Public-Key Encryption—
Welcome to Cryptomania!

Noah Stephens-Davidowitz

June 9, 2023

These are combined lecture notes for all of our lectures on public-key encryption. I combined them together because (1) they're really best presented together; and (2) there is a natural order to present this material in that doesn't really divide nicely into 75-minute chunks. So, these notes are written in this more natural ordering. I present the material in class in a slightly different order.

Welcome to Cryptomania!

1 Minicrypt vs. Cryptomania

In Russell Impagliazzo's famous paper introducing "Russell's five worlds" [Imp95], he defined two different worlds in which lots of cryptography is possible: Minicrypt and Cryptomania. Roughly speaking, Minicrypt is a world in which one-way functions exist, but nothing "beyond" that.[1] We saw that one-way functions imply many cryptographic primitives: PRGs, PRFs, semantically secure secret-key encryption schemes (and even CPA- and CCA-secure schemes), commitment schemes, zero-knowledge proofs for all of NP, identification schemes, MACs, and signatures! Minicrypt has plenty of fun primitives to keep a cryptographer busy. But, there's a world outside of Minicrypt. Russell called it Cryptomania, and today, we enter Cryptomania (insert drum roll/foreboding music).[2]

Here's the basic problem with Minicrypt. The secret-key encryption schemes that we've seen so far are extremely useful for two parties that share a secret key. But, we often don't have that luxury. For example, when we visit a website for the first time, we do not have a shared secret key with that website.

Of course, we still want to communicate securely in these cases. So, Alice would like some way to send encrypted messages to Bob *without* a shared secret. But, if Alice can encrypt a message without using any secret information, then so can anyone else! So, if we want something like this

[1] In Russell's other three worlds—Algorithmica, Heuristica, and Pessiland—one-way functions do not even exist, so very little cryptography is possible in these worlds.

[2] Boaz Barak defined a new world recently: Cryptofantasia. This is an even crazier world than Cryptomania. In Cryptofantasia, an extremely powerful cryptographic primitive called Indistinguishability Obfuscation exists. (Ok, as far as I know, Boaz only called it Cryptofantasia once in a blog post, and if you actually Google "Cryptofantasia," you get that one blog post and a bunch of super sketchy looking blockchain sites. Everyone actually calls this world "Obfustopia." But, clearly Cryptofantasia is the best name ever, so I call it Cryptofantasia even though nobody else does.) We won't enter Cryptofantasia in this course. It's scary there.

to work, we have to find a way to allow *anyone* to create a ciphertext that only Bob can decrypt. A reasonable person would probably find this idea implausible—maybe even impossible. (We're spoiled to have grown up in a world with public-key cryptography, so maybe we're rather jaded. But, public-key cryptography probably should surprise us.)

But, in Cryptomania, this *is* actually possible. And, as far as we know, we actually live in Cryptomania. (E.g., if "factoring is hard" by some reasonable definition, then we live in Cryptomania, as we will see.) In fact, the vast majority of internet traffic is encrypted with TLS, which is a complicated set of protocols, but is fundamentally built on public-key encryption. So, we've bet the security of the internet on the assumption that we live in Cryptomania.

But, before we even give a definition, let's devote a paragraph to thinking about what it would be like to live in a world without public-key cryptography: Minicrypt. How would Alice send a message to Bob? One can imagine various systems for accomplishing this—none satisfactory. Maybe Alice and Bob can meet in person to agree on a secret key. That solution is not so bad if Alice and Bob are good friends (although still rather tedious), but it wouldn't really work for the internet. Or maybe Alice could send Bob a secret key in the mail or via some courier service. Maybe Alice and Bob could each share a secret key with some trusted third party, and talk to each other via the third party. All of these solutions seem... not great. So, we should remember that we're lucky to live in Cryptomania (as far as we know).

Aside: Pure math is useful, and Hardy was (very very very very very) wrong

In 1940, G.H. Hardy (a very important number theorist, whom you may know because he was the mathematician who brought Ramanujan to Cambridge) wrote a famous essay called "A Mathematician's Apology". Hardy did *not* mean "apology" in the sense of saying he was sorry. He meant "apology" in a now old-fashioned sense that means something like "defense" or justification." (This old-fashioned use of the word "apology" lives on in the word "apologist," which means roughly someone who defends an idea—e.g., a capitalism apologist is someone who defends the idea of capitalism in response to people who criticize it.)

Hardy was defending himself from what he perceived as a criticism of the apparent uselessness of mathematics and, by extension, mathematicians. Specifically, he believed that the sort of mathematics that was his forte would never have any practical applications. He said, e.g., "We have concluded that the trivial mathematics is, on the whole, useful, and that the real mathematics, on the whole, is not."

Hardy believed deeply that mathematics is beautiful, and he defended himself by saying that the purpose of doing mathematics is to uncover this beauty. I agree! In fact, I agree quite strongly! But, he also believed that this was the *only* purpose of the sort of pure mathematics that he loved, and even that no other purpose would be discovered in the future. I probably would have agreed with Hardy at the time,[3] but it is amazing just how incredibly wrong he turned out to be.

Perhaps the most striking example of how wrong Hardy was is the incredible usefulness of the kind of public-key cryptography that we will see below, based purely on number theory. This is amazing both because of its extreme usefulness (it powers the internet!) and because of how thoroughly grounded it is in pure mathematics. These constructions work entirely by manipulating numbers, in much the same way that, say, Euler, Gauss, Sophie Germain, and Fermat would have done. I find it to be strange, a bit frightening, and quite beautiful that most of us now walk around

[3] In fact, even though I know that Hardy was wrong, I still have trouble truly reconciling myself with that fact.

all day with a device in our pockets that is quietly doing some number theory.

2 Public-key encryption (also known as magic)

Fortunately (and surprisingly), we probably live in Cryptomania. We know how to build public-key encryption. In Cryptomania, Bob has two keys: a public key pk and a secret key sk, generated simultaneously by some key-generation algorithm Gen. We think of the public key pk as, well, public. E.g., Bob may publish pk on his website, or he might simply send it to anyone who requests it—including adversaries. The secret key sk, on the other hand, must remain, well, secret. I.e., Bob holds on to that and does not share it with anyone—not even with Alice. This secret key is what allows him to decrypt. Using only pk, Alice (and anyone else), can compute some ciphertext $c \leftarrow \mathsf{Enc}(pk, m)$ encrypting her message m. And, Bob can use the secret key sk to decrypt it. I.e., $m = \mathsf{Dec}(sk, c)$. (The notation here suggests that Enc should be randomized and Dec should be deterministic. This is typically the case.)

Definition 2.1. *A* semantically secure public-key encryption scheme *is a triple of PPT algorithms* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *and a message space* $\mathcal{M}$ *with the following properties.*

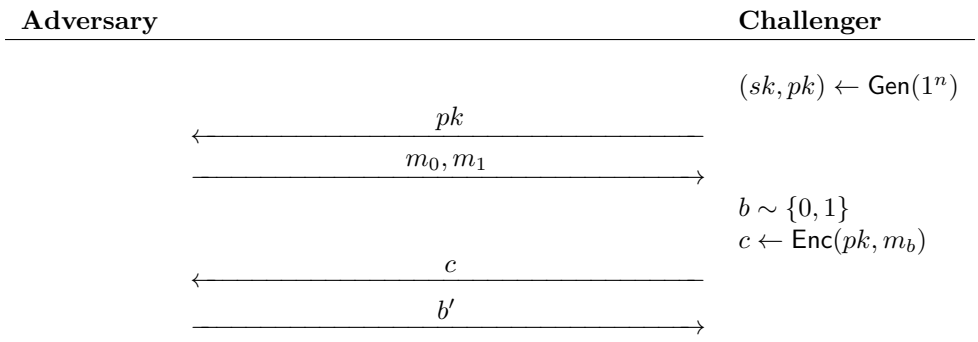
- **Correctness:** *For any message* $m \in \mathcal{M}$ *and any* $n \in \mathbb{N}$,

$$\Pr_{(sk,pk)\leftarrow\mathsf{Gen}(1^n)}[\mathsf{Dec}(sk, \mathsf{Enc}(pk, m)) = m] = 1 \ .$$

- **Semantic security:** *For any pair of PPT algorithms* $\mathcal{A}_0, \mathcal{A}_1$*, there exists a negligible* $\varepsilon(n)$ *such that for all* $n \in \mathbb{N}$,

$$\Pr_{(sk,pk)\leftarrow\mathsf{Gen}(1^n),b\sim\{0,1\}}[(\sigma, m_0, m_1) \leftarrow \mathcal{A}_0(1^n, pk); c \leftarrow \mathsf{Enc}(pk, m_b); \mathcal{A}_1(\sigma, c) = b] \leq 1/2 + \varepsilon(n) \ .$$

The definition of semantic security is a bit cumbersome as written above. We are describing an interactive game in which the adversary runs in two stages, maintaining a state σ between the two. It is far more natural to view this directly as an interactive game between an adversary and a challenger.



We say that the adversary wins this game if $b' = b$—i.e., if the adversary correctly guesses the bit b corresponding to the message m_b. The scheme is semantically secure if no PPT adversary wins this game with probability that is non-negligibly larger than $1/2$.

This idea (in a less modern form) is due to Diffie and Hellman [DH76] and Merkle [Mer78]. Intuitively, this definition solves the problem that we described above because, if such a scheme exists, Bob can simply, e.g., publish his public key on his website, and Alice can send him whatever message she likes encrypted under his public key. Welcome to Cryptomania!

2.1 Unpacking the definition

There are a few of things to notice about the above security notion. First, and most importantly, notice that the adversary is given the public key pk! This is, of course, what makes this key *public*. We saw the same thing with signatures, but here it really is quite surprising that we can achieve such a strong notion of security. The adversary has all the information that she needs to encrypt a message herself, but somehow she is not capable of reading an encrypted message.

This brings us to our second thing to notice: that a semantically secure public-key encryption algorithm Enc *must* be randomized. In particular, the adversary can compute $\mathsf{Enc}(pk, m_0)$ and $\mathsf{Enc}(pk, m_1)$ itself. If Enc were deterministic, then the adversary could simply check whether the challenge ciphertext c equals $\mathsf{Enc}(pk, m_0)$ or $\mathsf{Enc}(pk, m_1)$. So, if anything satisfies this definition, it better be randomized. In some sense, the random coins that Alice flips are the only things that she knows that the adversary Eve does not—Eve knows the public key; she knows what plaintext Alice might choose to send; she just doesn't know the random coins used to generate the ciphertext. (We saw roughly the same phenomenon for multi-message security in the secret-key setting on the homework. Here the problem with deterministic schemes is even more pronounced!)

Third, recall that in the secret-key setting, we emphasized the importance of *multi-message* semantic security. E.g., we saw that the one-time pad is secure for a single message, but completely broken for two messages. And, it took us many lectures to finally construct a scheme (based on PRFs, which we built using only one-way functions) that was secure for an arbitrary polynomial number of messages. After putting so much emphasis on multi-message security, it might seem very strange that the above definition only asks the adversary for a single pair of messages m_0, m_1! But, actually, the above definition is *equivalent* to the multi-message variant in which the adversary may choose polynomially many messages $m_{0,1}, \ldots, m_{0,\ell}$ and $m_{1,1}, \ldots, m_{1,\ell}$ and must guess the bit b given $(\mathsf{Enc}(pk, m_{b,1}), \ldots, \mathsf{Enc}(pk, m_{b,\ell}))$.

Intuitively, these definitions are equivalent for public-key encryption because, again, with access to the public key the adversary can generate encryptions herself. To prove the equivalence formally, we could use a simple hybrid argument, switching the messages $m_{b,1}, \ldots, m_{b,\ell}$ with $m_{1-b,1}, \ldots, m_{1-b,\ell}$ one at a time. We then show that an adversary that can distinguish between two hybrids can be used to break semantic security by simply computing all but one of the ciphertexts itself using the public key (and planting its own challenge ciphertext in the one remaining spot). (Some years I give this equivalence for homework.)

2.2 Cryptomania is hard to find

One-way functions seem to be pretty easy to find (though we seem to be very far from proving unconditionally that any function is one-way!). We saw that the existence of OWFs is equivalent to the existence of hard puzzles, and we even saw a universal OWF. So, if hard puzzles exist, then there is a universal secure construction of secret-key encryption (and PRGs, PRFs, ZKPs, ...).

As a result, we have a good number of candidate one-way functions with a variety of constructions. We therefore did not spend much time actually constructing one-way functions and instead

spent time on applications of one-way functions. (Of course, there is a large body of work studying how best to construct one-way functions and other Minicrypt primitives like secret-key encryption, PRGs, PRFs, etc. To a very rough approximation, much of this work is focused on coming up with constructions that are *very* efficient but still secure.)

Public-key encryption is far more difficult to construct. We sometimes say that this is because public-key cryptography requires hardness of "structured problems"—usually problems with some nice algebraic structure that we can exploit in order to build this magical primitive. After forty years of research, we have essentially only found a handful of constructions that are still believed to be secure (and a rather long list of failed constructions). Because there are so few, we will see a large fraction of them in these lectures. Most of these constructions fall into one of three paradigms. They either use trapdoor permutations, trapdoor predicates, or key agreement. We will also see lattice-based constructions, which don't obviously fall into any of these categories (although the construction that we will see is really a key-agreement-style scheme in disguise).

Trapdoor permutations

3 Public-key encryption from trapdoor permutations

Diffie and Hellman suggested a very natural primitive, called a trapdoor permutation, which they suggested using to build public-key encryption [DH76]. (Of course, since Diffie and Hellman were writing in 1976—before the era of careful formal definitions in cryptography—they did not have the formal definition below.)

Definition 3.1. *A* trapdoor function *is a family of functions $F_k : D_k \to R_k$ and PPT key-generation algorithm* Gen*, which takes as input 1^n and outputs a trapdoor τ and a key k, satisfying the following properties.*

- **Efficiently computable:** *There is a PPT algorithm $\mathcal{A}$ such that for any (τ, k) output by* Gen *and any $x \in D_k$, $\mathcal{A}(k, x) = F_k(x)$.*
- **Efficiently sampleable:** *There is a PPT algorithm $\mathcal{A}$ such that for any (τ, k) output by* Gen *and any $x \in D_k$,*[4]
$$\Pr[\mathcal{A}(k) = x] = 1/|D_k| \,.$$
(I.e., $\mathcal{A}$ outputs a random sample from D_k.)
- **Trapdoor invertible:** *There is a PPT inversion algorithm $\mathcal{I}$ such that for any (τ, k) output by* Gen *and any $x \in D_k$ with $y := F_k(x)$, $F_k(\mathcal{I}(\tau, y)) = y$. (I.e., $\mathcal{I}$ uses the trapdoor τ to find an inverse of y.)*

[4]The requirement that a trapdoor permutation be efficiently sampleable is probably best viewed as a minor technical annoyance. Of course, a function F_k isn't very useful if we cannot even efficiently find an element in its domain D_k. Here, we ask just a bit more: that we can actually efficiently sample a uniformly random element in the domain. One can be slightly more general and consider a trapdoor function to be defined together with a sampling algorithm Samp, but for the examples that we will see, this extra generality is unnecessary.

- **One way:** *For any PPT algorithm $\mathcal{A}$, there is a negligible function $\varepsilon(n)$ such that for all $n \in \mathbb{N}$,*

$$\Pr_{(\tau,k)\leftarrow \mathsf{Gen}(1^n),\ x\sim D_k}[x' \leftarrow \mathcal{A}(1^n, k, F_k(x));\ F_k(x') = F_k(x)] \leq \varepsilon(n) .$$

(I.e., F_k is a one-way function if we simply ignore the trapdoor τ.)

F_k is a trapdoor permutation *if it is a trapdoor function and for all k, $D_k = R_k$ and F_k is a bijection (i.e., if it is a trapdoor function that also happens to be a permutation). When F_k is a trapdoor permutation, we sometimes write $F_\tau^{-1}(y)$ instead of $\mathcal{I}(\tau, y)$, to emphasize that there is a unique inverse.*

Intuitively, to build a public-key encryption scheme from a trapdoor function, we should take the public key to be k and the secret key to be τ. It is not immediately obvious what to do next, though. Indeed, it is in general not known how to build public-key encryption from a trapdoor *function*. Instead, we will need trapdoor *permutations*.

Indeed, if F_k is actually a trapdoor *permutation*, then a natural idea (that does not work!) is to define $\mathsf{Enc}(pk, m) := F_k(m)$ and $\mathsf{Dec}(sk, c) := \mathcal{I}(\tau, c)$. This was the original idea behind trapdoor permutations and the original suggestion of Diffie and Hellman (and they are still sometimes used like this...) but the resulting scheme is actually *not* semantically secure, no matter which trapdoor permutation F_k is used to instantiate it. In fact, it's deterministic! To check if a ciphertext c is an encryption of m_0 or m_1, an adversary can simply compute $\mathsf{Enc}(pk, m_0)$ and $\mathsf{Enc}(pk, m_1)$ and check which one equals c. So, we have to be a bit more clever.

3.1 Using hardcore predicates of trapdoor permutations to encrypt

Notice that the definition of a trapdoor permutation only guarantees that it is hard to invert *on a random input*. So, if we want to use a trapdoor function securely, we should probably call it on a random input. I.e., our ciphertext c should probably include $F_k(x)$ for some randomly sampled $x \sim D_k$.

We might then try using x as a one-time pad, e.g., we could define $\mathsf{Enc}(pk, m) := (F_k(x), x \oplus m)$. Then, the decryption algorithm would be $\mathsf{Dec}(sk, (c_1, c_2)) = \mathcal{I}(\tau, c_1) \oplus c_2$. This is also not secure, though. To break this scheme, the adversary, given a challenge ciphertext $(c_1 = F_k(x), c_2 = x \oplus m_b)$, can simply compute $x^* := c_2 \oplus m_0$. If $F_k(x^*) = c_1$, then $m_b = m_0$. Otherwise, $m_b = m_1$.

The issue with this scheme is that a one-time pad only works when the pad is *indistinguishable from random*. While we chose x uniformly at random, if we give the adversary $y = F_k(x)$, then it is trivial for her to distinguish x from random (by simply applying F_k and seeing if the result is y)! (The above attack did exactly this.) So, we need to find some function $P(x)$ such that $P(x)$ is indistinguishable from random *even when the distinguisher knows $F_k(x)$*. Of course, this is exactly what a hardcore predicate guarantees.

So, let $P_k : D_k \to \{0,1\}$ be some hardcore predicate of F_k. I.e., given k and $y := F_k(x)$ for $x \sim D_k$, it is hard to compute $P_k(x)$ with probability non-negligibly larger than $1/2$.

While formally P_k depends on k (since its domain does), it will typically be a simple function that ignores k—such as the first-bit function. So, we will just write P. Then, here is our encryption scheme, which encrypts a single-bit message $m \in \{0,1\}$.

- $\mathsf{Gen}(1^n)$: Call $(\tau, k) \leftarrow \mathsf{Gen}_F(1^n)$ and output $(sk := \tau, pk := k)$.

- $\mathsf{Enc}(pk, m)$: Sample $x \sim D_k$ uniformly at random, and output $(c_1 := F_k(x), c_2 := P(x) \oplus m)$.
- $\mathsf{Dec}(sk, (c_1, c_2))$: Output $P(\mathcal{I}(\tau, c_1)) \oplus c_2$.

To see intuitively why this scheme is secure, simply notice that $P(x)$ is indistinguishable from random from the adversary's perspective. The formal proof is very similar to many of the proofs that we have already seen—in particular our proof that a very similar construction yielded a computationally hiding commitment scheme. We therefore leave it as an exercise.

We could now devote some time to worrying about whether such hardcore predicates exist for trapdoor permutations. Indeed, using the Goldreich-Levin theorem, one can generically convert a trapdoor permutation into one with a hardcore predicate. However, in practice, we know of very few trapdoor permutations, and in all cases there is a very simple hardcore predicate.

4 Rabin's trapdoor function (aka squaring)

So, we should be excited because we have just built our first public-key encryption scheme! However, it requires a trapdoor permutation, and we have not seen how to construct such an object yet.

So, now, let's see how to build a trapdoor function, using an absolutely beautiful construction due to Rabin [Rab79]. Unfortunately, it won't quite be a trapdoor permutation yet. That will wait until later. But, even though Rabin's function as presented is not a trapdoor permutation, (1) you will see in the homework how to make it a trapdoor permutation; and (2) it is secure assuming that factoring is hard! (The other schemes that we build will rely on more exotic assumptions.)

The function itself is very simple, though it requires a bit of number theory background to see what's going on. The key-generation function samples two random n-bit primes p and q and takes these as the trapdoor $\tau := (p, q)$. The public key is simply their product $k := N := p \cdot q$. The domain and range of Rabin's function are both $\mathbb{Z}_N^* := \{1 \leq z \leq N-1 \ : \ \gcd(z, N) = 1\}$—i.e., the set of numbers between 1 and $N-1$ that are coprime to N. We then simply define $F_N(x) := x^2 \bmod N$. (That's it!) We need to show two things: (1) that F_N is efficiently invertible given p and q; and (2) that it is one way (assuming that factoring is hard—or, more formally, that factoring $N = pq$ is hard for random n-bit primes p, q).

To see how to use the trapdoor, we need to remember the *Chinese Remainder Theorem* (CRT, which goes back at least to the 3rd century AD and a Chinese mathematician named Sunzi). CRT tells us that, for $N = pq$, $ab = c \bmod N$ if and only if $ab = c \bmod p$ and $ab = c \bmod q$. Furthermore, for any $a \in \mathbb{Z}_p^*$ and $b \in \mathbb{Z}_q^*$, there is a *unique* $c \in \mathbb{Z}_N^*$ such that $c = a \bmod p$ and $c = b \bmod q$. (The fancy way to say this is that $\mathbb{Z}_N^*$ is isomorphic to $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$ with isomorphism given by $x \mapsto (x \bmod p, x \bmod q)$.) In fact, such a c can be computed efficiently given a, b, p, q. To do so, we compute p^{-1}, the inverse of p modulo q, and q^{-1}, the inverse of q modulo p. (This notation is dangerous. Notice that there is no inverse of p or q modulo N. So, here it is crucial that these inverses are defined modulo q and p respectively.) We then set $c := qq^{-1}a + pp^{-1}b \bmod N$. (Notice that in this funny notation, it is *not* the case that $qq^{-1} = 1 \bmod N$. Equivalently, we can compute $c_p := q^{-1}q \bmod N$ such that $c_p = 1 \bmod p$ and $c_p = 0 \bmod q$ and $c_q := p^{-1}p \bmod N$ such that $c_q = 0 \bmod p$ and $c_q = 1 \bmod q$ and then take $c := ac_p + bc_q$.) It is easy to see that $c = a \bmod p$ and $c = b \bmod q$. (Notice that we have actually just stumbled into a proof of CRT!)

We will also introduce some notation. We sometimes simply write (a, b) or $(a \bmod p, b \bmod q)$ to represent the number $c \in \mathbb{Z}_N^*$ such that $c = a \bmod p$ and $c = b \bmod q$. This is sometimes called

the *CRT representation* of c. We write $\mathbb{QR}_N := \{x^2 \ : \ x \in \mathbb{Z}_N^*\}$ for the set of *quadratic residues* in $\mathbb{Z}_N^*$.

In order to compute a square root of $y \in \mathbb{QR}_N$, we first compute a square root x_p of y modulo p and then compute a square root x_q of y modulo q. (Recall that this can be done efficiently. For example, if $p, q = 3 \bmod 4$, then we can simply take $x_p = y^{(p+1)/4} \bmod p$ and $x_q = y^{(q+1)/4} \bmod q$. It's good to convince yourself that this works, using Fermat's little theorem. For the general case, see Appendix A.) Then, we use the procedure described above to find the unique $x \in \mathbb{Z}_N^*$ such that $x = x_p \bmod p$ and $x = x_q \bmod q$. The CRT implies that $x^2 = y \bmod N$. So, this procedure allows us to use the factors (p, q) for N to compute square roots of elements in $\mathbb{QR}_N$!

But, notice that our choices here were not unique. $y \in \mathbb{QR}_N$ has exactly *two* square roots modulo p, $\pm x_p \bmod p$, and y has two exactly square roots modulo q as well, $\pm x_q \bmod q$. Therefore, there are actually exactly *four* square roots of every square in $\mathbb{Z}_N^*$, corresponding to the four pairs $(\pm x_p, \pm x_q)$.

So, we can invert Rabin's function, but it is *not* a one-way permutation. It is actually a four-to-one function. In the next problem set, you will fix this. But, for now, let's content ourselves with building a trapdoor *function*. (In fact, a four-to-one trapdoor function is already enough to build public-key encryption, though we will not bother to show this.)

Now, in order to prove that F_N is one way, we will need to make some computational assumption. (If we could prove this unconditionally, we would in particular prove that $\mathsf{P} \neq \mathsf{NP}$, which we don't plan to do in this class.) As it turns out, this function is one way if and only if it is hard to factor N. (I.e., if and only if no PPT algorithm can factor a number N sampled as above with non-negligible probability.) This is quite nice because it means that we can build a trapdoor function from an assumption that we are very comfortable with, the hardness of factoring.

To see this, we show how to use an inverter for F_N to factor N, i.e., we show that if F_N is not one way and therefore that there exists an efficient algorithm that inverts F_N with non-negligible probability, then there exists an efficient algorithm that factors N with non-negligible probability. (Below is an example of the kind of proof sketch with lots of explanations and intuition that is great for lectures notes that are meant to explain what is happening, but not great for homework or research papers, which are meant to be easily checkable :).)

To that end, we make the following simple observation (which is at the heart of most modern factoring algorithms). Suppose that we know $x_1, x_2 \in \mathbb{Z}_N^*$ such that

$$x_1^2 = x_2^2 \bmod N \ .$$

Then, we can write $x_1^2 - x_2^2 = (x_1 + x_2)(x_1 - x2) = 0 \bmod N$. In other words, N divides $(x_1 + x_2)(x_1 - x2)$. Since p is prime, p must divide either $x_1 + x_2$ or $x_1 - x_2$, and similarly q must divide one of them. Of course, we might have $x_1 = \pm x_2 \bmod N$, in which case this is not particularly interesting. In particular, we might have that $x_1 - x_2 = 0$ or $x_1 + x_2 = N$ or something like this. But, let's suppose that $x_1 \neq \pm x_2 \bmod N$. Indeed, we saw above that there are four square roots of x_1^2, so for any fixed $x_1 \in \mathbb{Z}_N^*$, there are always exactly two choices of x_2 such that $x_2^2 = x_1^2 \bmod N$ but $x_2 \neq \pm x_1 \bmod N$. If p and q *both* divide, say, $x_1 - x_2$, then we have $x_1 = x_2 \bmod N$, a contradiction. Similarly, if p and q both divide $x_1 + x_2$, then $x_1 = -x_2 \bmod N$, also a contradiction. So, under the assumption that $x_1 \neq \pm x_2 \bmod N$, we must have either $\gcd(x_1 - x_2, N) = p$ and $\gcd(x_1 + x_2, N) = q$ or vice versa. Either way, we can use x_1 and x_2 to factor N efficiently!

So, in order to factor, it suffices to find $x_1, x_2 \in \mathbb{Z}_N^*$ such that (1) $x_1^2 = x_2^2 \bmod N$ and (2) $x_1 \neq \pm x_2 \bmod N$. The fancy way to say this is that x_1, x_2 form a *non-trivial pair of square roots*

of $y := x_1^2 \bmod N$. (A pair x_1, x_2 of square roots is *trivial* if $x_1 = \pm x_2 \bmod N$.) I call them "funny square roots" because they're kind of silly if you think about it. We want to show how to find such a funny pair of square roots given an inverter for F_N. But, an inverter for F_N only gives us *one* square root of $y \in \mathbb{QR}_N$. How do we find a funny *pair*?

To do so, we simply sample a random $x_1 \sim \mathbb{Z}_N^*$ and compute $y := x_1^2 \bmod N$. We then call our inverter on y, and receive as output x_2. If $x_2^2 = x_1^2 \bmod N$ (of course, sometimes our inverter might fail, but let's condition on it succeeding), then I claim that $x_2 \neq \pm x_1 \bmod N$ with probability exactly $1/2$. To see this, notice that, since we sampled x_1 randomly, x_1 is equally likely to be any of the four square roots of y. But, the inverter only sees y. So, from the inverter's perspective, x_1 is a uniformly random square root, and no matter which other square root x_2 it chooses, with probability $1/2$ we will have $x_2 \neq \pm x_1 \bmod N$.

4.1 A formal proof of security for Rabin's function

Ok, let's do a formal proof. First, we need our formal computational assumption.

Assumption 4.1 ("Factoring is hard")**.** *For any PPT $\mathcal{A}$ there is a negligible $\varepsilon(n)$ such that for all $n \in \mathbb{N}$,*

$$\Pr_{p,q \sim \mathbb{P}_N}[(p', q') \leftarrow \mathcal{A}(N) \ : \ \{p', q'\} = \{p, q\}] \leq \varepsilon(n) \ ,$$

where $\mathbb{P}_n$ is the set of n-bit primes.

Notice that the above assumption is necessarily an *average-case* assumption. In particular, this is much stronger than simply saying that it is hard to factor in the worst case. E.g., the above assumption might be false, but it might still be hard to factor *some* numbers, even if it is not hard to factor the product of uniformly random n-bit primes.

Theorem 4.1. *If factoring is hard, then Rabin's trapdoor function is hard to invert.*

Proof. Suppose Rabin's function F_N is *not* hard to invert. Then, there exists an efficient adversary $\mathcal{A}$ such that $\mathcal{A}$ such that

$$\varepsilon(n) := \Pr_{p,q \sim \mathbb{P}_N, x_1 \sim \mathbb{Z}_N^*}[x_2 \leftarrow \mathcal{A}(N, x_1^2 \bmod N) \ : \ x_1^2 = x_2^2 \bmod N]$$

is non-negligible.

We construct a factoring algorithm $\mathcal{B}$ as follows. $\mathcal{B}$ takes as input an integer N, where $N := pq$ for (unknown) uniformly random n-bit primes $p, q \sim \mathbb{P}_n$. It then samples $x_1 \sim \mathbb{Z}_N^*$, sets $y := x_1^2 \bmod N$, and computes $x_2 \leftarrow \mathcal{A}(N, y)$. Finally, it outputs $p' := \gcd(x_1 + x_2, N)$ and $q' := \gcd(x_1 - x_2, N)$.

Notice that the input to $\mathcal{A}$ is distributed identically to its input in the security game against F_N. Therefore, we have

$$\Pr[x_1^2 = x_2^2 \bmod N] = \varepsilon(n) \ .$$

Furthermore, as we noticed above, we have $\{p', q'\} = \{p, q\}$ whenever $x_1^2 = x_2^2 \bmod N$ *and* $x_1 \neq \pm x_2 \bmod N$ (i.e., whenever x_1, x_2 form a funny pair of square roots). The final thing to notice is simply that, *conditioned on* $x_1^2 = x_2^2 \bmod N$, the probability that $x_1 \neq \pm x_2 \bmod N$ is exactly $1/2$,

$$\Pr[x_1 \neq \pm x_2 \bmod N \mid x_1^2 = x_2^2 \bmod N] = 1/2 \ .$$

The reason for this is that the input y to $\mathcal{A}$ is independent of the specific choice of square root x_1.[5] Therefore,

$$\begin{aligned}\Pr[\{p',q'\}=\{p,q\}] &\geq \Pr[x_1^2 = x_2^2 \bmod N \text{ and } x_1 \neq \pm x_2 \bmod N] \\ &= \Pr[x_1^2 = x_2^2 \bmod N]\Pr[x_1 \neq \pm x_2 \bmod N \mid x_1^2 = x_2^2 \bmod N] \\ &= \varepsilon(n)/2\ ,\end{aligned}$$

which is non-negligible, as needed. □

So, Rabin's function is in fact a trapdoor function, assuming that factoring is hard (for numbers sampled from the appropriate distribution). Unfortunately, it is not a permutation. Below, we will see RSA, a true trapdoor permutation (whose security relies on a stronger assumption). We will also see different ways to construct public-key encryption!

5 RSA

We now present the RSA function, due to Rivest, Shamir, and Adleman [RSA78]. I like to think of it as a modification of Rabin's function that converts it into a trapdoor permutation, rather than "just" a trapdoor function. (You will see a rather different way to modify Rabin's function in the homework.) Historically, RSA was actually discovered *before* Rabin's function, so calling RSA a modification of Rabin's function is a little strange. Still, logically RSA is probably better viewed as a successor to Rabin's function—even if historically RSA came first. (Let me know if you agree or disagree!)

So, let's recall Rabin's function. The trapdoor consists of two random n-bit primes p and q, and the public key is $N := pq$. The function itself is then $G_N(x) := x^2 \bmod N$ for $x \in \mathbb{Z}_N^*$. (I'm writing G here and not F because I want to use F for RSA.) And, the "problem" with this function G_N is that it is four-to-one—i.e., there are always exactly four different elements $x_1, x_2, x_3, x_4 \in \mathbb{Z}_N^*$ such that $G_N(x_i) = G_N(x_j)$. This results from the fact that every residue $y = x^2 \bmod p$ in $\mathbb{Z}_p^*$ has *two square roots* $\pm x \bmod p$, and the same is true modulo q. By the Chinese Remainder Theorem, this yields four different square roots, which are $(\pm x \bmod p, \pm x \bmod q)$ in the CRT representation.

There is a natural solution to this, though. Instead of taking $G_N(x) := x^2 \bmod N$, what about $F_{N,e}(x) := x^e \bmod N$ for some exponent e? This is exactly the RSA function! By the CRT, this will give a permutation if and only if the maps $x \mapsto x^e \bmod p$ and $x \mapsto x^e \bmod q$ are both themselves permutations with domains $\mathbb{Z}_p^*$ and $\mathbb{Z}_q^*$ respectively.

So, when is the map $x \mapsto x^e \bmod p$ a permutation over $\mathbb{Z}_p^*$? Well, this is a permutation if and only if e is coprime to $p-1$. To see this, recall that if e is coprime to $p-1$, then there exists an *inverse* $d \in \mathbb{Z}_{p-1}^*$ *of* e *modulo* $p-1$, i.e., $de = 1 + k(p-1)$ for some integer k. So, suppose that $x^e = y^e \bmod p$ for some $x, y \in \mathbb{Z}_p^*$. Then, of course, we also have $x^{de} = y^{de} \bmod p$.

[5] E.g., consider the thought experiment in which we first sample a uniformly random quadratic residue $y \in \mathbb{QR}_N$, then we set $x_2 \leftarrow \mathcal{A}(N,y)$, and then we sample a uniformly random square root $x_1 \sim \{x \in \mathbb{Z}_N^* \ : \ x^2 = y \bmod N\}$. Clearly, in this though experiment,

$$\Pr[x_1 \neq \pm x_2 \bmod N \mid x_1^2 = x_2^2 \bmod N] = 1/2\ ,$$

since there are exactly two choices for x_1 where $x_1 = \pm x_2 \bmod N$ and two choices where $x_1 \neq \pm x_2 \bmod N$. However, this thought experiment actually produces x_1 and x_2 that are distributed identically to those generated in the reduction.

But, $x^{de} = x^{1+k(p-1)} = x \bmod p$, since by Fermat's little theorem $x^{p-1} = 1 \bmod p$. Similarly, $y^{de} = y \bmod p$. So, we conclude that $x = y \bmod p$, i.e., we conclude that the map $x \mapsto x^e \bmod p$ is a permutation over $\mathbb{Z}_p^*$.

Rabin's function with $e = 2$ failed to give a permutation specifically because $p-1$ and $q-1$ are always even (for odd primes p and q). But, if we choose e to be coprime to both $p-1$ and $q-1$, then we do in fact obtain a permutation. In practice, for efficiency reasons, it is common to take $e = 3$, which yields a permutation if and only if $p, q = 2 \bmod 3$. (Actually, there are complicated considerations that go into choosing e, and there are some issues with choosing e to be very small. Instead, people often $e = 2^{16} + 1$. Perhaps you can see why such an e would be a reasonable choice for efficiency, given that exponentiation is typically computed via repeated squaring.)

Furthermore, the function $x \mapsto x^e \bmod p$ is easy to invert over $\mathbb{Z}_p^*$ when $\gcd(p-1, e) = 1$ (if you know p and e!)—even easier than finding square roots. To invert this function, we simply compute $d \in \mathbb{Z}_{p-1}^*$, the inverse of e modulo $p-1$, so that $ed = 1 + k(p-1)$ for some integer k. (Recall that we can efficiently compute d given e and p using the Euclidean algorithm.) Then, by definition, $x^{ed} = x \cdot x^{1+k(p-1)} = x \bmod p$, since $x^{p-1} = 1 \bmod p$ for all $x \in \mathbb{Z}_p^*$.

More generally, to invert the map $F_{N,e}$ *given the prime factors* p, q *of* N, where $\gcd(e, \phi(N)) = 1$, we can choose $d \in \mathbb{Z}_{\phi(N)}^*$ to be the inverse of e in the group $\mathbb{Z}_{\phi(N)}^*$. (Recall that $\phi(N)$ is the order of the group $\mathbb{Z}_N^*$. In particular, for $N = pq$, $\phi(N) = (p-1)(q-1)$.) Given this trapdoor d and $y = F_{N,e}(x) = x^e \bmod N$, we can efficiently compute the inverse $x = y^d \bmod N$.

5.1 The RSA trapdoor permutation, formally

Given the legwork we've done above, the RSA trapdoor permutation is relatively straightforward. There are many design decisions, such as how to choose the primes p, q (should they be congruent to 2 mod 3? safe primes?) and how to choose e (should e random, or three, or $2^{16}+1$?) that we will simply leave unspecified. So, this is not really a description of a specific trapdoor permutation, but rather a framework for building trapdoor permutations, where the specific trapdoor permutation depend on the details of how one chooses p, q, and e. Specifically, the Gen algorithm below is not fully specified—we simply wrote "choose p, q, e" without specifying how they should be chosen.

- $\mathsf{Gen}(1)^n$: Choose two n-bit primes p, q, and $e \in \mathbb{Z}_{(p-1)(q-1)}^*$. The key is then $(N := pq, e)$, and the trapdoor is $(N, d := e^{-1} \bmod (p-1)(q-1))$.[6]
- $F_{N,e}(x) := x^e \bmod N$ for $x \in \mathbb{Z}_N^*$
- $F_{N,d}^{-1}(y) := y^d \bmod N$ for $y \in \mathbb{Z}_N^*$.

That's it! This is clearly efficiently computable, efficiently sampleable, and trapdoor invertible. But, is it one way? I.e., is it hard to invert? The answer, of course, is that we don't know—since we don't know how to prove even that one-way functions exist. However, in the case of Rabin's function, we were able to prove that inverting the function is as hard as factoring N (i.e., as hard as factoring a number sampled from the same distribution as the key).

For RSA, no such result is known (though, there is a very interesting partial result—from one of the first papers written by my good friend and constant collaborator Divesh Aggarwal—which shows that any "natural" algorithm for breaking RSA yields a factoring algorithm [AM09]). (The

[6] We could also take p, q as our trapdoor. But, it is a bit cleaner to just use d.

function that you construct in the homework will be a trapdoor permutation that is provably secure if factoring is hard.)

Instead, the best that we know how to do is simply to conjecture directly that the RSA function is one way. Formally, we conjecture that no polynomial-time algorithm given N, e, and $y := x^e \bmod N$, with N, e sampled as above and $x \sim \mathbb{Z}_N^*$, can find x with non-negligible probability. This is known as the RSA Assumption.

Assumption 5.1 (RSA)**.** *For any PPT $\mathcal{A}$ there is a negligible $\varepsilon(n)$ such that for all $n \in \mathbb{N}$,*

$$\Pr_{N,e \leftarrow \mathsf{Gen}(1^n),\ x \sim \mathbb{Z}_N^*}[\mathcal{A}(N, e, x^e \bmod N) = x] \leq \varepsilon(n) \ .$$

This is, of course, far less natural than the assumption that factoring is hard. And, since the RSA function can in fact be inverted given the factorization of N, the RSA Assumption is *stronger* than the assumption that factoring N is hard. I.e., if factoring is not hard, then the RSA assumption definitely fails. However, it could potentially be the case that factoring is hard but RSA can still be broken without factoring N.

Fortunately, the fastest attacks that we know of on RSA work by simply factoring the modulus N. (At least, this is true when it is implemented carefully and correctly. There are *many* bad ways to implement RSA that yield much faster attacks.) So, as far as we know, there is no faster way to break RSA than to factor the modulus.

5.2 An RSA-based encryption scheme

For completeness, we present a full encryption scheme based on RSA. (It is exactly the same as the scheme from trapdoor permutations described above, simply implemented using RSA.)

The original proposed RSA encryption scheme worked with plaintext space $m \in \mathbb{Z}_N^*$ and defined $\mathsf{Enc}((N, e), m) := m^e \bmod N$. However, as we saw in the previous lecture, this is *not* actually secure, since it is deterministic. (Vinod Vaikuntanathan likes to say to his cryptography class something to the effect of "if there is one thing that you should take away from this class, it's that $\mathsf{Enc}((N, e), m) := m^e \bmod N$ is *not* a secure encryption scheme.")

There are many ways to convert this into a secure encryption scheme. But, for our purposes, the simple predicate-based strategy for encrypting a single bit that we discussed above is sufficient. So, we need a hardcore predicate $P_{N,e} : \mathbb{Z}_N^* \to \{0, 1\}$ for the RSA function $F_{N,e}$. One possibility would be to use the Goldreich-Levin theorem to get this hardcore predicate. But, as it happens, the least-significant bit $P(x) := x \bmod 2$ is a hardcore predicate for RSA.[7] (Formally, if a PPT adversary can guess the least-significant bit of x with non-negligible advantage given x^e, then there is a PPT adversary that can break the RSA Assumption with non-negligible probability.) So, here is our first concrete example of a public-key encryption scheme, which is provably secure assuming that the RSA Assumption holds.

- $\mathsf{Gen}(1)^n$: Choose two n-bit primes p, q, and $e \in \mathbb{Z}_{(p-1)(q-1)}^*$. Output $sk = (N := pq, d := e^{-1} \bmod (p-1)(q-1))$ and $pk := (N, e)$.
- $\mathsf{Enc}((N, e), m \in \{0, 1\})$: Sample $x \sim \mathbb{Z}_N^*$. Output $(y := x^e \bmod N, b := m \oplus (x \bmod 2))$.

[7] Here, by $x \bmod 2$ for $x \in \mathbb{Z}_N^*$, we literally just mean the parity of x when, e.g., interpreted as an integer between 0 and $N - 1$. It's worth noting here that N is odd, so, e.g., x and $x^e \bmod N$ do not necessarily have the same parity.

- $\mathsf{Dec}((N,d),(y,b))$: Compute $x := y^d \bmod N$ and output $b \oplus (x \bmod 2)$.

That's it. It's surprisingly simple, right?!

Aside: Using such a scheme in practice (or not...)

Of course, the above scheme is horribly impractical, and would never be used as described to power the internet. To make the scheme secure against the best known attacks (as of this writing), we must take n to be in the thousands, so that N is thousands of bits long. Since the fastest known algorithm for modular exponentiation runs in time roughly $n^2 \log n$ (and in practice is significantly slower), this means that the decryption algorithm takes millions of CPU cycles. If one used this to encrypt and decrypt billions of bits, one bit at a time, it would require quadrillions of CPU cycles, which might take my laptop weeks. So, not acceptable! (I'm not a practitioner, so I'm being very hand-wavy here. I specified the decryption algorithm, because the encryption algorithm often takes $e = 3$ or $e = 2^{16} + 1$, which makes it faster by a factor of roughly n. And, frankly, I have no idea what a "CPU cycle" actually is, so take what I just wrote with a huge grain of salt.)

Of course, this can be improved significantly by figuring out how to encrypt more than one bit at a time. Indeed, there are now many tricks to do so using RSA, which are known as *padding schemes*. However, even with the best known padding schemes, RSA is still a pretty inefficient way to communicate if you want to have a long conversation. In practice, my understanding is that RSA encryption is now not used nearly as much as the scheme that we will see in Section 9 (which can be made very efficient by using some fancy group theory).

And, even then, one does not typically use public-key encryption directly to communicate. Instead, one uses ideas like those in Section 15 to use just *one* public-key ciphertext in order to establish a shared secret key. One can then communicate using secret-key encryption from then on, which is much much faster in practice.

Trapdoor predicates

6 Trapdoor predicates

You might have noticed that our public-key encryption scheme from trapdoor permutations seems not to use the full power of the trapdoor permutation. In particular, we had the decryption algorithm compute the *entire* preimage x of $F_k(x)$, but then we only used a single bit $P(x)$. E.g., for RSA, the decryption algorithm went through all of the trouble of computing $x = y^d \bmod N$, and then only used the single bit $x \bmod 2$. This seems wasteful, right? :)

Trapdoor predicates simply cut out the middle man. I.e., a trapdoor predicate is just explicitly some predicate that is easy to compute with a trapdoor but hard without it. We give the formal definition below, but we will then see a very elegant example that is perhaps more enlightening than the definition. (Admittedly, our first example will not be secure...)

Definition 6.1. *A* trapdoor predicate *is a family of predicates* $P_k : D_k \to \{0,1\}$ *and a PPT key-generation algorithm* Gen *satisfying the following properties.*

- **Efficiently sampleable:** *There is a PPT algorithm* $\mathcal{A}$ *such that for any key* k *and bit* b*,*

$\mathcal{A}(k,b)$ is a uniformly random element $x \in D_k$ subject to the constraint that $P_k(x) = b$. I.e., $\mathcal{A}(k,b)$ is uniformly random over the set $\{x \in D_k \ : \ P_k(x) = b\}$.

- **Trapdoor computable:** *There is a PPT algorithm $\mathcal{I}$ such that*

$$\Pr_{(\tau,k)\leftarrow\mathsf{Gen}(1^n),x\sim D_k}[\mathcal{I}(\tau,x) = P_k(x)] = 1 \ .$$

  (I.e., $\mathcal{I}$ uses the trapdoor τ to compute the predicate $P_k(x)$.)

- **Hard to predict:** *For any PPT algorithm $\mathcal{A}$, there is a negligible function $\varepsilon(n)$ such that for all $n \in \mathbb{N}$,*

$$\Pr_{(\tau,k)\leftarrow\mathsf{Gen}(1^n),\ x\sim D_k}[\mathcal{A}(1^n,k,x) = P_k(x)] \leq 1/2 + \varepsilon(n) \ .$$

  (I.e., $P_k(x)$ is hard to predict given x and k.)

Intuitively, a trapdoor predicate is just a predicate P_k that is hard to compute without the trapdoor τ but easy to compute with the trapdoor τ.

To build public-key encryption from such a scheme, we use the same idea as before. I.e., a ciphertext encrypting the plaintext bit $b \in \{0,1\}$ is simply x sampled uniformly at random subject to the constraint that $P_k(x) = b$. (Notice that we need the fancier notion of efficient sampleability in order to use this.)

Our example of a trapdoor predicate will be far nicer than the actual definition (at least, the example will be quite nice before we worry about some technicalities), so let's jump right in.

7 Quadratic residuosity—first attempt and the Legendre symbol

Remember that we write $\mathbb{QR}_N := \{x^2 \bmod N \ : \ x \in \mathbb{Z}_N^*\}$ for the set of *quadratic residues* mod N. We now also write $\overline{\mathbb{QR}}_N := \{x \in \mathbb{Z}_N^* \ : \ x \notin \mathbb{QR}_N\}$ for the set of non-residues.

Here's the rough idea behind the Goldwasser-Micali trapdoor predicate. As in all of our examples so far, our trapdoor will be $\tau = (p,q)$, two random n-bit primes. Our public key will be $N := pq$ *and* a uniformly random non-residue $y \in \overline{\mathbb{QR}}_N$ (so, the full key is $k = (N,y)$). In other words, y is a random element in $\mathbb{Z}_N^*$ that is *not* a square modulo N. Our domain D_N is simply $\mathbb{Z}_N^*$, and $P_N(x)$ is the predicate equal to 1 if $x \in \mathbb{QR}_N$ and zero if $x \in \overline{\mathbb{QR}}_N$. I.e.,

$$P_N(x) := \begin{cases} 1 & x \in \mathbb{QR}_N \\ 0 & x \in \overline{\mathbb{QR}}_N \end{cases} \ .$$

(You might see an issue with the probability that $P_N(x)$ equals one for random x, or perhaps you know what the Jacobi symbol is and see a larger issue. Let's not worry about this for now. We will change some of the details above to fix this soon.)

Notice that, given just N, we can efficiently sample a uniformly random element from $\mathbb{QR}_N$, by simply sampling $x \sim \mathbb{Z}_N^*$ and outputting $x^2 \bmod N$. Furthermore, given both y and N, we can efficiently sample an element in $\overline{\mathbb{QR}}_N$ by sampling $x \sim \mathbb{Z}_N^*$ and outputting $yx^2 \bmod N$. Here, we are using the fact that if $y_1 \in \overline{\mathbb{QR}}_N$ and $y_2 \in \mathbb{QR}_N$, then $y_1 y_2 \in \overline{\mathbb{QR}}_N$. (Notice that it is not at all obvious how we would generate an element in $\overline{\mathbb{QR}}_N$ if we did not have the element y. As we will see, though, the resulting element is *not* a uniformly random element from $\overline{\mathbb{QR}}_N$.)

As far as we know, it is hard to determine whether x is a square modulo some composite integer N. But, it is easy to determine this modulo a *prime* integer p. (In fact, we saw in Section 4 that it is even easy to compute square roots modulo primes p.) There is actually a fancy name for this. It's called the *Legendre symbol*, and it's written

$$\left(\frac{x}{p}\right) := \begin{cases} 1 & x \in \mathbb{QR}_p \\ -1 & x \in \overline{\mathbb{QR}}_p \end{cases} .$$

(Ok... it's actually typically written $\left(\frac{x}{p}\right)$, but this is terrible notation since it looks exactly like it just means x/p. It's unclear whether my notation is any better, but at least it doesn't look like a fraction...) There's actually a very nice way to compute the Legendre symbol; we have $\left(\frac{x}{p}\right) = x^{(p-1)/2} \bmod p$, which you saw in the homework.

Furthermore, if $N = pq$ for primes p and q, then we have already seen that x is a square if and only if $\left(\frac{x}{p}\right) = \left(\frac{x}{q}\right) = 1$ (i.e., x is a square modulo $N = pq$ if and only if it is a square modulo p and a square modulo q). So, we can use our trapdoor (p, q) to compute our predicate by first computing $\left(\frac{x}{p}\right) = x^{(p-1)/2} \bmod p$ and $\left(\frac{x}{q}\right) = x^{(q-1)/2} \bmod q$, and then simply checking if they are both equal to one.

Putting all of this together, we get a *correct* public-key encryption scheme. To encrypt a bit $m \in \{0, 1\}$ using only the public key (N, y), we sample $x \sim \mathbb{Z}_N^*$ and output $c := y^m x^2 \bmod N$. (This is just clever notation to say "output $yx^2 \bmod N$ if $m = 1$ and $x^2 \bmod N$ if $m = 0$.") To decrypt, we simply check if $c \in \mathbb{QR}_N$, which can be done efficiently using the trapdoor (p, q). If yes, then $m = 0$. Otherwise, $m = 1$.

But, is this scheme secure? It turns out that the answer is no. Certainly, as described this is not a trapdoor predicate on all of $\mathbb{Z}_N^*$ for a very simple reason: there are fewer quadratic residues than non-residues! Remember that every $x \in \mathbb{QR}$ has exactly *four* square roots in $\mathbb{Z}_N^*$. This means that only $1/4$ of the elements in $\mathbb{Z}_N^*$ are residues. So, if you give me a random element, it is easy to guess whether or not it's a square with probability better than $1/2$: I can just always guess that it's not a square! This silly adversary will be right with probability $3/4$.

This silly adversary does not yield an attack on the encryption scheme itself, since it behaves in exactly the same manner on encryptions of zero and encryptions of one. But, we will see a more sophisticated attack in the next section.

7.1 The Jacobi symbol

To truly understand the attack on the above scheme, we will need to think a bit more carefully about ciphertexts $c := yx^2 \bmod N$ that are encryptions of one. Remember that y part of the public key, so we will think of it as fixed for now. And, remember how our decryption algorithm works: it checks if c is a square mod p and if c is a square mod q and outputs zero if and only if it is *both*. But, notice that our decryption algorithm could potentially encounter four different situations: c might be a square mod both p and q, in which case $c \in \mathbb{QR}$; but c could also be a square mod p but not q, a square mod q but not p, or not a square mod either. We can encode this succinctly via the Legendre symbol: the four possibilities are

$$\left(\left(\frac{c}{p}\right), \left(\frac{c}{q}\right)\right) = (\pm 1, \pm 1) .$$

Notice that both $\left(\frac{c}{p}\right)$ and $\left(\frac{c}{q}\right)$ are unchanged if we multiply c by some square $(x')^2 \bmod N$. So, our ciphertexts of one, $c := yx^2 \bmod N$ will always have

$$\left(\left(\frac{c}{p}\right), \left(\frac{c}{q}\right)\right) = \left(\left(\frac{y}{p}\right), \left(\frac{y}{q}\right)\right) ,$$

regardless of our choice of x. In particular, ciphertexts of 1 are *not* uniformly random non-residues c. Instead, they are special non-residues c with the property that $\left(\frac{c}{p}\right) = \left(\frac{y}{p}\right)$ and $\left(\frac{c}{q}\right) = \left(\frac{y}{q}\right)$. (Of course, ciphertexts $c' := x^2 \bmod N$ of zero always satisfy

$$\left(\left(\frac{c'}{p}\right), \left(\frac{c'}{q}\right)\right) = (1, 1) ,$$

by definition.) This seems a bit suspicious. We'd intended our ciphertexts of one to be uniformly random elements from $\overline{\mathbb{QR}}_N$, but they're actually not. Instead, they are uniformly random in some smaller set: the set of all points whose Legendre symbol matches y modulo both p and q.

As it happens (and this is *not* obvious), though it is believed to be difficult to determine whether an element c is a square modulo N without knowing its factorization, we *can* learn some things about the two Legendre symbols $\left(\left(\frac{c}{p}\right), \left(\frac{c}{q}\right)\right)$, even if we don't know p and q. E.g., it is possible to efficiently distinguish the case

$$\left(\left(\frac{c}{p}\right), \left(\frac{c}{q}\right)\right) = (1, -1)$$

from the case when

$$\left(\left(\frac{c}{p}\right), \left(\frac{c}{q}\right)\right) = (1, 1) .$$

To see this, we must define the *Jacobi symbol*, introduced by Jacobi in 1837. It is defined to equal to the Legendre symbol when its base is prime, and such that

$$\left(\frac{x}{N}\right) := \left(\frac{x}{p}\right) \cdot \left(\frac{x}{q}\right)$$

for $N := pq$. More generally,

$$\left(\frac{x}{p_1 \cdots p_\ell}\right) = \left(\frac{x}{p_1}\right) \cdots \left(\frac{x}{p_\ell}\right) ,$$

i.e., the Jacobi symbol modulo a product $p_1 \cdots p_\ell$ is the product of the Jacobi symbols modulo each factor $p_1, \ldots, p_\ell$. (It is common to use the exact same notation for the Jacobi symbol and the Legendre symbol, under the convention that we always mean the Jacobi symbol when N is composite. Formally, the Legendre symbol is simply not defined for composite N, and since the two symbols are equal when N is prime, there is no risk of confusion.)

Gauss's celebrated law of quadratic reciprocity tells us that the Jacobi symbol satisfies the following startling identity

$$\left(\frac{x}{N}\right) = (-1)^{\frac{x-1}{2} \cdot \frac{N-1}{2}} \cdot \left(\frac{N \bmod x}{x}\right)$$

for odd x and N. This is not obvious at all, but it is true. (In fact, it is not *so* hard to prove. There are many beautiful proofs online, and if you take a number theory class you will almost certainly see at least one proof of this fact. Gauss himself published six different proofs, in spite of his notorious

reluctance to publish. He called quadratic reciprocity his "golden theorem.") Together with the rule

$$\left(\frac{2x}{N}\right) = (-1)^{\frac{N^2-1}{8}} \cdot \left(\frac{x}{N}\right) ,$$

this allows us to compute the Jacobi symbol efficiently.[8] Specifically, we can repeatedly apply these rules until we reduce the problem to a computation on small numbers. In fact, the algorithm for computing the Jacobi symbol is basically just the Euclidean algorithm, since at each step we replace (x, N) by $(N \bmod x, x)$ (plus some bookkeeping for even x). The proof that it terminates efficiently is therefore essentially identical to the same proof for the Euclidean algorithm.

So, our encryption algorithm is actually *not* secure. In particular, if

$$\left(\frac{y}{N}\right) = -1 ,$$

which happens with probability 2/3, then we can easily distinguish ciphertexts of zero from ciphertexts of one by checking the Jacobi symbol.

This is a very interesting example of a case where a perfectly innocent-looking cryptographic construction turns out to be insecure because of some rather deep mathematics.

8 Goldwasser-Micali—quadratic residuosity done right

After rushing through this beautiful mathematics, we are ready to present a trapdoor predicate based on a form of quadratic reciprocity that is believed to be secure. This construction is due to Shafi Goldwasser (the patron saint of this class) and Silvio Micali [GM82].

Let $\mathbb{J}_N := \{x \in \overline{\mathbb{QR}}_N,\ \left(\frac{x}{N}\right) = 1\}$ be the set of quadratic non-residues with Jacobi symbol equal to one. (This is not standard notation.) Then, we can build the same trapdoor predicate (and corresponding encryption scheme) as before with $\overline{\mathbb{QR}}_N$ replaced by $\mathbb{J}_N$. I.e., our trapdoor is still $t := (p, q)$, and our public key is now $k := (N := pq, y \sim \mathbb{J}_N)$. The predicate is again

$$P_N(z) := \begin{cases} 1 & z \in \mathbb{QR}_N \\ 0 & z \in \overline{\mathbb{QR}}_N \end{cases} ,$$

but our domain is now $\mathbb{J}_N \cup \mathbb{QR}_N$, rather than all of $\mathbb{Z}_N^*$. To sample efficiently from $\mathbb{QR}_N$, we simply compute $x \sim \mathbb{Z}_N^*$ and output $z := x^2 \bmod N$. To sample efficiently from $\mathbb{J}_N$, we just output

[8] The rule for even integers $2x$ might look complicated, but it is actually much easier to derive than the first. One first notices that the Jacobi symbol is multiplicative,

$$\left(\frac{xy}{N}\right) = \left(\frac{x}{N}\right) \cdot \left(\frac{y}{N}\right) .$$

Then, one simply needs to compute

$$\left(\frac{2}{N}\right)$$

to derive the rule for

$$\left(\frac{2x}{N}\right) .$$

Another way to state this rule that is less opaque is that for primes p equal to 1 or 7 modulo 8, 2 is a square modulo p. For primes congruent to 3 or 5 modulo 8, 2 is not a square.

$z := yx^2 \bmod N$ instead. It is now easy to see that such a z is in fact uniformly distributed in $\mathbb{J}_N$, independently of y.[9]

To compute the predicate given the trapdoor (p, q), we simply check whether z is a square modulo p and modulo q. (In fact, we only need to check one of these, since by construction we always have that $\left(\frac{z}{p}\right) = \left(\frac{z}{q}\right)$. And, remember, that this can be done a bit more efficiently than by attempting to compute the square root directly. To check if z is a square modulo a prime p, simply check if $z^{(p-1)/2} \bmod p$ is $+1$ or -1.)

This is a secure trapdoor predicate under the following computational assumption, which we believe to be true.

Assumption 8.1 (Quadratic residuosity)**.** *For any PPT algorithm $\mathcal{A}$, there exists a negligible $\varepsilon(n)$ such that for all $n \in \mathbb{N}$,*

$$\Pr_{p,q\sim\mathbb{P}_n, y\sim\mathbb{QR}_{pq}}[\mathcal{A}(y, pq) = 1] - \Pr_{p,q\sim\mathbb{P}_n, y\sim\mathbb{J}_{pq}}[\mathcal{A}(y, pq) = 1] \leq \varepsilon(n) \ ,$$

where we write $\mathbb{P}_n$ for the set of all n-bit primes.

To prove security, we apply the above assumption *to the public key element* y, rather than on $yx^2 \bmod N$ or $x^2 \bmod N$. I.e., we argue that, by the above assumption, no adversary can distinguish between the trapdoor predicate game in which $y \sim \mathbb{J}_N$ is sampled honestly from one in which $y \sim \mathbb{QR}_N$ is sampled from the quadratic residues. Notice that, if $y \in \mathbb{QR}_N$, then $yx^2 \bmod N$ is distributed identically to $x^2 \bmod N$. In other words, if we sample $y \sim \mathbb{QR}_N$, then "encryptions of zero" are identical to "encryptions of one." So, this immediately implies security. Here is the formal proof.

Theorem 8.1. *The Goldwasser-Micali trapdoor predicate is hard to predict, assuming Assumption 8.1.*

Proof. Suppose that we have some PPT adversary $\mathcal{A}$ that wins the trapdoor predicate game against the Goldwasser-Micali trapdoor predicate. In particular,

$$\varepsilon(n) := \Pr_{p,q\sim\mathbb{P}_n, y\sim\mathbb{J}_{pq}, z\sim\mathbb{QR}_{pq}\cup\mathbb{J}_{pq}}[\mathcal{A}(pq, y, z) = P_{pq}(z)] - 1/2$$

is non-negligible, where here we write $P_N(z)$ for the predicate that is one if and only if z is a quadratic residue mod N. (I.e., $\mathcal{A}$ is significantly more likely to output 1 and say "yep, that looks like a quadratic residue" when z is sampled uniformly at random from $\mathbb{QR}_{pq}$, as opposed to from $\mathbb{J}_{pq}$.)

We construct an adversary $\mathcal{B}$ in the quadratic residuosity game as follows. $\mathcal{B}$ takes as input $N := pq$ for $p, q \sim \mathbb{Z}_N^*$ and $y^* \in \mathbb{Z}_N^*$, where either $y^* \sim \mathbb{QR}_N$ or $y^* \sim \mathbb{J}_N$. (Our goal is to output 1 in the first case and zero in the second.) It samples $x \sim \mathbb{Z}_N^*$ and $b \sim \{0, 1\}$ uniformly at random and sets $z := y^b x^2 \bmod N$ (i.e., $z = x^2 \bmod N$ if $b = 0$ and $z = yx^2 \bmod N$ if $b = 1$). Finally, it runs $b' \leftarrow \mathcal{A}(pq, y^*, z)$ and outputs 0 if $b' = b$ and 1 if $b' \neq b$. I.e., it guesses that $y^* \in \mathbb{J}_N$ if and only if $b = b'$.

[9] Specifically, one simply notes that for any fixed element $c \in \mathbb{J}_N$, $yc^{-1} \bmod N$ is a quadratic residue (where c^{-1} means the inverse of c modulo N, which exists because $c \in \mathbb{J}_N \subset \mathbb{Z}_N^*$). Therefore, for every c, there are exactly four values of x such that $x^2 = yc^{-1} \bmod N$, since every quadratic residue has exactly four square roots. So, the probability that $z = c$ is exactly $4/|\mathbb{Z}_N^*|$.

To see why this works, first suppose that $y^* \sim \mathbb{J}_N$, i.e., that y^* is a uniformly random non-residue with Jacobi symbol one, just like the $\boldsymbol{y}$ that is used in the trapdoor predicate game that $\mathcal{A}$ plays. Then, notice that z is itself a uniformly random element from $\mathbb{QR}_N \cup \mathbb{J}_N$, independent of y. (We observed this in the discussion before this proof, and provided a proof in Footnote 9.) Therefore, from the perspective of $\mathcal{A}$, its view in this case is identical to its view in the trapdoor predicate game, and it follows that

$$\Pr_{p,q\sim\mathbb{P}_n, y^*\sim\mathbb{J}_{pq}}[\mathcal{B}(pq, y^*) = 1] = 1 - \Pr_{p,q\sim\mathbb{P}_n, y\sim\mathbb{J}_{pq}, z\sim\mathbb{QR}_{pq}\cup\mathbb{J}_{pq}}[\mathcal{A}(pq, y, z) = P_{pq}(z)] = 1/2 - \varepsilon(n) \ .$$

On the other hand, if $y^* \sim \mathbb{QR}_N$, then notice that z is a uniformly random quadratic residue *independent of the bit b*. Therefore, no matter how the adversary $\mathcal{A}$ behaves in this case, we must have $\Pr[b = b'] \leq 1/2$. Putting the two things together, we see that

$$\Pr_{p,q\sim\mathbb{P}_n, y^*\sim\mathbb{QR}_{pq}}[\mathcal{B}(pq, y^*) = 1] - \Pr_{p,q\sim\mathbb{P}_n, y^*\sim\mathbb{J}_{pq}}[\mathcal{B}(pq, y^*) = 1] \geq \varepsilon(n) \ ,$$

which is non-negligible, as needed. □

Unfortunately, just like the RSA assumption, Assumption 8.1 is not known to be equivalent to the hardness of factoring. In some sense, it actually seems like a far stronger assumption than RSA, since it's actually a *decisional assumption*. I.e., it says that it is hard to compute a *single bit* (with probability non-negligibly than 1/2), whereas the RSA assumption only says that it is hard to compute an entire n-bit number (with non-negligible probability). And, it's a bit unnerving that there's this non-trivial issue involving the Jacobi symbol. (Indeed, perhaps the most important thing to remember about the discussion above is that it's easy to build a scheme that's insecure. If Shafi and Silvio weren't so knowledgable about number theory, they might not have known about the Jacobi symbol and then published a totally insecure scheme.) Maybe one of you will find some additional subtlety that breaks Assumption 8.1?

On the other hand, quadratic residuosity is an extremely well studied problem by mathematicians going back centuries, which makes us quite confident that Assumption 8.1 holds. And, like for RSA, the fastest attacks that we know of work by factoring N. Nevertheless, it could possibly be the case that Assumption 8.1 is false, even if factoring is hard.

Key agreement and Diffie-Hellman

9 Diffie-Hellman key agreement

Finally, we see the third paradigm for public-key cryptography: *key agreement* (often also called key exchange). The key-agreement paradigm looks at public-key cryptography from a different perspective. Instead of thinking directly about how Bob can use Alice's public key to encrypt a message, we think about how Alice and Bob can together generate a shared (pseduo)-random secret key via only public communication. If they can do this, then sending a message securely is relatively easy: they can use a secret-key encryption scheme or just a one-time pad.

Here is the formal definition. Again, the actual construction that we present is far easier to understand than the formal definition. (It is, in my opinion, the most beautiful construction that we will see.)

Definition 9.1. *A* key-agreement protocol *is a protocol between two PPT algorithms $\mathcal{A}$ and $\mathcal{B}$ satisfying the following properties.*

- **Correctness.** *At the end of the protocol, $\mathcal{A}$ and $\mathcal{B}$ each output the same string $sk \leftarrow \langle \mathcal{A}, \mathcal{B} \rangle(1^n)$, with $sk \in \{0,1\}^n$.*
- **Security.** *For any PPT adversary $\mathcal{E}$, there is a negligible function $\varepsilon(n)$ such that*

$$\Pr_{(T,sk)\leftarrow\langle\mathcal{A},\mathcal{B}\rangle(1^n)}[\mathcal{E}(T,sk)=1] - \Pr_{(T,sk)\leftarrow\langle\mathcal{A},\mathcal{B}\rangle(1^n),x\sim\{0,1\}^n}[\mathcal{E}(T,x)=1] \leq \varepsilon \ ,$$

where we write T for the transcript generated by $\mathcal{A}$ and $\mathcal{B}$ (i.e., the messages that they send). In other words, the secret key is indistinguishable from a random string, even to an adversary that has seen all of the messages sent by $\mathcal{A}$ and $\mathcal{B}$.

Now, here is the beautiful protocol due to Diffie and Hellman. This was actually proposed in Diffie and Hellman's original paper [DH76]. We present it here under the assumption that there is some publicly known group G whose order is a known n-bit prime p together with a known generator g. (We can accomplish this by assuming the existence of some fixed sequence G_n of groups, or we can just have Alice choose this group and include it in her first message. We also have clearly cheated a little here, since our secret key is no longer in the domain $\{0,1\}^n$. Formally, we should either change our definition to allow for a different key space or change our protocol. But, such details get in the way of a beautiful protocol, so we ignore them.)

Alice		**Bob**
$a \sim \mathbb{Z}_p$		
	$A := g^a \longrightarrow$	
		$b \sim \mathbb{Z}_p$
	$\leftarrow B := g^b$	
`OUTPUT` B^a		
		`OUTPUT` A^b

Notice that Alice and Bob both output the same thing—g^{ab}. The DDH assumption is exactly the assumption that the above scheme is a secure key-agreement scheme. I.e., it says that an adversary with access (g, g^a, g^b) cannot distinguish g^{ab} from a random group element. Here is the formal assumption, which applies to some sequence of groups G_n with generator g_n and order p_n, an n-bit prime.

Assumption 9.1 (Decisional Diffie-Hellman (DDH))**.** *For any PPT $\mathcal{A}$ there is a negligible $\varepsilon(n)$ such that for all $n \in \mathbb{N}$,*

$$\Pr_{a,b\sim\mathbb{Z}^*_{p_n}}[\mathcal{A}(g_n, g_n^a, g_n^b, g_n^{ab})] - \Pr_{a,b,c\sim\mathbb{Z}^*_{p_n}}[\mathcal{A}(g_n, g_n^a, g_n^b, g_n^c)] \leq \varepsilon(n) \ .$$

The status of Assumption 9.1 is similar to the status of Assumption 8.1 and to the RSA assumption. The best attacks that we know on Diffie-Hellman in general work by computing the discrete logarithm. So, we would like to say that DDH holds if and only if the discrete logarithm is

hard to compute over G. However, there are certain specific prime-order groups G over which DDH is known to be easy, but the discrete logarithm is thought to be hard. (These groups—specifically, groups with *bilinear maps*—are actually quite useful for cryptographers, though we will not see them in this class.) Nevertheless, we believe that DDH holds, e.g., in a large prime-order subgroup of $\mathbb{Z}_q^*$. (DDH is *not* secure over $\mathbb{Z}_q^*$ itself. E.g., we can solve DDH simply by checking whether the last element in the tuple (g, g^a, g^b, g^c) is a square, which it will be with 3/4 probability.)

So, how do we use this? Well, if Alice and Bob wish to communicate securely but they do not share a random secret key, they simply run the above protocol. At the end of it, they *do* share a secret key. This key is *not* random (it is uniquely determined by the transcript of the protocol, which is public), but it is just as good. In particular, it is indistinguishable from a random key, even to an adversary who can see Alice and Bob's messages. So, they just use this secret key (interpreted appropriately as a bit string) to communicate as they like using their favorite secret-key encryption scheme.

In fact, more-or-less exactly this is what is done in practice. Diffie-Hellman key agreement is used to secure roughly half of all internet traffic. The protocols used in practice are a bit more involved than the one we have seen. Specifically, they use what is called *authenticated* key agreement, in which at least one party also uses a signature scheme to sign the transcript of the protocol. (Key agreement is most useful when we know who we are talking to. If Alice and Bob just used the above protocol, Eve might be able to simply respond to Alice's message with her own message B'. If Alice and Bob had no way to authenticate, whatever secrets Alice planned on sending to Bob will now get sent to Eve.)

9.1 ElGamal encryption

If Alice and Bob only want to exchange a single message, then they do not need to combine key agreement with a many-message semantically secure secret-key encryption scheme. Instead, they can just use the one-time pad. In fact, if you think about it, this actually yields a public-key encryption scheme, in which Alice's public key is her first message $A := g^a$, and Bob's ciphertext consists of $B := g^b$ and his message padded with the shared secret key g^{ab}. This is known as ElGamal encryption [ElG84], and it works as follows. It is easiest if we think of our message m as itself a group element $m \in G$.

- $\mathsf{Gen}(1^n)$: Sample $a \sim \mathbb{Z}_p$ and output $(sk := a, pk := A = g^a)$.
- $\mathsf{Enc}(pk, m)$: Sample $b \sim \mathbb{Z}_p$ and output $c := (B := g^b, C := A^b m)$.
- $\mathsf{Dec}(sk, (B, C))$: Output $m = B^{-a}C$.

It is easy to see that this scheme is secure if (and only if) Assumption 9.1 holds. Specifically, under Assumption 9.1, the ciphertext $c := (B := g^b, C := A^b m)$ is indistinguishable from $c := (B := g^b, C')$, where C' is a uniformly random group element, independent of m.

More generally, any two-message key-agreement protocol implies a public-key encryption scheme, where the public key is simply Alice's (first and only) message A and the ciphertext is simply Bob's message B together with $K \oplus m$, where K is Alice and Bob's shared secret key. And, any public-key encryption scheme implies a two-round key-agreement protocol in which Alice samples a fresh public key pk and sends it as her first message. Bob responds by sampling a uniformly random

string r and sending $\mathsf{Enc}(pk, r)$ to Alice. Alice decrypts, and they each output r as their shared secret key. So, two-round key agreement is equivalent to public-key encryption.

Lattice-based crypto and Learning with Errors

10 Bonus content: Decades later...

(*Feel free to skip this section entirely and to jump to Section 11.*)

The public-key cryptographic constructions that we have seen so far have all used number-theoretic (or at least group-theoretic) ideas, and they were all discovered in the late 1970s and early 1980s.

A couple of decades later (depending on how you count; see the history below), a very different (and, I would argue, much simpler) method for building public-key cryptography was discovered: lattice-based cryptography. (My own research is mostly related to lattice-based cryptography. Specifically, I study the computational assumptions needed for security of lattice-based cryptography and various related questions.) We will see that it turns out to be an extremely powerful tool. Personally, I think that public-key encryption is magical. But, with lattice-based cryptography, far more magical things are possible, like fully homomorphic encryption (as we will see!).

10.1 Aside: "Post-quantum cryptography"

One of the reasons that lattice-based cryptography is useful is because it is thought to be post-quantum. That is, it is thought to be secure even against a polynomial-time adversary with a quantum computer. In contrast, a quantum computer can efficiently factor large integers and compute discrete logarithms [Sho97]. So, the current cryptography that powers the internet is *not* post quantum and will be broken if someone builds a large-scale quantum computer. Progress in building quantum computers has been slow, but there *is* progress, and this has led to calls for a switch to post-quantum cryptography.

This "quantum insecurity" of the public-key cryptography that we currently use is really quite surprising to me. Quantum computers are extremely useful for certain very specific tasks, such as for simulating quantum systems. (E.g., if you want to know how a bunch of molecules are going to interact in a setting in which quantum mechanics is directly relevant—in a drug or in some fancy new material or whatever—quantum computers would be super useful!) But, the list of problems that we think are hard classically but for which we know efficient quantum algorithms is extremely short. The three main ones are, well, simulating quantum systems (e.g., predicting how certain molecules will interact), factoring, and the discrete logarithm. If it were not for lattice-based cryptography (and a few other less well-studied constructions that we will not see in this class), one might have conjectured that there is something fundamental about public-key cryptography that makes it susceptible to quantum attacks. But, it seems that what actually happened is that we happen to have rested the security of the internet on the hardness of some of the very few classically hard problems that happen to be broken by quantum computers.

Anyway, for this reason, lattice-based cryptography is in the process of being standardized, with

the intention of replacing many of the uses of factoring- and discrete-log-based cryptography over the next decade or so. In other words, we're going to be using lattice-based cryptography instead of factoring- and discrete-log-based cryptography soon.

Though this post-quantum property is perhaps the most important aspect of lattice-based cryptography from a practical perspective, in my opinion (which is not shared universally), it is the least interesting aspect from a theoretical perspective. See [Nat22] if you're interested to learn more about the standardization process.

Aside: History

The timeline of lattice-based cryptography is rather complicated, with many fits and starts. Below, we will just focus on the line of work started by Oded Regev (my advisor). (Oded is my advisor and I am therefore strongly biased towards presenting his work over others. But, this is also the standard presentation of lattice-based cryptography in an introductory class like this one.) But, let me attempt to briefly recount some of the history here. (I will certainly fail to give a decent description.)

Lattice-based cryptography traces its roots to "knapsack cryptosystems," which are proposed public-key cryptographic constructions whose security is meant to be based on the hardness of variants of the Subset Sum and Knapsack computational problems. The original knapsack-based scheme was proposed by Merkle and Hellman in 1978 [MH78] more-or-less around the same time as RSA. It was then *broken* by Shamir in 1984 [Sha84]. (I.e., the scheme simply is not secure. Don't use it!) This was followed by a sequence of similar new constructions which were essentially all broken. See [NS01].

These attacks on knapsack-based cryptography primarily worked by viewing the cryptosystems from the perspective of geometric objects called lattices, and it was Miklós Ajtai (in 1996) who originally had the idea of building a variant of knapsack cryptosystems that more directly incorporated lattices into the design [Ajt96]. Ajtai's original scheme was only a secret-key encryption scheme (presented simply as a one-way function), but it had the tremendous advantage of having a worst-case to average-case reduction (which we will discuss below), which made it seem much more likely to be secure than prior attempts. In 1997, Ajtai and Cynthia Dwork showed a *lattice-based* public-key encryption scheme that also had a worst-case to average-case reduction [AD97]. Around the same time (though published in 1998), Jeffrey Hoffstein, Jill Pipher, and Joe Silverman discovered the NTRU cryptosystem [HPS98], a very different lattice-based public-key encryption scheme that was much more efficient but lacked a worst-case to average-case reduction.

This sparked a flurry of works in the area. See Peikert's survey for more [Pei16]. The modern approach to lattice-based cryptography was invented by my advisor Oded Regev, who described the Learning with Errors assumption (LWE) that we will see below. Later, Craig Gentry [Gen09] showed that *fully homomorphic* encryption could be constructed from lattices! (Gentry's original scheme was based on rather complicated assumptions, which were later simplified. We will see a relatively simple construction in this course!) Indeed, lattice-based cryptography has been shown to be extremely powerful, with constructions of many crazy primitives like Identity-Based Encryption (an encryption scheme in which your public key is just your name), attribute-based encryption (an encryption scheme in which the person encrypting the message can designate a set of people who can decrypt the message according to their "attibutes"—e.g., "can be decrypted by anyone who was born in the summer, has taken CS 4830, but has not yet taken CS 6840"), functional encryption, etc. See Peikert's survey [Pei16].

Aside: GCHQ (probably) did not discover lattice-based cryptography

Recall that mathematicians and cryptographers working for the British spy agency GCHQ were actually the first to discover public-key cryptography. In 1970, James Ellis conceived of the idea of public-key cryptography, but did not propose a construction. In 1973, Clifford Cocks invented RSA, and in 1974, Malcolm Williamson invented Diffie-Hellman key agreement. So, put succinctly, they discovered both public-key cryptography "based on the discrete logarithm" and public-key cryptography "based on factoring." However, as far as we know, they never discovered lattice-based cryptography. (Of course, this is all based on documents that were kept classified for 27 years. So, it is possible that they discovered other things that remain classified—perhaps including lattice-based cryptography—in which case this description of history would be nonsense.)

Maybe it's just me, but I find this to be fascinating because we performed the same experiment twice and got the same result. Public-key encryption was developed twice independently. And both times the first ideas that we came up with more-or-less the same—schemes with security based roughly on the hardness of factoring or the discrete logarithm! This fact is quite striking once one considers the (perhaps debatable) simplicity of lattice-based cryptography. Why was it not discovered first? Is this a quirk of human history, a cultural phenomenon, or something fundamental about public-key cryptography?

11 Linear equations DON'T lead to secure encryption schemes

Before we see an actual lattice-based cryptographic scheme, let's first see a silly construction that very much does not work.

Here's the scheme. Let q be a prime number. (The fact that q is prime is just for simplicity. One of the nice things about lattice-based cryptographic schemes is that the specific choices of parameters don't seem to matter too much. E.g., very little changes in the discussion below if we take q to be an n-bit prime or a $(\log n)$-bit composite number or anything in between.) The secret key is a uniformly random vector $\boldsymbol{s} \sim \mathbb{Z}_q^n$. The public key is a uniformly random $m \times n$ matrix over $\mathbb{Z}_q$, i.e. $A \sim \mathbb{Z}_q^{m \times n}$, together with $\boldsymbol{b} := A\boldsymbol{s} \bmod q \in \mathbb{Z}_q^m$. We will later take $m \gg n$.

To "encrypt" a one-bit plaintext $\mu \in \{0, \lfloor q/2 \rfloor\}$ given $(A, \boldsymbol{b})$ (it should not be clear at all yet why I took the plaintext to be either 0 or $\lfloor q/2 \rfloor$—indeed, this should look quite strange), choose $\boldsymbol{t} \sim \mathbb{Z}_q^m$, set $p := \langle \boldsymbol{b}, \boldsymbol{t} \rangle \bmod q$ and $c := p + \mu \bmod q$, and $\boldsymbol{d}^T := \boldsymbol{t}^T A$. The ciphertext itself is then $(c, \boldsymbol{d})$. I.e., we use p as a sort of one-time pad to encrypt μ, obtaining c, and we think of $\boldsymbol{d} \in \mathbb{Z}_q^n$ as somehow providing some information about p that will allow for decryption.[10]

To decrypt using the secret key, compute $p' := \langle \boldsymbol{s}, \boldsymbol{d} \rangle \bmod q$. Notice that

$$p' = \langle \boldsymbol{s}, \boldsymbol{d} \rangle = \boldsymbol{d}^T \boldsymbol{s} = \boldsymbol{t}^T A \boldsymbol{s} = \langle \boldsymbol{b}, \boldsymbol{t} \rangle = p \bmod q \,.$$

Then compute $\mu := c - p' \bmod q$.

So, this scheme is correct. The "only" teensy-weensy little problem is that it is horribly insecure. There are many attacks, but perhaps the most obvious attack is simply to use linear algebra to

[10] Here, I am treating all of my vectors $\boldsymbol{s}, \boldsymbol{b}, \boldsymbol{t}, \boldsymbol{d}$ as column vectors (because I am an a civilized person). So, $\boldsymbol{t}^T$ and $\boldsymbol{d}^T$ are row vectors. It is common in the literature to not bother to write the transpose $\cdot^T$ and instead to just write $\boldsymbol{t}A$. The reader is then to understand that if a column vector appears to the left of a matrix, it should be treated as a row vector—since no other interpretation is possible. But, here, I will use the more pedantic notation and write $\boldsymbol{t}^T A$.

compute the secret key $\boldsymbol{s}$ from the public key, $(A, \boldsymbol{b} := A\boldsymbol{s} \bmod q)$. (If you have not seen linear algebra modulo q before, suffice it to say that the same techniques that you learned over the real numbers also work modulo q. E.g., Gaussian elimination works over $\mathbb{Z}_q$ for prime q.)

12 NOISY systems of linear equations DO lead to secure cryptography—Regev encryption

Now, let's modify the above scheme until we make it both correct and secure.

The first idea is to add "noise" to $\boldsymbol{b}$. I.e., instead of taking $\boldsymbol{b} := A\boldsymbol{s} \bmod q$, we take $\boldsymbol{b} := A\boldsymbol{s} + \boldsymbol{e} \bmod q$, where $\boldsymbol{e} = (e_1, \ldots, e_m) \in \mathbb{Z}_q^m$ and the $e_i \sim \chi$ are sampled independently from some distribution χ over $\mathbb{Z}_q^n$. We write this succinctly as $\boldsymbol{e} \sim \chi^m$.

More specifically, we will take χ to be "small," i.e., to consist of small numbers. For example, we can take the noise distribution χ_σ to be uniform in the interval $[-\sigma, \sigma]$ for some parameter $\sigma \ll q$. (The preferred distribution that we use for the error here is called the *discrete Gaussian distribution*. This was the topic of my dissertation, and I am quite fond of this distribution. But, we will stick to uniform noise in these notes for simplicity.)

If this were the only change that we made, then the scheme might be secure, but it would not be correct! In particular, now we have

$$p := \langle \boldsymbol{b}, \boldsymbol{t} \rangle = \boldsymbol{t}^T A\boldsymbol{s} + \langle \boldsymbol{e}, \boldsymbol{t} \rangle = p' + \langle \boldsymbol{e}, \boldsymbol{t} \rangle \bmod q \ .$$

So, we no longer have $p = p'$, which seems like a pretty big problem. But, suppose instead of choosing $\boldsymbol{t}$ uniformly from all of $\mathbb{Z}_q^n$, we choose $\boldsymbol{t}$ to be "small" as well. We could take $\boldsymbol{t} \sim \chi_\sigma^m$, just like $\boldsymbol{e}$, but it's easier to just take $\boldsymbol{t} \sim \{0,1\}^m$.

Then, as long as σ is small enough (e.g. $\sigma < q/(10m)$ suffices), $p - p'$ will lie in a small interval. In particular, $c - p' = \mu + p - p' \bmod q$ will lie in the interval, say, $[-q/4, q/4]$ if $\mu = 0$ but when $\mu = \lfloor q/2 \rfloor$, it will never lie in this interval. (This is why we chose $\mu \in \{0, \lfloor q/2 \rfloor\}$. If we took, e.g., $\mu \in \{0,1\}$, then the noise would be significantly larger than μ, and we would be unable to decrypt.) Therefore, we can decrypt by computing $c - p' \bmod q$ and checking whether it lies in this interval! (Here, we are assuming that numbers modulo q are represented as integers between, say, $-(q-1)/2$ and $(q-1)/2$ for convenience. If we instead represent elements in $\mathbb{Z}_q$ as numbers between 0 and $q-1$, then $\mu = 0$ yields numbers in $[0, q/4] \cup [3q/4, q-1]$, which is less elegant.)

Here is the cryptosystem written out formally. This scheme is originally due to Regev [Reg09], and it is often called Regev encryption. We assume that $q = q(n)$, $m = m(n)$, and $\sigma = \sigma(n) < q/(10m)$ are fixed public parameters (which, to be clear, depend on n). (For technical reasons, this scheme is only secure if $m > n \log q$, as we will see later.)

- $\mathsf{Gen}(1^n)$: Sample $A \sim \mathbb{Z}_q^{m \times n}$ and $\boldsymbol{s} \sim \mathbb{Z}_q^n$, and $\boldsymbol{e} \sim \chi_\sigma^m$. Output $sk := \boldsymbol{s}$ and $pk := (A, \boldsymbol{b} := A\boldsymbol{s} + \boldsymbol{e} \bmod q)$.
- $\mathsf{Enc}((A, \boldsymbol{b}), \mu \in \{0, \lfloor q/2 \rfloor\})$: Sample $\boldsymbol{t} \sim \{0,1\}^m$ and output $(\boldsymbol{d}^T := \boldsymbol{t}^T A \bmod q, c := \langle \boldsymbol{b}, \boldsymbol{t} \rangle + \mu \bmod q)$.
- $\mathsf{Dec}(\boldsymbol{s}, (\boldsymbol{d}, c))$: Output 0 if $c - \langle \boldsymbol{s}, \boldsymbol{d} \rangle \bmod q$ lies in the interval $[-q/4, q/4]$ and $q/2$ otherwise.

12.1 (Decisional) Learning with Errors (LWE)

The above scheme is secure under what is known as the (Decisional) Learning with Errors Assumption, or Decisional LWE (also known due to Regev). Put succinctly, the Decisional LWE Assumption is the assumption that the public key $(A, A\boldsymbol{s} + \boldsymbol{e} \bmod q)$ described above is pseudorandom. The Decisional LWE *problem* is the computational problem that asks us to distinguish $(A, A\boldsymbol{s} + \boldsymbol{e} \bmod q)$ from a uniformly random matrix of the same dimensions. There is also a Search LWE problem, in which the goal is to find $\boldsymbol{s}$, given $(A, A\boldsymbol{s} + \boldsymbol{e})$.[11]

Formally, this is really a family of problems and assumptions, which depends on the parameters $\sigma = \sigma(n), q = q(n), m = m(n)$. In class, we will see a slightly different form of this assumption in which the adversary can request as many samples $(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{s} + e \rangle)$ as it wants. This is equivalent to allowing the parameter m to be an arbitrarily large polynomial.

Definition 12.1 (Decisional LWE Assumption)**.** *For any PPT $\mathcal{B}$, there exists negligible $\varepsilon(n)$ such that*

$$\Pr_{A \sim \mathbb{Z}_q^{m \times n}, \boldsymbol{s} \sim \mathbb{Z}_q^n, \boldsymbol{e} \sim \chi_\sigma^m}[\mathcal{B}(A, A\boldsymbol{s} + \boldsymbol{e} \bmod q) = 1] - \Pr_{A \sim \mathbb{Z}_q^{m \times n}, \boldsymbol{b} \sim \mathbb{Z}_q^m}[\mathcal{B}(A, \boldsymbol{b}) = 1] \leq \varepsilon(n) \ .$$

It is not immediately obvious that this assumption is sufficient (or even necessary!) to imply the security of Regev encryption. Intuitively, this assumption tells us that the above scheme is as secure as a modified scheme (which no longer has a valid decryption algorithm) in which the public key is replaced by uniformly random $(A, \boldsymbol{b})$. But, it is not immediately obvious whether the plaintext μ could still be identified from the ciphertext $(\boldsymbol{t}^T A, \langle \boldsymbol{b}, \boldsymbol{t} \rangle + \mu \bmod q)$, even if A and $\boldsymbol{b}$ are uniformly random. For this, we need the Leftover Hash Lemma (which is often just called LHL).

Theorem 12.2 ((Special case of the) Leftover Hash Lemma [ILL89])**.** *For $\varepsilon \in (0, 1)$, $A \sim \mathbb{Z}_q^{m \times n}$ with $m \geq n \log q + 2 \log(1/\varepsilon) + 100$, and $\boldsymbol{t} \sim \mathbb{Z}_q^m$, the distribution $(A, \boldsymbol{t}^T A)$ is ε-close to the distribution $(A, \boldsymbol{u})$, where $\boldsymbol{u} \sim \mathbb{Z}_q^n$.*[12]

We can now prove the security of Regev encryption.

Theorem 12.3. *Regev encryption is semantically secure if Decisional LWE is true and, e.g., $m > 2n \log(q)$.*

Proof. The proof is via a sequence of two games. We define Game 1 as the semantic security game against Regev encryption. Game 2 is the same game, except that $\boldsymbol{b}$ is replaced by a uniformly random element of $\boldsymbol{u} \sim \mathbb{Z}_q^m$.

Notice that Decisional LWE is exactly the assumption that Game 1 is computationally indistinguishable from Game 2. (Formally, if there existed some PPT adversary $\mathcal{B}$ that had non-negligibly larger advantage in Game 1 than in Game 2, then we could use such an adversary to break the

[11] The problem gets its name from its similarity to machine learning problems. In particular, you can view Search LWE as a problem in which you are given many samples of the form $(\boldsymbol{a}_i, \langle \boldsymbol{a}_i, \boldsymbol{s} \rangle + e_i \bmod q)$, and the goal is to "learn" $\boldsymbol{s}$.

[12] When we say that two distributions D_1, D_2 are ε-close, we mean that $\frac{1}{2} \sum |\Pr[D_1 = x] - \Pr[D_2 = x]| \leq \varepsilon$, where the sum is over all x in the support of the distributions D_1, D_2. (This sum is called the *statistical distance* between D_1 and D_2.) Equivalently, for *any* (possibly computationally unbounded) adversary,

$$\Pr[\mathcal{A}(D_1) = 1] - \Pr[\mathcal{A}(D_2) = 1] \leq \varepsilon \ .$$

So, statistical closeness means "statistically indistinguishable."

Decisional LWE assumption by constructing $\mathcal{B}'$ that takes as input $(A, \boldsymbol{b})$, plays the role of the challenger in the semantic security game with $\mathcal{B}$ with public key $(A, \boldsymbol{b})$, and outputs 1 if and only if $\mathcal{B}$ wins this semantic security game. I am being lazy here :). See Theorem 8.1 for a similar proof.) Finally, the Leftover Hash Lemma (applied to the $((n+1) \times m)$-dimensional matrix $A' := (A, \boldsymbol{u})$) tells us that Game 2 cannot be won with non-negligible probability, even by an unbounded adversary. That is, in Game 2 the distribution of an encryption of 0 is ε-close to the distribution of an encryption of 1, for $\varepsilon := 2^{-(m - n \log q - 100)/2}$, which is negligible for $m > 2n \log q$. □

13 Bonus content: Search LWE and a search-to-decision reduction

As we mentioned above, there is another natural problem that seems related to decisional LWE, but is not exactly the same: search LWE. This is the computational problem in which we are given $(A, A\boldsymbol{s} + \boldsymbol{e} \bmod q)$, and the goal is to find $\boldsymbol{s}$. I.e., it is the computational problem that asks us to find the secret key given the public key. The Search LWE Assumption is then of course the assumption that Search LWE is hard. (As above, this is really a family of assumptions, indexed by $q(n), m(n), \sigma(n)$.)

Definition 13.1 (Search LWE Assumption)**.** *For any PPT $\mathcal{B}$ there exists negligible $\varepsilon(n)$ such that*

$$\Pr_{A \sim \mathbb{Z}_q^{m \times n}, \boldsymbol{s} \sim \mathbb{Z}_q^n, \boldsymbol{e} \sim \chi_\sigma^m} [\mathcal{B}(A, A\boldsymbol{s} + \boldsymbol{e} \bmod q) = \boldsymbol{s}] \leq \varepsilon(n)$$

(Of course, the above is vacuously true if there are very many choices of $\boldsymbol{s}', \boldsymbol{e}'$ such that $A\boldsymbol{s} + \boldsymbol{e} = A\boldsymbol{s}' + \boldsymbol{e}' \bmod q$, but one can verify that $\boldsymbol{s}$ and $\boldsymbol{e}$ are unique with high probability for $m \gg n \log q$ and $\sigma \ll q/2$.)

The following theorem shows that the search and decision problems are equivalent (at least for prime $q \leq \text{poly}(n)$). In particular, this justifies us simply referring to "LWE" and not bothering to specify whether we mean search or decision.

Theorem 13.2 (Informal, see [Reg09])**.** *For prime $q \leq \text{poly}(n)$, the Decisional LWE Assumption holds if and only the Search LWE Assumption holds.*

Proof sketch. The idea behind the proof is as follows. Suppose that $(A, \boldsymbol{b} := A\boldsymbol{s} + \boldsymbol{e})$ are as in the search LWE problem. Let $\boldsymbol{r} \sim \mathbb{Z}_q^m$, and let $R_i \in \mathbb{Z}_q^{m \times n}$ be the matrix whose ith column is $\boldsymbol{r}$ and whose other columns are all zero. Then, consider $(A' := A + R_i, \boldsymbol{b}' := \boldsymbol{b} + \widetilde{s}_i \boldsymbol{r} = A\boldsymbol{s} + \boldsymbol{e} + \widetilde{s}_i \boldsymbol{r} \bmod q)$ for some $\widetilde{s}_i \in \mathbb{Z}_q$.

If $\widetilde{s} = s_i$, then

$$\boldsymbol{b}' = A'\boldsymbol{s} + \boldsymbol{e} \bmod q \,,$$

so that $(A', \boldsymbol{b}')$ is distributed exactly as a valid LWE instance. On the other hand, if $\widetilde{s} \neq s_i$, then

$$\boldsymbol{b}' = A'\boldsymbol{s} + \boldsymbol{e} + (\widetilde{s}_i - s_i)\boldsymbol{r} \bmod q$$

is uniformly random.

So, intuitively, we can use an adversary for the Decisional LWE oracle to "check" whether $s_i = \widetilde{s}_i$. Since there are only polynomially many choices of $s_i \in \mathbb{Z}_q$, this allows us to find s_i in polynomial time.

To make this actually work, one needs to worry about how to work with an adversary with only small advantage, and one needs to deal with issues of independence. The easiest solution to the

first problem is to work with the versions of LWE that we defined in class, in which the adversary may request as many samples of the form $(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{s} \rangle + e \bmod q)$ as she wants, so that we can produce arbitrarily many independent sets samples to feed to the Decisional LWE adversary. One can then run the above "guess and check" procedure many times, and argue that by Chernoff bound, we will eventually converge on the right answer with high probability. There are more clever ways as well. To deal with independence issues, we must "rerandomize" the secret each time that we do this "guess and check" procedure. I.e., we must replace $(A, A\boldsymbol{s} + \boldsymbol{e} \bmod q)$ by $(A, A(\boldsymbol{s} + \boldsymbol{s}') + \boldsymbol{e} \bmod q)$ for a fresh uniformly random vector $\boldsymbol{s}' \sim \mathbb{Z}_q^n$. □

14 Bonus content: Worst-case to average-case reductions

One of the most striking features of lattice-based cryptography are the worst-case to average-case reductions. Notice that LWE is an *average-case problem*. That is, it is defined in terms of a distribution of $(A, A\boldsymbol{s} + \boldsymbol{e} \bmod q)$. And, the LWE Assumption says that it is hard to solve this problem on the *average* over this distribution.

Of course, all of the assumptions that we have seen so far have been like this. E.g., Rabin's trapdoor function is only secure if it is hard to factor $N = pq$ on the average, for p, q sampled from a certain distribution. But, one could imagine a world in which there is no polynomial-time algorithm that factors all integers N, but factoring numbers from this particular distribution is easy on the *average*. Indeed, in practice, choosing the right distribution over p, q is non-trivial because of specialized factoring algorithms that work for some choices of p, q but not others. In some sense, such issues seem inevitable, since we have to use randomness in choosing our keys, which inevitably means we have to choose some distribution.

We might similarly worry that, while it might be difficult to find $\boldsymbol{s}$ given $(A, A\boldsymbol{s} + \boldsymbol{e} \bmod q)$ for some choices of A, $\boldsymbol{s}$, q, and "small" $\boldsymbol{e}$, it might not be hard on the average for our particular choice of q. We do not quite know how to rule this out completely, but we do have very strong evidence to suggest that this is not the case. In particular, Regev showed that solving LWE in the average-case is at least as hard as solving a worst-case variant of LWE.

For a basis $\mathbf{B} \in \mathbb{Z}^{m \times n}$, let $\lambda_1(\mathbf{B}) := \min\{\|\mathbf{B}\boldsymbol{z}\| \ : \ \boldsymbol{z} \in \mathbb{Z}^n, \ \boldsymbol{z} \neq \mathbf{0}\}$. Computing $\lambda_1(\mathbf{B})$ is known as the Shortest Vector Problem (SVP), and it was shown by Ajtai to be NP-complete [Ajt98]. (People had tried for about 20 years to prove that SVP was NP-hard before Ajtai finally succeeded.) *Approximating* $\lambda_1(\mathbf{B})$ up to some factor of $\gamma \geq 1$ is known as, well, γ-approximate SVP, and it is thought to be hard even for very large approximation factors γ. (The fastest polynomial-time algorithm achieves $\gamma = 2^{n \log\log n / \log n}$, so nearly exponential in n.) A closely related problem, which looks more like LWE, is the γ-Bounded Distance Decoding problem (γ-BDD), in which the input is $\mathbf{B} \in \mathbb{Z}^{m \times n}$ and $\boldsymbol{t} \in \mathbb{Z}^n$, and the goal is to find $\boldsymbol{z} \in \mathbb{Z}^n$ such that $\|\mathbf{B}\boldsymbol{z} - \boldsymbol{t}\|$ is as small as possible, under the *promise* that there exists a $\boldsymbol{z}$ with $\|\mathbf{B}\boldsymbol{z} - \boldsymbol{t}\| < \lambda_1(\mathbf{B})/\gamma$.

Regev proved that LWE is at least as hard as these worst-case problems! This provides very strong evidence for the LWE Assumption! (It is not hard to show that the converse is true as well. That is, if you can solve γ-BDD or γ-SVP, then you can solve LWE. However, the specific approximation factors γ are not the same. So the problems are "equivalent up to a loss in the approximation factor.")

Theorem 14.1 ([Reg09])**.** *For any $\sigma > \sqrt{n}$, there is a reduction from the* worst-case problems *γ-SVP and γ-BDD to the* average-case *(Decisional) LWE problem, for $\gamma = \mathrm{poly}(n) \cdot (q/\sigma)$.*

Notice that the approximation factor γ depends on "noise rate" σ/q. This seems reasonable: as the noise rate goes down, the LWE problem becomes easier, and we are therefore only able to use an LWE solver to solve an easier problem.

In fact, Regev's original reduction required a quantum computer and only worked for prime $q \leq \text{poly}(n)$. But, this was improved to a fully classical reduction for all q in a sequence of works (e.g. [Pei09, BLP+13]).

Two notes

15 A note on efficiency and hybrid encryption

In practice, schemes like ElGamal, Goldwasser-Micali, and RSA are typically not used to encrypt long messages. The problem is that they are very inefficient. Goldwasser-Micali requires us to do one exponentiation per bit, and the version of RSA that we saw (using a hardcore predicate) has the same issue. ElGamal still requires us to do two exponentiations for every n bits or so. This is very expensive for large messages.

Instead, in practice, we almost always use a public-key primitive like public-key encryption or key agreement to create a shared secret key (as we already described in the section on key agreement). Then, we use some secret-key scheme to communicate. This is *much* faster, since the secret-key schemes that are used in practice (not the ones we've seen in class!) are very efficient. E.g., you have almost certainly downloaded videos that are encrypted under some secret-key scheme like AES, but you probably do not download ElGamal-encrypted videos.

A particular form of this is known as *hybrid encryption*, which works as follows. Let PEnc be some public-key encryption algorithm and SEnc be some secret-key encryption algorithm. If Bob wishes to send a long file m to Alice using her public key pk, he can simply sample a fresh key sk^* for SEnc and send Alice $c := (\mathsf{PEnc}(pk, sk^*), \mathsf{SEnc}(sk^*, m))$. I.e., he uses the public-key scheme to encrypt the secret key sk^* of a secret-key scheme, and then he uses the secret-key scheme to do most of the heavy lifting.

Even more cleverly, Alice and Bob can use public-key encryption *once* in order to establish a shared secret key sk^*. They can then communicate as much as they like using secret-key encryption.

16 A note on homomorphism

All of the public-key encryption schemes that we have seen so far are *homomorphic* in some way. In other words, if we are given two encryptions $\mathsf{Enc}(pk, m_1)$ and $\mathsf{Enc}(pk, m_2)$, we can compute a valid encryption of some function $f(m_1, m_2)$ of the two messages, without knowing m_1 or m_2 (or the secret key).

For example, take two ciphertexts in the Goldwasser-Micali scheme, $c_1 := y^{m_1} x_1^2 \bmod N$ and $c_2 := y^{m_2} x_2^2 \bmod N$, where $m_1, m_2 \in \{0, 1\}$. I claim that $c^* := c_1 c_2 \bmod N = y^{m_1+m_2}(x_1 x_2)^2 \bmod N$ is a valid encryption of the ciphertext $m_1 \oplus m_2$. To see this, first notice that if $m_1 = m_2 = 0$, then this is clearly true, since then $c^* = y^{m_1+m_2}(x_1 x_2)^2 = (x_1 x_2)^2 \bmod N$ is clearly a quadratic residue. On the other hand, if $m_1 \neq m_2$, then $c^* = y(x_1 x_2)^2 \bmod N$ is clearly an encryption of one. Finally, if $m_1 = m_2 = 1$, then $y^{m_1+m_2}(x_1 x_2)^2 = y^2(x_1 x_2)^2 = (y x_1 x_2)^2 \bmod N$ is again a quadratic

residue. So, given an encryption of m_1 and m_1 and m_2, we can create a valid encryption of $m_1 \oplus m_2$ without ever knowing m_1 or m_2.

This is both useful and troublesome. It is useful in some applications. E.g., maybe you can imagine how one might use this for a coin-flipping protocol? It is also quite troublesome. E.g., it does not seem great if an adversary Eve can change Bob's message from "YES!" to "Absolutely not!" (You can fix this by having Bob sign the message, but this requires Bob to have a secret signing key, and Alice to know the corresponding verification key.)

Similarly, ElGamal is homomorphic. Specifically, given $c_1 := (g^{b_1}, A^{b_1} m_1)$ and $c_2 := (g^{b_2}, A^{b_2} m_2)$, we can compute $c^* := (g^{b_1+b_2}, A^{b_1+b_2} m_1 m_2)$ by simply multiplying the corresponding parts of the two ciphertexts. Clearly, c^* is a valid encryption of $m_1 m_2$.

In the next two lectures, we will see a very extreme version of this property. In particular, we will see a *fully homomorphic* encryption scheme, i.e., a scheme in which it is possible to compute a valid encryption of $f(m_1, m_2)$ for *any* efficiently computable function f given only $\mathsf{Enc}(pk, m_1)$ and $\mathsf{Enc}(pk, m_2)$! This is extremely useful.

References

[AD97] Miklós Ajtai and Cynthia Dwork. A Public-Key Cryptosystem with Worst-Case/Average-Case Equivalence. In *STOC*, pages 284–293, 1997.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems. In *STOC*, 1996.

[Ajt98] Miklós Ajtai. The Shortest Vector Problem in L2 is NP-hard for randomized reductions. In *STOC*, 1998.

[AM09] Divesh Aggarwal and Ueli Maurer. Breaking RSA generically is equivalent to factoring. In *EUROCRYPT*, 2009. https://eprint.iacr.org/2008/260.pdf.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of Learning with Errors. In *STOC*, 2013. http://arxiv.org/abs/1306.0281.

[DH76] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 1976.

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, 1984.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, 2009.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *STOC*, 1982.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *ANTS*, pages 267–288, 1998.

[ILL89] R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *STOC*, 1989.

[Imp95] Russell Impagliazzo. A personal view of average-case complexity. In *Structure in Complexity Theory Conference*, 1995.

[Mer78] Ralph C. Merkle. Secure communications over insecure channels. *Commun. ACM*, 21(4), 1978. http://doi.acm.org/10.1145/359460.359473.

[MH78] R. Merkle and M. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Transactions on Information Theory*, 24(5):525–530, September 1978.

[Nat22] National Institute for Standards and Technologies (NIST). Selected algorithms 2022 - Post-quantum cryptography, 2022. https://csrc.nist.gov/Projects/post-quantum-cryptography/selected-algorithms-2022.

[NS01] Phong Q. Nguyen and Jacques Stern. The two faces of lattices in cryptology. In *CaLC*, pages 146–180, 2001.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case Shortest Vector Problem. In *STOC*, 2009.

[Pei16] Chris Peikert. A decade of lattice cryptography. *Foundations and Trends in Theoretical Computer Science*, 10(4):283–424, 2016.

[Rab79] M. O. Rabin. Digitalized signatures and public-key functions as intractable as factorization. Technical report, Massachusetts Institute of Technology, 1979. 01432.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2), 1978.

[Sha84] Adi Shamir. A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem. *IEEE Trans. Inform. Theory*, 30(5):699–704, 1984.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997.

A Computing square roots modulo a prime

Here, we prove that it is possible to compute square roots efficiently modulo a prime q. I.e., given a prime q and $y \in \mathbb{QR}_q$, we can efficiently compute x such that $x^2 = y \bmod q$. (As a side note, notice there are two solutions to this equation—x and $-x$.)

The high-level idea is best illustrated in the special case when $q = 4m + 3$ for some integer m. In this case, the point is that "$m + 1$ is kind of an inverse of 2 modulo $q - 1 = 4m + 2$," in the sense that $4 \cdot (m + 1) = 2 \bmod q - 1$.[13] So, we might expect that if $y = x^2 \bmod q$, then raising y to the $(m + 1)$st power should give us a square root, i.e., $y^{m+1} = x^{2(m+1)} \bmod q$ might not equal x

[13] Notice that, 2 does not have an inverse modulo $q - 1$, since 2 is not coprime to $q - 1$. So, there is no element x such that $2x = 1 \bmod q - 1$. However, there is an element x such that $4x = 2 \bmod q$, which is pretty close!

(because $2(m+1)$ certainly does *not* equal $1 + k(q-1)$ for some integer k), but maybe it's always a square root?

Indeed, in this case $(y^{m+1})^2 = y \bmod q$, i.e., $y^{m+1} \bmod q$ is a square root is a square root of of y modulo q. To see this, recall that the order of the group $\mathbb{Z}_q^*$ is $q-1$, and therefore for any $z \in \mathbb{Z}_q^*$, $z^{q-1} = 1$ (i.e., the order of the element z divides the order $q-1$ of the group $\mathbb{Z}_q^*$). Therefore, if $y = x^2 \bmod q$ for some $x \in \mathbb{Z}_q^*$, then

$$(y^{m+1})^2 = x^{4m+4} = x^{4m+2} \cdot x^2 = y \bmod q \ .$$

as needed. (Notice that we never guaranteed that $y^{m+1} = x \bmod q$. It is of course possible that $y^{m+1} = -x \bmod q$ as well.)

Now, let $q - 1 = 2^s k$ for some odd k. Of course, such a decomposition in terms of s and k always exists (and we can find it efficiently). Notice that, if $y^k = 1 \bmod q$, then by an argument that is more-or-less identical to the above, $y^{(k+1)/2} \bmod q$ is a valid square root of y modulo q. More generally, we will show how to find $z \in \mathbb{Z}_q^*$ such that

$$y^k z^2 = 1 \bmod q \ ,$$

so that $(y^{(k+1)/2} z)^2 = y \bmod q$.

Let $w \in \mathbb{Z}_q^*$ be a quadratic *non-residue*, i.e., w is an element that cannot be written as a square modulo q. (Such a w can be found efficiently by sampling a random $w \in \mathbb{Z}_q^*$ and recalling that w is a quadratic non-residue if and only if $w^{(q-1)/2} = -1 \bmod q$. So, we can just guess and check until we find such a w.) Let $W := w^k \bmod q$, and notice that W satisfies $W^{2^{s-1}} = w^{(q-1)/2} = -1 \bmod q$.

Set $a = 0$, and do the following repeatedly until you output something. Let $0 \leq s' \leq s-1$ be minimal such that $(y^k W^{2a})^{2^{s'}} = 1 \bmod q$. (Notice that $s' \leq s-1$ precisely because y is a square, so that $y^{2^{s-1}k} = x^{2^s k} = x^{q-1=}1 \bmod q$. Furthermore, notice that we can compute such an s' efficiently.) If $s' = 0$, then we take $z = W^a \bmod q$, and we are done. Otherwise, add $2^{s-s'-1}$ (which is an integer) to a and repeat the loop.

To see that this works, we claim that s' always decreases by at least one in each step, so that the algorithm terminates after a total number of loops bounded by $s \leq \log q$. (One can actually view this algorithm as identifying all the 1s in the binary representation of a, starting with the most significant digits.) In other words, we claim that $(y^k W^{2a+2^{s-s'}})^{2^{s'-1}} = 1 \bmod q$. Indeed, first notice that, before we update a, $(y^k W^{2a})^{2^{s'-1}} = -1 \bmod q$, since by definition this is a square root of 1 that is not 1. Therefore,

$$(y^k W^{2a+2^{s-s'}})^{2^{s'-1}} = -1 \cdot W^{2^{s-1}} = 1 \bmod q \ ,$$

as needed.