**Unit 1: Symmetric Cryptography**

**Introduction**

[Headshot]

Welcome to Applied Cryptography!

In this class, we’re going to learn about how to use secrets to solve problems in computing. We’ll learn the foundations of encryption, and protocols for using encryption to send messages securely, manage users and passwords, represent money, and sign documents.

For the first unit, we will introduce the basics of symmetric cryptography.

[end Headshot]

[write] Cryptography

underline: crypto means “secret” -

Ancient greek: κρύπτω - hide

-graphy = writing telegraphy, photography, etc.

cryptology – science of secrets

cryptanalysis – breaking/analyzing secrets

more correct name for this class would be cryptology – but cryptography is so common, we’ll use that.

[ Quiz ]

Which of these involve ***cryptology***?

[ ] Opening a door with a mechanical key

[ ] Playing poker

[ ] Logging into your udacity.com account

[ ] Doing a search using [google.com](http://www.google.com)

[ Answer ]

Mechanical key – shape of the key is the secret, not the physical object. If someone can take a picture of your key, even from a far distance, its not a secret anymore!

UCSD project http://www.jacobsschool.ucsd.edu/news/news\_releases/release.sfe?id=791

Poker – its all about secrets – wouldn’t be much of a game if everyone kept their cards face-up on the table.

Logging into udacity.com involves a lot of cryptography. Your password is a secret – you authenticate yourself to the website by showing that you know the secret. A website that manages passwords well never stores your actual password, but instead stores an encrypted version of your password. We’ll talk more in Unit 2 about how to manage passwords.

Google.com now defaults to using HTTPS for all signed-in users, so at least if you are logged into your google account, all the traffic between your machine and Google’s servers is encrypted. We’ll talk more in Unit 4 about how HTTPS works.

[ Course Overview ]

What will we do in this class:

Learn the foundations of symmetric and asymmetric cryptography

Symmetric: draw picture – same key for both encryption and decryptions

Asymmetric: different keys – also known as “public key cryptography”

This enables lots of cool thinks like secure web transactions

chains of trust, and digital signatures

We’ll see how to implement primitives for symmetric and asymmetric encryption – but I want to be very careful here! None of the code we will write for this will be secure enough to use for protecting national secrets, or even for your own personal information – writing secure encryption code is really hard, and you have to worry about more than just functional correctness. As one example, you have to worry about timing side channels.

[ Draw something ]

If the time your code takes depends in some way on the key or the data, this leaks information about your secrets. For most of the code we write in this class, we are trying to understand how things work by writing fairly simple code, not build fully secure implementations.

As a general rule,

DON’T IMPLEMENT YOUR OWN CRYPTO\*

[except for fun and learning]

You should use existing implementations in good libraries, that have been carefully vetted and tested.

[ Quiz ]

Should you use any of the code from this class to protect nuclear launch codes?

[ ] Yes

[ ] No

[ Answer ]

I hope everyone got that one right! If not, you’ll be hearing from our lawyers soon.

**Symmetric Encryption**

**Plaintext**

**Encrypt - decrypt**

**Ciphertext**

**M = set of messages**

**K = set of keys**

**C = set of ciphertexts**

**m \in M**

**k \in K**

**Kerckhoffs's principle – 1883**

Faut qu’il n’exige pas le secret, et qu’il puisse sans inconvenient tomber entre les mains de l’ennemi. ´

([A cipher] must not depend on secrecy, and it must not matter if it falls into enemy hands.)

**What parts of a cryptosystem should be kept secret?**

**[ ] Alice**

**[ ] Encryption Algorithm**

**[ ] Decryption Algorithm**

**[ ] Keys**

**[ ] Ciphertext**

Properties of a symmetric cipher:

Correctness:

D is the inverse of E: D\_k(E\_k(m)) = m for all m in M, k in K

Security: (will formalize this more later)

c = E\_k(m) does not reveal anything about m or k

**[ got to here ]**

**[ quiz ]**

**Which of these functions satisfy the correctness requirements for a symmetric cipher (but not necessarily the security requirements)? [assume M = {1, 2, …} N, set of natural numbers, K = {1, 2, … }]**

**[ x] E(m, k) = m + k**

**D(c, k) = c – k**

**[ x] E(m, k) = m**

**D(c, k) = c**

**[ x ] E(m, k) = m \* k**

**D(c, k) = c / k**

**[ ] E(m, k) = k**

**D(c, k) = c**

**A Perfect Cipher**

We’re going to start with a symmetric cipher that is actually unbreakable! In fact, it is really the only perfect cipher there is, at least in theory. In practice, it is rarely a practical cipher though, because it requires extremely long keys.

First, I want to introduce every cryptographers favorite function: XOR

A B A + B

0 0 0

0 1 1

1 0 1

1 1 0

What is the value of x + y + x?

[ ] 0

[ ] x

[ ] y

[ ] it depends on x

properties of xor:

x + x = 0

xor is commutative, associative and distributive. It is just addition mod 2!

x + y + y = x

M + K = C

C + K = M

One-time pad

E(P, K) = P ⊕ K

D(C, K) = C ⊕ K = (P ⊕ K) ⊕ K = P

Ciphertext: **0100111 1110101**

Key1: 1100100 0100110

Plaintext1: 1000011 1010011 = “CS”

What is a key that would make the given CipherText to “BS” instead? \_\_\_\_\_\_\_\_\_\_

1100100 0100110

Plaintext2: 1000010 1010011 = “BS”

**One-time pad**

History – Steve Bellovin’s discovery

**Analysis of One-Time Pad**

**Showing a Cipher is Secure**

Proving correctness is easy – just need to show invertibility; proving security is very hard.

How can we argue that a cipher is secure? This question will require some guessing, but see if you can answer it before I explain the answer. Order these by how convincing the argument is (1 is the best argument, 5 is the worst)

\_5\_\_ “I invented the cipher, and tried really hard to break it, but I couldn’t, so it must be super secure!”

\_3\_\_ Many very smart and highly motivated people tried very hard to break it but couldn’t, so it must be secure.

\_4\_\_\_ Since there are 834 quadrillion possible keys, it must be secure.

\_\_1\_ Mathematical proof that the ciphertext contains no information, without knowing the key.

\_2\_ Mathematical argument that breaking the cipher is at least as hard as solving some other problem which we believe is hard.

[ Answer ]

key size is an upper bound on how hard it is too break a cipher – small keys must be weak since an attacker could try them all, but large keys don’t provide any security guarantee.

The best is to have strong mathematical proofs --- but this is rarely possible, we usually have to rely on computational hardness proofs that argue that a cipher is secure because if someone could break it they could also break this other problem which we already know is secure. We’ll see an example of this with RSA and factoring in Unit 3, but we’ll get to that later. The one-time pad is a rare case where we can mathematically prove the cipher is secure, and this is what Claude Shannon did in the 1940s.

**Probability (Review?)**

Let \Omega be the universe of possible points. For now, assume \Omega is finite.

Simple example: flipping a coin: (show coin)

\Omega = { Heads, Tails }

If \Omega has a *uniform distribution*, each output is equally likely. We use P to mean the probability of some outcome.

P: outcome => [0, 1] real number

P(Heads) = 1/2

P(Tails) = ½

Mathematical coins that we hope to use in our cryptosystems behave like this, but real coins aren’t so perfect. Instead, let’s assume: <http://dilbert.com/strips/comic/1998-01-13/>

\Omega = { Heads, Tails, Edge }

P(Heads) = 0.49999

P(Tails) = 0.49999

P(Edge) = 0.00002

(results with your coins may vary)

An *event* is a subset of elements from a distribution.

P(E) = \sum P(w)

w \in E

[ quiz ]

What is P(E\_validtoss)?

E\_validtoss = { Heads, Tails }

[answer]

1 – P(edge) = 0.99998

Complementary Event Probability:

P (E) + P(\not E) = 1

\not E (bar over E) = \Omega **\** E

- set subtraction – draw picture

[quiz]

What is P({})?

[answer]

0

**Conditional Probability**

**Definition.** Given two events, A and B, in the same probability space (\Omega), the conditional probability of B given that A has occurred is:

P(B | A) = P(A \intersect B) / Prob(A)

[ Quiz ]

Given that a coin toss is valid, what is the probability that the coin is Heads?

\Omega = { Heads, Tails, Edge }

P(Heads) = 0.49999

P(Tails) = 0.49999

P(Edge) = 0.00002

E\_validtoss = { Heads, Tails }

1 – P(edge) = 0.99998

= P({ Heads, Tails } /\ { Heads }) / P(E\_validtoss)

= P({Heads}) / P(E\_validtoss) = 0.5 (note: not the case for real coin tosses!)

There’s a lot more to know about probabilities, but let’s get back to analyzing the one-time pad!

An event, E, is some subset of U: E <= U.

E\_heads = { ‘heads’ } # heads

E\_valid = { ‘heads’, ‘tails’ } # non-edge

If some event, x, is drawn from U.

Conditional Probability

P[A | B] = Probability of A given that B occurred

By definition: P[A | B] = P(A intersect B) / P(B)

Note if P(B) = 0 this doesn’t work, but conditional probability of A given B is meaningless

**[[[ start here ]]]**

**Perfect Cipher**

The ciphertext provides an attacker with no additional information about the message.

Which of these captures the property we want?

For k <- K, m, m\* \in M:

( - ) P[ m = m\* | E\_k(m) = c] = 1/|M|

(x ) P[ m = m\* | E\_k(m) = c ] = P [ m = m\* ]

( - ) P[ m = m\*] = P [ E\_k(m) = E\_k(m\*) ]

( - ) P[ m = m\* | E\_k(m\*) = c] = 1 / |K|

[ Answer ]

We want to know that an attacker who sees the ciphertext learns nothing about the actual message, a priori probability the message is m\*.

**One-Time Pad is Perfect Cipher**

[ messages ] -> [ ciphertexts ]

show complex mapping with Ki

P[E\_k(m) = c] = sum over all keys P[E\_ki(m) = c]

Which of these are the same?

[ x ] P[ c = E\_k(m) ] = P [ c = E\_k(m\*) ] <- same property!

**It’s perfect, why aren’t we done?**

* **Malleable: [draw a hammer]**
* **Impractical: | K | = | M |**

**Got to showing structure of machine**

**Not-quite one-time pads - Cryptanalysis**

Colossus

Channels

C = c\_0 c\_1 c\_2 … c\_n-1

<http://www.math.uwaterloo.ca/co/about/files/prof_tutte/corr98-39.pdf>

Each character is 5 bits – divide the ciphertext into channels

Z\_0 = c\_0 c\_5 c\_10 … = z\_0,0 z\_0,1 …

Z\_1 = c\_1 c\_6 c\_11 …

Z\_2 = c\_2 c\_7 c\_12 …

Z\_3 = c\_3 c\_8 c\_13 …

Z\_4 = c\_4 c\_9 c\_14 …

z\_c,i = m\_c,i + k\_c, i + s\_c, i

= output of the K wheels

= output of the S wheels

Let’s assume the k\_i’s are uniformly random. (but not that the real k\_i repeat frequently!)

What is P(z\_c, i == 0)?

[ ] less than ½ [x] exactly ½ [ ] more than ½

P(z\_c, i == 0) = m\_c,i + k\_c, i + s\_c, i

since we assume k is uniformly random,

P(k\_c,i == 0) == ½

So, whatever p\_r = P(m\_c,i XOR s\_c,i) is,

P(k\_c,i XOR m\_c,i XOR s\_c,i) == ½

\Delta z\_c,i = z\_c,i – z\_c,i+1

\Delta z\_0,i XOR \Deltz z\_1, i =

z\_0,i – z\_0,i+1 XOR z\_1,i – z\_1,i+1

Which of these has probability different from 1/2?

( ) P[z\_c, i XOR z\_c, i+1]

(x) P[z\_c, i XOR z\_c+1, i]

( ) P[z\_c, i XOR z\_c+1, i+1]

1. is 1/2:

m\_c,i XOR k\_c,i XOR s\_c,i XOR m\_c,i+1

XOR k\_c,i+1 XOR s\_c, i+1

**\*\*\* Finished breaking Lorenz – need to show Collosus**

**Modern Symmetric Ciphers and Cryptanalysis**

AES

Use in Python library

Simple use in keeping an encrypted file – all the problems with this that we’ll address in future units

**Jefferson’s Wheel**

Importance of separating key and mechanism

Introduction to randomized ciphers!

Set up unit 2: uses of symmetric encryption