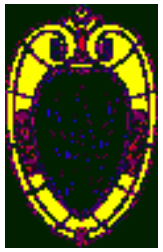


CS/ECE/Math 435
Introduction to Cryptography

Professor Chris DeMarco
Department of Electrical & Computer Engineering
University of Wisconsin-Madison

Spring Semester 2021

RSA System differs from previously studied encryption schemes in another important way: generation of keys (focus on p, q) costs significant computation.

Best strategics relic on selection of upper bound on allowable range of p, q; denote this bound X. Then candidate p, q values selected from integers $[2, X]$, tested for "primeness."

Issues / Questions

1) What % of integer values in $[2, x]$ are prime? i.e., what is probability of sampling a prime from this range?

2) What is the computational cost of testing a candidate p to ascertain prime/not prime

Issue (1) addressed by the Prime Number Theorem, which states $\{$ # of primes $\leq x\}$ approaches $\frac{x}{\log(x)}$ for large x.

$\Rightarrow \{$% of primes $\leq x\}$

or

$\left\{\begin{array}{l}\text{probability of selecting}\\ \text{a } p \text{ that is prime}\\ \text{from } [2, x]\end{array}\right\} \cong \frac{1}{\log(x)}$

Regarding issue (2), interesting trick to reduce (average) computational cost.

At computational cost less than a "full" prime test, we can employ a probabilistic / randomized test (dependent on random selection of another integer).

Given p that is truly prime; randomized algorithm <u>always</u> classifies it as prime.

Given p that truly is not prime, randomized algorithm has 75% probability of classifying it as not prime, but 25% probability of incorrectly classifying as prime.

Usefulness lies in its use as a screening test ⇒ "throw away" any candidate p that randomized algorithm classifies as not prime.

Cryptographic Attacks on RSA

At core of most attacks on RSA are attempts to recover prime factors p and q from the public key value of N.

Key Question: What is the computational cost of identifying prime factors of (large) integer N?

Number Field Sieve factorization

algorithm has number of operations

$$\leq \exp\left[\left(C + o(1)\right) \cdot \left(\log N\right)^{1/3} \cdot \left(\log \log N\right)^{2/3}\right]$$

$C \cong 1.92$

$o(1)$: value approaching zero for large N

U.S. National Institute for Standards and Technology (NIST) recommends N of 2048 bits or greater.

Drawback for symmetric encryption schemes include the need for some prior, secure communication of key. Seemingly infeasible to use in an environment in which all communication is public.

But ... adapting RSA-like ideas, we can create a mechanism for communicating a secret key over an entirely public network (and than this key can be used in a symmetric encryption scheme).

Diffie-Hellman Key Exchange

Relies on a network protocol manager, providing framework for communication between user A and user B.

- Protocal manager publishes a (large) prime number p, and a "special" associated integer $g \in \mathbb{Z}_p^*$.

The value g is termed the "generator." More on g shortly.

- A randomly selects $x \in \mathbb{Z}_p$, and (publically) sends $S_A = \text{mod}(g^x, p)$ to B. Note: The value of x remains internally secret to A.

- B randomly selects $y \in \mathbb{Z}_p$, and sends $S_B = \text{mod}(g^y, p)$ to A.

A computes:

$$K_A = \text{mod}\left[\left(\underbrace{\text{mod}(g^Y, p)}_{\text{this is } S_B \text{ that A received}} \right)^{X}, p \right]$$

(X: internally secret to A)

These are equal!

B computes:

$$K_B = \text{mod}\left[\left(\underbrace{\text{mod}(g^X, p)}_{\text{this is } S_A \text{ that B received}} \right)^{Y}, p \right]$$

(Y: internally secret to B)

Users A and B now have shared key $K = K_A = K_B$, without K itself having ever been communicated.

Cryptoanalysis Attacks on this scheme:

Essentially, attacker seeks the mapping

$$\underbrace{(g^x, g^y)}_{\text{publically communicated}} \longrightarrow \underbrace{g^{xy}}_{K = K_A = K_B}$$

} All computed mod p

At present, best know algorithms for attacking this protocol rest on computing discrete logarithms; i.e. given $\mod(g^x, p)$, compute x.

Recall that for p prime, $\mathbb{Z}_p^* = \{1, 2, \ldots p-1\}$ (i.e., all elements of $\mathbb{Z}_p$, except 0).

FACT: There exists a $g \in \mathbb{Z}_p^*$, known as the "generator," or "primitive root,"

such that for every $y \in \mathbb{Z}_p^*$, there exists a $x \in \mathbb{Z}_p^*$ satisfying $y = \mod(g^x, p)$. The value y is the discrete logarithm of y with respect to base g.

Key point:

- $x \longmapsto \mod(g^x, p)$ is a low cost computation
- $y \to x$, the discrete logarithm is a high cost computation

Example: For prime $p = 11$, $g = 7$ is a primitive root

$x \in \mathbb{Z}_p^*$	1	2	3	4	5	6	7	8	9	10
$\text{mod}(g^x, p)$	7	5	2	3	10	4	6	9	8	1

we recover all elements of $\mathbb{Z}_p^*$

How to identify a primitive root/ generator g for D-H Key Exchange algorithm?

Theorem:

$g \in Z_p^*$ is a primitive root if and only if

$$\mod\left(g^{(p-1)/q}, p\right) \neq 1$$

for each integer q that is a prime divisor of $(p-1)$.

Example: We showed $g = 7$ is a generator for $\mathbb{Z}_p^*$, $p = 11$.

To demonstrate the Theorem, observe $p - 1 = 10 = 2 \cdot 5$. Hence $q = 2, 5$. Compute:

$$\mathrm{mod}\left(7^{10/2}, 11\right) = 10 \neq 1$$

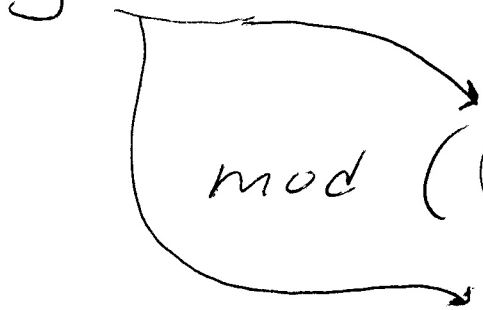
$$\mathrm{mod}\left(7^{10/5}, 11\right) = 5 \neq 1$$

↑ our candidate g

passes the test prescribed by Theorem

But consider a candidate generator,

$g = 3$ for $p = 11$.



$$\mod\left(3^{10/2}, 11\right) = 1 \quad !$$

$$\mod\left(3^{10/5}, 11\right) = 9$$

fails test prescribed by Theorem $\Rightarrow$

$g = 3$ is <u>not</u> a generator/primitive root for $\mathbb{Z}_{11}^*$

Added "tricks" to reduce computational cost in identifying p and g for D-H Key Exchange:

Factoring $p-1$ (to obtain g) is computationally expensive, for an arbitrary choice of p prime.

- One trick selects a smaller prime q, sets $p = 2q + 1$,

$(\Rightarrow (p-1) = 2 \cdot q)$.

- Focus choice first on selecting a candidate $(p-1)$ in known factored form, then test associated p for primality.