

## Exponentiation ciphers

Choose a large integer  $n$  for modulus. Encode plaintext as (blocks)  $0 \leq M < n$ . Encipher  $M$  as  $C = E(M) = M^e \bmod n$ . Decipher  $C$  as  $M = D(C) = C^d \bmod n$ .

This works, that is,  $D(E(M)) = M$  for all  $M$  in  $0 \leq M < n$ , provided that  $ed \equiv 1 \pmod{\phi(n)}$  since  $M^{\phi(n)} \equiv 1 \pmod{n}$  by Euler's Theorem. (Proof: Write  $ed = t\phi(n) + 1$  for some integer  $t$ .) This implies that  $e$  is relatively prime to  $\phi(n)$ .

## The RSA public-key cipher

Rivest-Shamir-Adleman.

Let  $n = pq$  be the product of two large primes. Then  $\phi(n) = \phi(pq) = (p - 1)(q - 1)$ , so  $ed \equiv 1 \pmod{(p - 1)(q - 1)}$ .

Each user of RSA has her own set of keys: Make  $n$  and  $e$  public, but keep  $d$  secret. The factors  $p$  and  $q$  are not needed after  $e$  and  $d$  are computed, but in any case should not be revealed.

If many users wish to communicate securely in pairs, then RSA requires fewer total keys to be stored than Pohlig-Hellman.

Preliminary cryptanalysis: Since  $n$  is public and one can easily compute  $d$  from  $e$  and the factors of  $n$ , a direct approach to breaking RSA is to factor  $n$ . For a modulus  $n$  of 300 decimal digits, this is too hard for current algorithms and computers.

## RSA Signatures

RSA has no direct authentication: Anyone can send any message to you and claim it came from anyone. However, one can sign RSA messages as follows:

Use the same notation for enciphering and deciphering algorithms:  $E_A$ ,  $D_B$ , etc. Alice can sign (and encipher) a message  $M$  to Bob by sending  $C = E_B(D_A(M))$  to Bob. Bob can decipher  $C$  by applying  $D_B$  to it (to get  $D_A(M)$ ) and then check the signature by applying  $E_A$  to the latter.

Note that Bob's cipher algorithms do not commute with Alice's because the modulus is different. Thus the order in which Bob applies the operations to  $C$  matters: Bob must do  $D_B$  first and then  $E_A$  second.

There is another problem caused by the different moduli.  $D_A$  and  $E_A$  do arithmetic modulo Alice's modulus  $n_A$  while  $E_B$  and  $D_B$  do arithmetic modulo Bob's modulus  $n_B$ . This works fine if  $n_A < n_B$  but part of the message will be lost if  $n_A > n_B$ .

There are three ways to solve this problem:

1. Re-block the message after  $D_A$  is applied.
2. Enforce an arbitrary threshold  $T$  and let every RSA user  $A$  have two complete sets of RSA keys, one with  $n_{A_1} < T$  and one with  $n_{A_2} > T$ . The keys with the smaller modulus  $n_{A_1}$  are used for signing messages from  $A$  and the keys with the larger modulus  $n_{A_2}$  are used to encipher messages going to  $A$ .

3. A more elegant solution is for Alice to sign (and encipher) a message  $M$  to Bob by sending  $C = E_B(D_A(M))$  to Bob when  $n_A < n_B$ , and by sending  $C = D_A(E_B(M))$  to Bob when  $n_A > n_B$ . In either case, Bob undoes these operations in reverse order.

What if Alice later denies sending  $M$  and Bob goes to an independent judge to prove that  $M$  bears Alice's signature? In the first case ( $n_A < n_B$ ), Bob gives the judge  $M$  and  $X = D_B(C)$ , the judge computes  $M' = E_A(X)$  and tests whether  $M' = M$ . If so, the judge rules that Alice signed  $M$ . In the second case ( $n_A > n_B$ ), Bob gives the judge  $M$  and  $C$ , the judge computes  $X' = E_B(M)$  and  $X' = E_A(C)$  and tests whether  $X' = X$ . If so, the judge rules that Alice signed  $M$ .

## More RSA Cryptanalysis

So far we have described the “textbook” RSA. If you can factor the modulus, you can break it. But is this the only way to break it?

Let  $\text{GenModulus}$  be a polynomial-time algorithm that, on input  $1^n$ , outputs  $(N, p, q)$ , where  $N = pq$  and  $p$  and  $q$  are  $n$ -bit primes except with probability negligible in  $n$ . (Later we will say more about how to construct such primes.)

Factoring experiment  $\text{Factor}_{\mathcal{A}, \text{GenModulus}}(n)$ :

1. Run  $\text{GenModulus}(1^n)$  to get  $(N, p, q)$ .
2.  $\mathcal{A}$  is given  $N$  and outputs  $p', q'$ .
3. The output of the experiment is 1 if  $p' > 1$ ,  $q' > 1$  and  $N = p' \cdot q'$ , and 0 otherwise.

Of course, if the output of  $\text{Factor}_{\mathcal{A}, \text{GenModulus}}(n)$  is 1, then  $\{p', q'\} = \{p, q\}$  except with negligible probability (in  $n$ ).

**Definition:** We say that factoring is hard with respect to  $\text{GenModulus}$  if for all PPT  $\mathcal{A}$  there exists a negligible function  $\text{negl}$  such that

$$\Pr[\text{Factor}_{\mathcal{A}, \text{GenModulus}}(n) = 1] \leq \text{negl}.$$

The *factoring assumption* is that there is a  $\text{GenModulus}$  with respect to which factoring is hard.

One way to construct a suitable  $\text{GenModulus}$  is to generate two random  $n$ -bit primes and let  $N$  be their product.

Factoring integers has been studied for hundreds of years. It probably is hard.

The problem of distinguishing prime numbers from composites, and of resolving composite numbers into their prime factors, is one of the most important and useful in arithmetic... The dignity of science seems to demand that every aid to the solution of such an elegant and celebrated problem be zealously cultivated.

— Carl Freidrich Gauss, *Disquisitiones Arithmeticae*, 1801.

Suppose, for example, that two 80-digit numbers  $p$  and  $q$  have been proved prime... Suppose further that the cleaning lady gives  $p$  and  $q$  by mistake to the garbage collector, but that the product  $pq$  is saved. It must be felt as a defeat for mathematics that, in these circumstances, the most promising approaches [for finding  $p$  and  $q$ ] are searching the garbage dump and applying mnemo-hypnotic techniques.

— Hendrik W, Lenstra, Jr. *Computational Methods in Number Theory*, 1981.

Breaking RSA is a different problem from factoring.

Let GenRSA be a PPT algorithm that, on input  $1^n$ , outputs  $(N, e, d)$ , where  $N$  is the product of two  $n$ -bit primes,  $e$  is a positive integer  $< N$  with  $\gcd(e, \phi(N)) = 1$ , and  $d$  is a positive integer  $< N$  with  $ed \equiv 1 \pmod{\phi(N)}$ . The algorithm is allowed to fail with negligible probability.

The RSA experiment  $\text{RSA-inv}_{\mathcal{A}, \text{GenRSA}}(n)$ :

1. Run  $\text{GenRSA}(1^n)$  to get  $(N, e, d)$ .
2. Choose  $1 < y < N$  at random with  $\text{gcd}(y, N) = 1$ .
3.  $\mathcal{A}$  is given  $N, e, y$  and outputs  $x$  in  $0 < x < N$ .
4. The output of the experiment is 1 if  $x^e \equiv y \pmod{N}$  and 0 otherwise.

**Definition:** We say that the RSA problem is hard with respect to  $\text{GenRSA}$  if for all PPT  $\mathcal{A}$  there exists a negligible function  $\text{negl}$  such that

$$\Pr[\text{RSA-inv}_{\mathcal{A}, \text{GenRSA}}(n) = 1] \leq \text{negl}.$$

## Outline of Algorithm GenRSA

Input: Security parameter  $1^n$

Output:  $N, e, d$  as above.

$(N, p, q) \leftarrow \text{GenModulus}(1^n)$

$\phi(N) := (p - 1)(q - 1)$

Find  $e$  with  $\text{gcd}(e, \phi(N)) = 1$

compute  $d := e^{-1} \pmod{\phi(N)}$ .

Return  $N, e, d$

With GenRSA as above, for which GenModulus is the RSA problem hard?

If  $N$  can be factored, then one can compute  $\phi(N)$  and  $d$  as in GenRSA. So if GenRSA is to be hard, then it must be hard to factor the  $N$  produced by GenModulus. Thus, the RSA problem is not harder than factoring. But might it be easier?

When  $N$  is the product of exactly two primes, one can factor  $N$  easily if  $N$  and  $\phi(N)$  are both known. So factoring  $N$  and computing  $\phi(N)$  are equally hard problems.

Also, given  $N$ ,  $e$ ,  $d$  with  $ed \equiv 1 \pmod{\phi(N)}$ , one can factor  $N$  in PPT.

There is no known proof that there is no other way to solve the RSA problem.