Applied Cryptography and Network Security

Adam J. Lee
adamlee@cs.pitt.edu
6111 Sennott Square

Lecture #8: RSA
January 30, 2014

University of Pittsburgh

Didn't we learn about RSA last time?

During the last lecture, we saw what RSA does and learned a little bit about how we can use those features

Our goal today will be to explore

- Why RSA actually works
- Why RSA is efficient* to use
- Why it is reasonably safe to use RSA

In short, it's a details day…

Note: Efficiency is a general term ☺

RSA Overview / Roadmap

How do we choose large, pseudo-random primes?!

Key generation:

- Choose two large prime numbers p and q, compute $n = pq$
- Compute $\varphi(n) = (p-1)(q-1)$
- Choose an integer d such that $\gcd(d, \varphi(n)) = 1$
- Calculate e such that $ed \equiv 1 \pmod{\varphi(n)}$
- Public key: n, e
- Private key: p, q, d

Why is $\varphi(n) = (p-1)(q-1)$

How can we do this?

This seems tricky, too

Isn't this expensive?

Usage:

- Encryption: $M^e \pmod{n}$
- Decryption: $C^d \pmod{n} = M^{ed} \pmod{n} = M^{k\varphi(n)+1} \pmod{n} = M^1 \pmod{n} = M$

Why does this work?

Before we can do anything, we need a few large, pseudo-random primes

If our numbers are small, primality testing is pretty easy

- Try to divide n by all numbers less than $\sqrt{n}$
- The Sieve of Eratosthenes is a general extension of this principle

RSA requires big primes, so brute force testing is not an option (Why?)

To choose the types of numbers that RSA needs, we instead use a probabilistic primality testing method test : $Z \times Z \rightarrow \{T, F\}$

- test(n, a) = F means that n is composite based on the witness a
- test(n, a) = T means that n is probably prime based on the witness a

Often times with probability ≈ 1/2

To test a number n for primality:

1. Randomly choose a witness a
2. if test(n, a) = F, n is composite
3. if test(n, a) = T, loop until we're reasonably certain that n is prime

k repetitions means P[n composite] = 2^{-k}

Fermat's little theorem can help us!

Fermat's little theorem: Given a prime number p and a natural number a such that $1 \leq a < p$, then $a^{p-1} \equiv 1 \mod p$.

How does this help with primality testing?

- If $a^{p-1} \neq 1 \mod p$, then p is definitely composite
- If $a^{p-1} \equiv 1 \mod p$, then p is probably prime

Note: Some composite numbers will always pass this test (Yikes!)

- These are called Carmichael numbers
- Carmichael numbers are rare, but may still be found
- Other primality tests (e.g., Miller-Rabin) avoid detecting these numbers

This helps us test whether some number is prime. But how exactly does this help us generate RSA keys?

Putting it all together…

```
foundPrime = false
while (!foundPrime)
        let r = some large, odd, random number
        foundPrime = true
        for (iters = 0; iters < k; iters++)
                let a = random number less than r
                if (a^(r-1) ≠ 1 mod r)
                        foundPrime = false
                        break
return r
```

The prime number theorem tells us that, on average, the number of primes less than n is approximately n/ln(n)

- That is, P[n prime] ≈ 1/ln(n)
- Searching for a prime is hard, but not ridiculously so

RSA Overview / Roadmap

How do we choose large, pseudo-random primes?!

Key generation:

✔
- Choose two large prime numbers p and q, compute $n = pq$
- Compute $\varphi(n) = (p-1)(q-1)$
- Choose an integer d such that $\gcd(d, \varphi(n)) = 1$
- Calculate e such that $ed \equiv 1 \pmod{\varphi(n)}$
- Public key: n, e
- Private key: p, q, d

Why is $\varphi(n) = (p-1)(q-1)$

How can we do this?

This seems tricky, too

Isn't this expensive?

Usage:

- Encryption: $M^e \pmod{n}$
- Decryption: $C^d \pmod{n} = M^{ed} \pmod{n} = M^{k\varphi(n)+1} \pmod{n} = M^1 \pmod{n} = M$

Why does this work?

φ(n) is called Euler's totient function

Definition: The totient function, φ(n), counts the number of elements less than n that are relatively prime to n

For an RSA modulus n = pq, calculating φ(n) is actually pretty simple

Note that we need to consider each of the pq numbers ≤ n

- Clearly, all multiples of p share a common factor with n
  - There are q such numbers {p, 2p, 3p, …, qp}
- Similarly, all multiples of q share a common factor with n
  - There are p such numbers {q, 2q, 3q, …, pq}
- So, we have that φ{n} = pq – p – q + 1
- As a result, φ(n) = pq – p – q + 1 = (p-1)(q-1)

The +1 controls for subtracting pq twice

Note: Calculating φ(n) is easy because we know how to factor n!

RSA Overview / Roadmap

Key generation:

- ✔ Choose two large prime numbers p and q, compute $n = pq$
- ✔ Compute $\varphi(n) = (p-1)(q-1)$
- Choose an integer d such that $\gcd(d, \varphi(n)) = 1$
- Calculate e such that $ed \equiv 1 \pmod{\varphi(n)}$
- Public key: n, e
- Private key: p, q, d

Why is $\varphi(n) = (p-1)(q-1)$

How can we do this?

This seems tricky, too

Isn't this expensive?

Usage:

- Encryption: $M^e \pmod{n}$
- Decryption: $C^d \pmod{n} = M^{ed} \pmod{n} = M^{k\varphi(n) + 1} \pmod{n} = M^1 \pmod{n} = M$

Why does this work?

Review of greatest common divisors

Definition: Let a and b be integers, not both zero. The largest integer d such that $d \mid a$ and $d \mid b$ is called the greatest common divisor of a and b, and is denoted by $\gcd(a, b)$.

Note: We can (naively) find GCDs by comparing the common divisors of two numbers.

Example: What is the GCD of 24 and 36?

- Factors of 24: 1, 2, 3, 4, 6, 12
- Factors of 36: 1, 2, 3, 4, 6, 9, 12, 18
- ∴ gcd(24, 36) = 12

Wait. Aren't we dealing with numbers that are hard to factor?

Luckily, computing GCDs is not all that hard…

Intuition: Rather than computing the GCD of two big numbers, we can instead compute the GCD of smaller numbers that have the same GCD!

Interesting observation: $\gcd(x, y)$ is the same as $\gcd(x-y, y)$

Wait, what?

Divides

First, we must show that $d \mid x \land d \mid y \rightarrow d \mid (x - y)$

- If $d \mid x$ and $d \mid y$, then $x = kd$ and $y = jd$
- Then $x-y = kd - jd = (k-j)d$
- So, $d \mid x \land d \mid y \rightarrow d \mid (x - y)$

Ok, so d is a divisor of $(x - y)$, but is it the greatest divisor?

- The divisors of $(x - y)$ are a subset of of the divisors of x and the divisors of y
- Since $d = \gcd(x, y)$, it is the greatest of the remaining divisors

Euclid's algorithm optimizes this process!

Euclid's algorithm finds gcd(x,y) as follows:

- Set $r_{-1} = x$, $r_{-2} = y$, $n = 0$
- While r_{n-1} != 0
  - divide r_{n-2} by r_{n-1} to find q_n and r_n such that $r_{n-2} = q_n r_{n-1} + r_n$
  - $n = n + 1$
- $\gcd(x, y) = r_{n-2}$

Example: Computing gcd(414, 662)

n	q_n	r_n
-2	-	662
-1	-	414
0	1	248
1	1	166
2	1	82
3	2	2
4	41	0

That's all fine and good, but how does this help us compute $d \equiv 1 \bmod \varphi(n)$?

Method 1: Use Euclid's algorithm

- Choose a random d
- Use Euclid's algorithm to determine wither $\gcd(d, \varphi(n)) = 1$
- Repeat as needed

Method 2: We can just choose a large prime number $r > \max(p, q)$

Why does method 2 work?

- r is a prime number, so it has no divisors other than itself and 1
- r is larger than p and q, so $r \neq p$ and $r \neq q$

Note that in Method 2, d must be chosen from a large enough set that an adversary cannot simply find it through blind trial and error

RSA Overview / Roadmap

Key generation:

- ✔ Choose two large prime numbers p and q, compute $n = pq$
- ✔ Compute $\varphi(n) = (p-1)(q-1)$
- ✔ Choose an integer d such that $\gcd(d, \varphi(n)) = 1$
- Calculate e such that $ed \equiv 1 \pmod{\varphi(n)}$
- Public key: n, e
- Private key: p, q, d

How can we do this?

This seems tricky, too

Isn't this expensive?

Usage:

- Encryption: $M^e \pmod{n}$
- Decryption: $C^d \pmod{n} = M^{ed} \pmod{n} = M^{k\varphi(n)+1} \pmod{n} = M^1 \pmod{n} = M$

Why does this work?

It turns out that Euclid's algorithm can help us compute $e = d^{-1}$, too

If we maintain a little extra state, we can figure out numbers u_n and v_n such that $r_n = u_n x + v_n y$

If x and y are relatively prime, this will allow us to calculate x^{-1}

- $1 = u_n x + v_n y$ // If x and y are relatively prime, $r_n = 1$
- $u_n x = 1 - v_n y$ // Subtract $v_n y$ from both sides
- $u_n x \equiv 1 \bmod y$ // Definition of congruence
- So $u_n = x^{-1}$! // Definition of inverse

The extended Euclid's algorithm works as follows:

- Set $r_{-1} = y$, $r_{-2} = x$, $n = 0$, $u_{-2} = 1$, $v_{-2} = 0$, $u_{-1} = 0$, $v_{-1} = 1$
- While r_{n-1} != 0
  - divide r_{n-2} by r_{n-1} to find q_n and r_n such that $r_{n-2} = q_n r_{n-1} + r_n$
  - $u_n = u_{n-2} - q_n u_{n-1}$
  - $v_n = v_{n-2} - q_n v_{n-1}$
  - $n = n + 1$
- $\gcd(x, y) = r_{n-2} = u_{n-2}x + v_{n-2}y$

This makes $r_n = u_n x + v_n y$ for $n = -1$ and $n = -2$

How about an example?

Example: Find the inverse of 797 mod 1047

n	q_n	r_n	u_n	v_n
-2		797	1	0
-1		1047	0	1
0	0	797	1	0
1	1	250	-1	1
2	3	47	4	-3
3	5	15	-21	16
4	3	2	67	-51
5	7	1	-490	373

So, 1 = -490*797 + 373*1047

- -490*797 = 1 + (-373)1047
- -490*797 $\equiv$ 1 mod 1047
- In other words, -490 is the inverse of 797 mod 1047

RSA Overview / Roadmap

Key generation:

✔ • Choose two large prime numbers p and q, compute $n = pq$

✔ • Compute $\varphi(n) = (p-1)(q-1)$

✔ • Choose an integer d such that $\gcd(d, \varphi(n)) = 1$

✔ • Calculate e such that $ed \equiv 1 \pmod{\varphi(n)}$

- Public key: n, e
- Private key: p, q, d

This seems tricky, too

Usage:

- Encryption: $M^e \pmod{n}$
- Decryption: $C^d \pmod{n} = M^{ed} \pmod{n} = M^{k\varphi(n)+1} \pmod{n} = M^1 \pmod{n} = M$

Isn't this expensive?

Why does this work?

Isn't exponentiation really expensive?

Exponentiation can be sped up using a trick called successive squaring

```
int pow(int m, int e)
    if(e is even)
        return pow(m*m, e/2)
    else
        return m * pow(m, e – 1)
```

For example, consider computing 2^{15}

- Naive method: 2 * 2 * 2 * … * 2 = 32,768
- Fast method: $2^{15} = 2 * 4^7$

  $= 2 * 4 * 4^6$
  $= 2 * 4 * 16^3$
  $= 2 * 4 * 16 * 16^2$
  $= 2 * 4 * 16 * 256$
  $= 32,768$

O(e) multiplications

O(log(e)) multiplications

RSA Overview / Roadmap

Key generation:

- ✔ Choose two large prime numbers p and q, compute $n = pq$
- ✔ Compute $\varphi(n) = (p-1)(q-1)$
- ✔ Choose an integer d such that $\gcd(d, \varphi(n)) = 1$
- ✔ Calculate e such that $ed \equiv 1 \pmod{\varphi(n)}$
- Public key: n, e
- Private key: p, q, d

Usage:

- ✔ Encryption: $M^e \pmod{n}$
- Decryption: $C^d \pmod{n} = M^{ed} \pmod{n} = M^{k\varphi(n) + 1} \pmod{n} = M^1 \pmod{n} = M$

Isn't this expensive?

Why does this work?

Why *does* decryption work?

Note: Decryption will work if and only if $C^d \bmod n = M$

$C^d \bmod n = M^{ed} \bmod n$ // $C = M^e \bmod n$

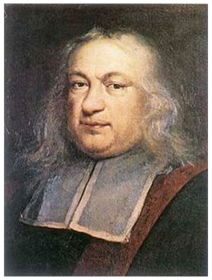
$= M^{k\varphi(n)+1} \bmod n$ // $ed \equiv 1 \bmod \varphi(n)$, so $ed = k\varphi(n) + 1$

$= M^1 \bmod n$ // ?!?

$= M \bmod n$ // $M^1 = M$

The only hitch in showing the correctness of the decryption process is proving that $M^{k\varphi(n)+1} \bmod n = M^1 \bmod n$

Fortunately, two smart guys can help us out with this…



Pierre de Fermat
160? - 1665

Leonhard Euler
1707 - 1783

First, we need to learn about the set Z_n^*

Definition: Z_n^* is the set of all integers relatively prime to n

Example: Z_{10}^*

×	1	3	7	9
1	1	3	7	9
3	3	9	1	7
7	7	1	9	3
9	9	7	3	1

Interesting note: $\forall$ a, b $\in Z_n^*$: ab $\in Z_n^*$

- a relatively prime to n means that $\exists$ $u_1, v_1 : u_1a + v_1n = 1$
- b relatively prime to n means that $\exists$ $u_2, v_2 : u_2b + v_2n = 1$
- Multiplying gives us $(u_1u_2)ab + (u_1v_2a + v_1u_2b + v_1v_2n)n = 1$

The above states that Z_n^* is closed under multiplication

This leads us to something called Euler's theorem

Theorem: $\forall\, a \in Z_n^* : a^{\varphi(n)} \equiv 1 \bmod n$

Proof:

- Multiply all $\varphi(n)$ elements of Z_n^* together, calling the product x
- Note that $x \in Z_n^*$, and has an inverse x^{-1}
- Now, multiply each element of Z_n^* by a and multiply each of the resulting elements together. This will give us $a^{\varphi(n)}x$.
- Multiplying each element of Z_n^* actually just rearranges these elements.
- As a result, we have that $a^{\varphi(n)}x = x$.
- If we divide both sides of the equation by x, we get that $a^{\varphi(n)} = 1$. ❑

Ok, so what does Euler's theorem have to do with RSA?

We can restate Euler's theorem so that it more clearly connects to RSA math

Theorem: $\forall\, a \in Z_n^*,\ k \in Z^+: a^{k\varphi(n)+1} \equiv a \bmod n$

Proof: $a^{k\varphi(n)+1} = a^{k\varphi(n)}a = a^{\varphi(n)k}a = 1^k a = a$ ❑

From Euler's theorem!

Now, in RSA

- All of our math is done mod n
- Our message space is chosen from elements of Z_n^*
- $ed \equiv 1 \bmod \varphi(n)$, so $ed = k\varphi(n) + 1$ for some k
- $\therefore M^{ed} \bmod n = M^{k\varphi(n)+1} \bmod n = M^1 \bmod n = M$

Decryption works!

RSA Overview / Roadmap

Key generation:

- ✔ Choose two large prime numbers p and q, compute $n = pq$
- ✔ Compute $\varphi(n) = (p-1)(q-1)$
- ✔ Choose an integer d such that $\gcd(d, \varphi(n)) = 1$
- ✔ Calculate e such that $ed \equiv 1 \pmod{\varphi(n)}$
- Public key: n, e
- Private key: p, q, d

Usage:

- ✔ Encryption: $M^e \pmod{n}$
- ✔ Decryption: $C^d \pmod{n} = M^{ed} \pmod{n} = M^{k\varphi(n)+1} \pmod{n} = M^1 \pmod{n} = M$

But why is RSA *safe* to use?

Now, why exactly is RSA safe to use?

In the original RSA paper*, the authors identify four avenues for attacking the mathematics behind RSA

1. Factoring n to find p and q
2. Determining φ(n) without factoring n
3. Determining d without factoring n or learning φ(n)
4. Learning to take e^{th} roots modulo n

As it turns out, all of these attacks are thought to be hard to do

- But you shouldn't take my word for it…
- Let's see why!

*R.L. Rivest, A. Shamir, and L. Adleman, A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, Communications of the ACM 21(2): 120-126, Feb. 1978.

It turns out that factoring is a hard* problem

First of all, why is factoring an issue?

- n is the public modulus of the RSA algorithm
- If we can factor n to find p and q, we can compute φ(n)
- Given φ(n) and e, we can easily compute the decryption exponent d

Fortunately, mathematicians believe that factoring numbers is a very difficult problem. History backs up this belief.

The fastest general-purpose algorithm for integer factorization is called the general number field sieve. This algorithm has running time:

$$O(e^{(c+o(1))(\log n)^{\frac{1}{3}}(\log \log n)^{\frac{2}{3}}})$$

Note: This running time is sub-exponential

- i.e., Factoring can be done faster than brute force
- This explains why RSA keys are larger than AES keys
  - RSA: Typically 1024-2048 bits
  - AES: Typically 128 bits

What about computing $\varphi(n)$ without factoring?

Question: Why would the ability to compute $\varphi(n)$ be a bad thing?

- It would allow us to easily compute d, since $ed \equiv 1 \bmod \varphi(n)$

Good news: If we can compute $\varphi(n)$, it will allow us to factor n

- Note 1: $\varphi(n) = n - p - q + 1$
  $= n - (p + q) + 1$
- Rewriting gives us $(p + q) = \varphi(n) - n - 1$
- Note 2: $(p - q) = \sqrt{(p+q)^2 - 4n}$
- Note 3: $(p + q) - (p - q) = 2q$
- Finally, given q and n, we can easily compute p

$$\begin{aligned}(p + q)^2 - 4n &= p^2 + 2pq + q^2 - 4n \\ &= p^2 + 2pq + q^2 - 4pq \\ &= p^2 - 2pq + q^2 \\ &= (p - q)^2\end{aligned}$$

What does this mean?

- If factoring is actually hard, then so is computing $\varphi(n)$ without factoring
- This is called a reduction

What about computing d without factoring n or knowing $\varphi(n)$?

As it turns out, if we can figure out d without knowing $\varphi(n)$ and without factoring n, d can be used to help us factor n

Given d, we can compute ed-1, since we know e

Note: ed - 1 is a multiple of $\varphi(n)$

- $ed \equiv 1 \bmod \varphi(n)$
- $ed = 1 + k\varphi(n)$
- $ed - 1 = k\varphi(n)$ ✔

It has been shown that n can be efficiently factored using any multiple of $\varphi(n)$. As such, if we know e and d, we can efficiently factor n.

Are there any other attacks that we need to worry about?

Recall: $C = M^e \bmod n$

- e is part of the public key, so the adversary knows this
- If we could compute e^{th} roots mod n, we could decrypt without d

It is not known whether breaking RSA yields an efficient factoring algorithm, but the inventors conjecture that this is the case

- This conjecture was made in 1978
- To date, it has either been proved or disproved

Conclusion: Odds are that breaking RSA efficiently implies that factoring can be done efficiently. Since factoring is hard, RSA is probably safe to use.

RSA Wrap Up

Hopefully you now have a better understanding of RSA

- How each step of the process works
- How these steps can be made reasonably efficient
- Why RSA is safe to use

Unfortunately, this is not the end of the story…

- Although theoretically secure, implementations can be broken
- We'll revisit this in a later lecture

Next time: Secret sharing and threshold cryptography