Example 163. 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 such 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 such **safe primes** p, which are primes such that $(p-1)/2$ is prime, too. In that case, there is 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 164. 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 165. 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.

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

  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

  One source of this unsettling state of affairs is the use of "bad" PRGs (including seeds with too little entropy).

  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 166. (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).

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

Example 167. The following story from two years ago made lots of headlines:

`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 168. (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 $m = xy$.

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 and always use a securely padded version instead!

Example 169. 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 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.]