CS/COE 1501

www.cs.pitt.edu/~lipschultz/cs1501/

The RSA Cryptosystem

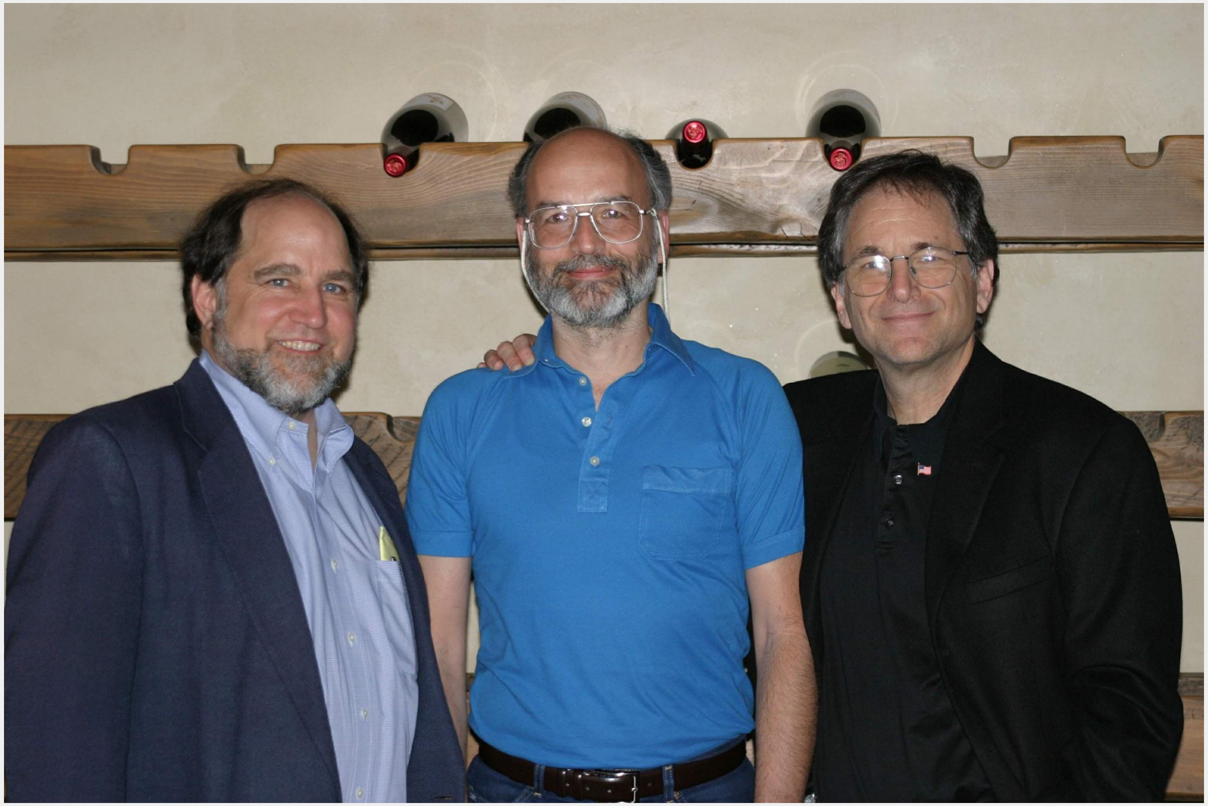
We ended lecture last time…

- Mentioning some of the shortcomings with symmetric key encryption
- Today we'll be talking about public-key encryption
  - Each user has their own pair of keys
    - A public key that can be revealed to anyone
    - A private key that *only they should know*
- Eases key distribution problems
  - Public key can simply be published/advertised
    - Posted repositories of public keys
    - Added to an email signature
  - Each user is responsible only for their own keypair

Cryptographic keys

- For symmetric ciphers (e.g., AES), keys are just numbers of a given bitlength (e.g., 128, 256)
- In public key crypto, we have *keypairs*
  - In RSA:
    - Public key is two numbers, which we will call n and e
    - Private key is a single number we will call d
- The length of n in bits is the key length
  - i.e., 2048 bit RSA keys will have a 2048 bit n value

RSA



RSA Cryptosystem

- What are public/private keys?
- How messages encrypted?
- How are messages decrypted?
- How are keys generated?
- Why is it secure?

Encryption

Say Alice wants to send a message to Bob

1. Looks up Bob's public key
2. Convert the message into an integer: m
3. Compute the ciphertext c as:
    - $c = m^e \pmod{n}$
4. Send c to Bob

Decryption

Bob can simply:

1. Compute m as:
    - $m = c^d \pmod{n}$
2. Convert m into Alice's message

n, e, and d need to be carefully generated

1. Choose two prime number p and q
2. Compute $n = p * q$
3. Compute $\varphi(n)$
   - $\varphi(n) = \varphi(p) * \varphi(q) = (p - 1) * (q - 1)$
4. Choose e such that
   - $1 < e < \varphi(n)$
   - $GCD(e, \varphi(n)) = 1$
     - I.e., e and $\varphi(n)$ are co-prime
5. Determine d as $d = e^{-1} \bmod (\varphi(n))$

What’s φ?

- Here, we mean φ to be Euler’s totient
- φ(n) is a count of the integers < n that are co-prime to n
  - I.e., how many k are there such that:
    - 1 <= k <= n AND GCD(n, k) = 1
- p and q are prime..
  - Hence, φ(p) = p - 1 and φ(q) = q -1
- Further, φ is multiplicative
  - Since p and q are prime, they are co-prime, so
    - φ(p) * φ(q) = φ(p * q) = φ(n)
      - I won’t detail the proof here...

OK, now what about multiplicative inverses mod φ(n)?

- $d = e^{-1} \bmod(\varphi(n))$
  - $d = (1/e) \bmod(\varphi(n))$
  - $e * d = 1 \pmod{\varphi(n)}$
- Now, *this* can be equivalently stated as $e * d = z * \varphi(n) + 1$
  - For some z
- Can further restate this as: $e * d - z * \varphi(n) = 1$
- Or similarly: $1 = \varphi(n) * (-z) + e * d$
- How can we solve this?
  - Hint: recall that we know $GCD(\varphi(n), e) = 1$

Use extended Euclidean algorithm!

- $GCD(a, b) = i = ax + by$
- Let:
  - $a = \varphi(n)$
  - $b = e$
  - $x = -z$
  - $y = d$
  - $i = 1$
- $GCD(\varphi(n), e) = 1 = \varphi(n) * (-z) + e * d$
- We can compute d in linear time!

RSA keypair example

- Remember:
    - p and q must be prime
    - $n = p * q$
    - $\varphi(n) = (p - 1) * (q - 1)$
    - Choose e such that
        - $1 < e < \varphi(n)$ and $GCD(e, \varphi(n)) = 1$
    - Solve $XGCD(\varphi(n), e) = 1 = \varphi(n) * (-z) + e * d$

OK, but how does $m^{ed} = m \bmod n$?

- Feel free to look up the proof using Fermat's little theorem
  - Knowing this proof is **NOT** required for the course
  - Knowing how to generate RSA keys and encrypt/decrypt **IS**
- For this course, we'll settle with our example showing that it *does* work

Why is RSA secure?

- 4 avenues of attack on the math of RSA were identified in the original paper:
  - Factoring n to find p and q
  - Determining $\varphi(n)$ without factoring n
  - Determining d without factoring n or learning $\varphi(n)$
  - Learning to take e^{th} roots modulo n

Factoring n

- This is *hard*
  - A 768 bit RSA key was factored one time using the best currently known algorithm
    - Took 1500 CPU years
      - 2 years of real time on hundreds of computers
    - Hence, large keys are safe
      - 2048 bit keys are a pretty good bet for now

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

- Would allow us to easily compute d because ed = 1 mod $\varphi(n)$
- Note:
  - $\varphi(n) = n - p - q + 1$
    - $\varphi(n) = n - (p + q) + 1$
    - $(p + q) = n + 1 - \varphi(n)$
  - $(p + q) - (p - q) = 2q$
  - Now we just need (p - q)...
    - $(p - q)^2 = p^2 - 2pq + q^2$
    - $(p - q)^2 = p^2 + 2pq + q^2 - 4pq$
    - $(p - q)^2 = (p + q)^2 - 4pq$
    - $(p - q)^2 = (p + q)^2 - 4n$
    - $(p - q) = \sqrt{((p + q)^2 - 4n)}$
- If we can figure out $\varphi(n)$ efficiently, we could factor n efficiently!

Determining d without factoring n or learning φ(n)?

- If we know, d, we can get a multiple of φ(n)
  - $ed = 1 \bmod \varphi(n)$
  - $ed = k\varphi(n) + 1$
    - For some k
  - $ed - 1 = k\varphi(n)$
- It has been shown that n can be efficiently factored using any multiple of φ(n)
  - Hence, this would provide another efficient solution to factoring!

Learning to take e^{th} roots modulo n

- Conjecture was made in 1978 that breaking RSA would yield an efficient factoring algorithm
  - To date, it has been not been proven or disproven

This all leads to the following conclusion

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

Implementation concerns

- Encryption/decryption:
  - How can we perform efficient exponentiations?
- Key generation:
  - We can do multiplication, XGCD for large integers
  - What about finding large prime numbers?

Exponentiation for RSA

Does this solve our problems?

```
res = 1
foreach bit in y:
    res = (res^2 mod n)
    if bit == 1:
        res = (res*x mod n)
```

- How can we improve runtime for RSA exponentiations?
  - Don’t actually need x^y
    - Just need (x^y mod n)

Still slower (generally) than symmetric encryption

- If only we could have the speed of symmetric encryption without the key distribution woes!
  - What if we transmitted symmetric crypto keys with RSA?
    - RSA Envelopes!
- Going back to Alice and Bob
  - Alice generates a random AES key
  - Alice encrypts her message using AES with this key
  - Alice encrypts the key using Bob's RSA public key
  - Alice sends the encrypted message and encrypted key to Bob
  - Bob decrypts the AES key using his RSA private key
  - Bob decrypts the message using the AES key

Prime testing option 1: BRUTE FORCE

- Try all possible factors
  - 1 .. sqrt(x)
    - aka 1 .. sqrt($2^{|n|}$)
      - For a total of $2^{(|n|/2)}$ factor checks
- A factor check should take about the same amount of time as multiplication
  - $|n|^2$
- So our runtime is $\Theta(2^{(|n|/2)}|n|^2)$

Option 2: A probabilistic approach

- Need a method test : $Z \times Z \longrightarrow \{T, F\}$
  - If test(x, a) = F, x is composite based on the witness a
  - If test(x, a) = T, x is probably prime based on the witness a
- To test a number x for primality:
  - Randomly choose a witness a
    - if test(x, a) = F, x is composite
    - if test(x, a) = T, loop
- Possible implementations of test(x, a):
  - Miller-Rabin, Fermat's, Solovay–Strassen

often probability ≈ 1/2

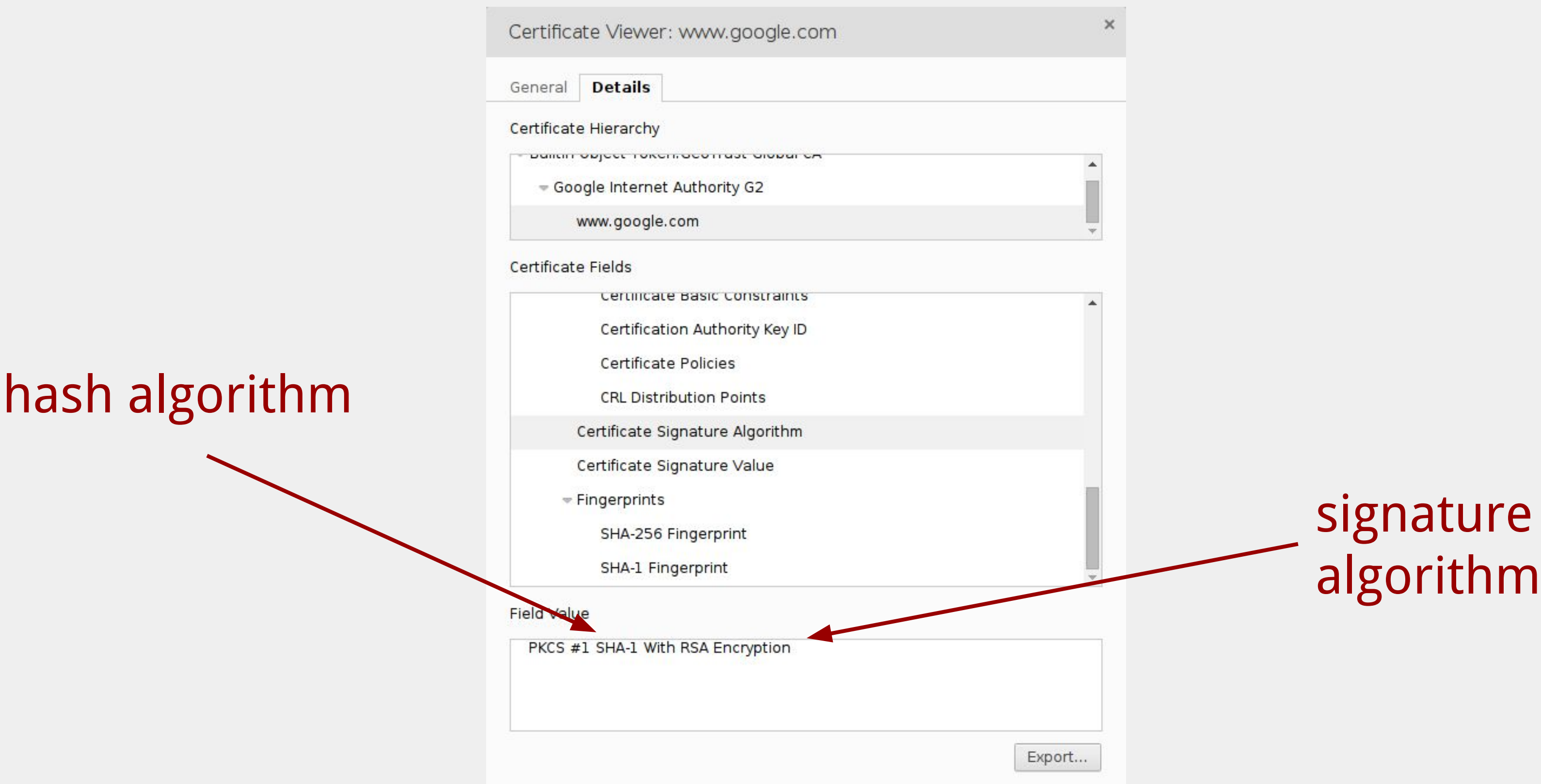
k repetitions leads to probability that x is composite ≈ $1/2^k$

Another fun use of RSA...

- Notice that encrypting and decrypting are inverses
  - $m^{ed} = m^{de} \pmod{n}$
- We can “decrypt” the message first with a private key
- Then recover the message by “encrypting” with a public key
- Note that anyone can recover the message
  - However, they know the message *must* have come from the owner of the private key
    - Using RSA this way creates a digital signature

How do we avoid the downsides of RSA here?

- We encrypted symmetric crypto keys before
- For digital signatures, instead of signing the whole message, we simply sign a hash of the message!



What about collisions?

- If Bob signs a hash of the message “I’ll see you at 7”

- It could appear that Bob signed any message whose hash collides with “I’ll see you at 7”...
  - If h(“I’ll see you at 7”) == h(“I’ll see you after I rob the bank”), Bob could be in a lot of trouble

- An attack like this helped the Flame malware to spread
- This is also the reason Google is aiming to deprecate SHA-1

Public key isn't perfect, however

What do you when a private key is compromised?

Final note about crypto

NEVER IMPLEMENT YOUR OWN CRYPTO

Use a trusted and tested library.