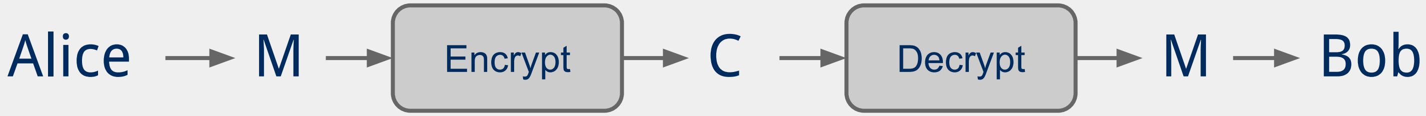
CS/COE 1501

www.cs.pitt.edu/~nlf4/cs1501/

An Introduction to Cryptography

Introduction to crypto

- Cryptography - enabling secure communication in the presence of third parties
    - Alice wants to send Bob a message without anyone else being able to read it



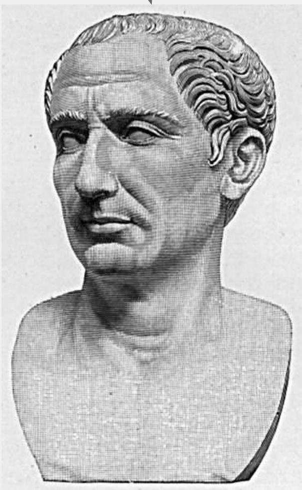
Enter the adversary

- Consider the adversary to be anyone that could try to eavesdrop on Alice and Bob communicating
  - People in the same coffee shop as Alice or Bob as they talk over WiFi
  - Admins operating the network between Alice and Bob
    - And mirroring their traffic to the NSA…
- Will have access to:
  - The *ciphertext*
    - The encrypted message
  - The encryption algorithm
    - At least Alice and Bob should assume the adversary does
- The key material is the only thing Bob knows that the adversary does not

Cryptography has been around for some time

- Early, classic encryption scheme:
  - Caesar cipher:
    - "Shift" the alphabet by a set amount
    - Use this shifted alphabet to send messages
    - The "key" is the amount the alphabet is shifted

Yes, that Caesar



Alphabet

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
XYZABCDEFGHIJKLMNOPQRSTUVW
```

Shift 3

By modern standards, incredibly easy to crack

- BRUTE FORCE
  - Try every possible shift
    - 25 options for the English alphabet
    - 255 for ASCII
- OK, let's make it harder to brute force
  - Instead of using a shifted alphabet, let's use a random permutation of the alphabet
    - Key is now this permutation, not just a shift value
  - R size alphabet means R! possible permutations!

By modern standards, incredibly easy to crack

- Just requires a bit more sophisticated of an algorithm
- Analyzing encrypted English for example
  - Sentences have a given structure
  - Character frequencies are skewed
  - Essentially playing Wheel of Fortune

So what is a good cipher?

- One-time pads
  - List of one-time use keys (called a *pad*) here
- To send a message:
  - Take an unused pad
  - Use modular addition to combine key with message
    - For binary data, XOR
  - Send to recipient
- Upon receiving a message:
  - Take the next pad
  - Use modular subtraction to combine key with message
    - For binary data, XOR
  - Read result
- Proven to provide perfect secrecy

One-time pad example

Encoding:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

Pad: Q J C W T

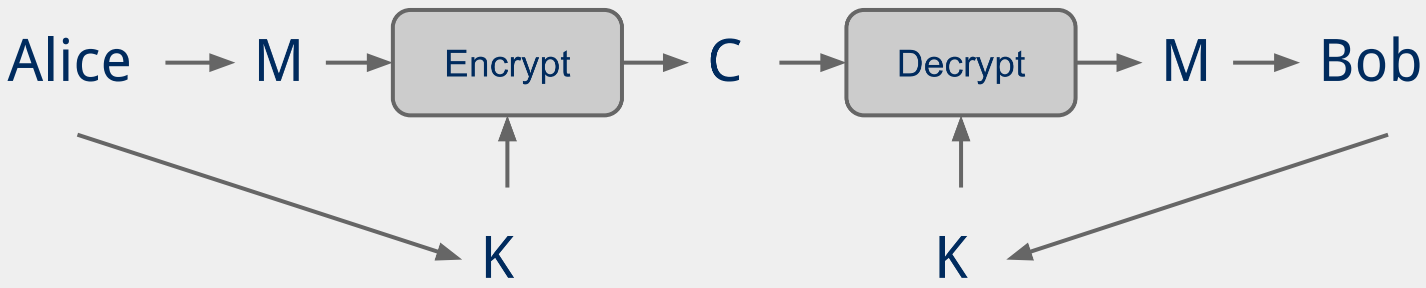
16 9 2 22 19

```
Message:      H  E  L  L  O

              7  4 11 11 14

        +    16  9  2 22 19     (mod 26)
         -------------------------------
             23 13 13  7  7

Encrypted     X  N  N  H  H
Message:
             23 13 13  7  7

        -    16  9  2 22 19     (mod 26)
         -------------------------------
              7  4 11 11 14

              H  E  L  L  O
```

Difficulties with one-time pads

- Pads must be truly random
- Both sender and receiver must have a matched list of pads in the appropriate order
- Once you run out of pads, no more messages can be sent

Symmetric ciphers



- E.g., DES, AES, Blowfish
- Users share a single *key*
  - Numbers of a given bitlength (e.g., 128, 256)
  - Key is used to encrypt/decrypt many messages back and forth
- Encryptions/decryptions will be fast
  - Typically linear in the size the input
- Ciphertext should appear random
- Best way to recover plaintext should be a brute force attack on the encryption key
  - Which we have shown to be infeasible for 128bit AES keys

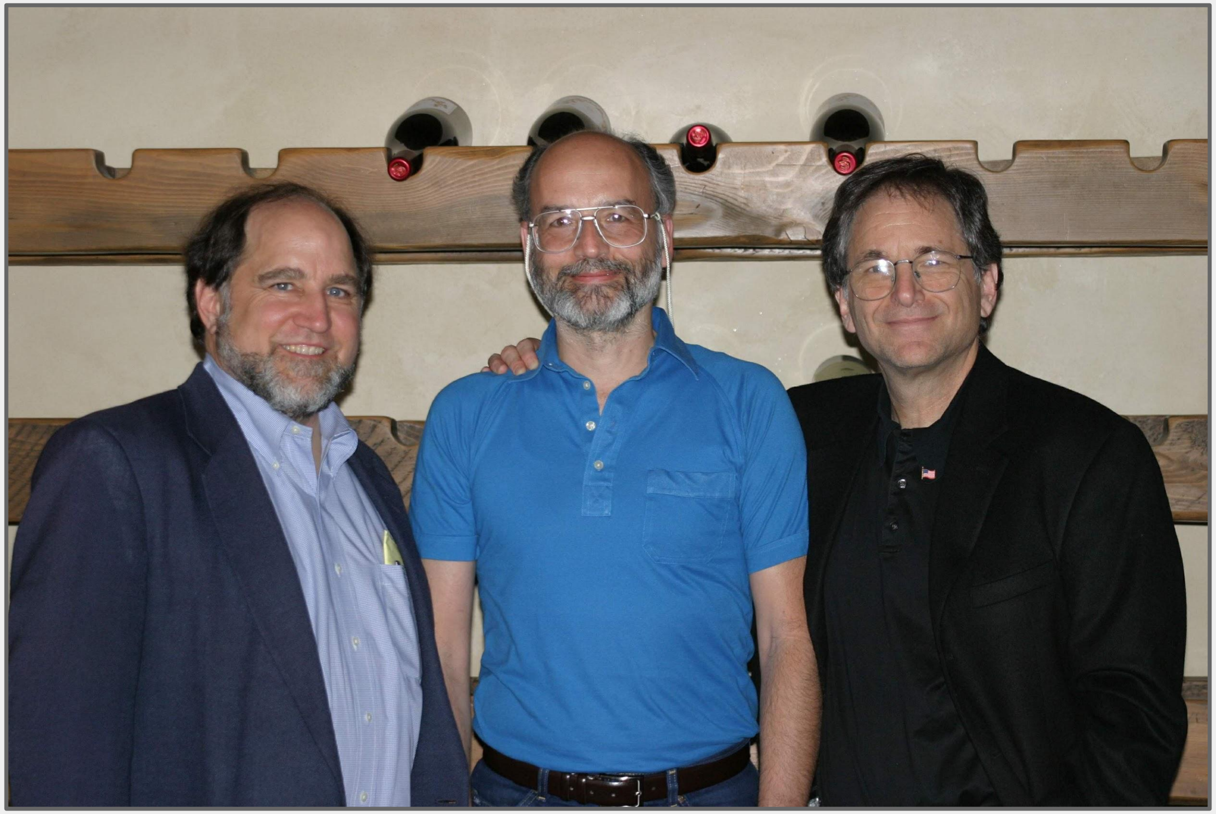
Problems with symmetric ciphers

- Alice and Bob have to both know the same key
    - How can you securely transmit the key from Alice to Bob?
- Further, if Alice also wants to communicate with Charlie, her and Charlie will need to know the same key, a different key from the key Alice shares with Bob
    - Alice and Danielle will also have to share a different key…
    - etc.

Enter 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
- How does this solve our problem?
    - 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
- Let's look at a public-key crypto scheme in detail...

RSA



RSA Cryptosystem in-depth

- What are RSA keypairs?
- How messages encrypted?
- How are messages decrypted?
- How are keys generated?
- Why is it secure?

RSA keypairs

- *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
    - Note that "n" will be used to indicate the RSA public key component for our discussion of RSA...

Encryption

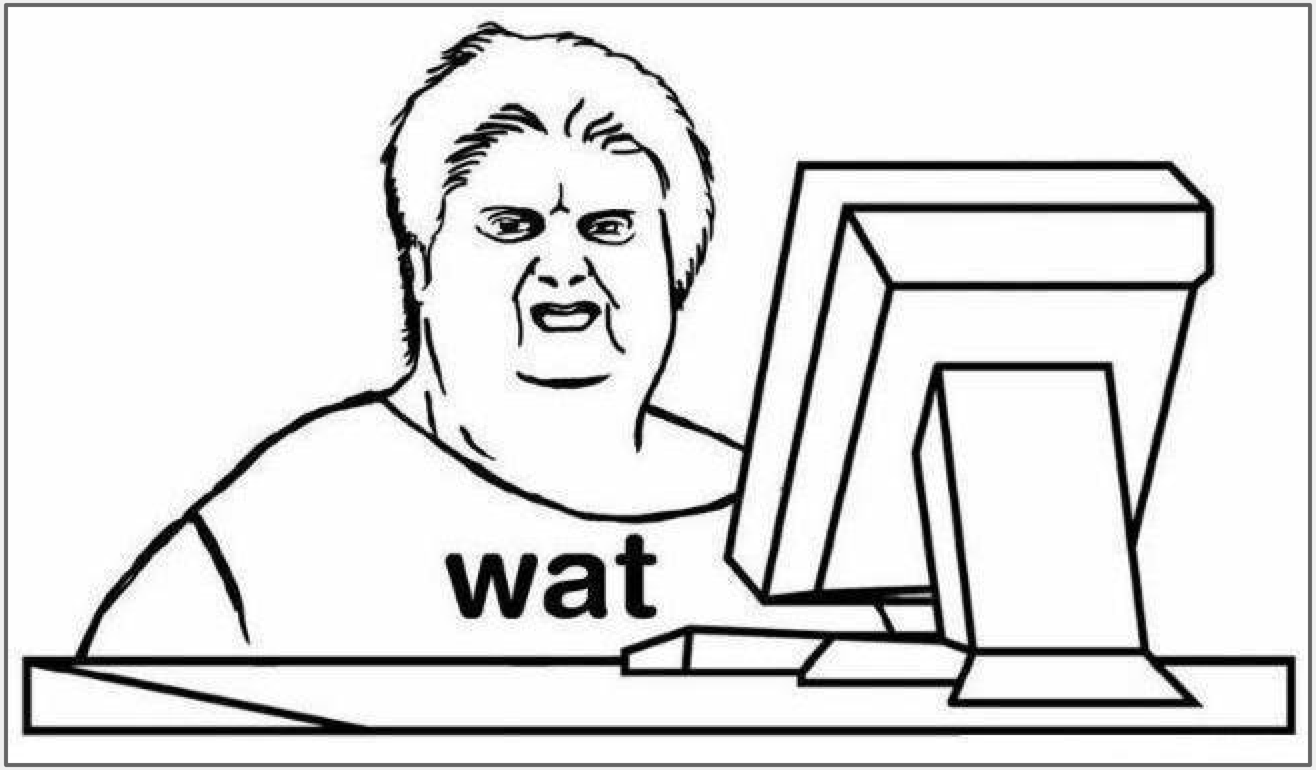
Say Alice wants to send a message to Bob

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


wat

n, e, and d need to be carefully generated

1. Choose two prime numbers 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 the 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)?

- $d = e^{-1} \bmod(\varphi(n))$
- Means that $d = 1/e \bmod(\varphi(n))$
- Means that $e * d = 1 \pmod{\varphi(n)}$
- 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 notes

- 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 $XGCD(\varphi(n), e) = 1 = \varphi(n) * (-z) + e * d$
- Compute the ciphertext c as:
  - $c = m^e \pmod{n}$
- Recover m as:
  - $m = c^d \pmod{n}$

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

- To the best of our knowledge, 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 pretty safe
      - 2048 bit keys are a pretty good bet for now

What about determining φ(n) without factoring n?

- Would allow us to easily compute d because $ed = 1 \bmod \varphi(n)$
- 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 φ(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 probably 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?

Efficient exponentiation for RSA

Does this solve our problems?

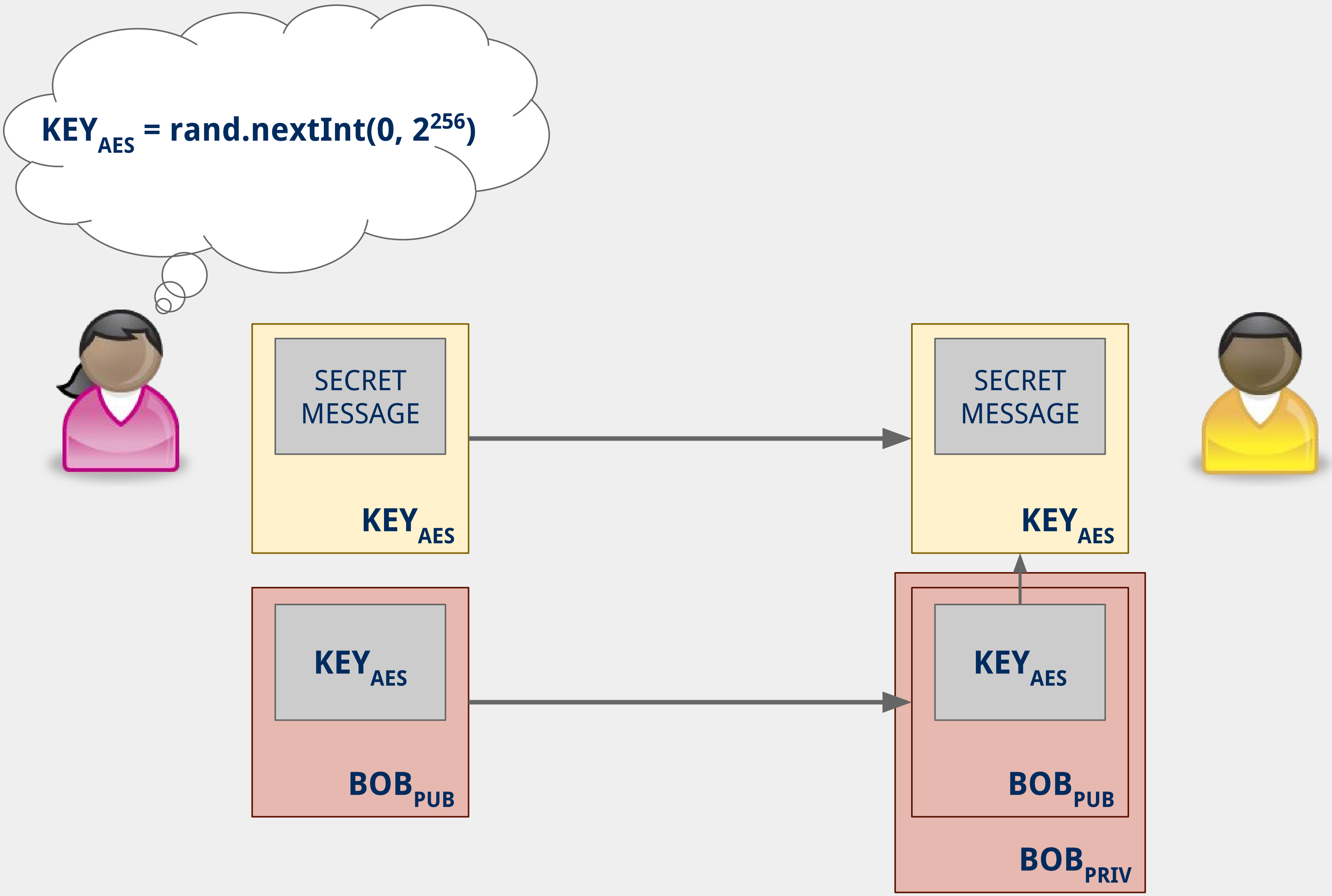
```
ans = 1
foreach bit in y:
    ans = (ans^2 mod n)
    if bit == 1:
        ans = (ans*x mod n)
```

- 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

RSA Envelope example



Prime testing option 1: BRUTE FORCE

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

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*

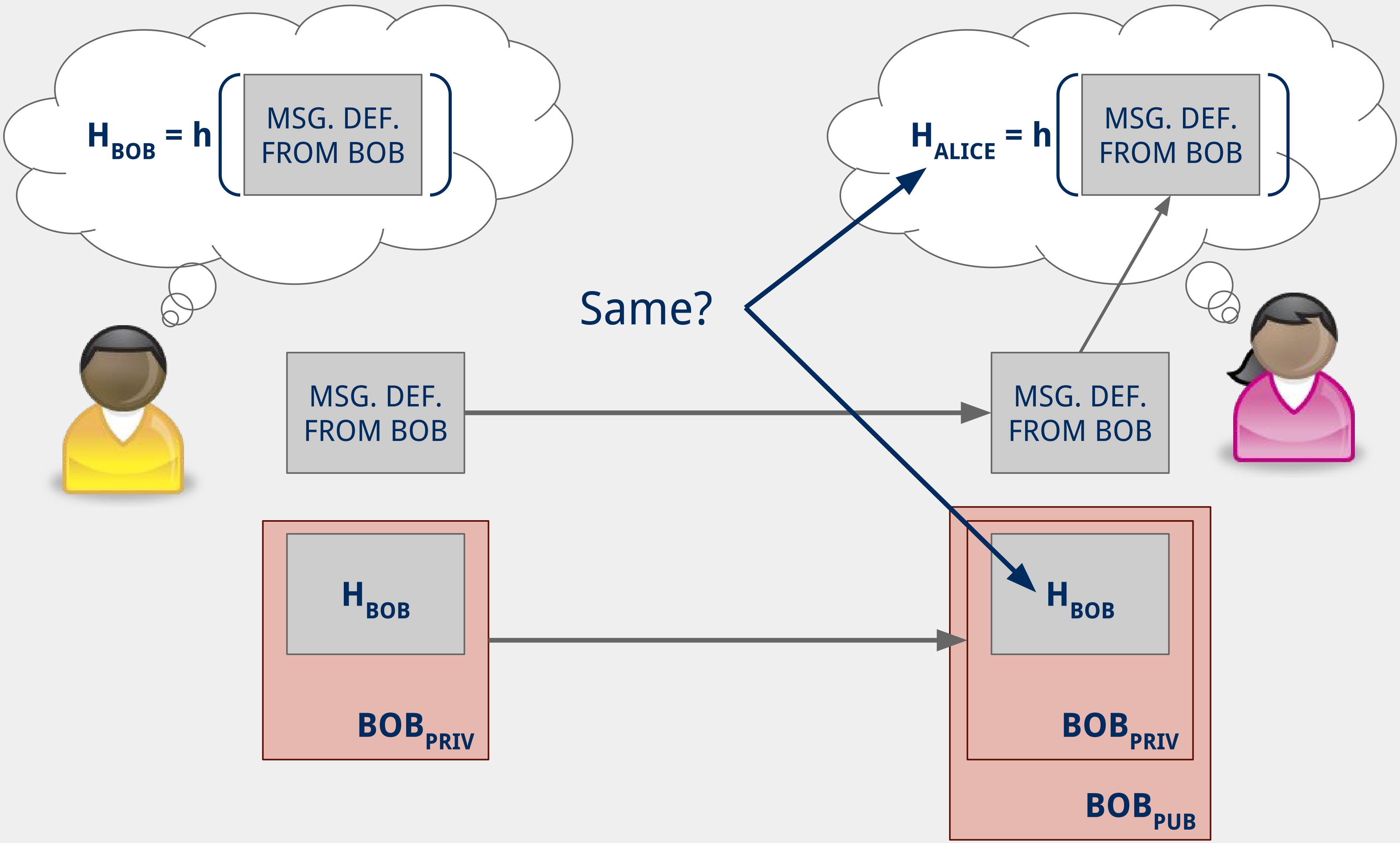
Do RSA signatures need to be slow?

- We encrypted symmetric crypto keys before to speed things up…
  - We'll need another crypto primitive to help out here
  - Cryptographically secure hash functions

Hashing for security (similarities)

- Cryptographically secure hash functions share properties with the hash functions we've already talked about:
  - Map from some input domain to a limited output range
    - Though output ranges are much larger here
      - For modern algorithms 224-512 bit output sizes
  - Time required to compute the hash is proportional to the size of the item being hashed
    - Though, practically, cryptographic hash functions are more expensive

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

Hashing for security (differences)

- This is why cryptographically secure hash functions must support additional properties:
  - It should be infeasible to find two different messages with the same hash value
  - It should be infeasible to recover a message from its hash
    - Should require a brute force approach
  - Small changes to a message should change the hash value so extensively that the new hash value appears uncorrelated with the old hash value

Public key isn't perfect

What do you when a private key is compromised?

Final note about crypto

NEVER IMPLEMENT YOUR OWN CRYPTO

Use a trusted and tested library.