be dealt with by the lattice attack. For $(m, n) = (640, 256)$, this means deducing 300 entries. The idea is that, for $1 \leq j \leq n$,

$$C_{1,j} = \sum_{\substack{i=1,\\ \mathbf{A}_{i,j}=1}}^{m} \mathbf{u}_i.$$

This time, we use it to form probabilities from each $C_{1,j}$, in that we say a member of the set $A_j = \{i : \mathbf{A}_{i,j} = 1\}$ has $C_{1,j}/\#A_j$ likelihood of corresponding to $\mathbf{u}_i = 1$. This idea is that we will be guessing the i with the highest average probability in this way has $\mathbf{u}_i = 1$.

To do this, we also make use of being able to quite accurately guess the weight of $\mathbf{u}$ by finding the mean of the entries of C_1 divided by the mean weight of the columns of $\mathbf{A}$ and multiplying by m. Then we say the members of the set $B_j = \{i : \mathbf{A}_{i,j} = 0\}$ have $(\mathbf{u}_{weight} - C_{1,j})/\#B_j$ likelihood of having $\mathbf{u}_i = 1$. Then when we sum up the probabilities over all n entries of C_1, each $\mathbf{u}_i$ has been given exactly n probabilities, and to make our guess we simply find the i that is the greatest difference from the mean total, and if it is higher than the mean we would guess $\mathbf{u}_i = 1$ or if it is lower than the mean we would guess $\mathbf{u}_i = 0$. We can continue guessing this way until enough have been made so that the lattice attacks can be used.

Experiments using $(m, n) = (640, 256)$ gave a correct guess about 99% of the time (988, 988 and 989 out of 1000 trials). They also showed that the probability to get subsequent guesses after the first decayed at a rate of about 1% for each guess for the first 20 guesses. It would be reasonable to assume this decay occurs asymptotically to 50% as there is no reason that this method should ever be worse than simply guessing randomly, however this could not be confirmed because of the obvious weakness of this method: it is simply not accurate enough to guess correctly for 50 straight guesses, let alone 300.

As well as the inevitable random variation leading to incorrect results, some of the inaccuracy would be due to our value for $\mathbf{u}_{weight}$ not being perfectly accurate. Our method of finding $\mathbf{u}_{weight}$ is equivalent to finding the proportion of the entries of $\mathbf{A}$ that correspond to $\mathbf{u}_i = 1$. Taking an average of n instances of m $\{0, 1\}$ variables, each with chance $\mathbf{u}_{weight}/m$ chance to be 1 yields a variance of

$$\frac{1}{n}\left(\mathbf{u}_{weight} - \frac{(\mathbf{u}_{weight})^2}{m}\right).$$

This means when using $(m, n) = (640, 256)$ we would expect to see our estimate of $\mathbf{u}_{weight}$ almost always within ± 3 of the true value as this is 3.8 standard deviations for the true $\mathbf{u}_{weight}$ being 320. This concurs with experimental results.

The decay of accuracy as more guesses are taken is also inevitable. When $m = 640$ and on average $\#A_j, \#B_j = 320$ and $\mathbf{u}_{weight}$ is around 320, the mean probability for i such that $\mathbf{u}_i = 1$ for each $C_{1,j}$ will be

$$\frac{1 + (\#A_j - 1)(\frac{\mathbf{u}_{weight}-1}{m-1})}{\#A_j} \approx 0.50078.$$

For i such that $\mathbf{u}_i = 0$ the mean probability for each $C_{1,j}$ will be

$$\frac{(\#A_j - 1)(\frac{\mathbf{u}_{weight}}{m-1})}{\#A_j} \approx 0.49922.$$

The approx 320 $\mathbf{u}_i = 1$ will form a distribution with mean as the upper number $\times n$ and the 320 $\mathbf{u}_i = 0$ form a distribution around the lower number $\times n$. Incorrect guesses occur when in the combined distribution the point furthest from $256 \times 1/2 = 128$ is greater than 128 and yet it was from the lower group, or the furthest point was less than 128 and yet it came from the higher group. As we make more and more guesses, this outcome becomes more likely,

as each correct guess removes either the highest point of the $\mathbf{u}_i = 1$ distribution or the lowest point of the $\mathbf{u}_i = 0$ distribution, leaving it more likely that the highest/lowest remaining point is from the wrong one.

The limited accuracy and the unavoidable decay of accuracy lead us to say that this method is not useful beyond reducing the size of problems only slightly. It may be able to solve $(m, n) = (360, 256)$ reliably but certainly not $(640, 256)$. It is also worth noting that there may be ways to improve this method slightly, given that if a $C_{1,j}$ gave a probability of 1 or 0 for some entries of $\mathbf{u}$ to have $\mathbf{u}_i = 1$ but overall those entries had an unexceptional average because of their probabilities in all the other entries of C_1, we would not end up making a guess for those entries of $\mathbf{u}$ even though we certainly have the information to guess them correctly. However it would unreasonable to expect that the improvements from this would make this method viable for large sized problems.

C LWE with constrained entries is well-defined

We now show that LWE is well-defined even when the entries of $\mathbf{A}$ are not chosen uniformly from $\{0, 1, \ldots, q-1\}$. As usual, in order to show that LWE is well-defined we consider the problem in the setting of a fixed secret vector $\mathbf{s}$ and where there is an oracle that outputs fresh instances $(\mathbf{a}^T, b = \mathbf{a}^T\mathbf{s} + e \pmod q)$, where $\mathbf{a}$ is a length n vector with entries independently and identically distributed (i.i.d.) with respect to some distribution, and where e is chosen according to the discrete Gaussian distribution on $\{-(q-1)/2, \ldots, -1, 0, 1, \ldots, (q-1)/2\}$ with mean 0 and variance σ^2.

Proving that LWE is well-defined requires showing that there is a single value for the secret vector $\mathbf{s}$ that is overwhelmingly more likely to be the value used by the oracle than any other vector $\mathbf{s}'$. We establish this by considering an algorithm (the distinguisher) that decides whether or not a guess $\mathbf{s}'$ for $\mathbf{s}$ is correct with overwhelming probability after making queries to the oracle.

Theorem 1. *LWE is well-defined when the entries of $\mathbf{a}$ are independently and uniformly chosen from an interval $\{N_1, N_1 + 1, \ldots, N_2 - 1, N_2\} \subseteq \mathbb{Z}_q$ (where $-(q-1)/2 \leq N_1 < N_2 \leq (q-1)/2$).*

Proof. Let $\mathbf{s}'$ be a candidate value for $\mathbf{s}$. Write $\mathbf{s}' = \mathbf{s} - \mathbf{u}$ where $\mathbf{u} = (u_1, \ldots, u_n)$. For any value $(\mathbf{a}^T, b = \mathbf{a}^T\mathbf{s} + e \pmod q)$ output by the LWE oracle set $e' = b - \mathbf{a}^T\mathbf{s}' \pmod q$ (as always, $-(q-1)/2 \leq e' \leq (q-1)/2$). Then

$$e' \equiv b - \mathbf{a}^T\mathbf{s}' \equiv e + \mathbf{a}^T\mathbf{u} \equiv e + \sum_{j=1}^{n} a_j u_j \pmod q.$$

Let p be the probability that an integer e chosen from the error distribution is such that $|e| \leq \sigma$. Typically $0.68 \leq p \leq 0.8$.

The distinguisher is parameterised by two values m and ϵ. The distinguisher simply requests m samples from the LWE oracle and counts the number m' of them such that $|e'| \leq \sigma$. If $|p - (m'/m)| < \epsilon$ then the distinguisher returns the value "yes" (to denote that $\mathbf{s}'$ is a correct guess for $\mathbf{s}$) and otherwise returns "no".

The key to the proof is the Chernoff bound (e.g., see Theorem 6.13 of Shoup [13]). It states that, when $\mathbf{s} = \mathbf{s}'$ and so $\mathbf{u} = 0$, then

$$\Pr(|p - (m'/m)| \geq \epsilon) \leq 2\exp(-m\epsilon^2/2).$$

Hence, if m is sufficiently large then the distinguisher will correctly recognise when $\mathbf{s}' = \mathbf{s}$ with overwhelming probability.

To complete the proof we need to show that when $\mathbf{s}' \neq \mathbf{s}$ then the probability that the distinguisher answers "yes" is negligible (indeed, we need to argue that this holds for all $\mathbf{s}' \neq \mathbf{s}$). The main idea is that if $\mathbf{s}' \neq \mathbf{s}$ then $\mathbf{u} \neq (0, \ldots, 0)$ and so the distribution of the values e' is obtained by averaging the error distribution with various shifts of it (corresponding to the possible values for $\mathbf{a}_j$ such that $\mathbf{u}_j \neq 0$). For any such vector $\mathbf{u}$ and for any choice $\{N_1, N_1 + 1, \ldots, N_2 - 1, N_2\}$, where $-(q-1)/2 \leq N_1 < N_2 \leq (q-1)/2$, for the entries in $\mathbf{s}$, we denote by p' the probability that $|e'| \leq \sigma$. It is easy to see that we cannot have $\sum_{j=1}^{n} a_j u_j \equiv 0 \pmod q$ for the fixed value $\mathbf{u}$ and *all* values for $\mathbf{a}$ (for this it is essential that the values u_j are considered as $-(q-1)/2 \leq u_j \leq (q-1)/2$ and that there are at least two *consecutive* choices for a_j for each j). Hence p' is obtained by averaging various shifts of the original

error distribution. Any shift of the error distribution has a strictly lower value for $\Pr(|e| \leq \sigma)$. It follows that $p' < p$. Setting $\epsilon = (p - p')/2$ and choosing m as above shows that the algorithm succeeds with overwhelming probability.

Since we need the algorithm to succeed for all of the $(q^n - 1)$ values $\mathbf{s}' \neq \mathbf{s}$ it suffices to take m sufficiently large that $(q^n - 1)$ times the probability of a "false positive" is negligible. In other words, increase m by a multiplicative factor of $n \log(q)$.

For example, taking $q = 4093$ and $\sigma = 3$ we compute the discrete Gaussian distribution with mean 0 and variance σ^2 on $\{-(q-1)/2, \ldots, -1, 0, 1, \ldots, (q-1)/2\}$. The probability that elements chosen from the error distribution satisfy $|e| \leq \sigma$ is $p \approx 0.758848$. Now, suppose the entries of $\mathbf{s}$ are chosen independently and uniformly from $\{0, 1\}$ and that $\mathbf{u} = (1, 0, \ldots, 0)$. Then $e' = e + a_1$, where $a_1 \in \{0, 1\}$. It follows that $\Pr(e' = k) = \frac{1}{2}\Pr(e = k) + \frac{1}{2}\Pr(e = (k-1))$. One can easily check that the probability that $|e'| \leq \sigma$ in this case is $p' \approx 0.745855$. Note that $p - p' \approx 0.013$, so taking $\epsilon = 0.006$ seems a sensible choice. Note that to get $2\exp(-m\epsilon^2/2) < 2^{-80}$ (so that the distinguisher is correct with overwhelming probability) requires taking $m \geq 81\log(2)/(\epsilon^2/2) > 3 \cdot 10^6$. It is necessary to use an even larger m so that this result holds for all the $(q^n - 1)$ possible values for $\mathbf{s}' \neq \mathbf{s}$. Taking $n = 256$ and $q = 4093$ requires multiplying m by $\log(q^n - 1) \approx 2129$.

Similarly, if $\mathbf{u} = (1, -1, 0, \ldots, 0)$ with binary $\mathbf{a}$ then $p' \approx 0.721027$ and if $\mathbf{u} = (1, 0, \ldots, 0)$ and entries in $\mathbf{a}$ are chosen uniformly in $\{-1, 0, 1\}$ then $p' \approx 0.745855$ (same value as before).

As with Regev's result that standard LWE is well-defined, care is needed to interpret the result for a fixed key $\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + \mathbf{e} \pmod{q}$. The issue is whether, among the $q^n - 1$ choices for $\mathbf{s}' \neq \mathbf{s}$, there is one such that $\mathbf{b} - \mathbf{A}\mathbf{s}'$ is a short vector (such a vector leads to an attack on the encryption scheme). If m is small enough then, even with standard LWE, such a case might arise. In our case such a scenario is certain. Indeed, if $\mathbf{A}$ were a 640×256 binary matrix then the probability that $\mathbf{b}$ would also have arisen from a vector such as $\mathbf{s}' = \mathbf{s} + (1, 0, \ldots, 0)$ (or similar) is high. From the point of view of security this does not seem to be a serious problem. The computational assumption is that it is hard to find a vector $\mathbf{s}'$ such that $\mathbf{b} - \mathbf{A}\mathbf{s}' \pmod{q}$ is short. This is exactly the closest vector problem in a lattice, and is a reasonable computational assumption on which to base the security of the scheme. Indeed, it is the same problem that Lindner and Peikert study and use to set the parameters for their scheme.