CSCI 3330 Cybersecurity

Word of the day

EPSILON 1. n. A small quantity of anything. "The cost is epsilon." 2. adj. Very small, negligible. "We can get this feature for epsilon cost." 3. `within epsilon of': Indistinguishable for all practical purposes, even closer than being `within delta of'. "That's not what I asked for, but it's within epsilon of what I wanted." Alternatively, it may mean not close enough, but very little is required to get it there: "My program is within epsilon of working."

Agenda for today

- RSA
- Diffie-Helman

RSA derive public key

- Find two primes p and q
- Choose e that shares no factors with neither (p - 1) nor (q – 1)
  - Typically 65537
- Let $n = p \times q$
- Public key = (n, e)

RSA derive public key

- Solve for d such that $(d \times e) \% ((p - 1) \cdot (q - 1)) = 1$ (use Extended Euclidean Algorithm)
- d is e's multiplicative inverse
- Intuition: even though d is an integer, it effectively acts as $^1/_e$ in the multiplicative group of order $(p - 1) \cdot (q - 1)$
- Private key = (d, e)

RSA crypto

- $n = p * q$
- $x = (x^e)^d \% n$
- $E(x) = x^e \% n$
- $D(x) = x^d \% n$
- $sign(x) = x^d \% n$
- $verify(x) = x^e \% n$

RSA key sizes

- In 2025, we want *n* to be 4096 bits
- AES keys are only 256 bits.
- Why are RSA keys so large?

Property of modular arithmetic

- “Fundamental property of multiplication in modular arithmetic”
- $(a \times b) \% n = ((a \% n) \times (b \% n)) \% n$
- How can this be helpful when calculating $x^e \% n$?

Property of modular arithmetic

- We have $x^e \% n = ((x^{e-k} \% n) \times x^k) \% n$
- For example: $x^5 \% n =$
  $((((((x \times x) \% n) \times x) \% n) \times x) \% n) \times x) \% n$
- Thus we can do modular arithmetic efficiently

Property of modular arithmetic

- In python:
- `5 ** 10000000 % 259`
- `(((5 **  5000000) % 259) * (5 **  5000000)) % 259`
- `pow(5, 10000000, 259)`

RSA encryption

- Recall: public key = (n, e) and n = $p \times q$
- If I share (n, e), why can't someone derive the private key?
- If p and q are large, n cannot be easily factored
- We *think* prime factorization cannot be done in polynomial time

Prime factorization

- Factor *n*: test all primes from 2 through sqrt(*n*) until you find one
- More complicated algorithms exist as well…

RSA in-class exercise

- $n = p \times q$
- $x = (x^e)^d \% n$
- $E(x) = x^e \% n$
- $D(x) = x^d \% n$
- $sign(x) = x^d \% n$
- $verify(x) = x^e \% n$

RSA malleability

- Suppose I have eavesdropped on a transmission of the following two things:
- A ciphertext c encrypted with 128-bit AES key *k*
- The RSA-encrypted key k^e
- How do I exploit this?

RSA malleability

- I calculate a new encrypted key:
  - $k'^e = k^e \times (2^{127})^e = (k \times 2^{127})^e$
  - $\Rightarrow k' = k \times 2^{127}$
  - What does this effectively do to the key?
- I create my own message *m*.
- Encrypt *m* with AES key *g* = [1, ... 127 zeroes…]
- Transmit encrypted *m* and k'^e.

RSA malleability

- Effectively I am guessing that the leftmost bit in *k* is a 1
- Do I get an error response? (E.g., a shorter response or even no response)
- If I get an error, then the leftmost bit was a 0
- Otherwise it was a 1!
- Call the recovered bit *b*.

RSA malleability

- I calculate a new encrypted key:
  - $k'^e = k^e \times (2^{126})^e = (k \times 2^{126})^e$
  - $\Rightarrow k' = k \times 2^{126}$
  - What does this effectively do to the key?
- I create my own message *m*.
- Encrypt *m* with AES key *g* = [*b*, 1, ... 126 zeroes…]
- Transmit encrypted *m* and k'^e.

RSA malleability

- Effectively I am guessing that the 2nd-leftmost bit in *k* is a 1
- Do I get an error response? (E.g., a shorter response or even no response)
- If I get an error, then the leftmost bit was a 0
- Otherwise it was a 1!
- Call the recovered bit *b*.
- **Keep repeating until all bits of k are recovered.**

Oracle requirements

- Returns a measurably different response (such as an error) when a message contains “random” bits
- Ignores bits that are “too high” in the decrypted symmetric key

RSA malleability

- Fix: OAEP padding
- Uses cryptographic hashing to ensure integrity

RSA in Assignment 6

```
RSA_PUBLIC_KEY =
23031139172164797501358931518141760217658633232914241675310934798937917
28362122942187339911198352881657123247917457139832226993776536877577094
41555766749232998066100180740183648968441411266998154575052542671483556
82097718249831721480482926491254278424681015395431396587270225006622307
96906840396607326506942419607009564238522306089609971263741261513588325
39077067014127970035905069700827639281180992446841778135830149233110079
10123929922647880687432206805838057491981427746974024273757066023180270
44340247719139149936789940561959505009861896669205397645289589817157443
76788249801436946328420051006942959501056914827O1, 65537

...

def rsa_encrypt(m_int, e_int, n_int):

    return pow(m_int, e_int, n_int)
```

Example from Assignment 6

https://brainhammer.com/cbc/?c=58d9f18eb1a68a928dc14ad17527f6bc04ddf9ca4aac64fb0aa83cdd5ab54c4185e8870321d3b24c78eb21057a961dad&iv=e486b3afb05e477d1ec8e2dae6efdc74&k=51905de251e6b496307787fd8104623a0bcdf895377ad56b42836c19db39f4b64f0f15008487a98db6e4183cc9ab001a8e3d09d1bb44c3553f8ccb27a76302279a4d02477eeea0a61c82039feefb0b85736c8de2a108066e483cafce7f2dff77a6958090a200355f786b45c5a92c3074c8f1a926bee2b6a85cb9c83d1c643116c971c429c3aaafcd0698f5f85b8097f8d157d492ec9bada481822398abf530e38b4ff7eb0248bb4cb6d3a6204a02236ffb28cb50af6c3ec63c7c880635431fe6e77855c2246a6b66c0339e78c97854fae550de47f2bf6b90076ce2d5e6aa1758d78afc9ff7c60ab88c8c727b2756c3090a6cb47c88bdb57d2c44b73d5b399239

Example from Assignment 6

- c=58d9f18eb1a68a928dc14ad17527f6bc04ddf9ca4aac64fb0aa83cdd5ab54c4185e8870321d3b24c78eb21057a961dad
- iv=e486b3afb05e477d1ec8e2dae6efdc74
- k=51905de251e6b496307787fd8104623a0bcdf895377ad56b42836c19db39f4b64f0f15008487a98db6e4183cc9ab001a8e3d09d1bb44c3553f8ccb27a76302279a4d02477eeea0a61c82039feefb0b85736c8de2a108066e483cafce7f2dff77a6958090a200355f786b45c5a92c3074c8f1a926bee2b6a85cb9c83d1c643116c971c429c3aaafcd0698f5f85b8097f8d157d492ec9bada481822398abf530e38b4ff7eb0248bb4cb6d3a6204a02236ffb28cb50af6c3ec63c7c880635431fe6e77855c2246a6b66c0339e78c97854fae550de47f2bf6b90076ce2d5e6aa1758d78afc9ff7c60ab88c8c727b2756c3090a6cb47c88bdb57d2c44b73d5b399239

RSA in Assignment 6

- But Assignment 6 uses OAEP padding
- So it should be impervious to *this* attack

Asymmetric comparison

- RSA: assumes prime factorization is hard
  - factor(n)
- DSA: assumes discrete log problem is hard
  - find x such that $g^x \% p = h$
- ED25519: assumes that the elliptic-curve discrete logarithm problem is hard

Assignment 6

- Deadline: November 6, 2025, 10:00pm
- CBC padding oracle attacks
  https://tildesites.bowdoin.edu/~j.knockel/csci3330f25/assignment6.pdf