Fall 2006
Chris Christensen
MAT/CSC 483

# Introduction to RSA and to Authentication

The most famous of the public key cryptosystem is RSA which is named after its three developers Ron Rivest, Adi Shamir, and Leonard Adleman. At the time of the algorithm's development (1977), the three were researchers at the MIT Laboratory for Computer Science. Their algorithm was first announced in Martin Gardner's "Mathematical Games" column in the August, 1977, *Scientific American*. Their formal paper "A method for obtaining digital signatures and public-key cryptosystems" was published in 1978 in the *Communications of the Association for Computing Machinery*.

## A Bit of Mathematics

Recall that public key cryptosystems are designed around mathematical processes that are (relatively) easy to do but for which the inverse is very difficult to do without additional information (the private key).

For RSA, the (relatively) easy part is multiplying pairs of large primes. The hard part is factoring large integers.

The basic mathematics goes back to the French mathematician Pierre de Fermat (1601 – 1665). A result known as Fermat's Little Theorem states that for any prime number *r* and any integer *a* not divisible by *r*, $a^{r-1} \equiv 1 \bmod r$.

Actually, what is directly used is a corollary of a more general result due to the Swiss mathematician Leonard Euler (1707 –1783). In the form in which it is used in the RSA cryptosystem, this corollary states that if *r* and *s* are prime numbers and *a* is an integer that has no common divisors with either *r* or *s*, then $a^{(r-1)(s-1)} \equiv 1 \bmod rs$.

Here is the RSA cryptosystem.

## Encryption

The receiver, say Josh, needs to construct two large prime numbers denoted *r* and *s*. The product of *r* and *s* is denoted $n = rs$. In practice, the prime numbers *r* and *s* are each about the same number of digits long and are selected so that their product *n* is 200 or more digits long. For our examples, we will use small primes, say $r = 53$, $s = 61$, and $n = rs = 3233$. Josh then selects an integer *e* that has a multiplicative inverse modulo $(r-1)(s-1)$. This is called the encryption exponent. For our example, we will choose $e = 37$ which has a multiplicative inverse modulo

$$(r-1)(s-1) = (53-1)(61-1) = 52 \times 60 = 3120.$$

Josh then constructs (using the Euclidean algorithm) the multiplicative inverse of *e* modulo $(r-1)(s-1)$; that number is the decryption exponent and is denoted *d*. For our example, $d = 253$.

Josh's public key consists of two numbers – *n* and *e*. These two numbers are available to anyone who might want to send an encrypted message to Josh. Josh keeps the decryption exponent *d* secret; it is his private key. The private key is used by Josh to decrypt any messages sent to him that have been encrypted with his public key.

For someone to cryptanalyze a message sent to Josh using his public key, they would have to be able to construct the private key *d* which is the inverse of *e* modulo $(r-1)(s-1)$. This can be easily done if *r* and *s* are known, but to get *r* and *s*, the cryptanalyst would need to be able to factor the other half of Josh's public key *n*. That is the problem. There is no efficient way to find *r* and *s* even if *n* is known (provided that *n* is large and *r* and *s* are approximately the same size).

Say, Brad wants to send the message `Midway` to Josh using Josh's public key. First, the message must be converted to a string of numbers. In practice, ASCII (American Standard Code of Information Interchange) numbers are usually used; we will use our usual `a = 01`, `b = 02`, …, `z = 26`. So, `Midway` would be converted to the string `130904230125`. The string is broken into blocks each of which is small than the modulus -- 3233 in our case. Four-digit blocks will work.

```
1309 0423 0125
```

Ciphertext is obtained by raising each plaintext block to the exponent *e* modulo *n*.

$$p^e \bmod n = C$$

$$1309^{37} \bmod 3233 = 3027$$

$$0423^{37} \bmod 3233 = 0635$$

$$0125^{37} \bmod 3233 = 1699$$

The message that Brad transmits to Josh is

```
3027   0635   1699
```

Decryption

How does Josh decrypt the message from Brad?

Decryption depends on Euler's corollary to Fermat's Little Theorem. Recall that Euler's corollary says that $a^{(r-1)(s-1)} \equiv 1 \bmod rs$. Because $n = rs$, this says that $a^{(r-1)(s-1)} \equiv 1 \bmod n$. Now *e* and *d* are inverses modulo $(r-1)(s-1)$; i.e., $ed \equiv 1 \bmod (r-1)(s-1)$. Another way of saying this is that *ed* is 1 plus a multiple of $(r-1)(s-1)$; i.e., $ed = 1 + k(r-1)(s-1)$.

Ciphertext is $C \equiv p^e \bmod n$. When Josh receives the ciphertext blocks, he raises each of them to the power *d*, his decryption exponent.

$$C^d \equiv (p^e)^d \equiv p^{ed}$$

$$\equiv p^{1+k(r-1)(s-1)} \equiv p^1 p^{k(r-1)(s-1)}$$

$$\equiv p\left(p^{(r-1)(s-1)}\right)^k \equiv p\left(1\right)^k \equiv p \bmod n$$

which is back to plaintext.

Here is the decryption for our example.

$$C^d \bmod n = p$$

$$3027^{253} \bmod 3233 = 1309$$

$$0635^{253} \bmod 3233 = 0423$$

$$1699^{253} \bmod 3233 = 0125$$

So, the message decrypts to `1309 0423 0125` which in plaintext is `Midway`.

## Authentication

Public key encryption systems solve the problems of key distribution (the sending key is “published” and is available to anyone who wants to use it) and of key security (only the private key needs to be kept secure, and the receiver is the only person with that key), and public key encryption systems like RSA are secure, but public key encryption systems create a new problem – authentication. How does the receiver know that a message really came from the person who “signed” it?

How, for example, does Josh know that the message above came from Brad? Brad could “sign” the message; but, because Josh's encryption key is public, anyone could create a message, sign Brad's name, encrypt it with Josh's public key, and send it to Josh. How does Josh know that it really came from Brad and not an imposter?

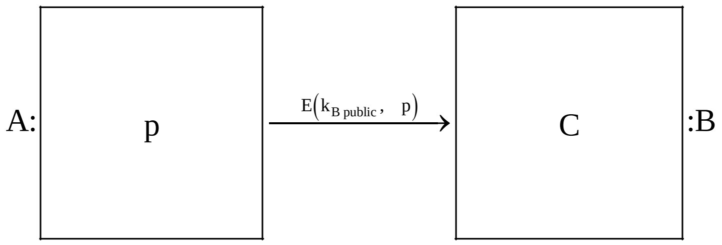
For classical ciphers, after a key has been exchanged by two people, a reasonable expectation by one of those people is that if a ciphertext message is received and the message decrypts to plaintext using the key that was exchanged, then the message came from the person with whom the key had been exchanged. (Sure, there are exceptions like when a key is stolen, torture is used to get the key or to force the sending of a message, etc.; but, in general, correct decryption implies authentication.)

However, this is not the case for public key ciphers.  Because everyone has access to the public key, anyone can send a message and "sign" any name that they want.  So, for public key ciphers authentication *is* a problem.

We won't go into the details of implementing ideas for authenticating messages; we will only discuss in general how authentication might be achieved in a public key situation.

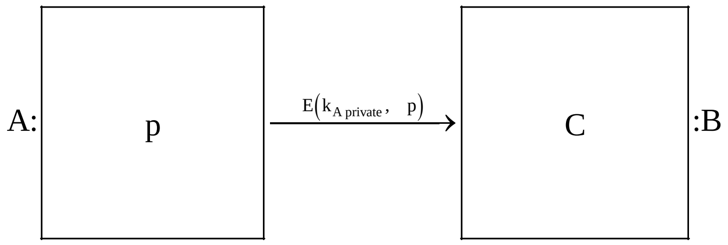
Suppose that Alice is sending a message to Bob.

**Security** is provided by Alice using Bob's public key.



But, are we sure that the message came from Alice?  Or might is be that someone else sent the message and "signed" Alice's name to it?  "Usual" public key encryption (Alice using Bob's public key to encipher) provides security but not authentication.

Alice can **authenticate** a message by sending it after encryption with *her private* key.  Alice is the only person who knows her private key.



Bob knows Alice's public key, Alice's public key is the inverse of her private key, and Alice is the only person who knows her private key.  So, if the message decrypts using Alice's public key, then Bob can be sure that the message came from Alice.  (Sure, some exceptions, but in general.)  Of course, this scheme provides no security because everyone has access to Alice's public key, and, therefore, everyone can decrypt the message.

“Reverse” public key encryption (Alice using her own private key to encipher) provides authentication but not security.

Of course, the message could be encrypted twice – once for security (using Bob’s public key) and once for authentication (using Alice’s public key); e.g.,

$$\begin{tikzcd}
\text{A:}\ \boxed{\quad p \quad} \arrow[r, "{E(k_{\text{B public}},\ p)}"] & \boxed{\qquad\quad} \arrow[r, "{E(k_{\text{A private}},\ p)}"] & \boxed{\quad C \quad}\ \text{:B}
\end{tikzcd}$$

The problem here is time. Public key algorithms are not fast; encrypting twice would make encryption an unreasonably long process for many messages.

There are various ways to implement this sort of encryption – security and authentication – but they often are the equivalent of encrypting twice.

A more efficient method is to use a **cryptographic hash function**.

A cryptographic hash function is a **message digest**; in some sense, the message is condensed. A very trivial hash function is:

$$h(m) = \begin{cases} 1 & \text{If the message contains an odd number of characters} \\ 0 & \text{If the message contains an even number of characters} \end{cases}$$

Another, used with German Enigma messages (not as a hash function but rather as an error check) was to append to the message the number of characters in the message. h(m) = the number of characters in the message.

Well, each of these tells us a little about the message but not much. A better digest is desirable.

But, in addition to being a digest of the message, a cryptographic hash function (for security reasons) should have some other properties:

It should be **one-way**.  Knowing h(m) it should not be feasible to determine m.

It should be **strongly collision free**.  It should be very unlikely that $h(m_1)=h(m_2)$ if $m_1 \neq m_2$.  Of course, the number of messages is much larger than the number of digests; so, collisions will occur but collisions should be unlikely.

There are two widely used families of cryptographic hash functions – the MD family (MD = message digest) and the SHA family (SHA = secure hash algorithm).  Rivest and RSA laboratories developed MD4 and now MD5. The original MD was never published; MD2 was the first of the family to appear, and it was followed by MD4.  The NSA developed SHA-1 and SHA-2.  Around February 2005, problems with SHA-1 became public.

Essentially hash functions allows authentication to occur without double encryption of the entire message.

Alice and Bob must agree on a hash function.  Then Alice can (for security) send her message using Bob's public key.  Also, she creates a hash of the plaintext and (for authentication) sends it using her private key.  Using his private key, Bob decrypts the ciphertext enciphered with his public key and creates a hash of the plaintext using the hash function that he and Alice have agreed to use.  Bob also decrypts the ciphertext of the hash function using Alice's public key.  The two hashes should be the same.  If they are, Bob can assume that the message came from Alice.

## More about the *Scientific American* Column

In his 1977 column, Martin Gardner posed a $100 ciphertext challenge.

```
9686   9613   7546   2206   1477   1409   2225   4355
8829   0575   9991   1245   7431   9874   6951   2093
0816   2982   2514   5708   3569   3147   6622   8839
8962   8013   3919   9055   1829   9451   5781   5254
```

The ciphertext was generated by the MIT team from a plaintext (English) message using $e = 9007$ and $n =$

114,381,625,757,888,867,669,235,779,976,146,612,010,218,296,721,
242,362,562,561,842,935,706,935,245,733,897,830,597,123,563,958,
705,058,989,075,147,599,290,026,879,543,541.

On April 26, 1994, a team of about 600 volunteers announced that they had completed the factorization of $n$. The two factors ($r$ and $s$) are:

3,490,529,510,847,650,949,147,849,619,903,898,133,417,764,638,
493,387,843,990,820,577

and

32,769,132,993,266,709,549,961,988,190,834,461,413,177,642,967,
992,942,539,798,288,533

Hence the team could determine the decryption exponent and decipher the message. (The message is: *the magic words are squeamish ossifage.*)

As mathematicians do more research about factoring, the algorithms are improving. It takes longer and longer keys ($n$ has 300 or more digits for the most secure transmissions) to guarantee security.

In 1994, a factoring algorithm for quantum computers Shor's algorithm was developed. No quantum computer yet exists, but if a quantum computer can be built, all RSA-encrypted messages are readable.

## Exercises

Josh's public key: $n$ = 3233 and $e$ = 37.
Josh's private key: $d$ = 253.

Brad's public key: $n$ = 2773 and $e$ = 17.
Brad's private key: $d$ = 157.

1. Encrypt the message `Purple` using Josh's public key.

2. Encrypt the message `Fermat` using Brad's public key.

3. Decrypt (using Brad's private key) the following ciphertext message which was encrypted using Brad's public key:

```
1643   0639   2556
```

4 RSA depends on long keys for security. Consider the case of a small key. Say, Beth has public key $n$ = 2701 and $e$ = 1037. Josh sends her the following message: `0642 2584 1992`. Find the decryption exponent $d$ and cryptanalyze the message.

5 Factor the following $n$s into prime factors:

5a $n$ = 91.
5b $n$ = 1850.
5c $n$ = 105400.
5d $n$ = 1678.
5e $n$ = 736163.

Discuss why it is important for the security of the RSA algorithm that $n$ be the product of two large primes of nearly equal size.