# Exploiting Vulnerabilities in Homomorphic Encryption with Weak Randomness

by

Leonard (Pi) Fisher

A Major Qualifying Project

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

in

Mathematical Sciences

by

April 2016

APPROVED:

Dr. William Martin, Major Advisor

**Abstract**

Suppose I give you a massively overdetermined yet consistent system of linear equations $Ax = b$ over a finite ring, but I change a few of the values of $b$ before showing it to you. Can you still solve for $x$? This is called the Learning With Errors problem, and it was introduced by Regev in 2009, who showed that it is hard on average. Today it is used in many homomorphic encryption schemes. Such schemes allow anyone to run computations on encrypted data without being able to learn what the data are. In this paper, I look at ways to exploit poor randomness in Learning With Errors. For extremely sparse noise models, I give polynomial-time solutions to the Learning With Errors problem.

## 1 Introduction

Cryptography is everywhere. From the military's secrets to hospital records to online purchases. Without cryptography, the only way to keep a secret would be to meet up in person when you wanted to let someone in on it. Imagine if everyone had to visit the bank each time they wanted to buy something from Amazon, and it's clear that we need encryption.

Just because something is encrypted, however, doesn't mean it's secure. Early encryption relied on the fact that most people couldn't read, and it was pretty successful at the time. The Roman Empire would encrypt messages by replacing each letter with one a few letters later in the alphabet, and the recipient would need to know how far each letter was shifted. For example, "encrypt" could be written as "gpetarv", with each letter replaced with the letter two spots after it (and saying that 'a' comes after 'z').

Nowadays, there are free, online tools that will take a message encrypted that way and quickly break the encryption [dCo16]. Modern encryption methods rely on problems that are thought to be too hard for computers to solve in a useful amount of time (e.g. hundreds of CPU years). Preferably, security would be based on problems classified as "NP-hard", which are the hardest problems for computers to solve.

Factoring huge numbers is a well-studied hard problem, but quantum computers would be able to do that. So far, quantum computers probably have not been made. If they have been made, only the people who have them know, and they don't want to tell anyone yet. New encryption methods not only need to be hard for normal computers to beat, but also need to be able to safe against attacks using quantum computers. In August of 2015, the NSA announced that it would be transitioning all its encrypted systems to use methods that are quantum resistant [Sch15].

While all of this is going on, people want to be able to do more and more with cryptography. One popular topic today is homomorphic encryption, where everyone is able to perform computations on encrypted data, but only someone with the password is able to see the results of the computations [GSW13]. A lot of people are getting worried about how much personal information big companies such as Google and Facebook have, but they use all that information in order to deliver relevant advertisements to us [Rep12]. Netflix keeps track of what movies all its users have seen, and how much they liked those movies. It would be impossible to suggest a movie to someone without any information on what they like. But wait, maybe a service can! There are homomorphic encryption methods that would allow Netflix to do everything it currently does except see the information themselves. When you watch a movie and rate it, that information would be encrypted before it went to Netflix. Then, their computers would use this encrypted information to pick a movie you might like, but even this would be encrypted, and they wouldn't be able to see what movie they were recommending. Then the recommendation gets to you and is decrypted so you can watch the movie.

Technology for homomorphic encryption hasn't yet advanced to that level, but it is being used in some smaller applications where not as many computations need to be performed on each number. Some banks have started using it for verification [Gau16]. Something that might not be far away is using homomorphic encryption when you enter your credit card number online. Currently, most companies store your credit card number when you make a purchase, and they give the number to the credit card company when they charge you. With homomorphic encryption, they wouldn't need to know your number in order to do this.

One thing that is important for many encryption schemes is the ability to generate random numbers [3SC05]. While random number generators are needed for a wide variety of applications, often they are not made very well. It is very difficult for a computer to come up with random numbers. When it does come up with random numbers, the numbers it comes up with are usually similar to a weighted die, where each number can come up, but some are far more likely. In September of 2013, a study was done in which researchers looked at RSA keys to see how secure they are [Ber+13]. The security of RSA is based on it being difficult to factor large numbers. Each RSA key is the product of two large primes. Unfortunately, if someone knows that two RSA keys share a prime, it is easy to factor both. This study found that, out of around two million keys, 103 shared a prime with another key, and they were able to factor those 103 keys. That may not seem like a lot, but it would be astounding if even two of the 2,000,000 keys shared a factor if the primes were truly random. Even though breaking RSA is supposed to be a hard problem, it's suddenly very easy to solve when it isn't implemented very well.

Several important homomorphic encryption methods are based on a problem called Learning With Errors, or LWE. In LWE, you're given a bunch of equations, but a small percentage of them are wrong. That is, there exists a unique "secret vector" which satisfies the vast majority of the given equations, but no vector can satisfy every equation. If you have the secret vector, it's easy to verify that it satisfies most of the equations. However, if you don't have the key, it's very difficult to figure out which equations are wrong. It's been proven that finding the secret vector for LWE is an NP-hard problem, but only when it's implemented well. For several examples of how a random number generator can be bad, this paper shows ways to solve the LWE problem with a reasonable amount of computation.

## 2 NP-Completeness

When given a problem, it is helpful to know how hard it will be to solve the problem. When computers solve problems, we want to know about how long it will take the computer to solve the problem. Specifically, if a computer can solve all problems of a certain type, we want to know how long it will take as a function of how big the input is. This is the subject known as computational complexity [Pap94]. For example, the larger two numbers are, the longer it takes to compute their product.

Because computers are being used to solve bigger and bigger problems every year, we want to look at how these functions behave as the size of the input approaches infinity. To help with this, we like to compare things to known functions, like $x^2$ or $e^x$. One way to compare functions is with Landau notation. We write $f(n) = O(g(n))$, or "$f(n)$ is Big-Oh of $g(n)$" to mean that, as $x$ gets very big, $f(n)$ is no bigger than some constant times $g(n)$. More formally, $f(n) = O(g(n))$ if and only if there exist positive constants $c$ and $N$ such that, for all $n > N$, $f(n) \leq cg(n)$.

Some problems, such as computing products, are not very difficult for a computer to solve. In 1965, Jack Edmonds said that an algorithm is quick for a computer if it can be done in polynomial time [Edm65]. That is, if $f(n)$ is how long it takes the computer in the worst case to perform a given algorithm with an input size of $n$, the algorithm is "quick" for the computer to perform if there exists some $k$ such that $f(n) = O(n^k)$. These algorithms are said to be performed in polynomial time. Computing the sum or product of two numbers can be done in polynomial time.

To make things more formal, we talk just about decision problems. For example, given an ordered triplet $(a, b, T)$, decide if the product $ab$ is greater than the threshold $T$. The set of decision problems for which there exists a polynomial time algorithm to solve is called P [Pap94]. Clearly, the above decision problem is in P.

Some decisions can be made quickly if a hint is provided. Consider the factoring problem: given an ordered pair $(n, T)$, decide if $n$ has a factor $k$ such that $1 < k < T$. If I give you the pair $(12319, 100)$, and if I give you hint of 97, you can quickly check that $1 < 97 < 100$ and that $\frac{12319}{97}$ is an integer, so you can quickly give a YES answer. NP is the set of decision problems for which an algorithm exists such that, when the answer is YES, said algorithm can find the answer in polynomial time, provided that it is given a small hint. When we say the hint is small, we mean that the size of the hint we need to give the algorithm is polynomial in terms of the size of the problem.

Notice that every problem in P is also in NP. I can multiply two numbers with a quick algorithm, and I could change that algorithm to let you give me a hint and then ignore the hint. Then the algorithm can get a small hint and solve the problem quickly.

A similar set of problems is coNP. coNP is the set of decision problems for which an algorithm exists such that, when the answer is NO, said algorithm can find the answer in polynomial time, provided that it is given a small hint. Factoring is also in coNP. If I give you the ordered pair $(12319, 50)$, and if I give you the hint $97, 127$, you can quickly check that 97 and 127 are each prime, $97 \cdot 127 = 12319$, and 50 is less than each of 97 and 127.

For encryption, problems that are in both NP and coNP are very nice. When someone gives you their guess as to the answer, it's easy to decide if they are correct or if they are incorrect. Unfortunately, most problems that are known to be in both NP and coNP are also known to be in P, and being in P is bad for security. If I encrypt something and tell you that the password is the product of 97 and 127, you can very quickly figure out my password. However, factoring is a problem in both NP and coNP, and it is not yet known if it is in P. If I tell you that my password is the prime factors of $12, 319$, it takes a lot more effort for you to find my password, but it's easy for a computer to check if the password you give it is correct (by multiplying the numbers together and testing if they are prime).

It is currently not known if there are any problems in NP but not in P. There is a set of problems called NP-complete problems. These are, in some sense, the hardest problems in NP. A problem $X$ is in NP-complete if, for every problem $Y$ in NP, there is a way to, with a quick algorithm, convert a problem from $Y$ to a problem from $X$, such that solving $X$ quickly will give a quick solution to $Y$. Thus, finding a quick solution to any NP-complete problem would be a proof that P=NP.

Proving from scratch that a problem is in NP-complete is very difficult. Luckily, Richard Karp did this in 1972 [Kar72]. Once you have one NP-complete problem $A$, proving that another NP problem $B$ is NP-complete is only as difficult as proving that $A$ can be modified into $B$. That is, showing that a solution to $B$ can be used to solve $A$.

One of Karp's original 21 NP-complete problems is Satisfiability, or SAT. In SAT, there is a list of variables $x_1, \ldots, x_n$ that each can be either TRUE or FALSE. There are also literals $x_1, \neg x_1, \ldots, x_n, \neg x_n$. Clauses are of the form $(\ell_1 \vee \cdots \vee \ell_k)$ where each $\ell_i$ is a literal. A clause is satisfied by a TRUE/FALSE assignment to $x_1, \ldots, x_n$ if at least one of its literals is true under that assignment. A SAT problem is a Boolean formula which is a conjunction of clauses, $C_1 \wedge \cdots \wedge C_m$. A SAT problem can be solved if there is an assignment of TRUE and FALSE to the variables such that each clause is satisfied. Note that such an assignment, if provided, is easy to confirm as a solution. For example, $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2)$ is solved with $x_1 =$ FALSE, $x_2 =$ TRUE. $x_2$ satisfies the first clause, and either of $x_1$ and $x_2$ will satisfy the second clause. This is a very small example, but it will come up again.

## 3 3D-Matching

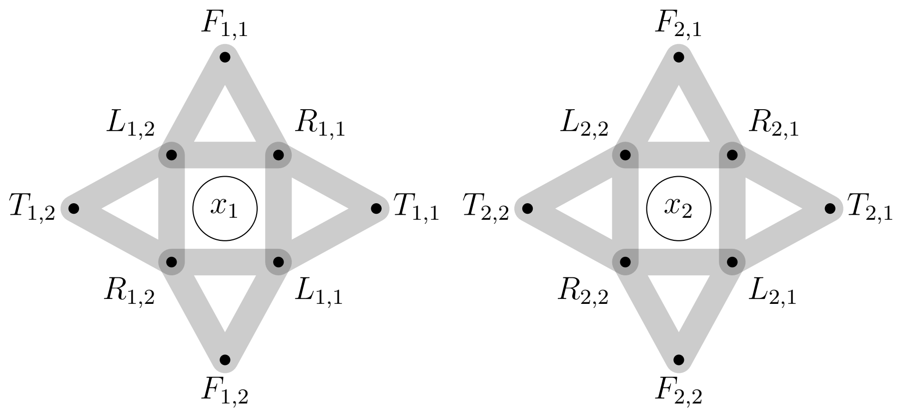
Another problem in NP is 3D-Matching. Given three sets of equal size $V_1, V_2, V_3$, and a set of ordered triples $E \subseteq V_1 \times V_2 \times V_3$, does there exist a subset $M \subseteq E$ such that $|M| = |V_1|$ and, for any $e, e' \in M$ with $e = (v_1, v_2, v_3)$ and $e' = (v_1', v_2', v_3')$, and for any $1 \leq i \leq 3$, no entries of $e$ and $e'$ match unless $e = e'$. If you are familiar with bipartite matching, you might see how this can be thought of as an extension of that problem into a higher dimension: $V_1 \cup V_2 \cup V_3$ is a partition of vertices, and $E$ is a set of hyperedges, where each hyperedge connects a vertex from each set. Note that a given subset $M$ can quickly be confirmed as a solution to the problem, so this problem is in NP.

To prove that 3D-matching is NP-complete, we will look at a way to convert any SAT problem into a 3D-Matching problem. We will use $(x_1 \vee x_2) \wedge (\neg x_1 \vee x_2)$ as an example throughout the construction.

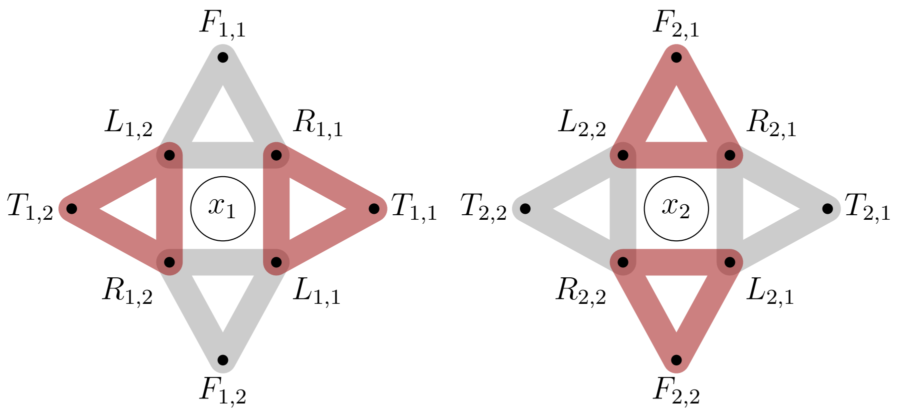
Consider an arbitrary Boolean expression of the form $C_1 \wedge \cdots \wedge C_m$. Let $n$ and $m$ be the numbers of variables and clauses, respectively. In our (tiny) example, $n = m = 2$. In general, these will not necessarily be equal. For each of the $n$ variables, we create $4m$ vertices. For $x_i$, we get the vertices $T_{i,1}, \ldots, T_{i,m}$, $F_{i,1}, \ldots, F_{i,m}$, $L_{i,1}, \ldots, L_{i,m}$, and $R_{i,1}, \ldots, R_{i,m}$. The $T$ and $F$

vertices will be used to indicate a truth assignment to $x_i$. The $L$ and $R$ vertices exist to require certain elements of $E$ to be included in $M$. We call them left and right because we will have several vertices that come in pairs. The second subscript for the $T$ and $F$ vertices is used to indicate a clause to which those vertices belong. The $T$ and $F$ vertices all belong to $V_1$. The $L$ vertices belong to $V_2$. The $R$ vertices belong to $V_3$.

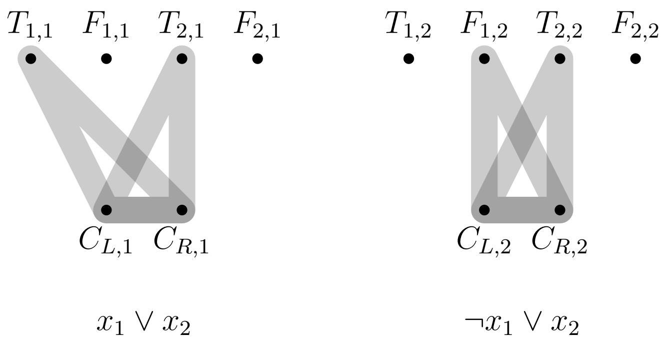
Now we write down some elements of $E$. For each $1 \leq i \leq n$, and for each $1 \leq j \leq m$, $E$ has triples $(T_{i,j}, L_{i,j}, R_{i,j})$ and $(F_{i,j}, L_{i,j+1}, R_{i,j})$. Note that the second subscripts are thought of modulo $m$, so $L_{i,m+1} = L_{i,1}$. Below, we see these vertices and hyperedges for our tiny example. We will refer to each of those subhypergraphs (with $4m$ vertices and $2m$ hyperedges) as "gadgets".



While the "T" and "F" vertices will be included in more hyperedges, the "L" and "R" vertices are in only these and no other hyperedges. Thus, any matching which solves this problem will use half of the hyperedges from each of these gadgets. For each $x_i$, either all the $T_{i,j}$ vertices or all the $F_{i,j}$ vertices will be left available after those hyperedges are chosen. For example, if we wanted to pick some of the above hyperedges to indicate $x_1$ = FALSE and $x_2$ = TRUE, we would select the red hyperedges in the following picture.



Next, for each clause $C_j$, we introduce two vertices, $C_{L,j}$ and $C_{R,j}$. These two vertices only exist in hyperedges with each other. For each literal $x_i$ in the clause, we make a hyperedge $(T_{i,j}, C_{L,j}, C_{R,j})$. For each literal $\neg x_i$ in the clause, we make a hyperedge $(F_{i,j}, C_{L,j}, C_{R,j})$. Below see these vertices and hyperedges for our example. Note that the diagram below doesn't include all of the vertices and hyperedges from before, as including them would make the picture difficult to understand.

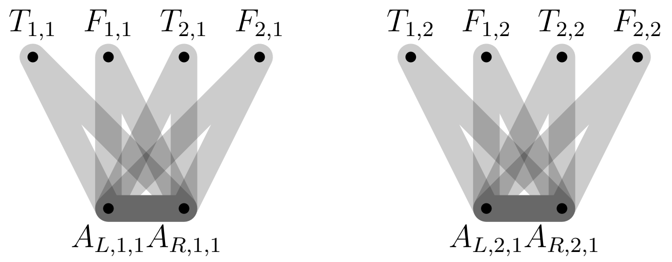


In this, we see that satisfying the clause $C_j$ turns into the problem of fitting vertices $C_{L,j}$ and $C_{R,j}$ into some hyperedge of our matching.

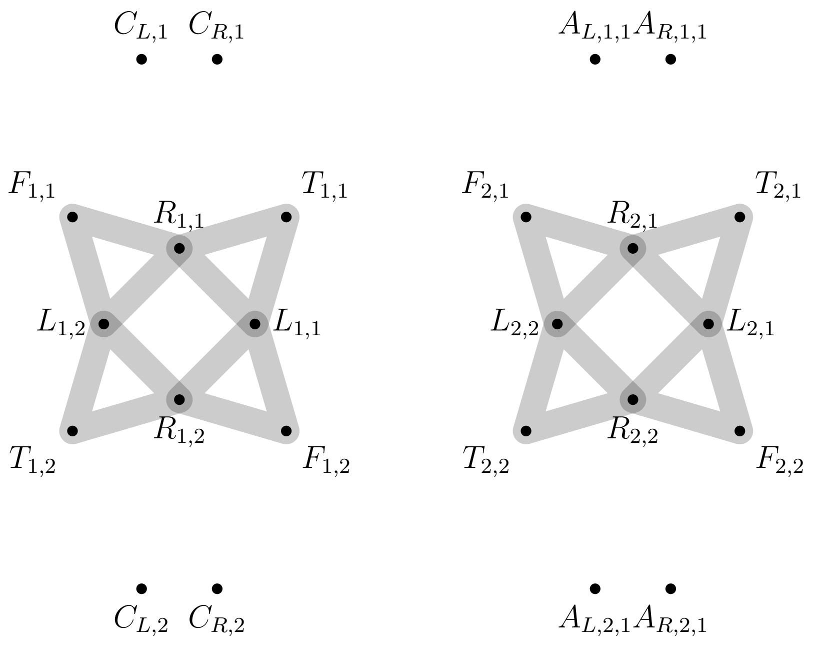
At this point, we're almost finished with our diagram. However, we are required to have three sets of equal size. $V_1$, which contains all the "T" and "F" vertices, has $2mn$ elements. The sets $V_1$ and $V_2$, which respectively contain the "L" and "R" vertices, each have size $mn+m$. Each of those sets needs an additional $mn-m$ auxiliary vertices.

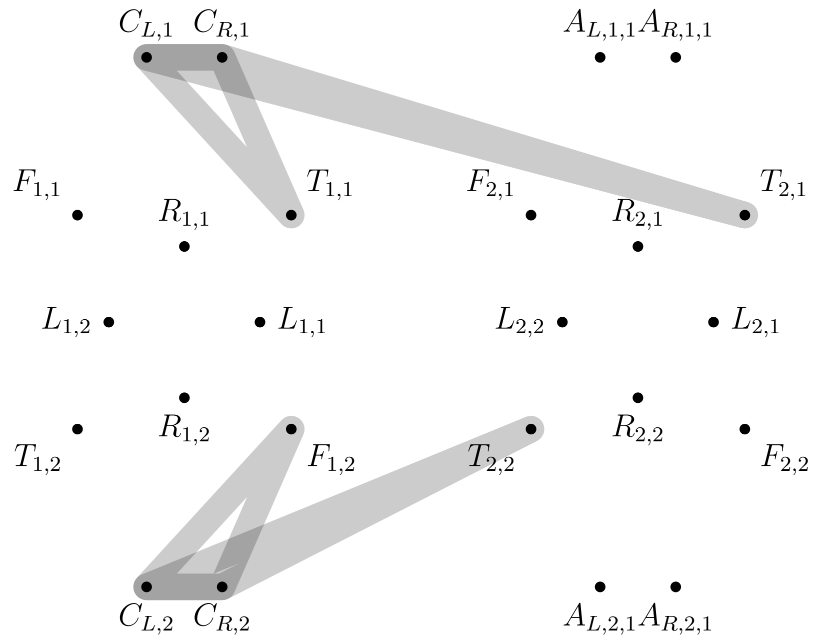
Grouping these auxiliary vertices by clause, we name them $A_{L,i,k}$ and $A_{R,i,k}$, where $1 \leq k \leq n-1$. Then we create a huge number of new hyperedges. For each $1 \leq i \leq n$, $1 \leq j \leq m$, and $1 \leq k \leq n-1$, we make two hyperedges $(T_{i,j}, A_{L,i,k}, A_{R,i,k})$ and $(F_{i,j}, A_{L,i,k}, A_{R,i,k})$. Note that $A_{L,i,k}$ and $A_{R,i,k}$ are always together. To see why we make these hyperedges, realise that every vertex needs to be in exactly one of the hyperedges in the matching. Of the "T" and "F" vertices, half will be included by hyperedges which show a truth assignment. After this, each variable has one vertex devoted to each clause. Satisfying a clause in the SAT problem is equivalent to selecting a hyperedge of one of these two types in the 3D-Matching problem.

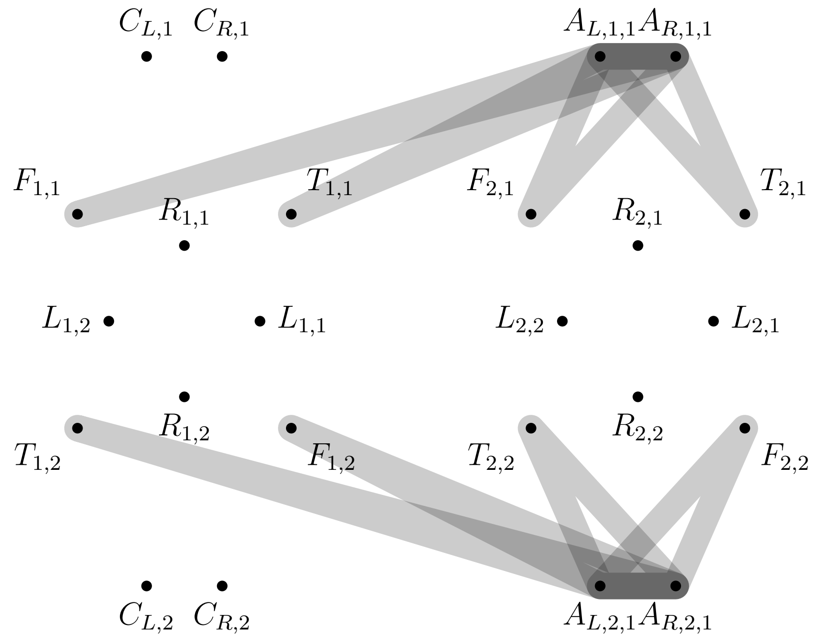
Now, each clause has $n-1$ more vertices devoted to it than is included in hyperedges. With these auxiliary vertices, we can include the rest of the vertices. For each clause, there are $n-1$ pairs of auxiliary vertices. Each of these pairs is matched to each "T" and "F" vertex for that clause. Thus, after some of the "T" and "F" vertices are includes by the truth assignment and the clause satisfaction, the rest will be able to be matched by the auxiliary vertices. The picture below shows the "T", "F", and "A" vertices for our example, along with hyperedges. Because our example is so small, the only value for $k$ is 1.

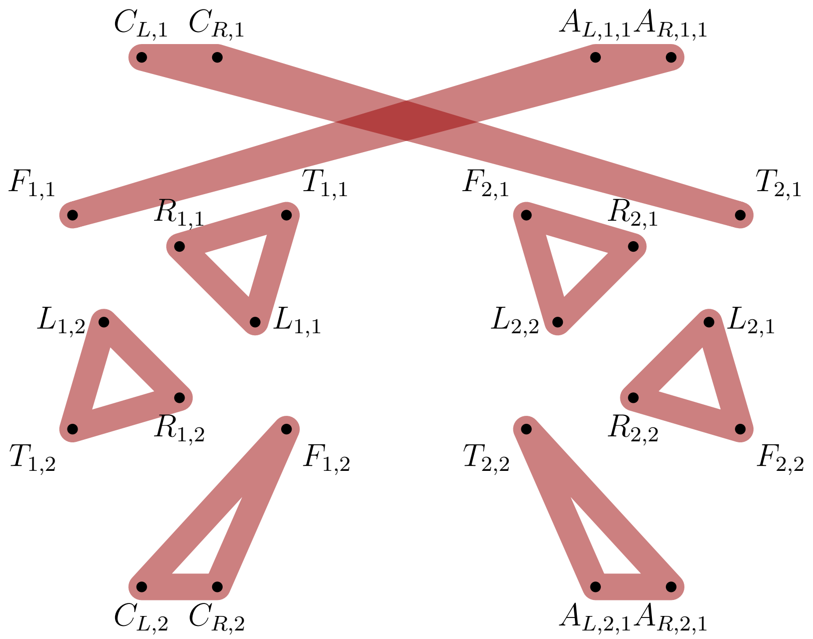


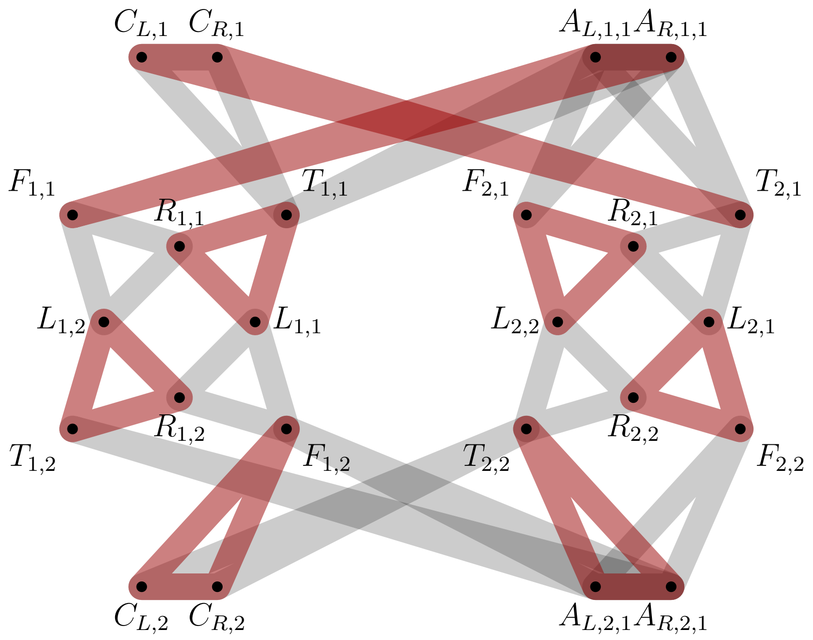
The next five diagrams each have all the vertices. The first diagram has only the hyperedges which show truth assignment. The second diagram has only the hyperedges that show satisfying the clauses. The third diagram has only the auxiliary hyperedges. The fourth diagram has a subset of hyperedges that solves the 3D-Matching problem. The last one has every hyperedge, and those in the matching are red. The last one is included mostly for completeness's sake. If you can't understand what's going on in it, don't worry.




$C_{L,1}$
$C_{R,1}$
$A_{L,1,1}$
$A_{R,1,1}$
$F_{1,1}$
$R_{1,1}$
$T_{1,1}$
$F_{2,1}$
$R_{2,1}$
$T_{2,1}$
$L_{1,2}$
$L_{1,1}$
$L_{2,2}$
$L_{2,1}$
$R_{1,2}$
$T_{1,2}$
$F_{1,2}$
$T_{2,2}$
$R_{2,2}$
$F_{2,2}$
$C_{L,2}$
$C_{R,2}$
$A_{L,2,1}$
$A_{R,2,1}$


$C_{L,1}$
$C_{R,1}$
$A_{L,1,1}$
$A_{R,1,1}$
$F_{1,1}$
$R_{1,1}$
$T_{1,1}$
$F_{2,1}$
$R_{2,1}$
$T_{2,1}$
$L_{1,2}$
$L_{1,1}$
$L_{2,2}$
$L_{2,1}$
$R_{1,2}$
$T_{1,2}$
$F_{1,2}$
$T_{2,2}$
$R_{2,2}$
$F_{2,2}$
$C_{L,2}$
$C_{R,2}$
$A_{L,2,1}$
$A_{R,2,1}$





Now that we have this construction, we need to prove two things. First, we need to show that the size of the 3D-Matching problem isn't too big. In order to show this is NP-complete, we want to show that a SAT problem can be turned into a 3D-Matching problem in polynomial time. Second, we need to show that these are the same problem. That is, one can be solved exactly when the other can be solved.

## 3.1 Conversion in Polynomial Time

This is a simple counting argument. With $n$ variables and $m$ clauses, we can compute how many vertices and hyperedges our hypergraph will have.

As for the "T" and "F" vertices, each of the $n$ variables brings one of each for each of the $m$ clauses. That makes $mn$ "T" vertices and $mn$ "F" vertices. The "L" and "R" vertices are counted the same way, and there are $mn$ of each. For each clause $C_i$, there are two vertices, $C_{i,L}$ and $C_{i,R}$. This makes another $2m$ vertices. Lastly, we have the "A" vertices. Earlier we counted $mn - m$ each of "$\mathrm{A_L}$" and "$\mathrm{A_R}$", for a total of $2mn - 2m$ auxiliary vertices. Now compute how many vertices are in each of the $V_i$ sets. $V_1$ has all the "T" and "F" vertices, so $|V_1| = 2mn$. $V_2$ has all the "L", "$\mathrm{C_L}$", and "$\mathrm{A_L}$" vertices. Thus, $|V_2| = (mn) + (m) + (mn - m) = 2mn$. Similarly, $V_3$ has the corresponding Right vertices, so $|V_3| = 2mn$. In total, there are $6mn$ vertices, and that is quadratic in terms of the input.

Now we count the hyperedges. For the gadgets which let us show truth assignments, we make $2m$ hyperedges for each variable, or $2mn$ hyperedges. Clauses are a bit harder to count, since clauses can have any positive number of literals in them. However, if there are $n$ variables, each clause has no more than $n$ literals in it. Each clause gets a hyperedge for each literal in the clause, so this is no more than $mn$ hyperedges in total. Lastly we have the auxiliary vertices and the hyperedges they are in. There are $mn - m$ pairs of auxiliary vertices. These are split by clause into $m$ groups of $n-1$ pairs. Each pair is in two hyperedges for each variable. That makes $m\big((n-1)(2n)\big)$ hyperedges. In total, that's no more than $2mn+mn+2mn(n-1) = 2mn^2+mn$ hyperedges. That's no more than cubic in the input, so we can create the 3D-Matching problem quickly.

## 3.2 Equivalent Solutions

In order for this construction to prove that 3D-Matching is NP-complete, it remains to be shown that the constructed 3D-Matching problem has a solution exactly when the original SAT problem has a solution.

First, assume that the original SAT problem has a solution. Consider one truth assignment that solves the original SAT problem. For each variable $x_i$, select a set of hyperedges based on $x_i$'s assignment. If $x_i$ is TRUE, select the hyperedges $\{F_{i,j}, L_{i,j+1}, R_{i,j}\}$ for every $j$. If $x_i$ is FALSE, select the hyperedges $\{T_{i,j}, L_{i,j}, R_{i,j}\}$ for every $j$. This will include each "L" and "R" vertex in a hyperedge.

For each clause $C_j$, there is at least one variable $x_i$ whose assignment satisfies $C_j$. If $x_i$ is TRUE, select the hyperedge $\{T_{i,j}, C_{L,j}, C_{R,j}\}$. Otherwise, select the hyperedge $\{F_{i,j}, C_{L,j}, C_{R,j}\}$.

Of the "T" and "F" vertices whose second subscript is $j$, so far $n+1$ of $2n$ are included in selected hyperedges. Each of those remaining vertices are in hyperedges with pairs $(A_{L,j,k}, A_{R,j,k})$, where $k$ ranges from 1 to $n-1$. Finding a matching for those vertices is equivalent to a bipartite matching problem where one of the partitions is the same "T" and "F" vertices and the other partition is new vertices $A_{j,k}$. This would be a complete bipartite graph with equal sized partitions, so there exists a matching for it. Thus, all the "T" and "F" vertices get matched. Since each of the three sets of vertex sets has the same cardinality, each vertex gets matched. Thus, the constructed 3D-Matching problem has a solution. □

Now assume that the constructed 3D-Matching problem has a solution. For each variable $x_i$, there are only two ways to match all the vertices $L_{i,j}$ and $R_{i,j}$, and these correspond to truth assignments as described earlier. Thus, we have a truth assignment for each $x_i$.

For each clause $C_j$, consider the vertices $C_{L,j}$ and $C_{R,j}$. These are included in a hyperedge in the matching. They only appear in hyperedges with each other, and the other vertex in these hyperedges corresponds to one of the literals in $C_j$. The third vertex in the hyperedge containing $C_{L,j}$ and $C_{R,j}$

is either $T_{i,j}$ or $F_{i,j}$, for some $i$. If it is $T_{i,j}$, $C_j$ is satisfied by $x_i$, and our truth assignment assigns TRUE to $x_i$. If it is $F_{i,j}$, $C_j$ is satisfied by $\neg x_i$, and our truth assignment assigns FALSE to $x_i$. Either way, our truth assignment satisfies $C_j$. Thus, our truth assignment satisfies every clause, so the original SAT problem has a solution. Now we have our requirement that the constructed 3D-Matching problem has a solution exactly when the original SAT problem has a solution. $\blacksquare$

So 3D-Matching is another NP-complete problem.

## 4 Learning With Errors

The Learning With Errors problem was first discussed in a paper by Regev in 2005 [Reg05]. Let $n$ and $q$ be positive integers. Let $\chi$ be a probability distribution on $\mathbb{Z}_q$. Let $s$ be a secret vector chosen uniformly at random from $\mathbb{Z}_q^n$. We want to look at pairs of random vectors $a$ and numbers $b$ which are sometimes equal to $\langle a, s\rangle$. More formally, define a new distribution, $L_{s,\chi}$ as giving samples of the form $(a, \langle a, s\rangle + \epsilon)$, where $a$ is sampled uniformly at random from $\mathbb{Z}_q^n$ and $\epsilon$ is sampled from $\chi$. This distribution is the starting point for Learning With Errors. Define a new distribution, $L_{s,\chi}^D$ as giving samples from $L_{s,\chi}$ with probability 0.5 and as giving samples uniformly at random from $\mathbb{Z}_q^n \times \mathbb{Z}_q$ with probability 0.5. With the distributions $L_{s,\chi}$ and $L_{s,\chi}^D$, we have two problems.

**Search LWE:** Given an oracle that can produce samples from $L_{s,\chi}$, find the vector $s$.
**Decision LWE:** Given an oracle that can produce samples from $L_{s,\chi}^D$, determine for each sample if it came from $L_{s,\chi}$.

In this paper, we will only discuss Search LWE, but Decision LWE was included for completeness. Typically, $q$ is assumed to have a size that is polynomial in $n$. Also, $\chi$ is usually assumed to be a discrete Gaussian over $\mathbb{Z}$ with a mean of 0 and a known variance.

In order to show that Search LWE is NP-complete, we will first introduce a problem known as Coset Weight:

Given a matrix $A$, a vector $y$, and a positive integer $w$, does there exist a vector $x$ such that $Ax = y$ and $x$ has no more than $w$ non-zero entries?

To show that a solution to Coset Weight will solve 3D-Matching, construct the matrix $A$ as follows [BMT78]. For each vertex in the hypergraph, $A$ has a row. Every column of $A$ has 3 ones and everything else zero, where the ones correspond to the three vertices in a hyperedge. $w$ is one-third the number of vertices, and $y$ is the all ones vector. Finding a solution to this system is easy using Gaussian elimination over $\mathbb{Z}_2$, but it is much more difficult when we put a limit on the number of non-zero entries of $x$. The matrix for the above example has 24 rows and 20 columns. It is given below.

$$
\begin{array}{l|cccccccc|cccc|cccccccc|}
T_{1,1} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
F_{1,1} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
T_{1,2} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
F_{1,2} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
T_{2,1} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
F_{2,1} & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
T_{2,2} & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
F_{2,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\cline{2-21}
L_{1,1} & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
L_{1,2} & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
L_{2,1} & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
L_{2,2} & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
C_{L,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
C_{L,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A_{L,1,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
A_{L,2,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
\cline{2-21}
R_{1,1} & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
R_{1,2} & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
R_{2,1} & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
R_{2,2} & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
C_{R,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
C_{R,2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
A_{R,1,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
A_{R,2,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1
\end{array}
$$

$$
x^T = \left[\begin{array}{cccccccc|cccc|cccccccc} 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \end{array}\right]
$$

Finally, we want to show that a solution to LWE solves this. Given a Coset Weight problem, construct an LWE problem as follows:

$$
\begin{bmatrix} A \\ A \\ \vdots \\ A \\ I \end{bmatrix} x \approx \begin{bmatrix} y \\ y \\ \vdots \\ y \\ 0 \end{bmatrix}
$$

Lastly, say that at most $w$ rows are allowed to not be solved by $x$. With $w+1$ copies of $A$ and $y$, we require $Ax = y$. With $Ix \approx 0$, we allow no more than $w$ entries of $x$ to not be 0. It is left to the reader to confirm that both of these reductions can be done in polynomial time.

At this time, we believe there is no (non-trivial) distribution $\chi$ for which Search LWE is known to not be NP-hard. The hardness of LWE was studied by Regev in 2009 by reductions to lattice-based problems. The hardness of these problems was in turn studied by Aharonov and Regev in 2005 [AR05]. The current best attack on general instances of LWE is exponential, and was designed by Blum, Kalai, and Wasserman [Avr03; Alb+12]. In this paper, we will consider a slightly different version of Search LWE (which henceforth will be abbreviated to LWE). In real life, when someone tries to generate random samples from a distribution, they often fail. It is very hard to get truly random data, so people settle for pseudo-random data. Because of this, we will allow ourselves the assumption that samples from $\chi$ are not independent, or $\chi$ is flawed in some other way (e.g. sparse). Thus, instead of looking at $m$ samples from $L_{s,\chi}$, we instead look at a matrix $A$ sampled uniformly from $\mathbb{Z}_q^{m\times n}$ and a vector $b = As + e$, where $e$ is sampled from some distribution $\chi$ on $\mathbb{Z}_q^m$. But we no longer assume the entries of $e$ will be independent.

## 5 LWE with Zero Errors

Henceforth, we will assume that $q$ is a prime, so $\mathbb{Z}_q$ is a finite field, $\mathbb{F}_q$. Suppose we were given an instance of LWE where $\chi$ always produces 0. Samples from this instance of LWE are of the form $(a, \langle a, s\rangle)$, where $a$ is uniformly sampled from $\mathbb{F}_q^n$. If this is known to be the case, it is trivial to solve for $s$. Gaussian elimination performed modulo $q$, applied to the matrix $[A|b]$, will find the solution $s$ for $As = b$. The only time this doesn't work is if $A$ doesn't have full column rank, but this happens with minuscule probability (discussed later). Unfortunately, this method doesn't directly extend itself to later problems, so we introduce a new method.

## 6 LWE with Exactly One Error

We will next consider an instance of LWE in which the matrix $A$ is $m \times n$ and $s$ is $n \times 1$ for fixed $m$ and $n$. Rather than each $\epsilon$ being chosen independently from a distribution $\chi$, an error vector $e$ is formed by choosing a row $i_0$ uniformly at random and choosing a non-zero field element $\epsilon$ uniformly at random, and letting $e_i$ be 0 when $i \neq i_0$ and $\epsilon$ when $i = i_0$. Rather than being given samples of the form $(a, \langle a, s\rangle + \epsilon)$, we are given $A$ and $b = As + e$. In this simplified version of the problem, we will consider a polynomial way to find $s$. Because the entries of $e$ are not chosen independently, we will not be able to use an arbitrary number of rows—we will have to work with exactly $m$ rows.

Since $A$ has $n$ columns, we will consider various subsystems of $A$ with $n$ rows. For a set $S = \{s_1, s_2, \ldots, s_n\}$ of row indices, define

$$A_S = \begin{bmatrix} a_{s_1,1} & a_{s_1,2} & \cdots & a_{s_1,n} \\ a_{s_2,1} & a_{s_2,2} & \cdots & a_{s_2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{s_n,1} & a_{s_n,2} & \cdots & a_{s_n,n} \end{bmatrix}, \qquad b_S = \begin{bmatrix} b_{s_1} \\ b_{s_2} \\ \vdots \\ b_{s_n} \end{bmatrix}.$$

Similarly define $[A|b]_S$ and $e_S$. Reducing $[A|b]_S$ over $\mathbb{F}_q$ will find any and all solutions to the equation $A_S x = b_S$, where $x$ may or may not be equal to $s$. From Linear Algebra, we can view this subsystem as $n$ hyperplanes in $\mathbb{F}_q^n$. The intersection of the $n$ hyperplanes is the set of all solutions to this subsystem. These $n$ hyperplanes will either intersect nowhere, in which case we call the system inconsistent; in multiple points, in which case we call the system underdetermined; or at a unique point $x$.

**Lemma 1** *The vector $s$ is a solution to $[A|b]_S$ if and only if $i_0 \notin S$.*

**Proof:** Because $b = As + e$, $b_S = A_S s + e_S$. If $S \ni i_0$, $e_S \neq 0$, so $A_S s \neq b_S$. Similarly, if $i_0 \notin S$, $e_S = 0$, and $b_S = A_S s + 0$. $\square$

**Lemma 2** *If a subsystem $[A|b]_S$ is inconsistent, $i_0 \in S$.*

**Proof:** Since the system is inconsistent, $s$ is not a solution, so Lemma 1 says that $i_0 \in S$. $\square$

**Lemma 3** *If two subsystems $[A|b]_S$ and $[A|b]_R$ each have unique solutions $x$ and $y$, respectively, and if $x \neq y$, then $i_0 \in (S \cup R)$.*

**Proof:** Note that $s$ solves every unperturbed row of $[A|b]$. At least one of $x$ and $y$ is not equal to $s$, so assume that $s \neq x$. Because all rows of $S$ share a unique solution other than $s$, at least one of those rows is not solved by $s$, so $i_0 \in S$. If we instead assume that $s \neq y$, then $i_0 \in R$.

**Lemma 4** *If two subsystems $[A|b]_S$ and $[A|b]_R$ share a unique solution $x$, then*

$$i_0 \in (S \cap R) \cup \overline{S \cup R}.$$

**Proof:** Consider first the case where $x = s$. In this case, since $s$ solves both subsystems, no row in either subsystem is perturbed (by Lemma 1), and $i_0 \in \overline{S \cup R}$. Now consider the case where $x \neq s$. Since the solution to each subsystem is unique, $s$ solves neither subsystem, so each subsystem contains a row not satisfied by $s$. Since only one row is not satisfied by $s$, it must be the same row in each subsystem that is not satisfied, and $i_0 \in (S \cap R)$.

## 6.1 Polynomial Solution to One Error Case

If we examine every square subsystem, that's sufficient to find $s$. As long as $m > n$, there will be a square subsystem that is unperturbed. The probability that such a system will have full rank is discussed later, but for now we just care that, for reasonable values of $q$ (at least 1000), the chances of a square matrix not having full rank are close to zero. Furthermore, we only need one of these matrices to have a unique solution.

As long as the number of systems to solve is polynomial in terms of the input, this is a polynomial solution (though it's really inefficient). Consider the case when $n$ is fixed and $m$ is allowed to grow. The number of matrices to examine is $\binom{m}{n}$.

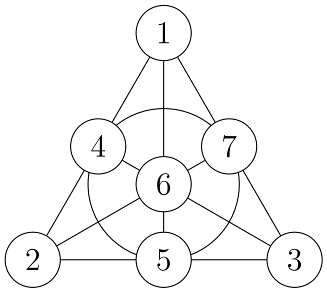
$$\binom{m}{n} = \frac{m(m-1)\cdots(m-n+1)}{n!} < \frac{1}{n!}m^n$$

Because $n$ is constant, this is polynomially bounded.

This result doesn't depend too heavily on there being a single error. If we replace the requirement of $m > n$ with a requirement on the number of errors $k$ such that $k \leq m - n$, we will still be guaranteed to have an unperturbed system. However, even though this is polynomial, we still want to make it better.

## 7 Seven Rows

The Fano plane is a 3-regular hypergraph with 7 vertices and 7 hyperedges. For now, all hyperedges are unordered triplets of vertices. Furthermore, every pair of vertices shares a unique hyperedge, and every pair of hyperedges has a unique intersection. For the following labelling of the Fano plane, the seven hyperedges are $\{1,2,4\}$, $\{2,3,5\}$, $\{3,4,6\}$, $\{4,5,7\}$, $\{1,5,6\}$, $\{2,6,7\}$, and $\{1,3,7\}$.



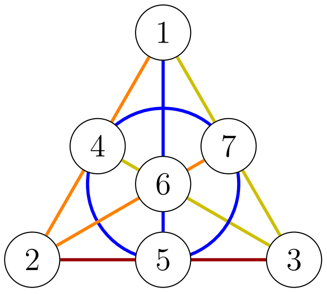
In the case when $A$ has 7 rows and 3 columns, we will only look at 7 subsystems to determine $s$ and $e$. To find our 7 subsystems of $[A|b]$, we will identify each row of $[A|b]$ with a vertex in the Fano plane. Each hyperedge now corresponds to a set of three rows of $[A|b]$, so each hyperedge $S$ is a subsystem $[A|b]_S$. Below, see an example of a system over $\mathbb{Z}_5$ and subsystems given by two of the hyperedges in the Fano plane.

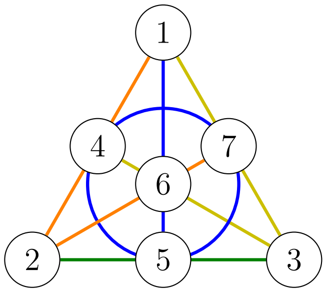
$$[A|b] = \left[\begin{array}{ccc|c} 4 & 1 & 4 & 0 \\ 1 & 2 & 3 & 4 \\ 3 & 4 & 2 & 3 \\ 0 & 4 & 1 & 2 \\ 2 & 1 & 1 & 1 \\ 3 & 2 & 2 & 4 \\ 0 & 1 & 4 & 3 \end{array}\right] \quad \begin{array}{l} [A|b]_{\{1,2,4\}} = \left[\begin{array}{ccc|c} 4 & 1 & 4 & 0 \\ 1 & 2 & 3 & 4 \\ 0 & 4 & 1 & 2 \end{array}\right] \\ [A|b]_{\{4,5,7\}} = \left[\begin{array}{ccc|c} 0 & 4 & 1 & 2 \\ 2 & 1 & 1 & 1 \\ 0 & 1 & 4 & 3 \end{array}\right] \end{array}$$

Using this identification, each hyperedge has a (possibly empty) set of solution vectors. Since the Fano plane is a hypergraph, we can colour its hyperedges. The colours we will use correspond to possible solution sets to our subsystems: no solutions, multiple solutions, and unique solutions. Specifically, the set of colours is $\mathbb{F}^3 \cup \big\{\{\}, \text{"underdetermined"}\big\}$. If a hyperedge is an inconsistent subsystem of $[A|b]$, it is given the colour $\{\}$. If a hyperedge is an underdetermined system, it is given the colour "underdetermined". If a hyperedge is a system with a unique solution $x$, it is given the colour $x$. In this paper, red will be used for $\{\}$, blue will be used for "underdetermined", and every other colour will represent a vector.

**Lemma 5** *If there is a unique vector $x$ such that $Ax - b$ has a single non-zero entry, then $i_0$ is determined from the seven subsystems constructed from the Fano plane and Lemmas 2 through 4*

**Proof:** There are only two colourings (up to graph isomorphism) in which it is unclear at which position the error is. They are:





Recall the rule where two systems each with distinct unique solutions implies that the error is in a row included by at least one of the systems. As orange and pink are both signifying a unique solution, pairs of those lines show that the error in each picture is in either position 2 or position 3. Let $x$ be the solution denoted by orange lines and let $y$ be the solution denoted by pink lines. Aside from Row 5, each other row is in a system accepting $x$ and a system accepting $y$. Since Rows 1 and 6 are not multiples of each other, and since they both accept solutions $x$ and $y$, their intersection is the line passing through $x$ and $y$. Since the system $\{1, 5, 6\}$ is underdetermined in each picture, Row 5 must also be satisfied by everything on that line. Thus, every row except for Row 2 is satisfied by $y$, and every row except for Row 3 is satisfied by $x$. So $Ax - b = (0, 0, p, 0, 0, 0, 0)$ and $Ay - b = (0, q, 0, 0, 0, 0, 0)$. In both of these cases, there are two solutions that solve $As + e = b$, where $e$ has exactly one non-zero entry.

## 8 Probability is a Thing

Because some matrices will not have unique solutions, and this attack hinges on finding a unique solution, we want to determine how likely it is that a matrix we try to solve will have a unique solution.

Suppose we are working over the field $\mathbb{F}_q$, and we have a matrix $A$ with $n$ columns. Recall that each row of $A$ is chosen uniformly from the vectors in $\mathbb{F}_q^n$.

**Proposition 1** *The size of a $k$-dimensional subspace of $\mathbb{F}_q^n$ is $q^k$.*

**Proof:** Let $S \subseteq \mathbb{F}_q^n$ be a $k$-dimensional subspace. Let a basis $\{w_1, w_2, \ldots, w_k\}$ for $S$ be given. Every vector $w \in S$ can be uniquely represented as a linear combination of the basis vectors: $w = \alpha_1 w_1 + \alpha_2 w_2 + \cdots + \alpha_k w_k$. Each $\alpha_i$ is chosen from $\{0, 1, \ldots, q-1\}$, and each vector $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_k) \in \mathbb{Z}_q^k$ uniquely determines a vector $w \in S$. Thus, there are $\left|\mathbb{Z}_q^k\right| = q^k$ linear combinations of the basis vectors, so $|S| = q^k$. $\square$

A $k \times n$ matrix $A$ chosen over $\mathbb{F}_q$ has rank-$k$ when the set of $k$ row vectors of $A$ form a linearly independent set. So the problem of finding a rank-$k$ matrix $A$ is the same as the problem of finding a $k$-tuple of linearly independent vectors.

**Lemma 6** *The number of $k$-tuples $(v_1, v_2, \ldots, v_k)$ of linearly independent vectors in $\mathbb{F}_q^n$ is given by*

$$(q^n - 1)(q^n - q)(q^n - q^2) \cdots (q^n - q^{k-1}) = q^{\binom{k}{2}} \prod_{r=0}^{k-1} (q^{n-r} - 1)$$

**Proof:** We will show this by induction. First consider $k = 1$. There are $q^n$ vectors in $\mathbb{F}_q^n$. With the exception of the zero vector, the set containing a single vector from $\mathbb{F}_q^n$ is a linearly independent set. So there are $q^n - 1$ one-tuples of linearly independent vectors.

Now assume that, for some $k$, the number of $k$-tuples of linearly independent vectors from $\mathbb{F}_q^n$ is $(q^n - 1)(q^n - q)(q^n - q^2) \cdots (q^n - q^{k-1})$. To construct a $(k+1)$-tuple of independent vectors, begin with a $k$-tuple $(v_1, v_2, \ldots, v_k)$. This set is a basis for a $k$-dimensional vector space, so there are $q^k$ choices for the vector $v_{k+1}$ that, when added to this set, will produce a linearly dependent set. Thus, there are $q^n - q^k$ choices for $v_{k+1}$ that can be added to this set to form a linearly independent set of size $k+1$. Multiplying these, we see that there are $(q^n - 1)(q^n - q)(q^n - q^2) \cdots (q^n - q^{k-1})(q^n - q^k)$ $(k+1)$-tuples of linearly independent vectors from $\mathbb{F}_q^n$. $\blacksquare$

Using this, we can count the number of invertible matrices over $\mathbb{F}_q$. An $n \times n$ matrix with rows $v_1$ through $v_n$ can be viewed as the $n$-tuple $(v_1, v_2, \ldots, v_n)$

of vectors from $\mathbb{F}_q^n$, and it's invertible when that $n$-tuple is a linearly independent set. Thus, there are $q^{\binom{n}{2}} \prod_{r=0}^{n-1}(q^{n-r}-1)$ invertible $n \times n$ matrices. The total number of $n \times n$ matrices is $q^{n \cdot n}$, as each of the $n^2$ entries of the matrix has $q$ possible values. Thus, we can divide to find the probability that an $n \times n$ matrix sampled uniformly at random from $\mathbb{F}_q^{n \times n}$ is invertible:

$$\frac{q^{\binom{n}{2}} \prod_{r=0}^{n-1}(q^{n-r}-1)}{q^{n \cdot n}} = \frac{q^{\frac{n(n-1)}{2}} \prod_{r=0}^{n-1}(q^{n-r}-1)}{q^{n \cdot n}} = \frac{1}{q^{\frac{1}{2}(n^2+n)}} \prod_{r=0}^{n-1} \left(q^{n-r}-1\right).$$

In the $3 \times 3$ case, we get a probability of

$$\frac{1}{q^6} \prod_{r=0}^{2} \left(q^{3-r}-1\right) = \frac{1}{q^6}(q^3-1)(q^2-1)(q-1) = 1 - \frac{1}{q} - \frac{1}{q^2} + \frac{1}{q^4} + \frac{1}{q^5} - \frac{1}{q^6}$$

which, for large values of $q$, is very close to $1-\frac{1}{q}$. For $q$ at least 2, it's between $1-\frac{2}{q}$ and $1-\frac{1}{q}$.

Next we want to look at the probability that an $m \times n$ matrix with $m \geq n$ has rank $n$. This is simply the probability that the $n$ column vectors (sampled uniformly from $\mathbb{F}_q^m$) are linearly independent. The number of ways to have those vectors be independent is given earlier as $q^{\binom{n}{2}} \prod_{r=0}^{n-1}(q^{m-r}-1)$, and the number of $m \times n$ matrices is $q^{mn}$. So the probability is

$$\frac{q^{\frac{1}{2}(n^2+n)}}{q^{mn}} \prod_{r=0}^{n-1} \left(q^{m-r}-1\right) = \frac{q^{\frac{1}{2}(n+1)}}{q^{m}} \prod_{r=0}^{n-1} \left(q^{m-r}-1\right).$$

From that, we can get an upper bound of the probability that, with $m$ rows, there is no invertible submatrix by subtracting it from 1.

Using the method described above for solving LWE with 1 error, the approximation above gives us a requirement on $q$ for us to be able to succeed with high probability. We'll compute for the case when $A$ has 7 rows.

With 7 rows, we look at seven $3 \times 3$ matrices, 3 of which include the row with an error. There are four unperturbed matrices. If at least one of those four is invertible, the vector $x$ that solves that subsystem of $Ax = b$ will solve 6 equations out of 7 and will thus solve the LWE instance. So we want to compute the probability that at least one of those four matrices is invertible. Because every pair of rows appears in only one matrix, the probabilities for each of the matrices being invertible are independent. The chance for a certain matrix to not be invertible is between $\frac{1}{q}$ and $\frac{2}{q}$, so the chance for all four to not be invertible is between $\frac{1}{q^4}$ and $\frac{16}{q^4}$. Thus, the chance that at least one is invertible is between $1 - \frac{16}{q^4}$ and $1 - \frac{1}{q^4}$. If we want to guarantee that the method will work with probability at least 0.999, we simply solve $1 - \frac{16}{q^4} \geq 0.999$ for $q$. Since $q$ has to be a positive integer, this means that $q > 11$ will guarantee that at least one unperturbed matrix is invertible with probability 0.999. The requirement on $q$ grows very slowly, especially considering that most primes chosen in real life applications are well over 1000, and the chance of being able to solve the instance when $q > 1000$ is about 0.999999999984.

Suppose we have an LWE instance with an $m \times n$ matrix $A$ with entries chosen from $\mathbb{F}_q$, and suppose up to $k$ entries of $b$ are perturbed by noise. One attack would be to pick $n$ rows from $[A|b]$ at random and try so solve for $s$. If a unique solution $x$ to the system is found, test how many rows of $[A|b]$ are solved by $x$. If at least $m - k$ rows are satisfied by $x$, terminate and produce $x$; otherwise repeat by picking a new set of $n$ rows. This is a randomised method very similar to the $t$-design approach earlier. The $t$-design approach requires having a design to use with a small enough minimum blocking set, so this method can be used when a suitable design isn't available. What follows is a computation that such a randomly chosen system will have $s$ as a unique solution.

The probability that the system has a unique solution has been shown above and is simply $\frac{1}{q^{\frac{1}{2}(n^2+n)}} \prod_{r=0}^{n-1} (q^{n-r} - 1)$. Next we want to find the probability that none of the $k$ perturbed rows are in our $n$. At worst, there will be $k$ rows with error, so we'll continue by assuming that. There are $\binom{m-k}{n}$ ways to choose rows that are error-free, and $\binom{m}{n}$ total ways to choose $n$ rows. Dividing gives the probability of the system being error-free as $\frac{(m-k)!(m-n)!}{m!(m-n-k)!}$. Thus, the probability that the randomly chosen subsystem has $s$ as a unique solution is no worse than

$$\frac{(m-k)!(m-n)!}{m!(m-n-k)!q^{\frac{1}{2}(n^2+n)}} \prod_{r=0}^{n-1} \left(q^{n-r} - 1\right).$$

Solving for this with $m = 7$, $n = 3$, and $k = 1$, we get a probability of $\frac{4(x-1)(x^2-1)(x^3-1)}{7x^6}$. When $q = 11$, which is when the combinatorial approach has a probability of 0.999 to succeed, this is about 0.5148. If we look at slightly more realistic numbers, with $m = 1000$, $n = 100$, $k = 10$, and $q = 1000$, we get a probability of about 0.3466. If we assume that each attempt done this way has indenpendent probability of success, this means we'd need to look at three or four systems, on average, in order to find $s$.

## 9 Periodic Noise

Another way the noise might be poorly implemented is if it is periodic. One way people generated random numbers is with a linear shift feedback register. When implemented reasonably well, the period is long enough to thwart our attacks. However, if we assume it is implemented poorly, we have a much better chance.

If we know that the cryptosystem is implemented with a randomly seeded LSFR with 50 bits, we know that the noise will have a period that is a factor of $2^{50} - 1$. A good seed will create noise with a period of $2^{50} - 1$, but $2^{50} - 1 = 3 \cdot 11 \cdot 31 \cdot 251 \cdot 601 \cdot 1801 \cdot 4051$. Of its 128 factors, 19 are smaller than one thousand, 55 are smaller than one million, and 99 are smaller than one billion. So even though the worst case for us is a period over one quadrillion, there are lots of cases that are much better for us.

If we think we know the period of the noise is $\tau$, then we can collect samples $(a_i, b_i)$ until the we have collected $(n + 1)\tau$ samples. For each $1 \leq i \leq \tau$, define the set $S_i = \{i, i + \tau, \ldots, i + n\tau\}$.

If the period of the noise is $\tau$, as we guess, then for all rows indexed by $S_i$, the following equation is true for some constant $k_i$: $As + \mathbb{1}k_i = b$. Using block matrices, this turns into $\begin{bmatrix} A & \mathbb{1} \end{bmatrix}_{S_i} \begin{bmatrix} s \\ k \end{bmatrix} = b_{S_i}$. This is something we can solve with Gaussian elimination!

$\begin{bmatrix} A & \mathbb{1} \end{bmatrix}_{S_i}$ is a square matrix, so it is invertible with high probability. Though the entries aren't all chosen at random, this is still true, which can be seen by considering the columns of $A_{S_i}$ to be chosen at random after the other column. This matrix actually have a (slightly) higher chance of being invertible than a random matrix, because we know our first column is not the zero vector. With even higher probability, there exists an $i$ such that $\begin{bmatrix} A & \mathbb{1} \end{bmatrix}_{S_i}$ is invertible. Solving one of the invertible systems, we get a guess $x$ for the secret vector, along with a guess for $k_i$. To see if $x = s$, compute $h = Ax - b$. If the vast majority of the entries of $h$ are zero, then $x = s$ with high probability. Otherwise, our guess that the period is $\tau$ is incorrect.

By iterating through guesses for $\tau$, we can break systems with periodic noise when the period isn't sufficiently long.

## 10 Burst Noise

Another way bad noise can be generated is as a burst. That is, the noise is 0 except in some small window. More formally, given $m$ samples $(a_i, b_i)$, there exist integers $t_i < t_f$ such that, for $i \notin [t_i, t_f]$, $\langle a_i, s \rangle = b$. Typically, there would be many such windows, but for now we'll assume a single burst.

For $1 \leq h \leq m$ and working with indices cyclically in $\mathbb{Z}_m$, define $S_h = \{h, h+1, \ldots, h+n-1\}$. Let $t = t_f - t_i$. We now have $m$ square matrices $A_{S_h}$, and $m - n - t$ of them will avoid the rows where the noise is not 0. Again, the chances that at least one of these matrices is invertible is very high. After solving a system to achieve a unique solution $x$, we again test it against the full system $[A|b]$ to see if the vast majority of the rows agree with $x$.

### 10.1 Combinations of Noise Models

Now that we have some attacks on a couple of weak noise models, we can look at ways to attack systems which use a combination of them.

Suppose that very sparse noise is added to periodic noise. Rather than assuming that the systems $\begin{bmatrix} A & \mathbb{1} \end{bmatrix}_{S_i} \begin{bmatrix} s \\ k \end{bmatrix} = b_{S_i}$ have no noise, we can collect more samples (adding them to the sets $S_i$ as appropriate) until we can apply the random method described in the probability section on the systems $\begin{bmatrix} A & \mathbb{1} \end{bmatrix}_{S_i} \begin{bmatrix} s \\ k \end{bmatrix} \approx b_{S_i}$.

If burst noise and periodic noise are combined, as long as the period $\tau$ is long enough in comparison to the window size $t+1$, the same approach as above will work. Once the correct period has been guessed, $S_i$ has very few rows from the window of the burst, so the noise is very sparse.

If there are multiple bursts, the same approach as for a single burst will still work, as long as the space between bursts is at least $n$.

## References

[3SC05] Donald E. Eastlake 3rd, Jeffrey I. Schiller, and Steve Crocker. *Randomness Requirements for Security.* 2005. URL: `http://tools.ietf.org/html/rfc4086`.

[Alb+12] Martin R. Albrecht et al. *On the Complexity of the BKW Algorithm on LWE.* Cryptology ePrint Archive, Report 2012/636. `http://eprint.iacr.org/`. 2012.

[AR05] Dorit Aharonov and Oded Regev. "Lattice problems in NP $\cap$ coNP". In: *J. ACM* 52.5 (2005), 749–765 (electronic). ISSN: 0004-5411. DOI: `10.1145/1089023.1089025`. URL: `http://dx.doi.org/10.1145/1089023.1089025`.

[Avr03] Hal Wasserman Avrim Blum Adam Kalai. "Noise-tolerant learning, the parity problem, and the statistical query model". In: *Journal of the ACM* 50.4 (2003), pp. 506–519. DOI: `http://dx.doi.org/10.1145/792538.792543`.

[Ber+13] Daniel J. Bernstein et al. *Factoring RSA keys from certified smart cards: Coppersmith in the wild.* Cryptology ePrint Archive, Report 2013/599. `http://eprint.iacr.org/`. 2013.

[BMT78] E. Berlekamp, R. McEliece, and H. van Tilborg. "On the inherent intractability of certain coding problems (Corresp.)" In: *IEEE Transactions on Information Theory* 24.3 (May 1978), pp. 384–386. ISSN: 0018-9448. DOI: `10.1109/TIT.1978.1055873`.

[dCo16] dCode. *Caesar Cipher. Caesar code shift - Decoder, encoder, decrypt, encrypt.* 2016. URL: `http://www.dcode.fr/caesar-cipher`.

[Edm65] Jack Edmonds. "Paths, Trees, and Flowers". In: *Canadian Journal of Mathematics* 17 (1965), pp. 449–467. DOI: `http://dx.doi.org/10.4153/CJM-1965-045-4`.

[Gau16] Gautham. *Homomorphic Encryption and Smart Contracts for Privacy and Transparency.* 2016. URL: `http://www.newsbtc.com/2016/04/17/homomorphic-encryption-and-smart-contracts-for-privacy-and-transparency/`.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. *Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based.* Cryptology ePrint Archive, Report 2013/340. `http://eprint.iacr.org/`. 2013.

[Kar72] Richard M. Karp. "Reducibility Among Combinatorial Problems". In: *New York: Plenum* (1972), pp. 85–103.

[Pap94] Christos H. Papadimitriou. *Computational Complexity.* Addison-Wesley Longman, 1994. ISBN: 978-0201530827.

[Reg05] Oded Regev. "On Lattices, Learning with Errors, Random Linear Codes, and Cryptography". In: STOC '05. Baltimore, MD, USA: ACM, 2005, pp. 84–93. ISBN: 1-58113-960-8. DOI: `10.1145/1060590.1060603`. URL: `http://doi.acm.org/10.1145/1060590.1060603`.

[Rep12] Consumer Reports. *Facebook & your privacy. Who sees the data you share on the biggest social network?* 2012. URL: `http://www.consumerreports.org/cro/magazine/2012/06/facebook-your-privacy/index.htm`.

[Sch15] Bruce Schneier. *NSA Plans for a Post-Quantum World*. 2015. URL: https://www.schneier.com/blog/archives/2015/08/nsa_plans_for_a.html.