CSCI3381-Cryptography

Lecture 7: RSA

October 9, 2014

Much of this is in Chapter 6 of the textbook.

1 General Issues in Public-key Cryptography

In symmetric cryptography, Alice and Bob share a single key K known only to them. In a public-key cryptosystem, one participant (let's say Bob) has a pair of keys K_{pub} and K_{priv}, the public and private keys. K_{pub} is known to everyone, and K_{priv} only to Bob. If Alice wants to send a message M to Bob, she encrypts it using Bob's public key and sends him

$$C = E_{K_{pub}}(M).$$

Bob decrypts it using his private key:

$$M = D_{K_{priv}}(C).$$

Of course, $D_{K_{priv}}$ is required to invert $E_{K_{pub}}$, and as usual, we assume that the algorithms E and D are known to everyone. However, no one should be able to determine K_{priv} given the public key K_{pub}. The public key is like a combination lock, distributed openly, and the private key is its combination, known only to one party.

1.1 Performance

In principle, Bob and Alice could hold a two-way conversation, but Bob would need Alice's public key to encrypt and Alice would use her own private key to decrypt. In practice, this is not really done. While all the algorithms underlying

RSA and other public key systems are 'easy', they are still much more time-consuming than symmetric block ciphers like AES and DES. The more common practice is this: Alice generates a key K for a symmetric cipher and encrypts it with Bob's public key; that is, she sends him

$$E_{K_{pub}}(K).$$

Bob decrypts, and all subsequent communication in the session is done using the symmetric cipher and the shared key K. Note that you get a new key K with each session, but the public and private keys are long-term.

1.2 Randomization of plaintext

Since the encryption key is public, an eavesdropper can tell whether intercepted ciphertext C is the encryption of a plaintext message M–she just has to encrypt M with the public key and compare! For this reason, data to be encrypted with a public key is usually padded with a large number of random bits before encryption. So instead of sending

$$E_{K_{pub}}(M)$$

to Bob, Alice sends

$$E_{K_{pub}}(M||\text{random} \quad \text{stuff}).$$

Doing so thwarts this kind of chosen-plaintext attack, because every time M is encrypted, the ciphertext is different. It also is effective against many of the attacks on RSA described below.

1.3 Active attack

Public-key cryptosystems are vulnerable to a *man-in-the-middle attack*. (This is not the same thing as the 'meet-in-the-middle attack' we saw earlier.) Here the attacker –let's call her Cruella– is not a passive eavesdropper, but someone capable of sending and altering messages on the communciation link. Cruella establishes her own public-private key pair (K'_{pub}, K'_{priv}) and masquerades as Alice. Let's say Bob and Alice want to use the public-key algorithm to set up an encrypted session with a symmetric key K. Since Bob has Cruella's public key, he sends

$$E_{K'_{pub}}(K).$$

This is intercepted by Cruella, who now shares the key K with Bob, who believes Cruella is Alice. Cruella generates a second symmetric key K', and uses Alice's

genuine public key K_{pub} to send $E_{K_{pub}}(K')$ to Alice, who believes the message came from Bob. Cruella now has secure links with both Alice and Bob. She receives messages from one party, decrypts them, alters them, and re-encrypts the altered message to the other party. Alice and Bob think they are talking to each other.

Man-in-the-middle attacks are a real problem. To thwart them, it is necessary to establish some means of authenticating users to one another. (To be concrete about it, how do you know that website really belongs to Amazon?) We will discuss authentication and signature methods, as well as 'public-key infrastructure,' a bit later in this course. In a sense, this forces us to bring back the Trusted Authority.

2 Security of RSA

The RSA algorithm itself is described in the last set of notes, and at the beginning of Chapter 6 of the text. We saw that it *works* (because decryption really does undo encryption), and that it is *easy* (because all the steps can be carried out efficiently), but is it *secure?* Can an eavesdropper recover the plaintext message M from $M^e \bmod N$?

2.1 Relation to hardness of factoring

If Eve can factor the modulus N, then she can repeat the key-generation phase of RSA and recover the private key d. For this reason, breaking RSA is easier than brute-forcing the key. Even the most naïve algorithm for factoring requires searching through no more than $\sqrt{N}$ values, and the current best algorithms are far faster than this, having successfully factored (with massive effort) numbers of more than 700 bits. For this reason, RSA moduli are much longer than keys used with symmetric block ciphers: 1024 bits and 2048 bits are commonly-used values.

Still, we believe that factoring is hard. Does this imply that breaking RSA is hard? In one sense, yes: We will prove that someone who possesses the private key d can factor N with an efficient probabilistic algorithm. So if factoring really is hard, then it is also hard to recover RSA private keys.

This next piece, which describes the factoring algorithm, is a bit involved, and you can skip it on first reading. We will use the standard notations p, q, e, d, N for the values in the RSA algorithm.

Since $ed-1$ is a multiple of both $p-1$ and $q-1$, we have

$$a^{ed-1} \equiv 1 \pmod{N}$$

for all a such that $\gcd(a, N) = 1$. Now if $a^k \bmod N = 1$ for all a relatively prime to N, then we have in particular $(-1)^k \bmod N = 1$, so k must be even. Secondly (we won't prove this), if $a^k \bmod N \neq 1$ for some a relatively prime to N, then this must be the case for at least half the a relatively prime to N.

We can then proceed as follows: We initially set $k = ed - 1$. For each k, sample a bunch of values $a < N$ at random and check whether $\gcd(a, N) = 1$. In the (very unlikely) event that you find $\gcd(a, N) \neq 1$, then the common factor will be a prime divisior of N and you're done. Otherwise, calculate $a^k \bmod N$. If $a^k \bmod N = 1$ for all the sampled a, then probably this holds for all a relatively prime to N, in light of our claim about the number of a for which this does not hold. Thus k is even, so replace it by $k/2$ and repeat. Eventually, we will arrive at values k and a such that $a^{2k} \mod N = 1$, and $a^k \bmod N \neq 1$.

This means that $r = a^k$ is a square root of 1 modulo N. There are four different square roots of 1, and they are the solutions to the congruences

$$x \equiv \pm 1 \pmod{p}$$

$$x \equiv \pm 1 \pmod{q}.$$

The Chinese Remainder Theorem tells us that all four solutions are different. If we find that r is 1 or -1, then we resample, trying out new values of a. Eventually we will find an r that is a square root of 1 modulo N different from ± 1. We have

$$N | r^2 - 1 = (r-1)(r+1).$$

Since N divides neither $r-1$ nor $r+1$, we must have $p|r-1$ and $q|r-1$, or vice-versa. Take the gcd of $r-1$ with N and you recover a prime factor.

Here is an example to illustrate the algorithm. We take $N = 8611$ and $e = 5$, and somehow recover the private key $d = 1685$. We will use this to factor N.

```
>>> N=8611
>>> e=5
>>> d=1685
>>> k=e*d-1
>>> As=[random.randint(2,N-1) for j in range(10)]
>>> pows=[pow(a,k,N) for a in As]
>>> pows
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```
>>> k=k/2
>>> As=[random.randint(2,N-1) for j in range(10)]
>>> pows=[pow(a,k,N) for a in As]
>>> pows
[1, 1, 1, 1, 1, 1, 1, 1, 5451, 1]
```

We have found a square root of 1 modulo N, namely 5451, and it is not equal to either 1 or -1. For the next step, we can use either 5450 or 5452. Computing the gcd with N gives one prime factor of N, and simple division gives the other.

```
>>> mrpt.extended_euclid(5450,N)
(109, (-30, 19))
>>> N/109
79
>>> 109*79
8611
```

This result is to be taken as theoretical evidence for the security of RSA: We cannot recover the secret key d unless factoring is easy. (But see below and the next homework assignment for an attack based on this algorithm.) It does not address is the more fundamental question of whether recovering the plaintext from ciphertext is as hard as factoring. To my knowledge, this is an open question: we really don't know whether breaking RSA is as hard as factoring. Security of RSA is based on a stronger conjecture about computational complexity:
RSA Conjecture. There is no easy algorithm which, given $N, e, M^e \bmod N$ where N is the product of two unknown primes p, q, and $\gcd(e, (p-1)(q-1))$, computes $M \bmod N$.

2.2 Assorted attacks on RSA

Here is a brief description of a number of attacks on RSA. Some of these can be prevented by random padding, others by following correct procedure in the generation of keys. You will get to try them all out on the next homework assignment.

2.2.1 Small message, small encryption exponent

A common choice for the encryption exponent is $e = 3$. If we are encrypting a small message, say using a 1024-bit RSA modulus to encrypt a 56-bit DES key

M, then the ciphertext C satisfies

$$C = M^3 \bmod N = M^3,$$

because in this case $M^3 < N$. We recover M by taking the cube root, which can be done quickly. (Observe the important distinction between taking the cube root mod N, which is believed to be hard in general, and taking the exact cube root, which is easy.)

2.2.2 Small encryption exponent

Again suppose that $e = 3$, and let us imagine that the same message M is sent to three different recipients, holding different public keys. We assume that the moduli N_1, N_2, N_3 of the keys are pairwise relatively prime (if not, we can factor two of the moduli and the game is up). So we have

$$M^3 \equiv C_i \pmod{N}_i$$

for $i = 1, 2, 3$. By the Chinese Remainder Theorem, there is a unique solution less than N_1, N_2, N_3, so this unique solution must be M^3. Once again, we take the cube root.

2.2.3 Two public keys with the same modulus

Your employer is lazy about generating primes, so he gives everyone in the company a public-private key pair N, e, d with different e and d for each employee, but a single N. Using your own private key d you can factor N, as outlined above, and thus recover all your colleagues' private keys.

2.2.4 Two public keys with the same modulus, again

You notice that the public keys of two users have the same modulus but different encryption exponents e_1, e_2. If e_1, e_2 are relatively prime and the two users encrypt the same message M, then you can recover M from the intercepted ciphertexts $C_1 = M^{e_1} \bmod N$ and $C_2 = M^{e_2} \bmod N$ by writing $1 = ae_1 + be_2$ and thus obtaining

$$M = M^{ae_1+be_2} \equiv C_1^a C_2^b \pmod{N}.$$

2.2.5 Two public keys with a common factor

If $N_1 \neq N_2$ are RSA moduli with a factor in common, taking the gcd reveals the factor. We thus factor N_1 and N_2 and can decrypt all the traffic encrypted with these public keys. (If you think this scenario, and the preceding ones, are far-fetched, see 'Mining Your P's and Q's: Detection of Widespread Weak Keys in Network Devices,' by N. Heninger *et. al.,* at

`https://factorable.net/weakkeys12.extended.pdf`

The authors observed many instances of common moduli and common factors in public keys obtained from the Internet).

2.2.6 Prime Factors too Close

If p and q are too close together ($|p - q| < N^{1/4}$,) then there is an efficient algorithm for recovering p and q from N. This will be outlined in the next assignment.

2.2.7 Short Plaintext

This is described in Section 6.2.2 of the textbook. If we encrypt a 56-bit DES key K using RSA, there is a good chance that K is the product of two integers x, y each less than 10^9. In this case the ciphertext C satisfies

$$Cy^{-e} \equiv x^e \pmod{N},$$

and we can recover x, y with a meet-in-the-middle attack.