CSC 580
Cryptography and Computer Security

The RSA Algorithm and Key Size Issues
(Section 9.2 and more)

March 15, 2018

Overview

Today:

- Overview/demo of research tools
- The RSA Algorithm - key sizes and factoring

Next:

- Read Sections 2.8, 10.1, and 10.2
- Complete ungraded homework 6
- Remember to be working on graded homework 2 (due next Thurs)

First up… some demos of research tools

Tools being demonstrated:

- Zotero (managing papers, citations, etc.)
- LaTeX and paper format templates
- BibTeX

Back to Crypto… Recap of last time

Miller-Rabin Primality Testing: There is an efficient randomized algorithm for testing if large numbers are prime (with very low probability of error).

- So: There is an efficient algorithm for *finding* large random prime numbers

Euler's totient function: $\phi(n)$ = number of integers from 1..n-1 that are relatively prime to n.

Euler's Theorem: For every a and n that are relatively prime (i.e., gcd(a,n)=1),

$$a^{\phi(n)} \equiv 1 \pmod{n} .$$

RSA Algorithm

Key Generation:

Pick two large primes p and q

Calculate $n=p*q$ and $\phi(n)=(p-1)*(q-1)$

Pick a random e such that $\gcd(e, \phi(n))$

Compute $d = e^{-1} \pmod{\phi(n)}$ [*Use extended GCD algorithm!*]

Public key is $PU=(n,e)$; Private key is $PR=(n,d)$

Encryption of message $M \in \{0,..,n-1\}$:

$E(PU,M) = M^e \bmod n$

Decryption of ciphertext $C \in \{0,..,n-1\}$:

$D(PR,C) = C^d \bmod n$

RSA Algorithm

Key Generation:

Pick two large primes p and q
Calculate $n=p*q$ and $\phi(n)=(p-1)*(q-1)$
Pick a random e such that gcd(e, $\phi(n)$)
Compute $d = e^{-1} \pmod{\phi(n)}$ [*Use extended GCD algorithm!*]
Public key is $PU=(n,e)$; Private key is $PR=(n,d)$

Encryption of message $M \in \{0,..,n\text{-}1\}$:

$E(PU,M) = M^e \bmod n$

Decryption of ciphertext $C \in \{0,..,n\text{-}1\}$:

$D(PR,C) = C^d \bmod n$

Correctness - easy when gcd(M,n)=1:

$$D(PR,E(PU,M)) = (M^e)^d \bmod n$$
$$= M^{ed} \bmod n$$
$$= M^{k\phi(n)+1} \bmod n$$
$$= (M^{\phi(n)})^k M \bmod n$$
$$= M$$

Also works when gcd(M,n)≠1, but slightly harder to show...

RSA Example

Simple example:

p = 73, q = 89

n = p*q = 73*89 = 6497

$\phi(n)$ = (p-1)*(q-1) = 72*88 = 6336

e = 5

d = 5069   [Note: 5*5069 = 25,345 = 4*6336 + 1]

Encrypting message M=1234:

1234^5 mod 6497 = 1881

Decrypting:

1881^{5069} mod 6497 = 1234

Note: If time allows in class, more examples using Python!

Status of breaking RSA and factoring

Observation: If we could factor fast, we could break RSA

- How: Factor the public modulus n, compute $\phi(n)$, and compute d

So factoring is ***sufficient*** to break RSA - is it ***necessary***?

Status of breaking RSA and factoring

Observation: If we could factor fast, we could break RSA

- How: Factor the public modulus n, compute $\phi(n)$, and compute d

So factoring is <u>*sufficient*</u> to break RSA - is it <u>*necessary*</u>?

- Answer: no one knows!
- This would be a great result if it could be proved…
- Note: Rabin's PK encryption system is based on a similar concept, and it has been shown that breaking it is equivalent to factoring
  - Rabin's scheme isn't used because it is very inefficient - bit-by-bit

<u>*What we know*</u>

Fast factoring ⇒ Break RSA

<u>*What we'd like*</u>

Break RSA ⇒ Fast factoring

<u>*Why?*</u> Look at logical contrapositive:

Can't factor fast ⇒ Can't break RSA

How fast can we factor?

Consider an algorithm with running time $\Theta\left(2^{c \cdot n^{\alpha} \cdot (\lg n)^{1-\alpha}}\right)$

With ɑ = 1: This is $2^{c\square n}$ -- pure exponential time

With ɑ = 0: This is $2^{c\square \lg(n)} = n^{c}$ -- pure polynomial time

Algorithm discovery for factoring has generally involved lowering ɑ

- ɑ = 1: Brute-force search for factors (exponential time)
- ɑ = ½: Quadratic Sieve (1981) - still the best for n<300 bits or so
- ɑ = ⅓: General Number Field Sieve (1990) - best for large numbers

But: Constants also matter (esp. the c in the exponent!)...

What are the real-world speeds and consequences?

Comparable Key Sizes

From NIST publication 800-57a

Issue: PK algorithms based on mathematical relationships, and can be broken with algorithms that are faster than brute force.

We spent time getting a feel for how big symmetric cipher keys needed to be

➔ How big do keys in a public key system need to be?

From NIST pub 800-57a:

Table 2: Comparable strengths

Security Strength	Symmetric key algorithms	FFC (e.g., DSA, D-H)	IFC (e.g., RSA)	ECC (e.g., ECDSA)
≤ 80	2TDEA[21]	$L = 1024$ $N = 160$	$k = 1024$	$f = 160\text{-}223$
112	3TDEA	$L = 2048$ $N = 224$	$k = 2048$	$f = 224\text{-}255$
128	AES-128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256\text{-}383$
192	AES-192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384\text{-}511$
256	AES-256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$