Cryptography

CMSC 414

October 2, 4, & 11, 2017

Goals of Cryptography

There are two main goals of cryptography:

- Keep secrets secret (*Confidentiality*, *Privacy*, *Anonymity*)
- Ensure that data is correct (*Integrity*, *Authenticity*)

Terminology

Cryptography and related topics have terminology all their own:

- Techniques
- Technologists
- Systems and operations
- Participants

Keeping Secrets

We want to replace some or all of our message with something else, and being able to recover the original

Encoding refers to replacing a semantic unit (generally, a word or phrase) with something else

Enciphering refers to replacing individual letters or bits with something else

Plaintext or **Cleartext** refers to a message as written and intended to be read

Ciphertext refers to a transformation of a plaintext so that it cannot be read, other than by the intended recipient

Code Examples

Codewords can replace single or multiple words

Telegraph operators have standard replacements using 4-letter non-words for common longer phrases

- CIMA: Reserved seats fully booked
- EZUK: I am arranging supply of trucks
- JOPO: No damages noted on discharge

A criminal organization replaced “kilo of cocaine” with “shirt”
$\Rightarrow$ Got awkward when the police intercepted a call promising delivery of $3\frac{1}{2}$ shirts

Cipher Types

Classified based on unit of operation

Substitution and **Stream Ciphers** transform one symbol at a time

Block Ciphers transform multiple symbols as a group

Substitution and Stream Ciphers

- **Vigenère Ciphers** encode by $C = P + K \mod 26$, and decode by $P = C - K \mod 26$
- **Monoalphabetic Ciphers** replace a letter with the same letter every time (might be **keyed**)
- **Polyalphabetic Ciphers** replace a letter with a letter from a rotating series of monoalphabetic ciphers
- **One-Time Pads** are like a polyalphabetic cipher with as many alphabets as there are characters in the message; that is, there is no repetition of the permuting pattern

Any substitution or stream cipher that repeats its permutations is vulnerable to a **frequency analysis** attack. This takes advantage of the fact that individual letters appear with different frequency in written English, as do various $n-$grams (patterns of n letters).

Group Exercise 1

A simple *keyed monoalphabetic cipher* uses a keyword or phrase, either with no repetitions or any repetitions removed (eg, "secret" would become "secrt"), and the remainder of the alphabet following to create a *permutation*. A plaintext "a" would be replaced with the first letter in the permutation, "b" with the second, etc. With "secrt" as the the keyword, the cipher would be:

```
 plaintext: abcdefghijklmnopqrstuvwxyz
ciphertext: secrtabdfghijklmnopquvwxyz
```

The `crypto-exercises` repository has frequencies up to 9−grams for English in `frequencies.txt`. There is also `cipher1.txt`, containing a message enciphered with a keyed monoalphabetic cipher. Try to recover the plaintext *and* the keyword. Division of labor among your group is strongly advised.

One-Time Pads

Consider a stream of bits that form a message M, where bit i is denoted M_i

In parallel, consider a stream of bits K that are randomly generated, so that any given bit has equal probability of being a 0 or 1; bit i is denoted K_i

The ciphertext C is defined as $C_i = M_i \oplus K_i$

Provides *information-theoretic security* — a ciphertext can conceivably decipher to *any* plaintext of the same length, and C_i has equal probability of being a 0 or 1

Requires a reliable stream of random numbers, shared by sender and receiver, with no reuse of the stream (this is what *Quantum Key Distribution* does)

Block Ciphers

- *Playfair* replaces pairs of letters based on rectangles within a grid
- **DES** and **AES** replace fixed-size blocks of bits according to a **key** and a complex algorithm of shifts and mathematical operations
- **RSA** also operates on fixed-size blocks, but has two keys, which are related by a mathematical property

Technologists

Cryptographers create ciphers

Cryptanalysts break ciphers

Cryptologists study ciphers, both how they're created and how they're broken

- generally not as thoroughly as cryptographers and cryptanalyts
- often focus on how to *use* cryptography

Systems and Operations

Encryption is the process of transforming a plaintext into the corresponding ciphertext

Decryption is the process of transforming a ciphertext into the corresponding plaintext

A **Cryptosystem** is the combination of an encryption and decryption scheme, which may involve multiple algorithms to prepare and modify the inputs and outputs, as well as protocols to use those schemes

Participants

We generally use the following names:

Alice, **Bob**, and **Carol** are the typical participants in a cryptosystem. They may be *cooperative* or *adversarial*.

Eve is an eavesdropper, and may have wiretapping abilities:

Passive Wiretapping Eve can see all messages exchanged

Active Wiretapping Eve can also add, remove, or modify messages between when they're sent and received

Trudy is an intruder

Some Notation

Cryptography involves math, so let's review

Specifying a function's *domain* and *range*:

$$f : \mathbb{D} \mapsto \mathbb{R}$$
$$H : \{0,1\}^* \mapsto \{0,1\}^n$$
$$E : \{0,1\}^k \mapsto \{0,1\}^k$$

Specifying a function's transformation on an input:

$$f : x \to f(x)$$
$$E : m \to m^e \mod n$$
$$D : c \to c^d \mod n$$

Categorizing Algorithms

A **Cryptographic Hash** function is a *one-way function* of the form

$$H : \{0,1\}^* \mapsto \{0,1\}^n$$

such that $c = H(m)$ is easy to compute, but $m = H^{-1}(c)$ (the **preimage**) is infeasible

A **Symmetric-Key Cryptosystem** has a single key, shared by the sender and receiver; $c = E(k, m)$ and $m = D(k, c)$ are easy to compute with the key k, but infeasible to compute otherwise

An **Asymmetric-Key Cryptosystem** has a public key that can be shared with any potential senders and a private key known only to the key owner; $c = E(K_{\text{pub}}, m)$ can be computed by anyone, but $m = D(K_{\text{priv}}, c)$ can only be computed efficiently by the key owner

Data Correctness

Cryptography can provide more than just secrecy

Cryptographic hashes can provide an *Integrity* check on data

Need some way to check the integrity of the hash
$\Rightarrow$ **Digital Signatures**

Digital Signatures

Asymmetric cryptosystems provide the following functions:

$$E : \{0,1\}^b \mapsto \{0,1\}^b; D : \{0,1\}^b \mapsto \{0,1\}^b$$

Since the domain and range of E and D are identical, we can switch them:

$$s = D(K_{\text{priv}}, m); m = E(K_{\text{pub}}, s)$$

s is a **digital signature** of m using the private key K_{priv}, which can be verified using the public key K_{pub}

Typically, $m = H(M)$ for some cryptographic hash function H

Evaluating Cryptosystems

The **Random Oracle** model is the standard model of cryptographic analysis

The oracle produces random output for any input, but the same input will always produce the same output

A **cryptographic primitive** is **cryptographically secure** if it's *indistinguishable* from a random oracle

Birthday Paradox

Counter-intuitive statistical result

Given m samples from a set of N options, the probability of a collision within the samples becomes appreciable when $m \approx \sqrt{N}$

$H : \{0,1\}^* \mapsto \{0,1\}^b$ likely to have collisions if $2^{b/2}$ inputs are tried

This has implications for

- choice of block size based on computing capabilities
- number of times a block cipher can safely be used

Cryptosystem Principles

Kerckhoffs's Principle: "A cryptosystem should be secure even if everything about the system, except the key, is public knowledge."

Schneier's Law: "Any person can invent a security system so clever that he or she can't imagine a way of breaking it."

$\Rightarrow$ ***There is no security through obscurity!***

Group Exercise 2

Playfair uses a keyword or phrase, similar to the monoalphabetic substitution cipher in the previous exercise. However, it arranges the alphabet in a 5x5 grid (with "j" excluded and replaced by "i" in the plaintext), and performs pairwise substitutions. Double letters are split with an "x" if they would be in the same block, and the message *might* be padded to make the total length an even number. With "secrt" as the the keyword, the grid would be:

```
secrt          some substitutions:
abdfg          el -> ri       to -> eu
hiklm          il -> km       ou -> pn
nopqu          ck -> dp       ly -> qr
vwxyz
```

The file `cipher2.txt` contains a Playfair-enciphered message. Again, try to recover the plaintext and the keyword, though you may only be able to recover the plaintext in class (using digram frequency analysis).

Symmetric Key Cryptosystems

Also called **Shared Key** cryptosystems

Sender and *receiver* have the *same key*

Key has to be transmitted by some **out-of-band** mechanism or generated by a **key-agreement protocol**

You can also use this to encrypt your own files, in which case you are in *both* roles, and there is no need to transmit the key

XOR Block Cipher

K is a block of length b

M is a message of length b

$C = M \oplus K$

Similar to One-Time Pad, *but...*

Breaking a Short XOR

```
AA D5 BF C0 A8 CA AA D5 AF C0 BC CF
```

We (somehow) already know $b = 16$ (2-byte XOR)

Break into 2 1-byte groups:

```
1: AA BF A8 AA AF BC
2: D5 C0 CA D5 C0 CF
```

Note repetitions — these are the same character in the plaintext!

$AA_1, D5_2, C0_2$

Breaking a Short XOR

Recall:

```
1: AA BF A8 AA AF BC
2: D5 C0 CA D5 C0 CF
```

$AA_1 \Rightarrow E(45) \Rightarrow K_1 = EF$

`E ? P ? G ? E ? @ ? S ?`

$AA_1 \Rightarrow T(54) \Rightarrow K_1 = FE$

`T ? A ? V ? T ? Q ? B ?`

$AA_1 \Rightarrow A(41) \Rightarrow K_1 = EB$

`A ? T ? C ? A ? D ? W ?`

Let's try some common letters!

Looks promising...

$D5_2 \Rightarrow T(54) \Rightarrow K_2 = 81$

`A T T A C K A T D A W N`

Longer blocks require more ciphertext, but technique is the same

This can be automated, based on derived letter frequencies

Better Symmetric-Key Ciphers

A good *block cipher* will employ:

Confusion Each bit in C depends on many bits in K (ideally non-linearly)

Diffusion Flip a bit in $M \Rightarrow$ flip about half the bits in C
Flip a bit in $C \Rightarrow$ flip about half the bits in M

Our simple XOR has *neither*

SP-Networks

Substitution and **Permutation**

S-boxes mix input bits into output bits (*confusion*)

- each input bit mixes into every output bit
- changing an input bit changes roughly half of the output bits
- *invertible* matrix
- usually block size is several S-boxes wide

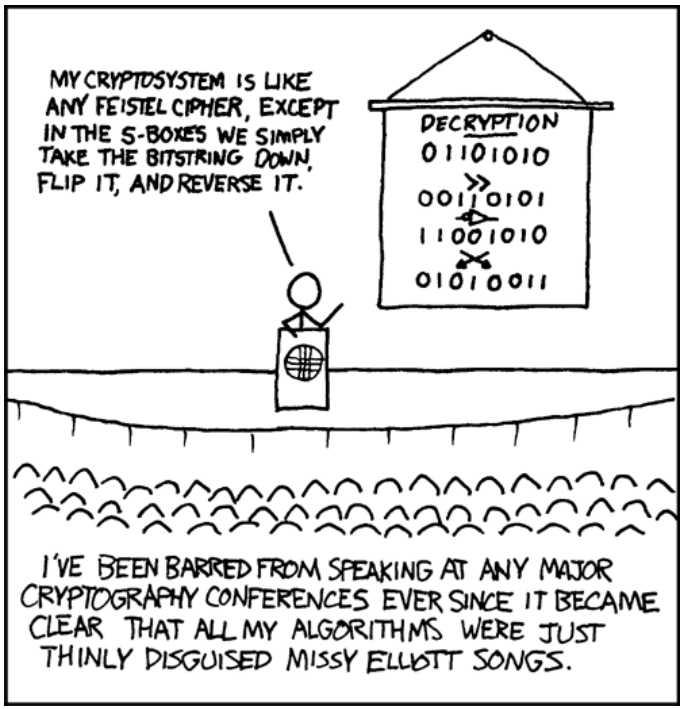
P-boxes permute outputs of S-boxes (*diffusion*)

- 1-1 mapping
- full-width of the block, to mix output from separate S-boxes

This defines *one round*, of which there are typically several

Feistel Ciphers

In general, Feistel Cipher is $\psi(f_1, f_2, ..., f_{2k})$
$\psi^{-1}(f_1, f_2, ..., f_{2k}) = \psi(f_{2k}, ..., f_2, f_1)$
Often, f_i built from a function F and keys K_i



Block size b bits
$F : \{0,1\}^{b/2} \mapsto \{0,1\}^{b/2}$
$K \to \{K_i\}, i \in [0, n]$ (**Key Schedule**)
$P \to (L_0, R_0)$ — L and R both $b/2$ bits
$L_{i+1} = R_i \qquad R_{i+1} = L_i \oplus F(R_i, K_i)$
$C = (R_{n+1}, L_{n+1})$
$R_i = L_{i+1} \qquad L_i = R_{i+1} \oplus F(L_{i+1}, K_i)$
$P = (L_0, R_0)$

F does *not* need to be invertible

DES

Data Encryption Standard

64-bit blocks, 56-bit keys (plus 8 parity bits)

16-round Feistel cipher, operating on 32 bits of data in each Feistel function

F is 8 different S-boxes and a P-box

S-boxes have 6-bit input and 4-bit output

DES is not secure $\Rightarrow$ 3DES is three applications of DES, with 3 (sometimes 2) different DES keys
$C = E_{K3}(D_{K2}(E_{K1}(P))), P = D_{K1}(E_{K2}(D_{K3}(C)))$

AES

3DES was good enough for a while; eventually replaced with **AES** (*Advanced Encryption Standard*)

Not a Fesitel cipher

Supports multiple key/block sizes: 128 (10 rounds), 192 (12 rounds), and 256 (14 rounds)

No practical attacks known (yet) against algorithm

Side-channel attacks exploit implementation details (an issue for all algorithms)

Group Exercise 3

Let's design an S-box! It will probably not be secure, but we won't worry about that.

Keeping things simple, let's have 4 bits of input and 4 bits of output. Things to consider:

1. Each bit of the input should contribute to every bit of the output.
2. Changing an input bit should change roughly half the output bits.
3. It must be invertible (perhaps with a different S-box)!

How might you include a symmetric key to create the full-block substitution step?

Modes of Operation

Rare to use a block cipher on only one block

Need a way to extend this to multiple blocks

- Electronic Code Book (ECB)
- Cipher Block Chaining (CBC)
- Output Feedback (OFB)
- Counter Encryption (CTR)
- Message Authentication Code (MAC)

Electronic Code Book

Simplest extension: $C_\mathrm{i} = E_K(P_\mathrm{i})$

Easy to parallelize, since each block is independent

Repeated plaintext blocks have same ciphertext

Even if from different messages!

Vulnerable to **splicing attacks**

OK for *challenge-response* systems

If you see a "secure" connection with "ECB" in the *ciphersuite*, be wary!

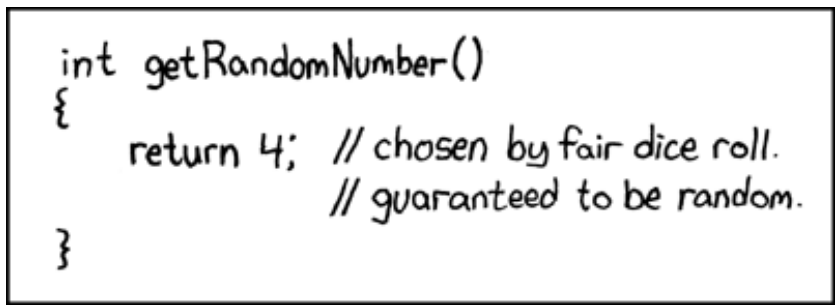
Cipher Block Chaining

$C_i = E_K(P_i \oplus C_{i-1})$

This obscures repeated patterns, and provides resistance against *splicing*

What about the first block?

Initialization Vector (IV) as C_0
$\Rightarrow$ *must be random!*



Can't parallelize block encryptions

Output Feedback

Does not run block cipher on plaintext

Instead, encrypts an IV:
$K_i = E_K(K_{i-1})$
$K_0 = IV$

Creates a **key stream**, used to create an *additive stream cipher*

$C_i = P_i \oplus K_i$

If you know a particular P_i, you can recover K_i
$\Rightarrow$ can replace with any block you like!
$\Rightarrow$ no *message integrity*

Can parallelize as many blocks as we are willing/able to store the keystream

Counter Encryption

Similar to OFB, *but*

$K_i = E_K(IV + i)$

Can compute K_i as needed to parallelize!

Still vulnerable to the attack described for OFB

CTR has a longer cycle length, though (2^n instead of $2^{n/2}$ for CBC and OFB)

Message Authentication Code

Not an encryption mode, but an integrity mode

CBC-MAC builds off of CBC mode, keeping only the *last* block

Effective for fixed-length messages

CMAC mixes K into the *next-to-last* block encryption to prevent an **extension attack**, where a variable-length message is concatenated with a malicious extension

We can also use a *cryptographic hash function* with a key to create a MAC

Cryptographic Hash Functions

$H : \{0,1\}^* \mapsto \{0,1\}^b$

Small changes in the input should result in large changes in the output

Given $H(x)$, it should be *very hard* to find x (there will be many)
$\Rightarrow 2^{b/2}$ guesses, on average

Algorithm	*b*	*Maximum message size*
SHA-1	160	$2^{64} - 1$
SHA-256	256	$2^{64} - 1$
SHA-512	512	$2^{128} - 1$

We can also build cryptographic hash functions from block ciphers
$\Rightarrow$ must consider Birthday Paradox!

Using Hash Functions

Given a good hash function H and a key K, we can create a good MAC

$MAC = H(K|M)$

- M is the message being authenticated
- $a|b$ denotes concatenation of a and b

We can also create a Feistel cipher:
$\psi(f_1, f_2, f_3, f_4) \qquad f_i(x) = H(K_i|x)$

The *Luby-Rackoff Result* proves this is a good block cipher

Encryption and Integrity

Encrypt-then-MAC

- Encrypt plaintext
- Keyed hash on ciphertext
- Used in IPsec

Encrypt-and-MAC

- Encrypt plaintext
- Keyed hash on plaintext
- Used in SSH

MAC-then-Encrypt

- Keyed hash on plaintext
- Plaintext and hash encrypted as a single message
- **CCM**: CBC-MAC and CTR
- Used in SSL/TLS

Group Exercise 4

In python, the `hashlib` module has methods like `sha1()`, `sha256()`, and `sha512()`. We know that we can use cryptographic hash functions to create Feistel block ciphers. Recall:

$P \rightarrow (L_0, R_0)$

$L_{i+1} = R_i \qquad R_{i+1} = L_i \oplus F(R_i, K_i)$

$C = (R_{n+1}, L_{n+1})$

$R_i = L_{i+1} \qquad L_i = R_{i+1} \oplus F(L_{i+1}, K_i)$

$P = (L_0, R_0)$

Try to create a Feistel block cipher using one of these hash functions. Make sure you define your block size, and use the hash output appropriately. You should be able to encrypt and decrypt messages using this cipher. ECB mode is fine. Can you come up with another way to create a stream cipher from a cryptographic hash function?

Limitations of Secret Key Cryptosystems

Secret keys are **pairwise** $\Rightarrow$ for n principals, $O(n^2)$ keys/exchanges

If someone distributes an encrypted file and then disappears, no way to verify file correctness or obtain a new key $\Rightarrow$ requires principals to be **online**

Key exchanges are often vulnerable to *man-in-the-middle* attacks, so we often **lack authentication**

How can we overcome these limitations?

Trusted Third Parties

One way is to use a **Trusted Third Party** (TTP)

If *Trent* is a trusted third party, then Alice and Bob can use Trent as an intermediary

Each principal only needs to exchange keys with Trent $\Rightarrow O(n)$, *good*

Possible to strongly authenticate Trent, since it only needs to be done once $\Rightarrow$ *good*

Trent becomes a bottleneck and central point of failure $\Rightarrow$ *bad*

Trust Models

Just like a *Security Model* or *Threat Model*, we also need a **Trust Model**

- Who are we trusting?
- Who is doing the trusting?
- What are we trusting them with?
- What are we trusting them to do or not do?

Trusting Trent

What is the trust model for our TTP?

- *Who are we trusting?*
  Trent
- *Who is doing the trusting?*
  Alice, Bob, and other communicants
- *What are we trusting them with?*
  all of the communications, including timely delivery
- *What are we trusting them to do or not do?*
  deliver messages with strong *Confidentiality* and *Integrity*, and with constant *Availability*; do not disclose messages, drop or delay them, nor modify them

Kerberos

Kerberos uses symmetric key cryptography and a TTP

Delegates service support and decentralizes *some* of the TTP functionality

Authentication Server (AS) grants Alice a **Ticket Granting Ticket** (TGT) for a **Service Server** (SS), encrypted with that service's *symmetric key*
$\Rightarrow$ *Alice cannot decrypt this ticket*, but the Service Server can in order to verify her credentials

The AS is required for users to log in, but SS can interpret the TGT whether the AS is available or not

Further tickets can be granted hierarchically

Asymmetric Key Cryptosystems

Also called **Public Key** cryptosystems

Sender and *receiver* have *different* keys

Public key is used to encrypt, and can be given to anyone

Private key is used to decrypt, and must be kept secret by the **owner**

This is incredibly useful for communications, like email and messaging

Can also be used to generate *digital signatures*

Hard Problems

Form the core of any public key cryptosystem

One-way Trapdoor Functions

With some piece of information (the *private key*), a function inverse is easy to compute $\Rightarrow$ this is the *trapdoor* part

Without it, the inverse is infeasible to compute $\Rightarrow$ this is the *one-way* part

- *Factoring products of large prime numbers*
- *Discrete logarithms*

Modular Arithmetic

Most public key cryptography uses *modular arithmetic*

$a \equiv b \pmod{n} \iff a \mod n = b \mod n$

Some things are easy

- $a + b \mod n$
- $a - b \mod n$
- $a * b \mod n$
- $a^b \mod n$

Some things are hard

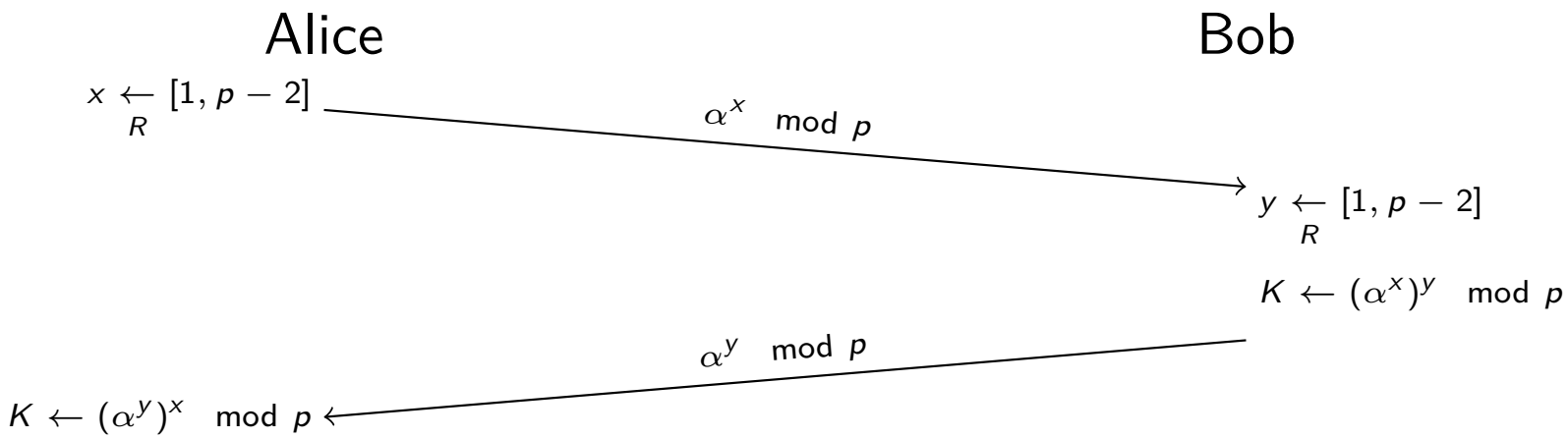
- find b such that $c \equiv a^b \pmod{n}$

Diffie-Hellman

Hard problem: given

- prime p
- α that generates $\mathbb{Z}_p^*$ (integers modulo p, excluding 0)
- $\alpha^a \mod p$
- $\alpha^b \mod p$

find $\alpha^{ab} \mod p$



These shared keys are discarded after a session completes

Group Exercise 5

Python handles large integers natively, making it a good prototyping environment for simplified cryptosystems. The ** operator does exponentiation. The `random` module can provide you with random numbers.

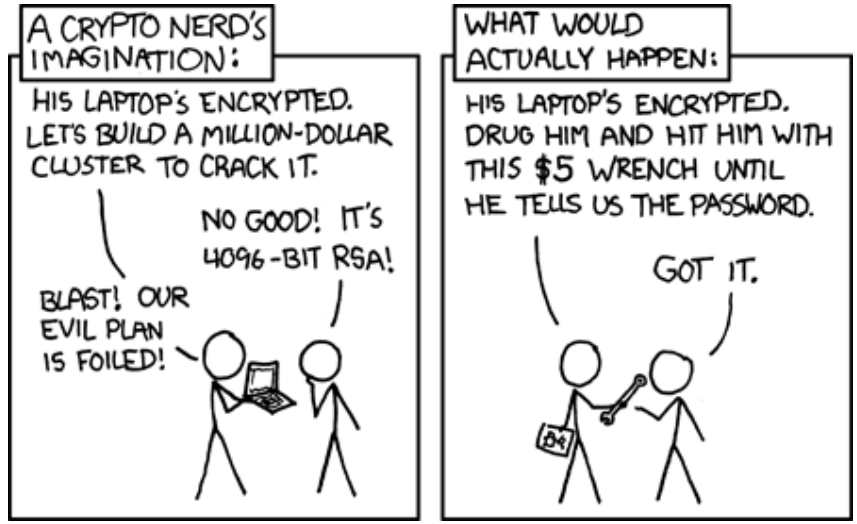
Write a simple program to implement Diffie-Hellman key agreement. You don't need to worry about communicating over the network, just demonstrate that you can create the shared key with each party only knowing its own random number.

Note that you'll need to find a suitable prime number. If you're testing n for primality, keep in mind that you only need to test up to $\sqrt{n}$.

RSA

Ron Rivest, Adi Shamir, Leonard Adleman — 1978

Most widely-used public key algorithm



First developed by GCHQ in 1973, but Official Secrets Act meant they couldn't take credit until 1997

RSA

Hard problem: given

- $n = p \cdot q$ for p, q prime (but known only to one party)
- $e \in \mathbb{Z}_n^*$ (integers modulo n, excluding $0, p \cdot a, q \cdot a \quad (\forall a)$)
- d s.t. $e \cdot d \equiv 1 \pmod{(p-1)(q-1)}$ (known only to one party)
- $m^e \mod n$

find m

With d, this is easy: $m \equiv (m^e)^d \mod n$

Computing d is hard without factoring n into p and q, even given e

$K_{\text{pub}} = (n, e), \quad K_{\text{priv}} = (p, q, d)$ e is usually 3 or 65537

$c = E(K_{\text{pub}}, m) = m^e \mod n, \quad m = D(K_{\text{priv}}, c) = c^d \mod n$

Weaknesses of RSA

The same message m encrypted multiple times will always be the same, allowing an attacker to launch a **chosen-plaintext attack** by trying potential messages

If a message m is encrypted with the same e and different n, n', the **Chinese Remainder Theorem** can be used to decrypt the message
$\Rightarrow$ easier with a small e, so $e = 65537$ is generally recommended

Random Padding of plaintexts can alleviate these problems, and several standards for padding exist

Practical Considerations of RSA

No two principals can have the same modulus n, or they will be able to decrypt each other's messages

A message m must be less than n

Due to the need for padding, m must be even smaller (fewer bits of padding means greater vulnerability to chosen-plaintext attacks)

Encryption is fairly fast, but decryption is rather slow

Long messages have to be split into many blocks

We will re-visit this soon

Other Public Key Cryptosystems

ElGamal generates a *pair* of elements as a ciphertext, and employs a random exponent which is incorporated into both elements

- It relies on the difficulty of solving the *Discrete Logarithm* problem

Elliptic-Curve Cryptograpy is based on a variant of the *Discrete Logarithm* problem as it relates to calculations involving elliptic curves

- There are multiple cryptosystems that employ elliptic curves
- Elliptic-curve cryptography typically requires much shorter keys than RSA for the same level of security

Digital Signatures

As previously discussed, a public key *encryption* scheme can be used to create a *digital signature* scheme

We will limit our discussion to RSA signatures, but there are other digital signature schemes to be aware of:

- The *Digital Signature Algorithm* is based on the ElGamal cryptosystem
- The *Schnorr* signature algorithm is similar to DSA, but is based on more general group theoretical principles

RSA Signatures

Define

$$S(K_{\text{priv}}, M) = D(K_{\text{priv}}, H(M))$$
$$V(K_{\text{pub}}, s, M) = H(M) \stackrel{?}{\equiv} E(K_{\text{pub}}, s) \pmod{n}$$

H is a cryptographic hash function, such as SHA-1 or SHA-256

If $s = S(K_{\text{priv}}, M) = H(M)^d \mod n$, then
$E(K_{\text{pub}}, s) = (H(M)^d \mod n)^e \mod n = H(M)$

A signature scheme thus requires both an *encryption* scheme and a *cryptographic hash* function

E and D are called *encrypt* and *decrypt* operations; S and V are called **sign** and **verify** operations