

Case 2. Suppose that $n = qr$, where q, r are co-prime. Among all line 7 nonwitnesses, find a nonwitness for which the -1 appears in the largest position in the sequence in line 5 of the algorithm (note that -1 is a line 7 nonwitness, so the set of these nonwitnesses is not empty). Let x be such a nonwitness and let j be the position of -1 in its sequence, where the positions are numbered starting at 0; $x^{s \cdot 2^j} \equiv -1 \pmod{n}$ and $x^{s \cdot 2^{j+1}} \equiv 1 \pmod{n}$. The line 7 nonwitnesses are a subset of $S_2 := \{a \in \mathbb{Z}_n^* \mid a^{s \cdot 2^j} \equiv \pm 1 \pmod{n}\}$, and S_2 is a subgroup of \mathbb{Z}_n^* .

By the CRT there exists $t \in \mathbb{Z}_n$ such that

$$\begin{aligned} t &\equiv x \pmod{q} & \Rightarrow & t^{s \cdot 2^j} \equiv -1 \pmod{q} \\ t &\equiv 1 \pmod{r} & & t^{s \cdot 2^j} \equiv 1 \pmod{r} \end{aligned}$$

Hence t is a witness because $t^{s \cdot 2^j} \not\equiv \pm 1 \pmod{n}$ but on the other hand $t^{s \cdot 2^{j+1}} \equiv 1 \pmod{n}$.

Problem 6.6. Show that $t^{s \cdot 2^j} \not\equiv \pm 1 \pmod{n}$.

Therefore, just as in case 1, we have constructed a $t \in \mathbb{Z}_n^*$ which is not in S_2 , and so S_2 can be at most half of \mathbb{Z}_n^* , and so at least half of the elements in \mathbb{Z}_n are witnesses. \square

Problem 6.7. First show that the sets S_1 and S_2 (in the proof of theorem 6.5) are indeed subgroups of \mathbb{Z}_n^* , and that in case 2 all nonwitnesses are contained in S_2 . Then show that at least half of the elements of \mathbb{Z}_n are witnesses when n is composite, without using group theory.

Note that by running the algorithm k times on independently chosen a , we can make sure that it rejects a composite with probability $\geq 1 - \frac{1}{2^k}$ (it will always accept a prime with probability 1). Thus, for $k = 100$ the probability of error, i.e., of a false positive, is negligible.

6.4 Public Key Cryptography

A *Public Key Cryptosystem* (PKC) consists of three sets: K , the set of (pairs of) *keys*, M , the set of *plaintext* messages, and C , the set of *ciphertext* messages. A pair of keys in K is $k = (k_{\text{priv}}, k_{\text{pub}})$; the *private* (or *secret*) key and the *public* key, respectively. For each k_{pub} there is a corresponding *encryption* function $e_{k_{\text{pub}}} : M \rightarrow C$ and for each k_{priv} there is a corresponding *decryption* function $d_{k_{\text{priv}}} : C \rightarrow M$.

The property that the encryption and decryption functions must satisfy is that if $k = (k_{\text{priv}}, k_{\text{pub}}) \in K$, then $d_{k_{\text{priv}}}(e_{k_{\text{pub}}}(m)) = m$ for all $m \in M$. The necessary assumption is that it must be difficult to compute $d_{k_{\text{priv}}}(c)$ just from knowing k_{pub} and c . But, with the additional *trapdoor* information k_{priv} it becomes easy to compute $d_{k_{\text{priv}}}(c)$.

In the following sections we present three different encryption schemes; Diffie-Hellman, which is not really a PKC but rather a way of agreeing on a secret key over an insecure channel, as well as ElGamal and RSA. All three require large primes (in practice about 1,000 bit long); a single prime for Diffie-Hellman and ElGamal, and a pair of primes for RSA. How to go about it? The answer will of course involve the Rabin-Miller algorithm from the previous section.

Here is how we go about it: we know by the prime number theorem that there are about $\pi(n) = n/\log n$ many primes $\leq n$. This means that there are $2^n/n$ primes among n -bit integers, roughly 1 in n , and these primes are fairly uniformly distributed. So we pick an integer at random, in a given range, and apply the Rabin-Miller algorithm to it.

6.4.1 Diffie-Hellman key exchange

If p is prime, then one can show—though the proof is difficult and we omit it here—that there exists a $g \in \mathbb{Z}_p^*$ such that $\langle g \rangle = \{g^1, g^2, \dots, g^{p-1}\} = \mathbb{Z}_p^*$. This g is called a *primitive root* for \mathbb{Z}_p^* . Given an $h \in \mathbb{Z}_p^*$, the *Discrete Log Problem* (DLP) is the problem of finding an $x \in \{0, 1, \dots, p-1\}$ such that $g^x \equiv h \pmod{p}$. That is, $x = \log_g(h)$.

For example, $p = 56609$ is a prime number and $g = 2$ is a generator for \mathbb{Z}_{56609}^* , that is $\mathbb{Z}_{56609}^* = \{2^1, 2^2, 2^3, \dots, 2^{56608}\}$, and $\log_2(38679) = 11235$.

The DLP is assumed to be a difficult problem. We are going to use it to set up a way for Alice and Bob to agree on a secret key over an insecure channel. First Alice and Bob agree on a large prime p and an integer $g \in \mathbb{Z}_p^*$. The numbers p, g are public knowledge, that is, $k_{\text{pub}} = \langle p, g \rangle$.

Then Alice picks a secret a and Bob picks a secret b . Note that Diffie-Hellman is not really a fully-fledged PKC; it is just a way for two parties to agree on a secret value over an insecure channel.

Alice computes $A := g^a \pmod{p}$ and Bob computes $B := g^b \pmod{p}$. Then Alice and Bob exchange A and B over an (insecure) link. On her end, Alice computes $A' := B^a \pmod{p}$ and Bob, on his end, computes $B' := A^b$

(mod p). Clearly,

$$A' \equiv_p B^a \equiv_p (g^b)^a \equiv_p g^{ab} \equiv_p (g^a)^b \equiv_p A^b \equiv_p B'.$$

This common value $A' = B'$ is their secret key.

Suppose that Eve is eavesdropping on this exchange. She is capable of gleaning the following information from it: $\langle p, g, g^a \pmod{p}, g^b \pmod{p} \rangle$. Computing $g^{ab} \pmod{p}$ (i.e., $A' = B'$) from this information is known as the *Diffie-Hellman Problem* (DHP), and it is assumed to be difficult when p is large.

But suppose that Eve has an efficient way of solving the DLP. Then, from $g^a \pmod{p}$ she computes a , and from $g^b \pmod{p}$ she computes b , and now she can easily compute $g^{ab} \pmod{p}$. On the other hand, it is not known if solving DHP efficiently yields an efficient solution for the DLP.

Problem 6.8. Consider the following algorithm.

Algorithm 6.2 Shank's Babystep-Giantstep Algorithm

- 1: On input g, p, h
 - 2: $n = 1 + \lfloor \sqrt{p} \rfloor$
 - 3: Compute list $g^0, g^1, g^2, \dots, g^n \pmod{p}$
 - 4: Compute list $hg^0, hg^{-n}, hg^{-2n}, \dots, hg^{-n^2} \pmod{p}$
 - 5: Find a match between the two lists, say $g^i = hg^{-jn}$
 - 6: **return** $x = jn + i$
-

Show that Shank's algorithm computes x , such that $g^x \equiv h \pmod{p}$ in time $O(n \log n)$ that is, $O(\sqrt{p} \log(\sqrt{p}))$.

6.4.2 ElGamal

This is a true PKC. Alice and Bob agree on public p, g . Alice also has a private a and a public $A := g^a \pmod{p}$. Bob wants to send a message m to Alice, so he creates an *ephemeral* key b , and sends the following pair c_1, c_2 to Alice:

$$\langle c_1 := g^b \pmod{p}, \quad c_2 := mA^b \pmod{p} \rangle.$$

Then, in order to read the message, Alice computes:

$$c_1^{-a} c_2 \equiv_p g^{-ab} m g^{ab} \equiv_p m.$$

Note that to compute c_1^{-a} Alice first computes the inverse of c_1 (in \mathbb{Z}_p^*), which she can do efficiently using the Extended Euclidean algorithm, and then computes the a -th power of the result.

Problem 6.9. We say that we can “break” ElGamal, if we have an efficient way for computing m from $\langle p, g, A, c_1, c_2 \rangle$. Show that we can break ElGamal if and only if we can solve the DHP efficiently.

6.4.3 RSA

Choose two odd primes p, q , and set $n = pq$. Choose $k \in \mathbb{Z}_{\phi(n)}^*$, $k > 1$. Advertise f , where $f(m) \equiv m^k \pmod{n}$. Compute $l = k^{-1}$ (inverse of k in $\mathbb{Z}_{\phi(n)}^*$). Now $\langle n, k \rangle$ are public, and the key l is secret, and so is the function g , where $g(C) \equiv C^l \pmod{n}$. (Note that $g(f(m)) \equiv_n m^{kl} \equiv_n m$.)

Note that computing the inverse of k in $\mathbb{Z}_{\phi(n)}^*$, that is l , can be done in polytime using the extended Euclidean algorithm. Just observe that if $k \in \mathbb{Z}_{\phi(n)}^*$, then $\gcd(k, \phi(n)) = 1$, so $\exists s, t$ such that $sk + t\phi(n) = 1$, and further s, t can be chosen so that s is in $\mathbb{Z}_{\phi(n)}^*$ (first obtain any s, t from the extended Euclidean algorithm, and then just add to s the appropriate number of (positive or negative) multiples of $\phi(n)$ to place it in the set $\mathbb{Z}_{\phi(n)}^*$, and adjust t by the same number of multiples (of opposite sign)). Set $l := s$.

Obviously RSA relies on the hardness of factoring integers for its security; if we were able to factor n , we would obtain p, q , and hence $\phi(n) = \phi(pq) = (p-1)(q-1)$, and so we would be able to compute l .

The first question is: why $m^{kl} \equiv_n m$? Observe that $kl = 1 + (-t)\phi(n)$, where $(-t) > 0$, and so $m^{kl} \equiv_n m^{1+(-t)\phi(n)} \equiv_n m \cdot (m^{\phi(n)})^{(-t)} \equiv_n m$, because $m^{\phi(n)} \equiv_n 1$. Note that this last statement does not follow directly from Euler’s Theorem, because $m \in \mathbb{Z}_n$, and not necessarily in \mathbb{Z}_n^* ; in fact m must be in $\mathbb{Z}_n - \{0, p, q, pq\}$, so we could insist that the messages m are small relative to n , so that $0 < m < \min\{p, q\}$ —in fact, we break a large message into those small pieces. By Fermat’s little theorem, we know that $m^{(p-1)} \equiv_p 1$ and $m^{(q-1)} \equiv_q 1$, so $m^{(p-1)(q-1)} \equiv_p 1$ and $m^{(q-1)(p-1)} \equiv_q 1$, thus $m^{\phi(n)} \equiv_p 1$ and $m^{\phi(n)} \equiv_q 1$. This means that $p|(m^{\phi(n)} - 1)$ and $q|(m^{\phi(n)} - 1)$, so, since p, q are distinct primes, it follows that $(pq)|(m^{\phi(n)} - 1)$, and so $m^{\phi(n)} \equiv_n 1$.

We now discuss very briefly two issues related to the security of RSA. The first one is that the primes p, q cannot be chosen “close” to each other. Note that

$$n = \left(\frac{p+q}{2}\right)^2 - \left(\frac{p-q}{2}\right)^2.$$

Since p, q are close, we know that $s := \frac{p-q}{2}$ is small, and $t := \frac{p+q}{2}$ is

only slightly larger than $n^{\frac{1}{2}}$, and $t^2 - n = s^2$ is a perfect square. So we try the following candidate values for t :

$$\lceil n^{\frac{1}{2}} \rceil, \lceil n^{\frac{1}{2}} \rceil + 1, \lceil n^{\frac{1}{2}} \rceil + 2, \dots$$

until $t^2 - n$ is a perfect square s^2 . Clearly, if s is small, we will quickly find such a t , and then $p = t + s$ and $q = t - s$.

The second issue is the following: suppose that Eve can compute $\phi(n)$ from n . Then she can easily compute the primes p, q (of course, if she can compute $\phi(n)$ she can directly compute l , and she does not need p, q). To see this note that $\phi(n) = \phi(pq) = (p-1)(q-1)$. Then,

$$\begin{aligned} p + q &= n - \phi(n) + 1 \\ pq &= n, \end{aligned} \tag{6.2}$$

and from these two equations,

$$(x - p)(x - q) = x^2 - (p + q)x + pq = x^2 - (n - \phi(n) + 1)x + n.$$

Thus, we can compute p, q by computing the roots of this last polynomial, and using the quadratic formula $x = (-b \pm \sqrt{b^2 - 4ac})/2a$, we obtain that p, q are

$$\frac{(n - \phi(n) + 1) \pm \sqrt{(n - \phi(n) + 1)^2 - 4n}}{2}.$$

Suppose that Eve is able to compute l from n and k . If Eve knows l , then she knows that whatever $\phi(n)$ is, it divides $kl - 1$, so she has equations (6.2) but with $\phi(n)$ in the first equation replaced by $(kl - 1)/a$, for some unknown a . There is a randomized polytime procedure to find the appropriate a , and obtain p, q , but we do not describe it here.

If Eve is able to factor she can obviously break RSA; on the other hand, if Eve can break RSA (by computing l from n, k), then she would be able to factor in randomized polytime. Conceivably, Eve may be able to break RSA *without* computing l , so these observation do not imply that breaking RSA is as hard as factoring.

6.5 Answers to selected problems

Exercise 6.4. To see why this is true, assume that $\gcd(a, p) \neq 1$. By the proposition 8.4 we know that if $\gcd(a, p) \neq 1$, then a does not have an inverse in \mathbb{Z}_p . Thus, it is not possible for $a^{(p-1)} \equiv 1 \pmod{p}$ to be true, since then it would follow that $a \cdot a^{(p-2)} \equiv 1 \pmod{p}$, and hence a would have a (multiplicative) inverse.

Exercise 6.6. To see why $t^{s \cdot 2^j} \not\equiv \pm 1 \pmod{n}$ observe the following: suppose that $a \equiv -1 \pmod{q}$ and $a \equiv 1 \pmod{r}$, where $\gcd(q, r) = 1$. Suppose that $n = qr|(a+1)$, then $q|(a+1)$ and $r|(a+1)$, and since $r|(a-1)$ as well, it follows that $r|[(a+1) - (a-1)]$, so $r|2$, so $r = 2$, so n must be even, which is not possible since we deal with even n 's in line 1 of the algorithm.

Exercise 6.7. Showing that S_1, S_2 are subgroups of \mathbb{Z}_n^* is easy; it is obvious in both cases that 1 is there, and closure and existence of inverse can be readily checked.

To give the same proof without group theory, we follow the cases in the proof of theorem 6.5. Let t be the witness constructed in case 1. If d is a (stage 3) nonwitness, we have $d^{p-1} \equiv 1 \pmod{p}$, but then $dt \pmod{p}$ is a witness. Moreover, if d_1, d_2 are distinct (stage 3) nonwitnesses, then $d_1 t \not\equiv d_2 t \pmod{p}$. Otherwise, $d_1 \equiv_p d_1 \cdot t \cdot t^{p-1} \equiv_p d_2 \cdot t \cdot t^{p-1} \equiv_p d_2$. Thus the number of (stage 3) witnesses must be at least as large as the number of nonwitnesses.

We do the same for case 2; let d be a nonwitness. First, $d^{s \cdot 2^j} \equiv \pm 1 \pmod{p}$ and $d^{s \cdot 2^{j+1}} \equiv 1 \pmod{p}$ owing to the way that j was chosen. Therefore $dt \pmod{p}$ is a witness because $(dt)^{s \cdot 2^j} \not\equiv \pm 1 \pmod{p}$ and $(dt)^{s \cdot 2^{j+1}} \equiv 1 \pmod{p}$.

Second, if d_1 and d_2 are distinct nonwitnesses, $d_1 t \not\equiv d_2 t \pmod{p}$. The reason is that $t^{s \cdot 2^{j+1}} \equiv 1 \pmod{p}$. Hence $t \cdot t^{s \cdot 2^{j+1} - 1} \equiv 1 \pmod{p}$. Therefore, if $d_1 t \equiv d_2 t \pmod{p}$, then $d_1 \equiv_p d_1 t \cdot t^{s \cdot 2^{j+1} - 1} \equiv_p d_2 t \cdot t^{s \cdot 2^{j+1} - 1} \equiv_p d_2$. Thus in case 2, as well, the number of witnesses must be at least as large as the number of nonwitnesses.

6.6 Notes

It was the randomized test for primality that stirred interest in randomized computation in the late 1970's. Historically, the first randomized algorithm for primality was given by [Solovay and Strassen (1977)]; a nice self-contained exposition of this algorithm can be found in [Papadimitriou (1994)][§ 11.1], and another in [von zur Gathen and Gerhard (1999)][§ 18.5].

R. D. Carmichael first noted the existence of the Carmichael numbers in 1910, computed fifteen examples, and conjectured that though they are infrequent there were infinitely many. In 1956, Erdős sketched a technique for constructing large Carmichael numbers ([Hoffman (1998)]), and a proof was given by [Alford *et al.* (1994)] in 1994.

The first three Carmichael numbers are 561, 1105, 1729, where the last