Review. Side-channel attacks target the implementation of a cryptosystem and use information such as timing, power consumption, cache usage, or electromagnetic/acoustic/optical data.

After advertising open implementations, let us add a cautionary example in that regard.

Example 186. 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 Of course, the issue has been fixed and the composite number has been replaced by the developers with a large prime.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 187. (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 $x = 1, 2, ..., 10^9$
- $y^e \pmod N$ for $y = 1, 2, ..., 10^9$

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 188. 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 189. (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 sends 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 190. (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 191. 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 192. (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 193. 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.