1 Public-key encryption

Public-key encryption is a type of protocol by which Alice can send Bob secret messages over public channels without them having exchanged any secret information in advance. It has the following form:

1. **Key generation:** Bob runs a randomized key generation algorithm Gen to produce a pair of keys: A secret key SK and a public key PK. He posts his public key for anyone who is interested to use.
2. **Encryption:** To communicate a message M, Alice runs a randomized encryption algorithm $Enc(PK, M)$ and sends its output to Bob.
3. **Decryption:** Upon receiving ciphertext C, Bob runs a decryption algorithm $Dec(SK, C)$ to recover the message.

The functionality requirement stipulates that for every message M, $Dec(SK, Enc(PK, M)) = M$ with probability one.

In analogy with secret-key encryption, the simplest notion of security is that of message simulatability/indistinguishability. Message indistinguishability says that the information observed by Eve, which includes not only the encryption but also the public key, looks the same for different messages. Message simulatability says that Eve can simulate public keys and encryptions.

Definition 1. A public-key encryption scheme (Gen, Enc, Dec) is (s, ε)-message indistinguishable if for every pair of messages M and M', $(PK, Enc(PK, M))$ and $(PK, Enc(PK, M'))$ are (s, ε)-indistinguishable.

Definition 2. A public-key encryption scheme (Gen, Enc, Dec) is (s, ε)-message simulatable by size t if there exists a sampler Sim of size t whore output is (s, ε)-indistinguishable from $(PK, Enc(PK, M))$ for every M.

As for secret-key encryption, the two definitions are equivalent up to some loss in parameters.[1]

Unlike for secret-key encryption, perfectly secure schemes do not exist, regardless of key size. For Bob to be able to decrypt, for any two distinct messages M and M', the encryptions $Enc(PK, M)$ and $Enc(PK, M')$ can never be equal to one another, so the random variables $(PK, Enc(PK, M))$ and $(PK, Enc(PK, M'))$ have disjoint support. A computationally unbounded Eve can distinguish the two with full advantage.

Another difference, as their names suggest, is that Alice's secret key cannot be the same as Bob's public key: If Eve had any efficient mechanism for deriving the secret key from the public key, she would be able to decrypt messages.

We can also consider security for multiple messages. For public-key encryption security for multiple messages is equivalent to security for a single message. This means in particular that public-key encryption must be randomized.

[1]An alternative definition of simulation is that Eve can simulate encryptions given the public key, i.e. $(PK, Sim(PK))$ and $(PK, Enc(PK, M))$ should be indistinguishable. That too is equivalent.

Claim 3. *If a public-key encryption scheme is (s, ε)-message indistinguishable, then it is $(s - (q - 1)t, q\varepsilon)$-message indistinguishable for q messages, where t is the size of the encryption circuit.*

Proof. For simplicity let's take $q = 2$. If the random variables

$$X = (PK, Enc(PK, M_1), Enc(PK, M_2)) \quad \text{and} \quad X' = (PK, Enc(PK, M'_1), Enc(PK, M'_2))$$

can be distinguished by some circuit D for some messages M_1, M_2, M'_1, M'_2 with advantage 2ε, then D distinguishes one of them from $Y = (PK, Enc(PK, M'_1), Enc(PK, M_2))$ with advantage ε. Without loss of generality assume this is true for X. Then the circuit D' that on input (PK, C) outputs $D(PK, C, Enc(PK, M_2))$ distinguishes $(PK, Enc(PK, M_1))$ and $Enc(PK, M'_1))$ with the same advantage. □

2 Public-key encryption via obfuscation

As there is no perfectly secure public-key encryption, we have to start somewhere else in our search of a plausible scheme. The next best thing is to take a symmetric-key encryption scheme and try to turn it into a public-key one.

Here is one idea. Take your favorite private-key scheme (Enc, Dec). Bob chooses a secret key K for this scheme. What should the corresponding public key be? Alice ought to be able to encrypt messages using the public key. Why not make the public key be *the encryption circuit itself* with the key K hardwired into it? A circuit/program is just a piece of data; it can be encoded by a string that describes the underlying graph and the gates.

Such a scheme would be perfectly functional, but it is not at all clear why we should expect it to be secure. A naive encoding of a circuit may reveal a lot about its inner workings, potentially providing Eve forbidden information about the secret key K that was hardwired into it. To prevent this information leak, what Bob would really like to do is to "rewire" the encryption circuit in a way that hides the secret key and reveals as little as possible about the encryption circuit.

This brings us to obfuscation. What does this python2 program do?

```
def F(x):
    return map(lambda o:(map(lambda c:map(lambda l:
    o.__setslice__(l[0],l[1],l[2]),([o[2]+3,o[2]+4,[o[0]]],[0,3,[o[1],
    reduce(lambda x,o:x+o,o[:2]),o[2]+1]])),range(x)),o)[1],[[1,1,0]+
    range(x)])[0][3:][-1]
```

The code doesn't make much sense, but you can try it out and see that it works:

```
>>> F(20)
6765
>>> F(11)
89
>>> F(101)
573147844013817084101L
```

Obfuscation is the art of writing correct programs that reveal little about *how* the computation happens. To turn this art into a science, we would like to have a "compiler" that takes any piece of source code and tinkers with it in a way that preserves functionality, but makes the code unintelligible to program analysis. This is what some software companies try to do to prevent reverse-engineering of their source code.

An *obfuscator* is a circuit Obf that takes the description of a circuit C (the source code) as an input and produces the description of another circuit C' (the obfuscated code) as its output. The functionality requirement says that C and C' must compute the same function (with probability one if the obfuscator is randomized).[2]

The security definition should say something like no efficient adversary can learn much by looking at the obfuscated code. This is not quite right because one thing the adversary can do with the obfuscated circuit is run the circuit on different inputs and observe the outputs. The security requirement for obfuscation is that this is the only type of information that the adversary can gain: Whatever the adversary can learn from the circuit, it can simulate by oracle access to the circuit. This notion goes by the name of *virtual black box* (VBB) security: An obfuscated circuit should look like a "black box" that computes the correct functionality and reveals nothing more.

Definition 4. Obf is (s, ε)-virtual black box secure (with simulator size t) if for every circuit L of size s there exists an oracle circuit Sim (of size at most t) for which the distributions $L(Obf(C))$ and Sim^C are (s, ε)-indistinguishable for every circuit C of the right size.[3]

Armed with an obfuscator, we can go back to upgrading private-key encryption into public key. The public key is now an obfuscation of the private-key encryption circuit $Enc_K = Enc(K, \cdot)$ with the private key K hardwired into it, namely the key-generation algorithm samples a secret key K and outputs the pair

$$SK = K, \qquad PK = Obf(Enc_K). \tag{1}$$

To encrypt, Alice applies the obfuscated circuit PK to her message. Decryption is the same as in the private-key scheme. Since the circuits PK and Enc_K are equivalent, the public-key encryption scheme is functional as long as the private-key scheme (Enc, Dec) is.

Let us now consider an adversary who observes the public key (but nothing else). If the obfuscation is VBB-secure, this adversary has little more power than one with oracle access to Enc_K. This is very much like the chosen plaintext attack and we already know how to design schemes that are secure against it.[4]

An adversary who tries to break message simulatability observes not only the public key PK, but also an encryption $Enc(PK, M)$ of some unknown message. The definition of obfuscation no longer promises that with this extra information available, the public key circuit can be replaced by an oracle, so this argument is not quite sufficient to establish CPA-security.

There is another pressing issue: Efficient VBB-secure obfuscators do not exist. This is a theorem.

Nevertheless, there is some evidence that a construction of public-key encryption along these lines may be workable. Virtual black-box security in particular implies that the obfuscations of any two circuits C and C' that compute the same function (and are of the same size) are $(s, 2\varepsilon)$-indistinguishable. This weaker security notion that goes by the name *indistinguishability obfuscation* is potentially achievable. There are concrete proposals about how it should be implemented.

Indistinguishability obfuscators can upgrade private-key encryption schemes into public-key. The starting point is a slight modification of the CPA-secure private-key encryption from Lecture 3.

[2]This definition is only sensible when C is deterministic. When obfuscating randomized circuits we implicitly treat the randomness as part of the input.

[3]L and Sim can be assumed to output a single bit, so the (s, ε)-indistinguishability requirement can be simplified to $|\Pr[L(Obf(C)) = 1] - \Pr[Sim^C = 1]| \leq \varepsilon$. Do you see why?

[4]This adversary is in fact a bit more powerful because it also gets to choose the randomness that goes into Enc_K, but security is still not too hard to prove.

Transformation (1) turns it into a message-indistinguishable public-key encryption scheme as long as Obf produces indistinguishable obfuscations. The proof of security is not difficult, but I won't cover it because this is very far from how actual public-key encryption schemes are built in this day in age.

To give more realistic examples we will need to get into the nuts and bolts of pseudorandom generators. In our discussion of private-key encryption, the pseudorandom generator was just some box that we wired up in various ways to get the desired security properties. Public-key encryption cannot be obtained in this way, so we will have to look at how specific implementations work. We will look at two proposals: Decisional Diffie-Hellman and Learning with Errors.

3 The Decisional Diffie-Hellman assumption

The Decisional Diffie-Hellman (DDH) assumption has to do with exponentiation in finite groups. Let's start with the integers $\mathbb{Z}_p$ modulo p, where p is a very large prime number (think thousands of bits). Recall that we can do not only addition and subtraction, but also multiplication and division modulo p, as long as we do not divide by zero. If we exclude zero, we obtain the multiplicative group $\mathbb{Z}_p^*$ of size $p-1$ in which multiplication and division of any two numbers is well-defined.

Let's talk about exponentiation. Given a group element g and an integer x, the power g^x can be calculated by multiplying g with itself x times. We'll be interested in the setting where x itself is very large, on the order of p itself. There is then a much faster way to exponentiate: If x is even, first calculate g^2 then raise it to the $(x/2)$-th power recursively. If x is odd, do the same with $x-1$ and multiply the result by g. This fast exponentiation algorithm reduces the number of multiplications from x to $O(\log x)$, so the whole procedure can be implemented in time roughly quadratic in the logarithm of p, namely the number of bits it takes to write down a number in $\mathbb{Z}_p^*$.

The inverse of exponentiation is taking discrete logarithms: Given g and h, find x so that g^x equals h (if one exists). How difficult is this? While this is not known, the complexity of the best known algorithms is governed by the largest prime factor of $p-1$. So if $p-1$ were prime, finding discrete logarithms in $\mathbb{Z}_p^*$ might be exponentially harder than exponentiating.

Unfortunately p and $p-1$ cannot both be prime. (Don't ask the obvious question.) The next best thing is to have $p-1$ equal twice some prime number q. Such prime numbers p are called *safe primes*. Mathematicians believe but cannot prove that there are infinitely many of them. In any case, there are explicit safe primes of the right order of magnitude for cryptography.

The Decisional Diffie-Hellman assumption postulates that discrete logarithms are not only difficult to find, but they are pseudorandom in the following sense. Let g be an element of $\mathbb{Z}_p^*$. We'll assume that p, q, and g are publicly known.

DDH assumption for base g: The random variables (g^X, g^Y, g^{XY}) and (g^X, g^Y, g^Z) are (s, ε)-indistinguishable, where X, Y, Z are random integers modulo $p-1$.

The reason that X, Y, and Z are taken modulo $p-1$ is that exponentiation wraps around modulo the order of the group, i.e. $h^{p-1} = 1$ for all elements h of $\mathbb{Z}_p^*$.

If discrete logarithms were easy to calculate then the DDH assumption couldn't be true: A distinguisher could recover X and Y from g^X and g^Y and use this to distinguish g^{XY} from the random group element g^Z. The converse does not hold: There are choices for g for which DDH does not hold but taking discrete logarithms might still be difficult.

To explain how these arise let's take $p = 11$, which is a safe prime, and $g = 2$ (see Figure 1). The

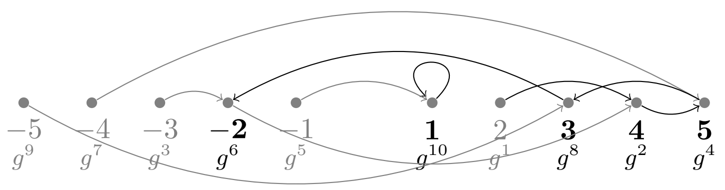

Figure 1: Powering in $\mathbb{Z}_{11}^*$. Arrows indicate squaring. The squares (in boldface) form the subgroup $\mathbb{G}$ of quadratic residues modulo 11. Each has one positive and one negative square root.

chances that g^{XY} is an even power of g is 3/4; this happens when at least one of X and Y is even. But g^Z is an even power of g only with probability 1/2. So a distinguisher that can tell apart even and odd powers of g breaks the DDH assumption. This can be done efficiently: h is an even power of g if and only if $h^5 = 1$.

This strategy works for any safe prime p as long as g generates the group $\mathbb{Z}_p^*$, namely the powers $g, g^2, \dots, g^{p-1}$ cover the whole group: To distinguish $h = g^{XY}$ from $h = g^Z$, check if $h^{(p-1)/2} = 1$.

Had we chosen $g = 4$ instead of $g = 2$ the distinguisher wouldn't have worked. In this case, g does not generate the whole group $\mathbb{Z}_p^*$, but only its subgroup $\mathbb{G} = \{1, 3, 4, 5, -2\}$ of *quadratic residues* modulo 11, namely the even powers of the old g. Now g^{XY} and g^Z are uniformly random in $\mathbb{G}$, and the DDH assumption says that (g^X, g^Y, g^{XY}) are indistinguishable from three independent random elements of $\mathbb{G}$.[5]

The DDH assumption is believed to be plausible whenever g is a quadratic residue (except 1), that is the square of a number, modulo a safe prime p. The best known attacks have complexity exponential in the cube root of p, so DDH might hold with parameters $s/\varepsilon \approx \exp(\sqrt[3]{p})$.[6] There is, however, a famous *quantum* algorithm that finds discrete logarithms in any (abelian) group in time polynomial in the logarithm of the group size. A quantum circuit of scale about p^2 would break the DDH assumption and many cryptosystems that depend on it.

4 El Gamal encryption

The DDH assumption gives an easy mechanism by which Alice and Bob can securely agree on a private key without prior interaction. This is the beautiful Diffie-Hellman key exchange:[7]

0. Alice chooses a random X.
   Bob chooses a random Y.

1. Alice sends g^X to Bob.
   Bob sends g^Y to Alice.

2. Alice raises Bob's message to the Xth power.
   Bob raises Alice's message to the Yth power.

Alice and Bob both obtain g^{XY}, which they declare to be their secret key. What Eve sees from this interaction are the values g^X and g^Y. The DDH assumption says that she gains no information

[5] X, Y, and Z can now be taken modulo q instead of $p - 1 = 2q$ because g^X etc. still cover all of $\mathbb{G}$ uniformly.
[6] There are other so-called elliptic curve groups in which DDH is thought to be even more secure.
[7] Yet another Turing award (2015).

about the secret key g^{XY} because relative to g^X and g^Y, g^{XY} can be simulated by an independent random group element g^Z.

El Gamal encryption is a slightly more elaborate application of the DDH assumption.

1. Key generation: Secret key is a random X; public key is $PK = g^X$.
2. Encryption: Message M is an element of $\mathbb{G}$. Its encryption is $(g^R, g^{XR} \cdot M)$ for random R.
3. Decryption: To decrypt (h, C), output $h^{-X} \cdot C$.

The scheme is functional: The decryption of $(g^R, g^{XR} \cdot M)$ equals $(g^R)^{-X} \cdot g^{XR} \cdot M = M$.

Claim 5. *If the (s, ε)-DDH assumption holds then El Gamal encryption is $(s - t_\times, \varepsilon)$-message simulatable in the time it takes to sample three random elements of $\mathbb{G}$, and $t_\times$ is the cost of group multiplication by a fixed element.*

Proof. The simulator outputs three random elements of $\mathbb{G}$. If message simulatability fails then $(PK, Enc(PK, M)) = (g^X, g^R, g^{XR} \cdot M)$ can be distinguished from three random elements of $\mathbb{G}$ by some D. Then the distinguisher $D'(h, i, j) = D(h, i, j \cdot M^{-1})$ breaks the DDH assumption at the cost of one extra multiplication. $\square$

One annoying aspect of this scheme is that the message space is the group of quadratic residues $\mathbb{G}$ which is not nicely structured. It would be more convenient if the messages were say integers between 1 and q. Fortunately there is an efficient conversion between the two representations

$$\mathbb{G} \leftrightarrow \{1, \dots, q\}.$$

To convert an integer into a quadratic residue, square it; to convert a quadratic residue h into an integer, calculate $h^{(q+1)/2}$ modulo p as an integer between $-q$ and q and take its absolute value. This works because $h^{(q+1)/2}$ and $-h^{(q+1)/2}$ both square to $h^{q+1} = h$, so the positive one must be equal to the integer representation of h. (See Figure 1; the elements of $\mathbb{G}$ are exactly the squares of the numbers 1 to 5. To go in the other direction, take a cube and then the absolute value.)

As a byproduct we obtain our first example of a pseudorandom generator: Assuming DDH, the function $G(X, Y) = (g^X, g^Y, g^{XY})$ viewed as a map from $\{1, \dots, q\}^2$ to $\{1, \dots, q\}^3$, is a pseudorandom generator.

5 Learning with errors

The Learning with Errors (LWE) assumption is about the hardness of solving noisy linear equations in rings. The domain is the ring $\mathbb{Z}_q$ of integers modulo q. You can think of the modulus q as a prime number if you like, but this doesn't make much of a difference.

A natural way to create a random *solvable* system of m linear equations in n variables modulo q is to choose everything at random. First choose a random solution x consisting of n numbers, (i.e., $s \sim \mathbb{Z}_q^n$) then choose a random $m \times n$ matrix A of coefficients for the equations also at random (i.e., $A \sim \mathbb{Z}_q^{m \times n}$) and output the left-hand side A and the right-hand side Ax. You will get something like this (for $m = 4, n = 3, q = 191$):

$$\begin{aligned} 143x_1 + 156x_2 + 161x_3 &= 106 \\ 156x_1 + 30x_2 + 33x_3 &= 119 \\ 150x_1 + 138x_2 + 49x_3 &= 63 \\ 127x_1 + 24x_2 + 154x_3 &= 125 \end{aligned}$$

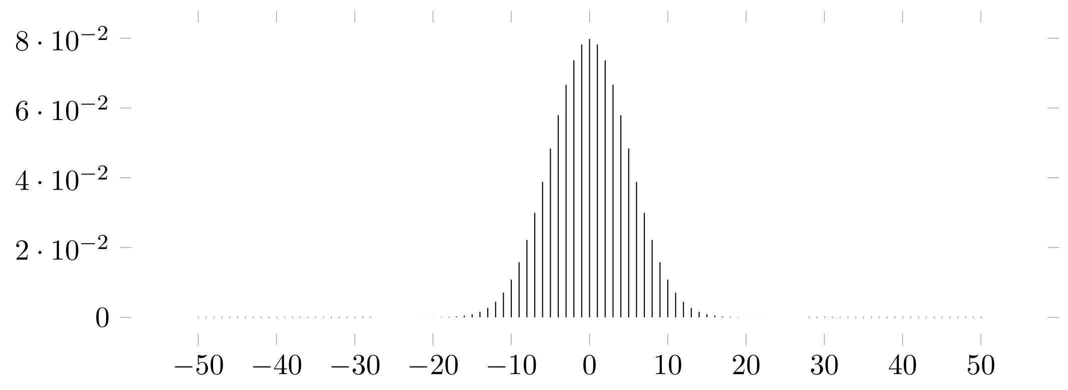

Figure 2: Discrete Gaussian noise over $\mathbb{Z}_{101}$ with $\sigma = 5$.

As you learned in high school this type of system isn't too hard to solve. In this case the solution is $x = (92, 150, 69)^\top$. Things change dramatically if a bit of noise is added to the right-hand side:

$$\begin{aligned}
143x_1 + 156x_2 + 161x_3 &\in 109 \pm 5 \\
156x_1 + 30x_2 + 33x_3 &\in 117 \pm 5 \\
150x_1 + 138x_2 + 49x_3 &\in 63 \pm 5 \\
127x_1 + 24x_2 + 154x_3 &\in 130 \pm 5.
\end{aligned}$$

There are two obvious strategies: Try out all possible solutions and try out all possible noise cancellations. The first one takes time q^n; the second one takes time b^m, where b is an upper bound on the magnitude of the noise ($b = 5$ in the above example). We will see shortly how this can be improved to about b^n. There is another attack that runs in time roughly $\exp(b^2)$ assuming at least this many equations are available. In particular, if the noise magnitude b is roughly larger than $\sqrt{n}$ there is no known attack that substantially improves brute-force search.

The search version of the Learning with Errors assumption states that recovering the secret remains hard even when the noise is additive and chosen at random from a suitable distribution ν over $\mathbb{Z}_q$.

Search-LWE assumption: There is no circuit of size s that, given A and $Ax + e$ as inputs, finds x with probability at least ε, where e is a random vector of m independent entries sampled from ν (for all m).

An appealing choice for ν is the discrete Gaussian distribution modulo q (see Figure 2). In this distribution, each integer j is sampled with probability proportional to $e^{-j^2/2\sigma^2}$ then reduced modulo q. The standard deviation parameter σ determines the typical magnitude of the noise and plays essentially the same role as b in the discussion so far.

At this point things may look very confusing with an n, an m, a q, and a σ hanging in the air. As long as $\sigma > 2\sqrt{n}$ all search-LWE assumptions with discrete Gaussian noise follow from a one-parameter assumption: The hardness of approximating the shortest vector in an n-dimensional lattice to within a factor of about $\sqrt{n}q/\sigma$. No efficient algorithms — classical or quantum — for this problem are known unless this factor is as large as about $2^{n/\log n}$.

The important aspect for us is that the noise magnitude can much smaller than the modulus q, even by factors that grow polynomially in the dimension n. A "typical" choice of parameters could be $b = n$, and $q = n^{10}$ or even $q = 2^{n/(\log n)^2}$, with n being the security parameter.

One slight annoyance with discrete Gaussian noise is that it is unbounded. Because Gaussians are so heavily tailed, the probability that a sample exceeds many standard deviations is extremely small. This can be made precise, but let's just assume that the noise distribution is b-bounded for some b that is of the same order of magnitude as the standard deviation σ.

Just like in the case of DDH, for designing public-key encryption a decisional assumption is more useful. The (decisional) LWE assumption states that the right-hand side of the noisy system of equations is indistinguishable from random.

Learning with errors (LWE) assumption: The random variables $(A, Ax+e)$ and (A, r), where $r \sim \{0,1\}^m$ is independent and uniform, are (s,ε)-indistinguishable (for all m).

In other words, the function $G(A, x, e) = (A, Ax + e)$ is a pseudorandom generator.

The LWE assumption looks stronger than the search-LWE assumption: If I can solve for x given A and $y = Ax + e$, then I can verify that all m entries of the vector $y - Ax$ are b-bounded. On the other hand, given A and r, the probability that there even exists an x for which all the entries or $r - Ax$ are b-bounded is tiny. For a fixed x, the probability that $r - Ax$ is b-bounded is the fraction of b-bounded vectors in $\mathbb{Z}_q^m$, namely $(b/q)^m$. So the probability that there even exists a b-bounded vector of the form $r - Ax$ is $2^n(b/q)^m$, which is extremely small.

Unlike in the case of DDH, the converse is also true: search-LWE implies LWE.

6 Ding-Lin key exchange

We'll need a small strengthening of the LWE assumption. LWE assumes that the secret solution s is chosen uniformly at random. The next lemma says that LWE remains secure even if the secret was chosen from the noise distribution ν. Let's call this variant shortLWE.

Lemma 6. *(s,ε)-LWE implies $(s - \mathrm{poly}(n), \varepsilon + q^{n-n'})$-shortLWE for every n'.*

The advantage of the shortLWE assumption is that the secret entries are of bounded magnitude. We can put this to use in the analysis of this key-exchange protocol. We write ν^n for n independent samples of the noise distribution.

0. Alice samples a random column vector x from ν^n.
   Bob samples a random row vector x' from ν^n.
1. Alice and Bob sample a uniformly random n by n matrix A.
2. Alice sends $Ax + e$ to Bob, where $e \sim \nu^n$ is a column vector.
   Bob sends $e' + x'A$ to Alice, where $e' \sim \nu^n$ is a row vector.
3. Alice multiplies Bob's message by x on the right.
   Bob multiplies Alice's message by x' on the left.

Alice obtains the value $(e' + x'A)x = e'x + x'Ax$. Bob obtains the value $x'(Ax + e) = x'Ax + x'e$. Since A is a random matrix, both of them are statisically very close to uniformly random elements of $\mathbb{Z}_q$.[8] But they are not the same!

This is where the shortLWE assumption kicks in: Since x, x', e, and e' are b-bounded, the dot products $x'e$ and $e'x$ have magnitude at most b^2n. If q is sufficiently larger than b^2n, Alice's and Bob's keys are still very close to one another relative to their magnitude. In Question 4 of

[8]Some error is incurred by the event $x = 0$ or $x' = 0$, but this is extremely rare.

Homework 1 you showed that this weaker type of key agreement is still sufficient for symmetric-key encryption.

How about security? Eve observes the messages $(A, Ax+e, e'+x'A)$. Her public key is $(e'+x'A)x$. Security for key exchange requires that the messages and the public key[9] should be computationally indistinguishable from a pair of independent random variables. By the LWE assumption, $(A, Ax+e, e'+x'A, (e'+x'A)x)$ is indistinguishable from $(A, Ax+e, r', r'x)$ for a random r'.

We would now like to argue that the last random variable is indistinguishable from purely random entries but it is not clear how to do that. One way to proceed is to modify the protocol so that Alice adds a little bit of extra noise to her key:

3. Alice multiplies Bob's message by x on the right and adds noise $e'' \sim \nu$ to it.

We can now complete the security proof.

Theorem 7. *Assuming (s, ε)-shortLWE, the messages exchanged by the Ding-Lin protocol together with Alice's public key are $(s-t, 2\varepsilon)$-indistinguishable from independent random strings over $\mathbb{Z}_q$, where t is the time it takes to sample $Ax+e$ on input A.*

Proof. We need to show that the random variable $(A, Ax+e, e'+x'A, (e'+x'A)x+e'')$ is $(s-t, 2\varepsilon)$-indistinguishable from independent random strings. First, this is $(s-t, \varepsilon)$-indistinguishable from $(A, Ax+e, r', r'x+e'')$ assuming (s, ε)-shortLWE, for a distinguisher between these two can be turned into one for shortLWE at an additional cost of t. But $(A, Ax+e, r', r'x+e'')$ is exactly a shortLWE instance with $n+1$ equations if we regard r' as the coefficients of the last equation, so it is $(s-t, \varepsilon)$-indistinguishable from four random strings (A, r, r', r''). □

Just like Diffie-Hellman key exchange upgrades into El Gamal encryption, the Ding-Lin protocol can be modified to give public-key encryption. You'll figure this out in Homework 2.

Proof Sketch of Lemma 6. To prove the lemma, we describe a transformation T that maps LWE equations into shortLWE equations, and maps equations with a random right-hand side into equations with a random right-hand side. So any distinguisher for shortLWE can be turned into one for LWE with the same advantage.

The LWE equations will be split into two parts: The first n or so LWE equations will be used to initialize the transformation, while the rest will be converted into shortLWE equations one by one.

Let's take the first n LWE equations (A, y) where $y = Ax + e$. Since A is an n by n matrix this system can usually be inverted so that s can be expressed as a linear function of the noise e, namely $x = A^{-1}y - A^{-1}e$. The vector e is the intended secret of the shortLWE instance.

When the next LWE equation $(a', y' = a'x + e')$ comes along we can rewrite its right-hand side in terms of e in the form $y' = a'A^{-1}y - a'A^{-1}e + e'$. If we multiply the left-hand side a' by $-A^{-1}$ and subtract $a'A^{-1}y$ from the right-hand side we get the equation $(-a'A^{-1}, -a'A^{-1}e + e')$ which has exactly the desired shortLWE form. In summary, the transformation T is given by

$$T(a', y') = (-a'A^{-1}, y' - a'A^{-1}y).$$

Continuing in this way we can convert all subsequent LWE equations into shortLWE ones. The left-hand sides of the equations are random since they are obtained by applying an invertible linear

[9]In the noisy setting the keys are not exactly identical so we'll consider Alice's key. That is the one that will later be used for symmetric-key encryption.

transformation to a random vector. The right-hand sides are exactly of the form left hand side times e plus some independent ν-distributed noise.

It remains to check that if the equations are uniformly random so is the outcome of the transformation. This is true because when y' is random and independent of a' so is $y' - a'A^{-1}y$.

One issue that we ignored is that the matrix A might not be invertible. By taking $n' > n$ equations in the initialization phase we can ensure that an invertible submatrix exists (and can be found) with probability at least $1 - q^{n-n'}$ and the same argument works. $\square$