Encryption and Surveillance

Previously we covered:

1. Schemes: How do we do encryption and decryption?
    a. The way by which you encrypt and decrypt messages (the code you run)
2. Conditions of what it means to be secure
    a. This entails an analysis of the system and its properties
3. Proof that the scheme satisfies the definition

Most of this class will be focused on Schemes and definitions. Cryptography focuses more on the proofs, but it is important for our class to understand the difference.

One time pad scheme (OTP) for one bit messages:
Alice has a key of length 1 bit (k), Bob has the same key
Encryption of the message m using k to create cipher text c: c = m xor k
Decryption of cipher text c using k: m = c xor k

New Material

1. Kerckhoff's Principle**:** "The enemy knows your system." In other words, a crypto system should be secure even if everything about the system is public **EXCEPT** for the secret key
    a. **DON'T** make up a secret encryption algorithm and keep it private. You shouldn't assume that just because no one currently knows about your system that no one can potentially know about your system.
    b. Attack surface should be minimized
    c. Many countries have many standards used for encryption.
    d. Secure systems arise from secure keys
    e. Everything about this "game" should be public so that as many people can play the game as possible so that the vulnerabilities would come to light sooner and publicly
2. Don't use the same key twice for One Time Pad Scheme
    a. Say you want to use the OTP to encrypt an l-bit message
        i. l = m1 m2 … ml
    b. Choose an l bit random key
        i. k = k1 k2 … kl
    c. Create the cipher text by performing bitwise xor on l and k
        i. c = l xor k
3. Reusing the same key for different messages is a terrible idea
    a. DONT DO THIS
        i. C1 = k xor m1
        ii. C2 = k xor m2
        iii. …

        iv. Cl = k xor ml
    b. Performing a brute force attack on One Time Pad Scheme with repeated key involves only two attempts, but if you choose an l-bit random key, you need 2^l
4. Flaws with OTP
    a. c1 xor c2 = (m1 xor k) xor (m2 xor k) = m1 xor m2 xor k xor k = m1 xor m2 xor 0
        i. Consider that m1 is an internet packet and you know that the first bit must be 0 (hypothetically).
            1. Since you know m1 should be 0, you can use the parts of the plain text are known, you can find other parts of the plain text
    b. This is also horrible because you can perform a frequency analysis on, not just letters, but xors of letters too (ie E xor E is most popular, E xor T is second most popular, T xor T is third, etc)
    c. One time pad is not used in practice because for each bit you send, you'd need to store another bit to decrypt/encrypt
5. <u>Stream Ciphers</u>
    a. The process
        i. Given a uniformly random 128-bit seed (2^128 equally likely possibilities), a pseudorandom generator (PRG) produces a pseudorandom number (endless stream of bits that look random but aren't, hence "pseudorandom")
            1. Ex 128 bit seed, s -> generator expands the seed -> <u>keystream</u>: a long series of bits (010101010….)
            2. Given an l-bit message, m and a seed, s we can generate an l-bit keystream and perform xor on it to decrypt and encrypt
            3. Alice and Bob
                a. Alice feeds seed into PRG to get keystream. Performs bitwise xor between k and m to encrypt. Bob would put a seed into PRG to get **the same** keystream since the algortihm used in the PRG is **deterministic** (inputting the same seed multiple times will result in the same keystream)
    b. **The randomness comes from the seed, not from the PRG algorithm.**
    c. Pass through the limitations of the one time pad because instead of the key being the same size as the message, we just need to use the short seed as a smaller representation of the actual key used
    d. Don't reuse bits from the keystream (for the same reason why you shouldn't for OTP)
6. What is randomness?
    a. With knowledge of the seed, the output of the PRG is deterministic as explained before
    b. However, if you don't know the seed (it's random and unknown), the output of the PRG **should look** random
7. Definition of a secure PRG (don't worry about implementation details for now)

    a. PRG is secure if for all polynomial time adversary they are playing the following game
        i. Adversary is playing the game in one of two worlds without the seed but can't know which world he's in
        ii. World 1: Adversary ask a box for bits, the box returns pseudorandom bitstring
        iii. World 0: Adversary asks a box for bits, the box returns a random bitstring
        iv. If adversary had the seed, the adversary can easily know which world he's in
8. Block cipher: Pseudorandom permutation (PRP)
    a. Permutation is a mathematical term where each input maps to exactly one output and each output maps to exactly on input (1 to 1 mapping)
        i. Hash function is not a permutation since multiple inputs can live to a nondistinct output
        ii. Permutation of size 4: 4! Possible permutations
            1. Pick one of these 4! Permutations to get a random permutation
    b. Gives a random permutation instead of a keystream from PRG
        i. Key + Input ---------Block Cipher containing PRP---------> output
        ii. Given that the pseudorandom permutation is {a:e, b:q, d:f, c:g}
            1. Inputting a into the PRP would spit out e
            2. Inputting d into the PRP would spit out f
            3. Suppose a, b, c, and d were bitstrings of size 128. Input space is 2^128. Output space is 2^128. Number of random permutations (1-1 mappings) between input and output spaces is (2^128)!
    c. Just like the stream cipher, if you know the key it doesn't look random, but otherwise seems so.