Sketch of Lecture 32

Wed, 4/11/2018

Example 170. (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 171. 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 172. (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 173. (homework) 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, maybe 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 looses this property when padded.

Example 174. (common modulus attack on RSA) 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.

Show that Eve can figure out the plaintext from c and c_2!!

Solution. Eve knows $c \equiv m^e \pmod{N}$ as well as $c_2 \equiv m^{e_2} \pmod{N}$.

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

She knows m if she can arrange x and y such that $ex + e_2 y = 1$.

Since $e - e_2 = \pm 2^r$, we have $\gcd(e, e_2) = 1$ (why?!). Hence, Eve can indeed find such x and y using the extended Euclidean algorithm.

Comment. 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.

Application: hash functions

A hash function H is a function, which takes an input x of arbitrary length, and produces an output $H(x)$ of fixed length, say, b bit.

Example 175. (error checking) When Alice sends a long message m to Bob over a potentially noisy channel, she also sends the hash $H(m)$. Bob, who receives m' (which, he hopes is m) and h, can check whether $H(m') = h$.

Comment. This only protects against accidental errors in m (much like the check digits in credit card numbers we discussed earlier). If Eve intercepts the message $(m, H(m))$, she can just replace it with $(m', H(m'))$ so that Bob receives the message m'.

Eve's job can be made much more difficult by sending m and $H(m)$ via two different channels. For instance, in software development, it is common to post hashes of files on websites (or announce them otherwise), separately from the actual downloads. For that use case, we should use a one-way hash (see next example).

- The hash function $H(x)$ is called **one-way** if, given y, it is computationally infeasible to compute m such that $H(m) = y$. [Also called **preimage-resistant**.]

This makes the hash function (weakly) **collision-resistant** in the sense that given a message m it is difficult to find a second message m' such that $H(m) = H(m')$. [Also called **second preimage-resistant**.]

- It is called **(strongly) collision-resistant** if it is computationally infeasible to find two messages m_1, m_2 such that $H(m_1) = H(m_2)$.

  Comment. Every hash function must have many collisions. On the other hand, the above requirement says that finding even one must be exceedingly difficult.

Example 176. (error checking, cont'd) Alice wants to send a message m to Bob. She wants to make sure that nobody can tamper with the message (maliciously or otherwise). How can she achieve that?

Solution. She can use a one-way hash function H, send m to Bob, and publish (or send via some second route) $y = H(m)$. Because H is one-way, Eve cannot find a value m' such that $H(m') = y$.

Some applications of hash functions include:

- **error-checking:** send m and $H(m)$ instead of just m
- **tamper-protection:** send m and $H(m)$ via different channels (H must be one-way!)

  If H is one-way, then Eve cannot find m' such that $H(m') = H(m)$, so the cannot tamper with m without it being detected.
- **password storage:** discussed later (there are some tricky bits)
- **digital signatures**: more later
- **blockchains**: used, for instance, for cryptocurrencies such as Bitcoin

Some popular hash functions:

	published	output bits	comment
CRC32	1975	32	not secure but common for checksums
MD5	1992	128	common; used to be secure (now broken)
SHA-1	1995	160	common; used to be secure (collision found in 2017)
SHA-2	2001	256/512	considered secure
SHA-3	2015	arbitrary	considered secure

- CRC is short for **Cyclic Redundancy Check**. It was designed for protection against common transmission errors, not as a cryptographic hash (for instance, CRC is a linear function).
- SHA is short for **Secure Hash Algorithm** and (like DES and AES) is a federal standard selected by NIST. SHA-2 is a family of 6 functions, including SHA-256 and SHA-512 as well as truncations of these.

  SHA-3 is not meant to replace SHA-2 but to provide a different alternative (especially following successful attacks on MD5, SHA-1 and other hash functions, NIST initiated an open competition for SHA-3 in 2007). SHA-3 is based on Keccak (like AES is based on Rijndael; Joan Daemen involved in both). Although the ouput of SHA-3 can be of arbitrary length, the number of security bits is as for SHA-2.

  https://en.wikipedia.org/wiki/NIST_hash_function_competition
- MD is short for **Message Digest**. These hash functions are due to Ron Rivest (MIT), the "R" in RSA. Collision attacks on MD5 can now produce collisions within seconds. For a practical exploit, see: https://en.wikipedia.org/wiki/Flame_(malware)

  MD6 was submitted as a candidate for SHA-3, but later withdrawn.