Example 175. (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 $x = 1, 2, \ldots, 10^9$

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.

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 and always use a securely padded version instead!

Example 176. (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 crypotographic algorithm.

See Section 6.2.3 in our book for more details 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).

Example 177. (homework) For RSA, does double (or triple) encryption improve security? That is, instead of using a single public key (N, e_1), Bob also uses a second public key (N, e_2), and asks people to send him messages first encrypted with (N, e_1) and then encrypted with (N, e_2).

Solution. 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 $(N, e_1 e_2)$.

Comment. Even if double encryption would improve security, it still wouldn't be a good idea. The reason is that an attacker able to determine the secret key for (N, e_1) is likely just as able to determine the secret key for (N, 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 bit. 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 178. (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 179. 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 $m \to c$ 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 180. (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 181. (homework) Show that ElGamal is multiplicatively homomorphic.

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