# CSCI 3330
# Cybersecurity

# Word of the day

**USER-FRIENDLY adj.** Programmer-hostile. Generally used by hackers in a critical tone, to describe systems that hold the user's hand so obsessively that they make it painful for the more experienced and knowledgeable to get any work done. See also menuitis, drool-proof paper, Macintoy, user-obsequious.

# Agenda for today

- More attacks against textbook RSA
- Diffie-Hellman

# 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:
  - $k'^e = k^e \times (2^{127})^e = (k \times 2^{127})^e \pmod{n}$
  - $\Rightarrow k' = k \times 2^{127} \pmod{n}$
  - 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

- 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:
  - $k'^e = k^e \times (2^{126})^e = (k \times 2^{126})^e \pmod{n}$
  - $\Rightarrow k' = k \times 2^{126} \pmod{n}$
  - 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

- 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!
- **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 when bits are “too high” in the decrypted symmetric key

# Oracle requirements

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

# Check for divisibility

- $k'^e = k^e \times (2^{-1})^e = (k/2)^e \pmod{n}$
- $\Rightarrow k' = k/2 \pmod{n}$
- $k'^e = k^e \times (3^{-1})^e = (k/3)^e \pmod{n}$
- $\Rightarrow k' = k/3 \pmod{n}$
- …
- When divisible, no “too high” bits set because it cleanly divides.
- Otherwise, “too high” bits set.

# Factor *k* for all factors less than *B*

- You have some large factor *F* (e.g., $F = 2 \cdot 2 \cdot 11 \cdot 29 \cdot \ldots$)
- Let $R = k / F$. Thus, to find *k*, it is sufficient to find *R*.
- Binary search for *x* such that $(x + 1)R \geq 2^{128}$ and $x \cdot R < 2^{128}$.
- Thus, $2^{128}/(x + 1) \leq R < 2^{128}/x$.
- Brute force *R* given these bounds.
- If the search space is too large, you must find more factors…

# 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

# Key derivation in python

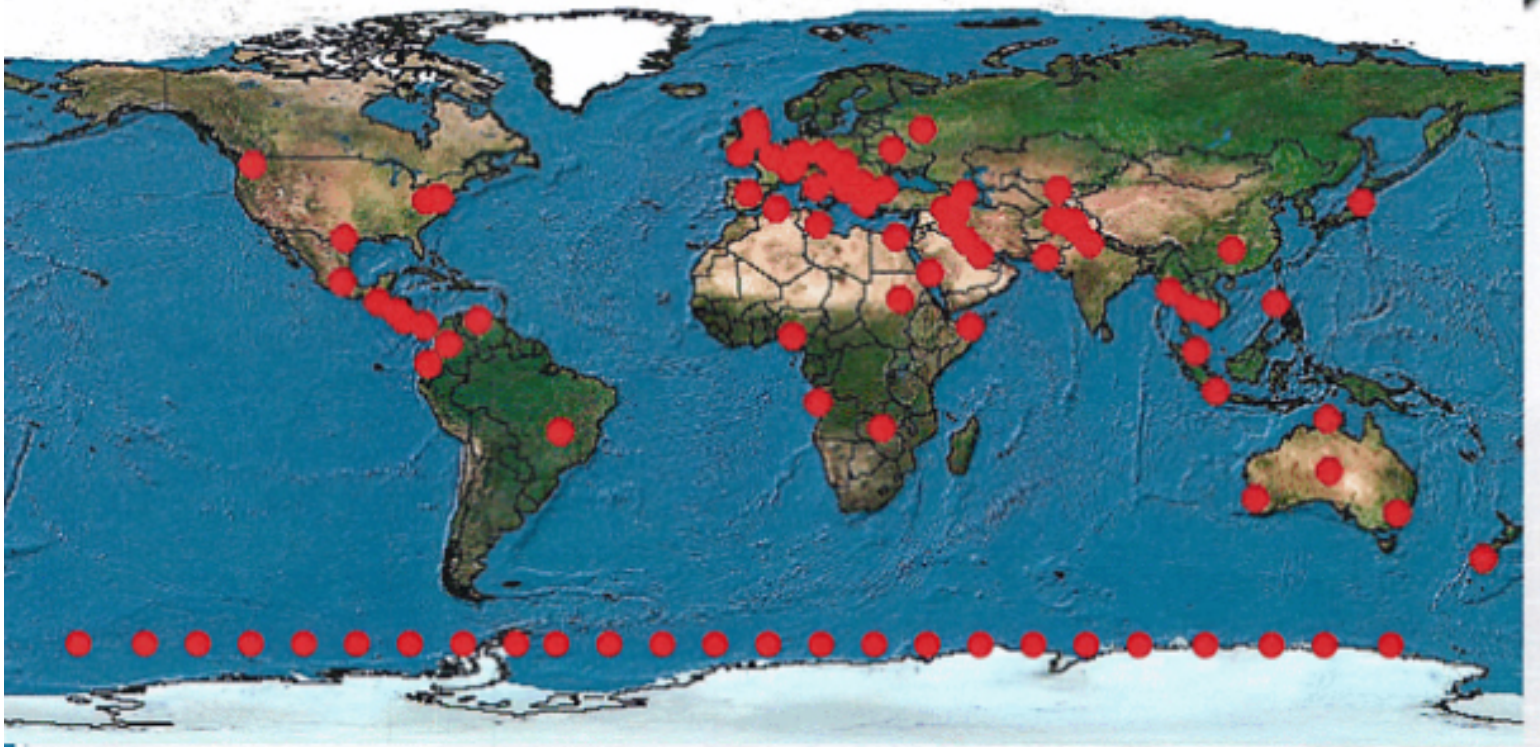
```
def lcm(a, b):
    return a // gcd(a, b) * b


def derive_rsa_keys(p, q, e=65537):
    n = p * q
    totient = lcm(p - 1, q - 1)
    if gcd(e, totient) != 1:
        raise ValueError('e must be coprime to totient')
    d = pow(e, -1, totient)
    return (e, n), (d, n)
```

# Why does this all matter?

- Knowing what I’m teaching you gives you immense power, to do either right or wrong
- The kinds of attacks that we are learning are regularly, routinely, and continuously at population scale

# Where is X-KEYSCORE?



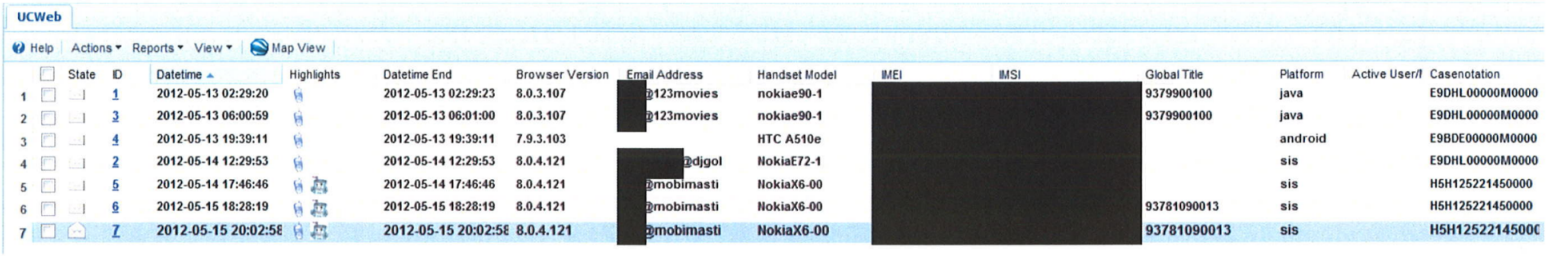
Approximately 150 sites

Over 700 servers

# Success Stories

* UCWeb mobile browser identification
  * Discovered by GCHQ analyst during DSD workshop
  * Chinese mobile web browser – leaks IMSI, MSISDN, IMEI and device characteristics

# UCWeb – XKS Microplugin



UCWeb

Help | Actions | Reports | View | Map View

| | State | ID | Datetime | Highlights | Datetime End | Browser Version | Email Address | Handset Model | IMEI | IMSI | Global Title | Platform | Active User/I | Casenotation |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | 1 | 2012-05-13 02:29:20 | | 2012-05-13 02:29:23 | 8.0.3.107 | [redacted]@123movies | nokiae90-1 | [redacted] | | 9379900100 | java | | E9DHL00000M0000 |
| 2 | | 3 | 2012-05-13 06:00:59 | | 2012-05-13 06:01:00 | 8.0.3.107 | [redacted]@123movies | nokiae90-1 | | | 9379900100 | java | | E9DHL00000M0000 |
| 3 | | 4 | 2012-05-13 19:39:11 | | 2012-05-13 19:39:11 | 7.9.3.103 | | HTC A510e | | | | android | | E9BDE00000M0000 |
| 4 | | 2 | 2012-05-14 12:29:53 | | 2012-05-14 12:29:53 | 8.0.4.121 | [redacted]@djgol | NokiaE72-1 | | | | sis | | E9DHL00000M0000 |
| 5 | | 5 | 2012-05-14 17:46:46 | | 2012-05-14 17:46:46 | 8.0.4.121 | [redacted]@mobimasti | NokiaX6-00 | | | | sis | | H5H125221450000 |
| 6 | | 6 | 2012-05-15 18:28:19 | | 2012-05-15 18:28:19 | 8.0.4.121 | [redacted]@mobimasti | NokiaX6-00 | | | 93781090013 | sis | | H5H125221450000 |
| 7 | | 7 | 2012-05-15 20:02:58 | | 2012-05-15 20:02:58 | 8.0.4.121 | [redacted]@mobimasti | NokiaX6-00 | | | 93781090013 | sis | | H5H12522145000 |

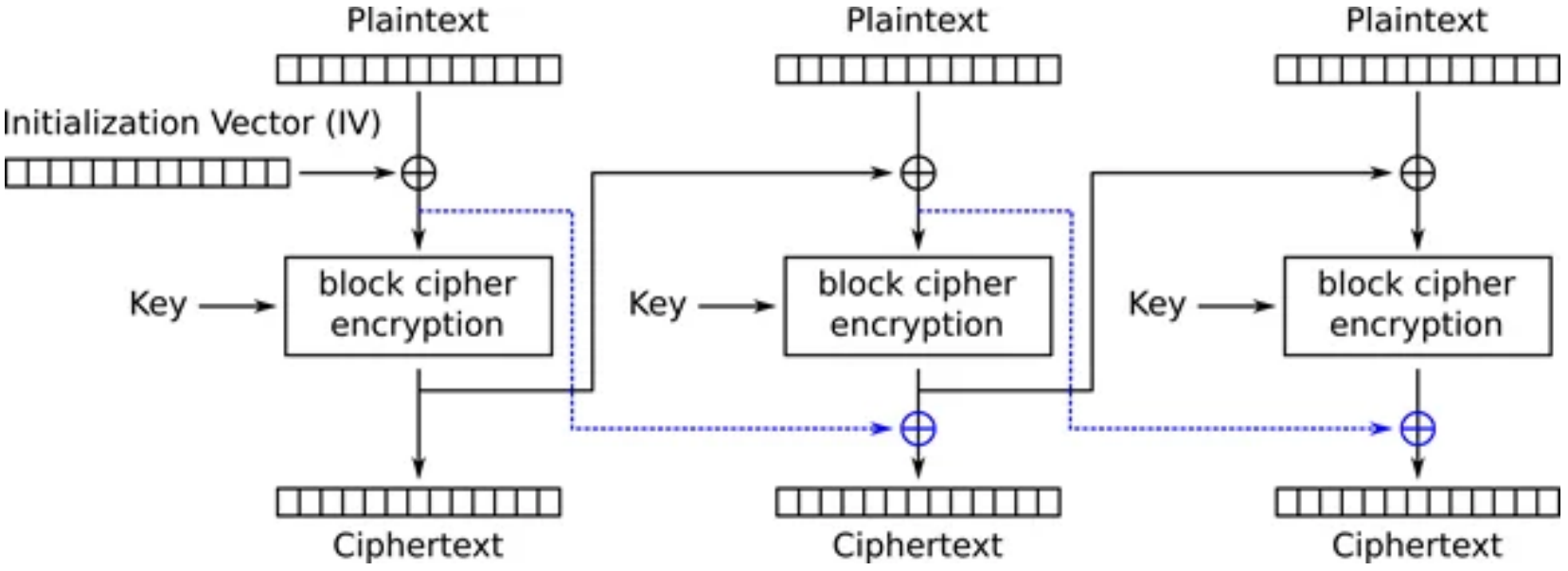
# UCWeb

* Led to discovery of active comms channel from [redacted]

*(S//SI//REL TO USA, FVEY) The CONVERGENCE team helped discover an active communication channel originating from [redacted] that is associated with the [redacted] as they are known within the [redacted] hierarchy area of responsibility is for covert activities in Europe, North America, and South America. The customer [redacted] leveraged a* ***Convergence Discovery capability that enabled the discovery of a covert channel associated with smart phone browser activity in passive collection****. The covert channel originates from users who use UCBrowser (mobile phone compact web browser).* ***The covert channel leaks the IMSI, MSISDN, Device Characteristics, and IMEI back to server(s) in*** *[redacted] Initial investigation has determined that perhaps malware can be associated when the covert channel is established. [redacted] covert exfil activity identifies SIGINT opportunity where potentially none may have existed before. Target offices that have access to X-KEYSCORE can search within this type of traffic based on their IMSI or IMEI to determine target presence*

# Problems in browsers

- Non-standard algorithms (XOR masks, etc.)

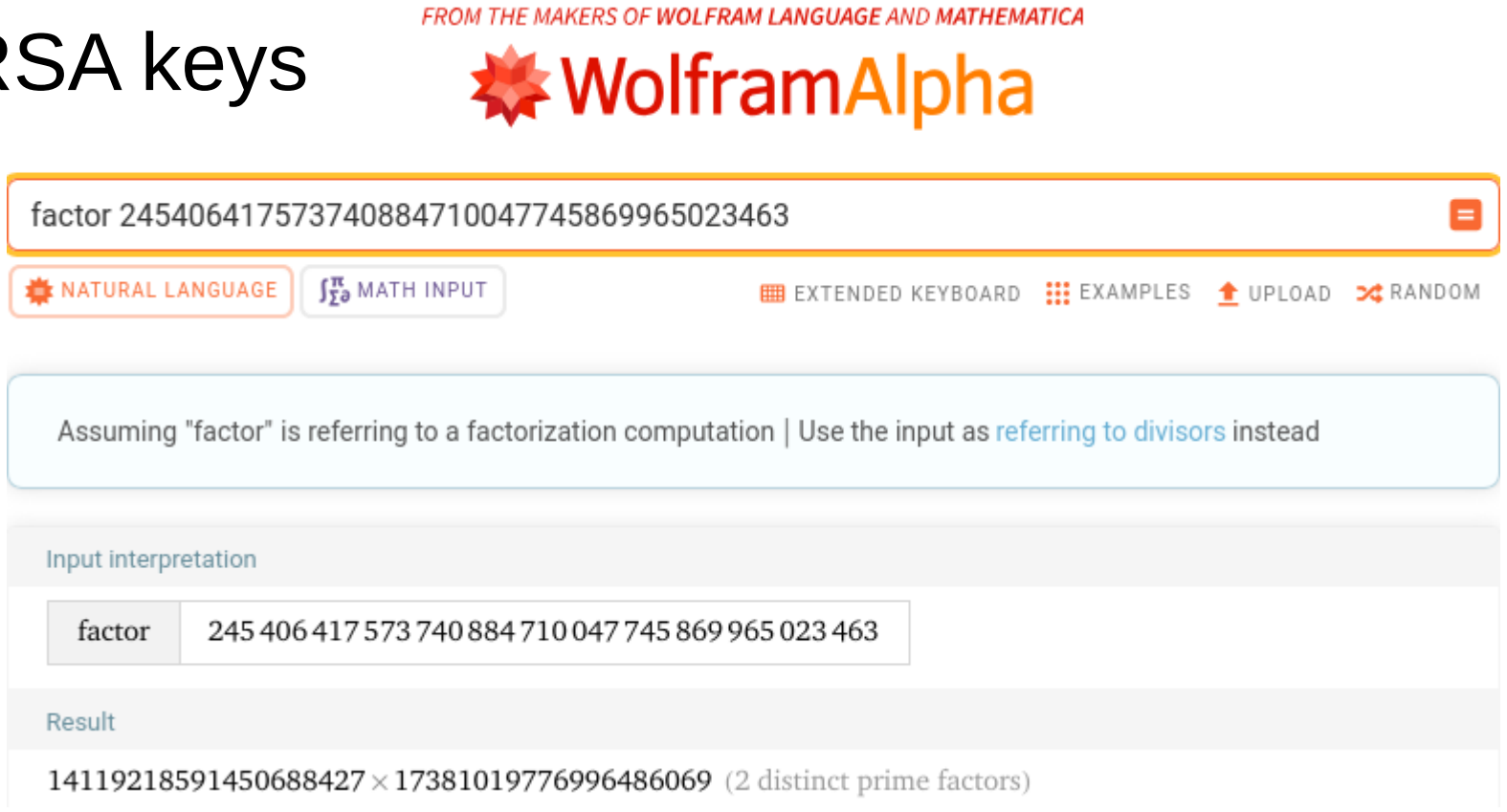


# Problems in browsers

- “Buggy” implementations
  - AES with a missing round
  - DES with byte-swapped constants (wrong endianness)
- Not always clear if intentional or not
  - Security through obscurity
  - Kerckhoffs’s principle

# Problems in browsers

- 128-bit RSA keys



# Problems in browsers

- RSA malleability problem

# Stalkerware

- It’s supposed to surveil you…
- But it’s network security is often insufficient, leaving victims open to surveillance by additional parties

# Keyboard apps

- For AI prediction, they may transmit everything you typed over the Internet
- We found that keyboard apps used by collectively one billion people had network security vulnerabilities
- **Eavesdroppers could read what you typed**

# Moral of the story

- **DON’T ROLL YOUR OWN CRYPTO**
- **USE STANDARD ENCRYPTION ALGORITHMS LIKE TLS/HTTPS**

# Moral of the story

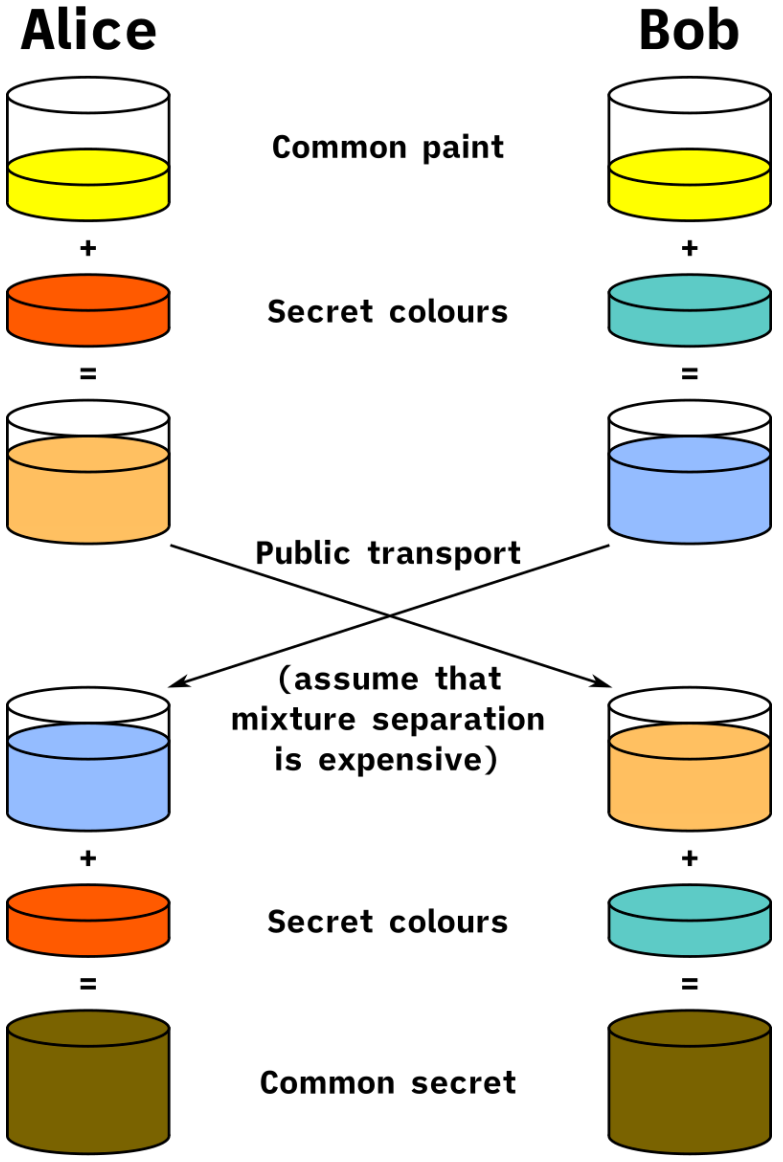
- If you want to help keep people safe
- Or protect vulnerable people’s rights
- Become one of the good guys
- Analyze popular software people use
- Help protect them from abusive surveillance

# Forward secrecy

- “Past communications cannot be decrypted even if the long-term private keys are found”
- Requires an *active* attack versus a *passive* attack
  - Passive: reading
  - Active: modifying

# Diffie-Hellman

- Key exchange algorithm
- Assumes difficulty of the discrete logarithm problem
- Agree on a symmetric key without ever transmitting it



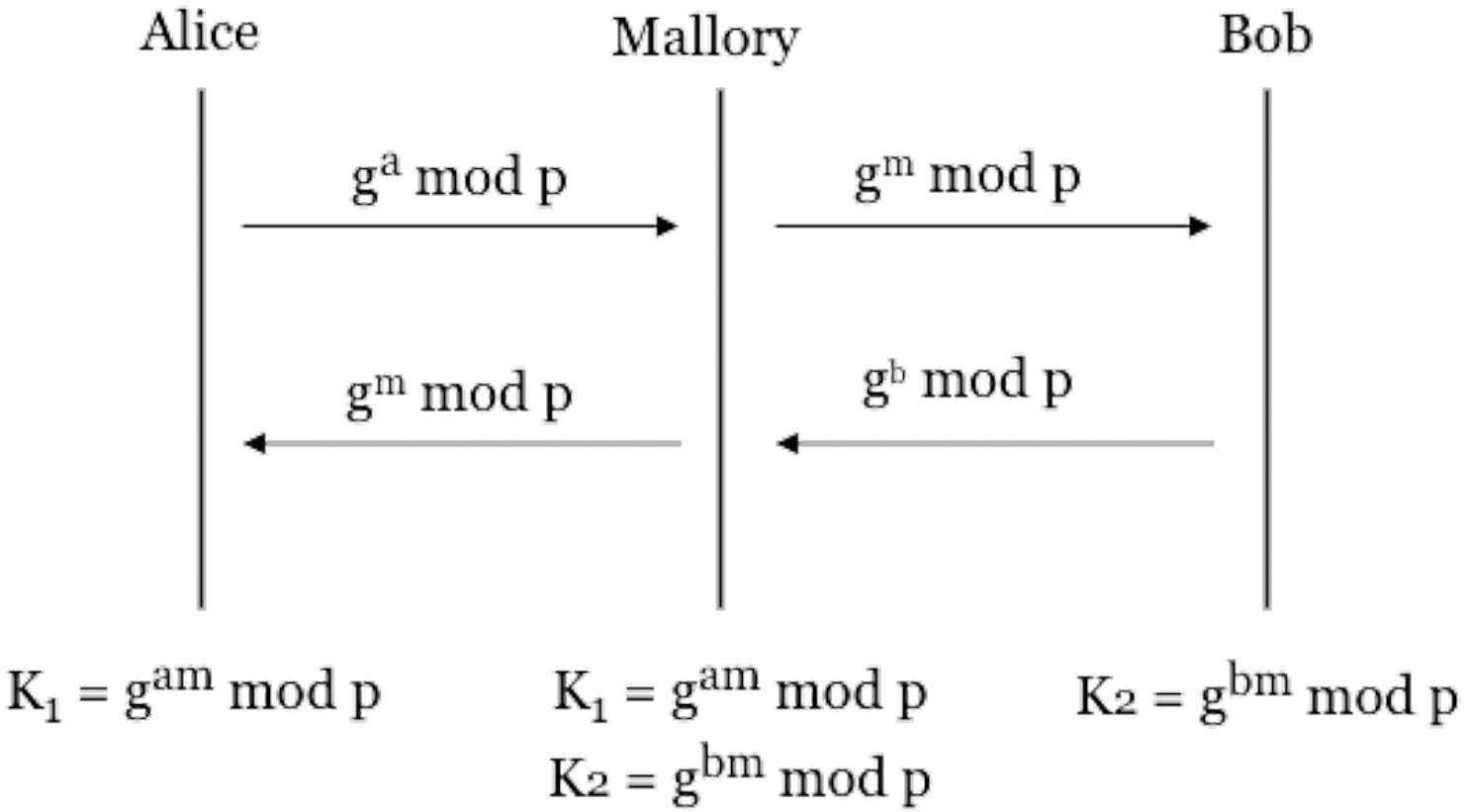
# Exchange between Alice and Bob

- Let $p$ be prime and $g$ be a primitive root mod $p$
- Assume $p = 23$, $g = 5$
- Alice picks secret $a = 12$ and sends $g^a \% p$ to Bob
- Bob picks secret $b = 4$ and sends $g^b \% p$ to Alice
- Alice computes $(g^b \% p)^a \% p = g^{ab} \% p$
- Bob computes $(g^a \% p)^b \% p = g^{ab} \% p$

# So we don’t need RSA?

- Just because we have asymmetric encryption doesn’t mean we don’t still need symmetric
- Similarly, we still need RSA (or another asymmetric algorithm) to exchange keys
- It just becomes *another* part of the process
- Diffie-Hellman by itself **provides no authentication**
- Server uses RSA to sign $g^x \% p$

# Active attack possible



# Assignment 6

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