Sketch of Lecture 26

Fri, 3/26/2021

Review. RSA

Example 160. If $N = 77$, what is the smallest (positive) choice for e?

Solution. Technically, $e = 1$ works but then we wouldn't be encrypting at all.
Note that e must be invertible modulo $\phi(N) = 6 \cdot 10 = 60$. Hence, $e = 2, 3, 4, 5, 6$ are not allowed.
The smallest possible choice for e therefore is $e = 7$.

Example 161. Bob's public RSA key is $N = 33$, $e = 13$. His private key is $d = 17$.

(a) Explain how the decryption of, say, $c = 26$ can be sped up using the CRT.

(b) Encrypt the message $m = 4$ and send it to Bob. Compare with the example from last class where $N = 33$, $e = 3$.

(c) Bob's choice of $e = 13$ is actually functionally equivalent to $e = 3$ and, similarly, d can be obtained as $e^{-1} \pmod{10}$, resulting in $d = 7$. Explain and generalize these claims!

(d) An RSA user is shocked by the previous part and exclaims "RSA is only half as secure as I thought...!" How shocked should we be?

Solution. Note that the private key is $d \equiv 13^{-1} \pmod{20} \equiv 17$.

(a) To decrypt, Bob needs to compute $m = c^d \pmod{N}$. Knowing that $N = pq = 3 \cdot 11$, we instead compute $c^d \pmod{p}$ and $c^d \pmod{q}$ [which is less work] and then use the CRT to recover $m \pmod{N}$.
Here, $26^{17} \equiv (-1)^{17} \equiv 2 \pmod{3}$ and $26^{17} \equiv 4^{17} \equiv 4^7 \equiv 4 \cdot 4^2 \cdot 4^4 \equiv 4 \cdot 5 \cdot 3 \equiv 5 \pmod{11}$.
Hence, $m = 26^{17} \pmod{33} \equiv 2 \cdot 11 \cdot (11)^{-1}_{\bmod 3} + 5 \cdot 3 \cdot (3)^{-1}_{\bmod 11} \equiv 22 \cdot (-1) + 15 \cdot 4 \equiv 5 \pmod{33}$.
Comment. Note that $(11)^{-1}_{\bmod 3}$ and $(3)^{-1}_{\bmod 11}$ can be precomputed and reused. In practice, using the CRT leads to about a 4-fold speed up.

(b) The ciphertext is $c = m^e \pmod{N}$. Here, $c \equiv 4^{13} \equiv \ldots \equiv 31 \pmod{33}$.
If $e = 3$ instead, then $c \equiv 4^3 = 64 \equiv 31 \pmod{33}$ so that we get the same ciphertext. See next item!

(c) If you look back at our proof of Theorem 157, you'll see that (again using the CRT) we only need $de \equiv 1 \pmod{(p-1)}$ and $de \equiv 1 \pmod{(q-1)}$ in order that $m^{de} \equiv m \pmod{pq}$.
So, instead of $d \equiv e^{-1} \pmod{(p-1)(q-1)}$, it is enough that $d \equiv e^{-1} \pmod{\operatorname{lcm}(p-1, q-1)}$.
Here, $\operatorname{lcm}(2, 10) = 10$, so that we only need $d = e^{-1} \pmod{10}$.

(d) It is definitely misleading that RSA is "half" as secure. It is indeed the case though that the key space for the secret key d is only half (or even less) as big as that RSA user initially thought.
However, that means that, for instance, if N is 2048 bit, then the secret key is one bit (possibly more) less than what the shocked RSA user expected. That hardly qualifies as "half as secure".
Comment. However, if $\operatorname{lcm}(p-1, q-1)$ is "too small", that is, $\gcd(p-1, q-1)$ is "too big" (so that we are loosing considerably more than 1 bit for the key size), then p, q should be discarded. If $\gcd(p-1, q-1) \approx 2^e$, then we are loosing about e bits for the key size.

Example 162. RSA is so cool! Why do we even care about, say, AES anymore?

Solution. RSA is certainly cool, but it is very slow (comparatively). As such, RSA is not practical for encrypting larger amounts of data. RSA is, however, perfect for sharing secret keys, which can then be used for encrypting data using, say, AES.

Example 163. Is it a problem that $m=1$ is always encrypted to $c=1$? (Likewise for $m=0$.)

Solution. Well, it would be a problem if we reply to questions using YES (say, 1) and NO (say, 0) and encrypt our reply. However, this would always be a terrible idea in any deterministic public key cryptosystem (that is, a system, in which a message gets encrypted in a single way)!

Why? That's because Eve can just encrypt both YES and NO (or any collection of expected messages) and see which matches the ciphertext she intercepted.

Important conclusion. We must not send messages taken from a small predictable set and encrypt them using a deterministic public key cryptosystem like RSA.

Once realized, this is easy to fix: for instance, Alice can just augment the plaintext with some random garbage in such a way that Bob can discard that garbage after decryption. This is done when RSA is used in practice.

Comment. This applies to any public key cryptosystem, in which a message gets encrypted in a single way. To avoid this issue, some randomness is typically introduced. For instance, for RSA, when used in practice, the plaintext would be padded with random noise before encryption. On the other hand, the ElGamal encryption we discuss next, has such randomness already built into it.

Comment. Note that this is not an issue with symmetric ciphers like DES or AES. In that case, even if the attacker knows that the plaintext must be one of "0" or "1", she still cannot draw any conclusions from intercepting the ciphertext.

Example 164. (extra) Bob's public RSA key is $N=55$, $e=7$.

(a) Encrypt the message $m=8$ and send it to Bob.

(b) Determine Bob's secret private key d.

(c) You intercept the message $c=2$ from Alice to Bob. Decrypt it using the secret key.

Solution.

(a) The ciphertext is $c=m^e \pmod{N}$. Here, $c\equiv 8^7 \pmod{55}$
$8^2\equiv 9$, $8^4\equiv 9^2\equiv 26$. Hence, $8^7=8^4\cdot 8^2\cdot 8\equiv 26\cdot 9\cdot 8\equiv 2 \pmod{55}$. Hence, $c=2$.

(b) $N=5\cdot 11$, so that $\phi(N)=4\cdot 10=40$.
To find d, we compute $e^{-1} \pmod{40}$ using the extended Euclidean algorithm:

$$\begin{array}{rlrcl} & \gcd(7,40) & \boxed{40} & = & 6\cdot\boxed{7}-2 \\ = & \gcd(2,7) & \boxed{7} & = & 3\cdot\boxed{2}+1 \\ = & 1 & & & \end{array}$$

Backtracking through this, we find that Bézout's identity takes the form

$$1=\boxed{7}-3\cdot\boxed{2}=\boxed{7}-3\cdot(6\cdot\boxed{7}-\boxed{40})=-17\cdot\boxed{7}+3\cdot\boxed{40}.$$

Hence, $7^{-1}\equiv -17\equiv 23 \pmod{40}$ and, so, $d=23$.

Comment. Actually, as discussed in Example 161, $\phi(N)=(p-1)(q-1)=4\cdot 10$ can be replaced with $\operatorname{lcm}(p-1,q-1)=\operatorname{lcm}(4,10)=20$. It follows that the pair $(e,d)=(7,23)$ is equivalent to the pair $(e,d)=(7,3)$.

(c) We need to compute $m=c^d \pmod{N}$, that is, $m=2^{23} \pmod{55}$.
$2^2=4$, $2^4=16$, $2^8\equiv 36\equiv -19$, $2^{16}\equiv 19^2\equiv 31 \pmod{55}$. Hence, $2^{23}=2^{16}\cdot 2^4\cdot 2^2\cdot 2\equiv 31\cdot 16\cdot 4\cdot 2\equiv 8 \pmod{55}$.
That is, $m=8$ (as we already knew from the first part).
Comment. As noted above, $d=3$ is equivalent to $d=23$. Indeed, $m=2^3=8 \pmod{55}$.

The ElGamal public key cryptosystem and discrete logarithms

Whereas the security of RSA relies on the difficulty of factoring, the security of ElGamal and Diffie–Hellman relies on the difficulty of computing discrete logarithms.

Discrete logarithms

Suppose $b = a^x \pmod{N}$. Finding x is called the **discrete logarithm problem** mod N. If N is a large prime p, then this problem is believed to be difficult.

Note. If $b = a^x$, then $x = \log_a(b)$. Here, we are doing the same thing, but modulo N. That's why the problem is called the discrete logarithm problem.

Example 165. Find x such that $4 \equiv 3^x \pmod{7}$.

Solution. We have seen in Example 150 that 3 is a primitive root modulo 7. Hence, there must be such an x. Going through the possibilities ($3^2 \equiv 2$, $3^3 \equiv 6$, $3^4 \equiv 4$), we find $x = 4$, because $3^4 \equiv 4 \pmod{7}$.

Example 166. Find x such that $3 \equiv 2^x \pmod{101}$.

Solution. Let us check that the solution is $x = 69$. Indeed, a quick binary exponentiation confirms that $2^{69} \equiv 3 \pmod{101}$. (Do it!)

The point is that it is actually (believed to be) very difficult to compute these **discrete logarithms**. On the other hand, just like with factorization, it is super easy to verify the answer if somebody tells us the answer.

Comment. We can check that 2 is a primitive root modulo 101. That is, $2 \pmod{101}$ has (multiplicative) order 100. That means every equation $2^x \equiv a \pmod{101}$, where $a \not\equiv 0$, has a solution.

Diffie–Hellman key exchange

(Diffie–Hellman key exchange)

- Alice and Bob select a large prime p and a primitive root $g \pmod{p}$.
- Bob randomly selects a secret integer x and reveals $g^x \pmod{p}$ to everyone.
  Alice randomly selects a secret integer y and reveals $g^y \pmod{p}$ to everyone.
- Alice and Bob now share the secret $g^{xy} \pmod{p}$.

  Indeed, Alice can compute $g^{xy} = (g^x)^y$ using the public g^x and her secret y.
  Likewise, Bob can compute $g^{xy} = (g^y)^x$ using the public g^y and his secret x.

Why is this secure? We need to see why eavesdropping Eve cannot (simply) obtain the secret $g^{xy} \pmod{p}$. She knows g, g^x, $g^y \pmod{p}$ and needs to find $g^{xy} \pmod{p}$. This is the **computational Diffie–Hellman problem** (CDH), which is believed to be hard (it would be easy if we could compute discrete logarithms).

Example 167. You are Eve. Alice and Bob select $p = 53$ and $g = 5$ for a Diffie–Hellman key exchange. Alice sends 43 to Bob, and Bob sends 20 to Alice. What is their shared secret?

Solution. If Alice's secret is y and Bob's secret is x, then $5^y \equiv 43$ and $5^x \equiv 20 \pmod{53}$.

Since we haven't learned a better method, we just compute $5^2, 5^3, \ldots$ until we find 43 or 20:

$5^2 = 25$, $5^3 \equiv 19$, $5^4 \equiv 19 \cdot 5 \equiv -11$, $5^5 \equiv -11 \cdot 5 \equiv -2$, $5^6 \equiv -2 \cdot 5 \equiv -10 \equiv 43 \pmod{53}$.

Hence, Alice's secret is $y = 6$. The shared secret is $20^6 \equiv 9 \pmod{53}$.

Note. We don't need to find Bob's secret. [It is $x = 11$.]

ElGamal encryption

Proposed by Taher ElGamal in 1985

The original paper is actually very readable: https://dx.doi.org/10.1109/TIT.1985.1057074

(ElGamal encryption)

- Bob chooses a prime p and a primitive root $g \pmod{p}$.
  Bob also randomly selects a secret integer x and computes $h = g^x \pmod{p}$.
- Bob makes (p, g, h) public. His (secret) private key is x.
- To encrypt, Alice first randomly selects an integer y.
  Then, $c = (c_1, c_2)$ with $c_1 = g^y \pmod{p}$ and $c_2 = h^y m \pmod{p}$.
- Bob decrypts $m = c_2 c_1^{-x} \pmod{p}$.

Why does decryption work? $c_2 c_1^{-x} = (h^y m)(g^y)^{-x} = ((g^x)^y m)(g^y)^{-x} = m \pmod{p}$

More conceptually, the key idea (featured in Diffie–Hellman) that makes ElGamal encryption work is that Alice (her private secret is y) and Bob (his private secret is x) actually share a secret: g^{xy}

Note that encryption is just multiplying m with the shared secret $h^y = g^{xy}$. Likewise, decryption is division by the shared secret $c_1^x = g^{xy}$.

Comment. For ElGamal, the message space actually is $\{1, 2, \ldots, p-1\}$. $m = 0$ is not permitted.

That's, of course, no practical issue. For instance, we could simply identify $\{1, 2, \ldots, p-1\}$ with $\{0, 1, \ldots, p-2\}$ by adding/subtracting 1.

Comment. p and g don't have to be chosen randomly. They can be reused. In fact, it is common to choose p to be a "safe prime" (see next comment), with specific pre-selected choices listed, for instance, in RFC 3526.

Advanced comment. Note that in order to check whether g is a primitive root modulo p, we need to be able to factor $p-1$, which in general is hard (2 is an obvious factor, but other factors are typically large and, in fact, we need them to be large in order for the discrete logarithm problem to be difficult). It is therefore common to start with a prime n and then see if $2n+1$ is prime as well, in which case we select $p = 2n+1$. Such primes p [primes such that $(p-1)/2$ is prime, too] are called **safe primes** (more later).

On the other hand, g doesn't necessarily have to be a primitive root. However, we need the group generated by g (the elements $1, g, g^2, g^3, \ldots$) to be large. For more fancy cryptosystems, we can even replace these groups with other groups such as those generated by elliptic curves.

Example 168. Bob chooses the prime $p = 31$, $g = 11$, and $x = 5$. What is his public key?

Solution. Since $h = g^x \pmod{p}$ is $h \equiv 11^5 \equiv 6 \pmod{31}$, the public key is $(p, g, h) = (31, 11, 6)$.

Comment. Bob's secret key is $x = 5$. In principle, an attacker can compute x from $11^x \equiv 6 \pmod{31}$. However, this requires computing a discrete logarithm, which is believed to be difficult if p is large.

Example 169. Bob's public ElGamal key is $(p, g, h) = (31, 11, 6)$.

(a) Encrypt the message $m = 3$ ("randomly" choose $y = 4$) and send it to Bob.

(b) Determine Bob's private key from his public key.

(c) Using Bob's private key, decrypt $c = (9, 13)$.

Solution.

(a) The ciphertext is $c = (c_1, c_2)$ with $c_1 = g^y \pmod{p}$ and $c_2 = h^y m \pmod{p}$.
Here, $c_1 = 11^4 \equiv 9 \pmod{31}$ and $c_2 = 6^4 \cdot 3 \equiv 13 \pmod{31}$. Hence, the ciphertext is $c = (9, 13)$.

(b) To find Bob's secret key x, we need to solve $11^x \equiv 6 \pmod{31}$. This yields $x = 5$.
(Since we haven't learned a better method, we just try $x = 1, 2, 3, \ldots$ until we find the right one.)

Comment. Alternatively, after having done the first part, we know that $m = c_2 c_1^{-x} \pmod{p}$ takes the form $3 = 13 \cdot 9^{-x} \pmod{31}$, which is equivalent to $9^x = 13 \cdot 3^{-1} \equiv 25 \pmod{31}$. While this also reveals $x = 5$, there is an issue with this approach. Can you see it?

[The issue is that 9 (which is c_1 and could be anything) does not have to be a primitive root. In fact, 9 is not a primitive root modulo 31. Accordingly, $9^x \equiv 25 \pmod{31}$ does not have a unique solution: $x = 20$ is another one (and does not correspond to Bob's private key).]

(c) We decrypt $m = c_2 c_1^{-x} \pmod{p}$.
Here, $m = 13 \cdot 9^{-5} \equiv 3 \pmod{31}$.

Comment. One option is to compute $9^{-1} \equiv 7 \pmod{31}$, followed by $9^{-5} \equiv 7^5 \equiv 5 \pmod{31}$ and, finally, $13 \cdot 9^{-5} \equiv 13 \cdot 5 \equiv 3 \pmod{31}$. Another option is to begin with $9^{-5} \equiv 9^{25} \pmod{31}$ (by Fermat's little theorem).

Example 170. (extra) Bob's public ElGamal key is $(p, g, h) = (23, 10, 11)$.

(a) Encrypt the message $m = 5$ ("randomly" choose $y = 2$) and send it to Bob.

(b) Encrypt the message $m = 5$ ("randomly" choose $y = 4$) and send it to Bob.

(c) Break the cryptosystem and determine Bob's secret key.

(d) Use the secret key to decrypt $c = (8, 7)$.

(e) Likewise, decrypt $c = (18, 19)$.

Solution.

(a) The ciphertext is $c = (c_1, c_2)$ with $c_1 = g^y \pmod{p}$ and $c_2 = h^y m \pmod{p}$.
Here, $c_1 = 10^2 \equiv 8 \pmod{23}$ and $c_2 = 11^2 \cdot 5 \equiv 6 \cdot 5 \equiv 7 \pmod{23}$. Hence, the ciphertext is $c = (8, 7)$.

(b) Now, $c_1 = 10^4 \equiv 18 \pmod{23}$ and $c_2 = 11^4 \cdot 5 \equiv 13 \cdot 5 \equiv 19 \pmod{23}$ so that $c = (18, 19)$.

(c) To find Bob's secret key x, we need to solve $10^x \equiv 11 \pmod{23}$. This yields $x = 3$.
(Since we haven't learned a better method, we just try $x = 1, 2, 3, \ldots$ until we find the right one.)

(d) We decrypt $m = c_2 c_1^{-x} \pmod{p}$.
Here, $m = 7 \cdot 8^{-3} \equiv 7 \cdot 4 \equiv 5 \pmod{23}$, as we knew from the first part.
[$8^{-1} \equiv 3 \pmod{23}$, so that $8^{-3} \equiv 3^3 \equiv 4 \pmod{23}$. Or, use Fermat: $8^{-3} \equiv 8^{19} \equiv 4 \pmod{23}$.]

(e) In this case, $m = 19 \cdot 18^{-3} \equiv 19 \cdot 16 \equiv 5 \pmod{23}$, as we knew from the second part.

Review. ElGamal encryption

- Like RSA, ElGamal is terribly slow compared with symmetric ciphers like AES.

  Encryption under ElGamal requires two exponentiations (slower than RSA); however, these exponentiations are independent of the message and can be computed ahead of time if need be (in that case, encryption is just a multiplication, which is much faster than RSA). Decryption only requires one exponentiation (like RSA).

- In contrast to RSA, ElGamal is randomized. That is, a single plaintext m can be encrypted to many different ciphertexts.

  A drawback is that the ciphertext is twice as large as the plaintext.

  On the positive side, an attacker who might be able to guess potential plaintexts cannot (as in the case of vanilla RSA) encrypt these herself and compare with the intercepted ciphertext.

Example 171. If Bob selects $p=23$ for ElGamal, how many possible choices does he have for g? Which are these?

Solution. g needs to be a primitive root modulo 23. Recall that, modulo a prime p, there are $\phi(\phi(p))=\phi(p-1)$ many primitive roots. Hence, Bob has $\phi(p-1)=\phi(22)=10$ choices for g.

Example 172. Does Alice have to choose a new y if she sends several messages to Bob using ElGamal encryption?

Solution. Yes, she absolutely has to randomly choose a new y every time! Here's why:

If she was using the same y to encrypt messages $m^{(1)}$ and $m^{(2)}$, Alice would be sending the ciphertexts $(c_1^{(1)}, c_2^{(1)})=(g^y, g^{xy}m^{(1)})$ and $(c_1^{(2)}, c_2^{(2)})=(g^y, g^{xy}m^{(2)})$.

That means, Eve can immediately figure out $c_2^{(1)}/c_2^{(2)}=m^{(1)}/m^{(2)}$ (the divison is a modular inverse and everything is modulo p). That's a combination of the plaintexts, and Eve should never be able to get her hands on such a thing.

(Note that Eve would know right away if Alice is doing the mistake of reusing y because $c_1^{(1)}=c_1^{(2)}$.)

Comment. The situation is just like for the one-time pad (in that case, reusing the key reveals $m^{(1)}\oplus m^{(2)}$).

The computational and decisional Diffie–Hellman problem

We indicated that the security of ElGamal depends on the difficulty of computing discrete logarithms. Here is a more precise statement.

Theorem 173. Obtaining m from c (without the private key) in ElGamal is exactly as difficult as the **computational Diffie–Hellman problem** (CDH).

The CDH problem is the following: given $g, g^x, g^y \pmod{p}$, find $g^{xy} \pmod{p}$. It is believed to be hard.

Proof. Recall that the public key is $(p,g,h)=(p,g,g^x)$. The ciphertext is $c=(g^y, h^y m)=(g^y, g^{xy}m)$.

Hence, determining m is equivalent to finding g^{xy}.

Since $g, g^x, g^y \pmod{p}$ are known, this is precisely the CDH problem. □

Example 174. In fact, even the **decisional Diffie–Hellman problem** (DDH) is believed to be difficult.

The DDH problem is the following: given $g, g^x, g^y, r \pmod{p}$, decide whether $r \equiv g^{xy} \pmod{p}$. Obviously, this is simpler than the CDH problem, where g^{xy} needs to be computed. Yet, it, too, is believed to be hard.

Comment. Well, at least it is hard (modulo p) if we always want to do better than guessing.

Here's how we can sometimes do better than guessing: if g^x or g^y are quadratic residues (this is actually easy to check modulo primes p using quadratic reciprocity and the Legendre symbol), then g^{xy} is a quadratic residue (why?!). Hence, if r is not a quadratic residue, we can conclude that $r \not\equiv g^{xy}$.

More on safe primes

Recall that p is a **safe prime** if both p and $(p-1)/2$ are prime. The next example illustrates why it is common to use safe primes for ElGamal.

In general, it is difficult to ensure that g is a primitive root, or almost a primitive root, modulo p.

Example 175. Suppose that p is a safe prime. Show that all residues $g \not\equiv 0, \pm 1 \pmod{p}$ have order $(p-1)/2$ or $p-1$.

In the latter case, g is a primitive root. In fact, if $p > 5$, then half of the residues $g \not\equiv 0, \pm 1$ are primitive roots.

Solution. Suppose $g \not\equiv 0, \pm 1 \pmod{p}$. Because p is a prime and $g \not\equiv 0$, g is invertible. Its multiplicative order N divides $\phi(p) = p-1$. But the prime factorization of $p-1$ is 2 times $(p-1)/2$. Hence, the only possible orders are $1, 2, (p-1)/2$ and $p-1$. The residues ± 1 are the only with order 1 and 2 (why?!). Thus, g must have order $(p-1)/2$ or $p-1$.

Finally, if $p > 5$ (so that $(p-1)/2$ is odd), note that the number of primitive roots is $\phi(p-1) = \phi(2)\phi((p-1)/2) = (p-3)/2$, which is exactly half of the residues g.

Advanced comment. Actually, it is easy to distinguish between the residues that have order $(p-1)/2$ and those that have order $p-1$. Recall that, if x has order $p-1$, then x^2 has order $\frac{p-1}{\gcd(p-1,2)} = \frac{p-1}{2}$. It follows that (among the $x \not\equiv 0, \pm 1$) quadratic residues have order $(p-1)/2$. (And, using quadratic reciprocity, it is computationally easy to determine whether a residue modulo p is a quadratic residue or not.)

Example 176. Is there any advantage for RSA if p is a safe prime? Potential issues?

Solution. If p is a safe prime, then $\gcd(p-1, q-1) = 2$. Why?!

Hence, the key space is as large as possible.

On the other hand, we need to think about whether we are weakening the security in case we might severely limit the number of possible p's to choose from.

Another issue is that generating random safe primes is considerably more work. On the other hand, Bob usually does not generate a public key frequently, so that this might not be much of an issue.

Sketch of Lecture 29

Further comments on RSA and ElGamal

Theorem 177. Determining the secret private key d in RSA is as difficult as factoring N.

Proof. Let us show how to factor $N = pq$ if we know e and d.

- Write $ed - 1 = 2^t m$, where t is chosen as large as possible such that 2^t divides $ed - 1$.
  Since $ed - 1 \equiv 0 \pmod{(p-1)(q-1)}$ and 2^2 divides $(p-1)(q-1)$, we have $t \geqslant 2$.
- Pick a random invertible residue x. Observe that $x^{ed-1} \equiv 1 \pmod{N}$. In other words, $(x^m)^{2^t} \equiv 1$.
  Hence, the multiplicative order of x^m must divide 2^t.
- Suppose that x^m has different order modulo p than modulo q.

  Note. This works for at least half of the (invertible) residues x. If we are unlucky, we just select another x.

  Since both orders must divide 2^t, we may suppose x^m has order 2^s modulo p, and larger order modulo q.
  Then, $x^{2^s m} \equiv 1 \pmod{p}$ but $x^{2^s m} \not\equiv 1 \pmod{q}$.
  Consequently, $\gcd(x^{2^s m} - 1, N) = p$ so that we have found the factor p of N.

  Note. Of course, we don't know s (because we don't know p and q), but we can just go through all $s = 1, 2, \ldots, t-1$. One of these has to reveal the factor p. □

However. It is not known whether knowing d is actually necessary for Eve to decrypt a given ciphertext c. This remains an important open problem.

Example 178. (homework) Bob's public RSA key is $N = 323$, $e = 101$. Knowing $d = 77$, factor N using the approach of the previous theorem.

Solution. Here, $de - 1 = 7776 = 2^5 \cdot 243$ so that $t = 5$ and $m = 243$.

- Let's pick $a = 2$. $a^m = 2^{243} \equiv 246 \pmod{323}$ must have order dividing 2^5.
  $\gcd(246^2 - 1, 323) = 19$ (so we don't even need to check $\gcd(246^{2^s} - 1, 323)$ for $s = 2, 3, 4$)
  Hence, we have factored $N = 17 \cdot 19$.

Comment. Among the $\phi(323) = 16 \cdot 18 = 288$ invertible residues a, only 36 would not lead to a factorization. The remaining 252 residues all reveal the factor 19.

Another project idea. Run some numerical experiments to get a feeling for the number of residues that result in a factorization.

Definition 179. Bob's public key cryptosystem is **semantically secure** if Eve cannot do better than guessing in the following challenge:

- Bob determines a random public and private key. The public key is given to Eve.
- Eve selects two plaintexts m_1 and m_2.
- Alice flips a fair coin and, accordingly, using the public key encrypts m_1 or m_2 as c.
- Eve now needs to decide whether c is the encryption of m_1 or m_2.

For this definition to make precise mathematical sense, we need to assume that Eve's computing power is somehow limited (typically, she is limited to polynomial-time algorithms).

Comment. Also, many variations exist of what semantic security exactly is. All of these try to capture the idea that an attacker does not learn anything about m from knowing c. The one above is often referred to as IND-CPA (Indistinguishability under Chosen Plaintext Attack).

Important comment. Realize that semantic security is a very strong property to ask for! In particular, this is much stronger than what we usually think about in terms of security: you might call a cipher secure if it is "impossible" for an attacker to get m from c. Semantic security is requiring that an attacker gets so little information from c that she cannot even tell whether it came from (her own choices) m_1 or m_2.

Example 180. Is vanilla RSA semantically secure?

Solution. No. Eve can just encrypt both m_1 and m_2 herself, and compare with c. She then knows for sure which of the two was encrypted.

Comment. As mentioned before, in practice, RSA is never used in its vanilla (or "textbook") version (unless random plaintexts are encrypted). Instead, it is randomized (like ElGamal is by design) by padding the plaintext with random stuff.

Check out OAEP: `https://en.wikipedia.org/wiki/Optimal_asymmetric_encryption_padding`

The resulting RSA-OAEP has been proven semantically secure (under the "RSA assumption" that finding m from c is hard).

Example 181. Is ElGamal semantically secure?

Solution. Essentially, yes.

Recall that the public key is $(p, g, h) = (p, g, g^x)$.

The ciphertext is $(c_1, c_2) = (g^y, h^y m) = (g^y, g^{xy} m)$. Eve needs to decide whether the m in there is m_1 or m_2.

Equivalently, she needs to decide whether $r = c_2 / m_1$ (or $r = c_2 / m_2$) equals g^{xy} or not.

This is essentially the DDH problem.

Strictly speaking. Because of the issue with quadratic residues mentioned when we introduced the DDH problem, ElGamal is not semantically secure in the sense we defined things. However, if we wanted (this is more of a theoretical point), this issue could be fixed by not computing with all invertible residues modulo p, but only with quadratic residues. We could further select p to be a **safe prime**, meaning that $(p-1)/2$ is prime again, in which case all quadratic residues (except 1) have order $(p-1)/2$ (so that no similar games can be played using orders of elements).

Practical implications. Indeed, Diffie–Hellman and ElGamal in practice often use safe primes p. In that case, as we observed in Example 175, there are no elements of small order (besides 1 and -1). Since generating such primes can be a bit expensive, it is common to use preselected ones. For instance, RFC 3526 lists six such primes (together with a generator g) with $1536, 2048, \ldots, 8192$ bits.

`https://www.ietf.org/rfc/rfc3526.txt`

Important. It is perfectly fine that p and g are not random in Diffie–Hellman or ElGamal. However, it is absolutely crucial that x (and y) are random (generated using a cryptographically secure PRG).

Example 182. What is your feeling? Can we make RSA even more secure by allowing N to factor into more than 2, say, 3 primes?

Solution. That doesn't seem like a good idea. Namely, observe that the security of RSA relies on adversaries being unable to factor N. Allowing more factors of N (while keeping the size of N fixed) makes that task easier, because more factors means that the factors are necessarily smaller.

Example 183. RSA has proven to be secure so far. However, it is easy to implement RSA in such a way that it is insecure. One important but occasionally messed up part of RSA is that ***p* and *q* must be unpredictable**, and the only way to achieve that is to choose p, q completely randomly in some huge interval $[M_1, M_2]$.

- For instance, if $N = pq$ has m digits and we know the first (or last) $m/4$ digits of p, then we can efficiently factor N.

  An adversary might know many digits of p if, for instance, we make the mistake of generating the random prime p by considering candidates of the form $10^{100} + k$ for small (random) values of k (10^{100} has no special significance; it can be replaced with any large number).

- Also, we must use a cryptographically secure PRG to generate p and q.

  If using a "bad" PRG or choosing seeds with too little entropy, then (especially among a large number of public keys generated this way) it becomes likely that (different) public keys N and N' share a prime factor p. In that case, everybody can determine $p = \gcd(N, N')$ and break both public keys.

  Indeed. For instance, in a study of Lenstra et. al., millions of public keys were collected and compared. Among the RSA moduli, about 0.2% shared a common prime factor with another one. That's terrible: if (different) public keys N and N' share a prime factor p, then everybody can determine $p = \gcd(N, N')$ and break both public keys.

  http://eprint.iacr.org/2012/064.pdf

- In that direction, is the security of public key cryptosystems like RSA in any way compromised when used by tens of millions of users?

  As noted above, millions of people using "bad" PRGs for generating RSA public keys make it likely that this weakness can be practically exploited.

  Similarly, for Diffie–Hellman and ElGamal, it is common to use fixed primes p. While fine in principle, this may be an issue if used by millions of users faced against an adversary Eve with vast resources. See, for instance: https://threatpost.com/prime-diffie-hellman-weakness-may-be-key-to-breaking-crypto/

Example 184. (side-channel attacks) For instance, by measuring the time it takes to decrypt messages as $m = c^d \pmod{N}$ in RSA, Eve might be able to reconstruct the secret key d.

This **timing attack**, first developed by Paul Kocher (1997), is particularly unsettling because it illustrates that the security of a system can be compromised even if mathematically everything is sound. This sort of attack is called a **side-channel attack**. It attacks the implementation (software and/or hardware) rather than the cryptographic algorithm.

See Section 6.2.3 in our book for more details on how d can be obtained in this attack.

In a similar spririt, there exist power attacks (measuring power instead of time during decryption) or fault attacks (for instance, injecting errors during computations):

https://en.wikipedia.org/wiki/Side-channel_attack

How to prevent? Implement RSA in such a way that no inferences can be drawn from the time and power consumption.

Lesson. Do not implement crypto algorithms yourself!! Instead, use one of the well-tested open implementations.

It's kind of sad, isn't it? Don't come up with your own ciphers. Don't implement ciphers yourself...

But it is important to realize just how easy it is to implement these algorithms in such a way that security is compromised (even if the idea, intentions and algorithms are all sound and secure).

After advertising open implementations, let us end this discussion with a cautionary example in that regard.

Example 185. The following story made lots of headlines in 2016:

https://threatpost.com/socat-warns-weak-prime-number-could-mean-its-backdoored/116104/

After a year, it was noticed that, in the open-source tool Socat ("Netcat++"), the Diffie-Hellman key exchange was implemented using a hard-coded 1024 bit prime p (nothing wrong with that), which wasn't prime! Explain how this could be used as a backdoor.

Solution. The security of the Diffie-Hellman key exchange relies on the difficulty of taking discrete logarithms modulo p. If we can compute x in $h = g^x \pmod{p}$, then we can break the key exchange.

Now, if $p = p_1 p_2$, then we can use the CRT to find x by solving the two (much easier!) discrete logarithm problems

$$h = g^x \pmod{p_1}, \quad h = g^x \pmod{p_2}.$$

This is an example of a **NOBUS backdoor** ("nobody but us"), because the backdoor can only be used by the person who knows the (secret) factorization of p.

Comment. In the present case, the Socat "prime" p actually has the two small factors 271 and 13597, and $p/(271 \cdot 13597)$ is still not a prime (but nobody has been able to factor it). This might hint more at a foolish accident than a malicious act.

Important follow-up question. Of course, the issue has been fixed and the composite number has been replaced by the developers with a large prime. However, should we trust that it really is a prime?

We don't need to trust anyone because primality checking is simple! We can just run the Miller–Rabin test N times. If the number was composite, there is only a 4^{-N} chance of us not detecting it. (In OpenSSL, for instance, $N = 40$ and the chance for an error, 2^{-80}, is astronomically low.) Both Fermat and Miller–Rabin instantly detect the number here to be composite (for certain).

Comment. This illustrates both what's good and what's potentially problematic about open source projects. The potentially problematic part for crypto is that Eve might be among the people working on the project. The good part is that (hopefully!*) many experts are working on or looking into the code. Thus, hopefully, any malicious acts on Eve's part should be spotted soon (in fact, with proper code review, should never make it into any production version). Of course, this "hope" requires ongoing effort on the parts of everyone involved, and the willingness to fund such projects.

*However, sometimes very few people are involved in a project, despite it being used by millions of users. For instance, see: https://en.wikipedia.org/wiki/Heartbleed

Example 186. (short plaintext attack on RSA) Suppose a 56bit DES key (or any other short plaintext) is written as a number $m \approx 2^{56} \approx 10^{16.9}$ and encrypted as $c = m^e \pmod{N}$.

Eve makes two lists:

- $cx^{-e} \pmod{N}$ for $x = 1, 2, \ldots, 10^9$
- $y^e \pmod{N}$ for $y = 1, 2, \ldots, 10^9$

If there is a match between the lists, that is $cx^{-e} = y^e \pmod{N}$, then $c = (xy)^e \pmod{N}$ and Eve has learned that the plaintext is $m = xy$.

This attack will succeed if m is the product of two integers x, y (up to 10^9). This is the case for many integers m.

Another project idea. Quantify how many integers factor into two small factors.

How to prevent? To prevent this attack, the plaintext can be padded with random bits before being encrypted. Recall that we should actually never use vanilla RSA (unless with random plaintexts) and always use a securely padded version instead!

Example 187. For RSA, does double (or triple) encryption improve security?

(a) Say, if Bob asks people to send him messages first encrypted with a first public key (N, e_1) and then encrypted with a second public key (N, e_2).

(b) Or, what if Bob asks people to send him messages first encrypted with a first public key (N_1, e_1) and then encrypted with a second public key (N_2, e_2).

Solution.

(a) No, this does not result in any additional security.
After one encryption, $c_1 = m^{e_1} \pmod{N}$ and the final ciphertext is $c_2 = c_1^{e_2} \pmod{N}$. However, note that $c_2 = m^{e_1 e_2} \pmod{N}$, which is the same as encryption with the single public key $(N, e_1 e_2)$.

(b) This adds only a negligible bit of security and hence is a bad idea as well. The reason is that an attacker able to determine the secret key for (N_1, e_1) is likely just as able to determine the secret key for (N_2, e_2), meaning that the attack would only take twice as long (or two computers). That's only a tiny bit of security gained, somewhat comparable to increasing N from 1024 to 1025 bits. If heightened security is wanted, it is better to increase the size of N in the first place.
[Make sure you see how the situation here is different from the situation for 3DES.]

Example 188. (common modulus attack on RSA) Alice encrypts m using each of the RSA public keys (N, e_1) and (N, e_2) so that the ciphertexts are $c_1 = m^{e_1} \pmod{N}$ and $c_2 = m^{e_2} \pmod{N}$. Eve might be able to figure out m from c_1 and c_2!! How and when?

Solution. The crucial observation is that $c_1^x c_2^y \equiv m^{e_1 x} m^{e_2 y} = m^{e_1 x + e_2 y} \pmod{N}$. Eve can choose x and y.

She knows m if she can arrange x and y such that $e_1 x + e_2 y = 1$. This is possible if $\gcd(e_1, e_2) = 1$, in which case Eve would use the extended Euclidean algorithm to determine appropriate x and y.

A scenario. Bob's public RSA key is (N, e). However, when Alice requests this public key from Bob, her message gets intercepted by Eve who instead sends (N, e_2) back to Alice, where e_2 differs from e in only one bit. Alice uses (N, e_2) to encrypt her message and sends c_2 to Bob. Of course, Bob fails to decrypt Alice's message and so resends his public key to Alice (this time, Eve doesn't intervene). Alice now uses (N, e) to encrypt her message and send c to Bob.

Since $e - e_2 = \pm 2^r$, we have $\gcd(e, e_2) = 1$ (why?!), so that Eve can determine m as explained above.

Comment on that scenario. From a practical point of view, we can argue that, if Eve can trick Alice into using a modified version of Bob's public key, then she might as well give a completely new public key (that Eve created) to Alice, in which case she can immediately decipher c_2. That's certainly true. However, that way, Eve's malicious intervention would be plainly visible as such.

Example 189. (chosen ciphertext attack on RSA) Show that RSA is not secure under a chosen ciphertext attack.

First of all, let us recall that in a chosen ciphertext attack, Eve has some access to a decryption device. In the present case, we mean the following: Eve is trying to determine m from c. Clearly, we cannot allow her to use the decryption device on c (because then she has m and nothing remains to be said). However, Eve is allowed to decrypt some other ciphertext c' of her choosing (hence, "chosen ciphertext").

You may rightfully say that this is a strange attacker, who can decrypt messages except the one of particular interest. This model is not meant to be realistic; instead, it is important for theoretical security considerations: if our cryptosystem is secure against this (adaptive) version of chosen ciphertext attacks, then it is also secure against any other reasonable chosen ciphertext attacks.

Solution. RSA is not secure under a chosen ciphertext attack:

Suppose $c = m^e \pmod{N}$ is the ciphertext for m.

Then, Eve can ask for the decryption m' of $c' = 2^e c \pmod{N}$. Since $c' = (2m)^e \pmod{N}$, Eve obtains $m' \equiv 2m$, from which she readily determines $m = 2^{-1}m' \pmod{N}$.

Comment. On the other hand, RSA-OAEP is provably secure against chosen ciphertext attacks. Recall that, in this case, m is padded prior to encryption. As a result, $2m$ or, more generally am, is not going to be a valid plaintext.

Example 190. What we just exploited is that RSA is **multiplicatively homomorphic**.

Multiplicatively homomorphic means the following: suppose m_1 and m_2 are two plaintexts with ciphertexts c_1 and c_2. Then, (the residue) $m_1 m_2$ has ciphertext $c_1 c_2$.

[That is, multiplication of plaintexts translates to multiplication of ciphertexts, and vice versa. Mathematically, this means that the map $m \rightarrow c$ is a homomorphism (with respect to multiplication).]

Indeed, for RSA, $c_1 = m_1^e$ and $c_2 = m_2^e$, so that $c_1 c_2 = m_1^e m_2^e = (m_1 m_2)^e \pmod{N}$ is the ciphertext for $m_1 m_2$.

Why care? In our previous example, being multiplicatively homomorphic was a weakness of RSA (which is "cured" by RSA-OAEP). However, there are situations where homomorphic ciphers are of practical interest. With a homomorphic cipher, we can do calculations using just the ciphertexts without knowing the plaintexts (for instance, the ciphertexts could be encrypted (secret) votes, which could be publicly posted; then anyone could add up (in an additively homomorphic system) these votes into a ciphertext of the final vote count; the advantage being that we don't need to trust an authority for that count). The search for a fully **homomorphic encryption** scheme is a hot topic. For a nice initial read, you can find more at:

https://blog.cryptographyengineering.com/2012/01/02/very-casual-introduction-to-fully/

Example 191. (chosen ciphertext attack on ElGamal) Show that ElGamal is not secure under a chosen ciphertext attack.

Solution. Recall, again, that in a chosen ciphertext attack, Eve is trying to determine m from c and Eve has access to a decryption device, which she can use, except not to the ciphertext c in question.

Suppose $c = (c_1, c_2) = (g^y, g^{xy}m)$ is the ciphertext for m. Then $(c_1, 2c_2) = (g^y, g^{xy}2m)$ is a ciphertext for $2m$. Hence, Eve can ask for the decryption of $c' = (c_1, 2c_2)$, which gives her $m' = 2m$, from which she determines $m = 2^{-1}m' \pmod{p}$.

In fact, again, the reason that ElGamal is not secure under a chosen ciphertext attack is that it is multiplicatively homomorphic.

Example 192. Show that ElGamal is multiplicatively homomorphic.

Solution. Let $(g^{y_1}, g^{xy_1}m_1)$ be a ciphertext for m_1, and $(g^{y_2}, g^{xy_2}m_2)$ a ciphertext for m_2.

The product (component-wise) of the ciphertexts is $(g^{y_1+y_2}, g^{x(y_1+y_2)}m_1 m_2)$, which is a ciphertext for $m_1 m_2$. So, again, the product of ciphertexts corresponds to the product of plaintexts.

A quick summary of some aspects of RSA and ElGamal.

- As long as appropriate key sizes are used, both RSA and ElGamal appear secure.

  About the same key size needed for both: at least 1024 bits. By now, better 2048 bits.

- The security of both RSA and ElGamal can be compromised by using a cryptographically insecure PRG to generate the secret pieces p, q (for RSA) or x (for ElGamal).

- It is important to have different ciphers, especially ones that rely on the difficulty of different mathematical problems.

  Comment. Factoring $N = pq$ and computing discrete logarithms modulo p are the two different problems for RSA and ElGamal, respectively. It is not known whether the ability to solve one of them would make it significantly easier to also solve the other one. However, historically, advances in factorization methods (like the number field sieve) have subsequently lead to similar advances in computing discrete logarithms. Both problems seem of comparable difficulty.

- Both are multiplicatively homomorphic, but RSA loses this property when padded.