-ography, -ology, -analysis

Cryptography (narrow sense): designing encryption
Cryptanalysis: breaking encryption
Cryptology: both of the above
Code (narrow sense): word-for-concept substitution
Cipher: the “codes” we actually care about

Caesar cipher

- Advance three letters in alphabet: A → D, B → E, ...
- Decrypt by going back three letters
- Internet-era variant: rot-13
- Easy to break if you know the principle

Keys and Kerckhoff’s principle

- The only secret part of the cipher is a key
- Security does not depend on anything else being secret
- Modern (esp. civilian, academic) crypto embraces openness quite strongly

Symmetric vs. public key

- Symmetric key (today’s lecture): one key used by all participants
- Public key: one key kept secret, another published
  - Techniques invented in 1970s
  - Makes key distribution easier
  - Depends on fancier math
**Goal: secure channel**

- Leaks no content information
  - Not protected: size, timing
- Messages delivered intact and in order
  - Or not at all
- Even if an adversary can read, insert, and delete traffic

**One-time pad**

- Secret key is truly random data as long as message
- Encrypt by XOR (more generally addition mod alphabet size)
- Provides perfect, “information-theoretic” secrecy
- No way to get around key size requirement

**Computational security**

- More realistic: assume adversary has a limit on computing power
- Secure if breaking encryption is computationally infeasible
  - E.g., exponential-time brute-force search
- Ties cryptography to complexity theory

**Key sizes and security levels**

- Difficulty measured in powers of two, ignore small constant factors
- Power of attack measured by number of steps, aim for better than brute force
  - E.g., exponential-time brute-force search
  - $2^{32}$ definitely too easy, probably $2^{64}$ too
- Modern symmetric key size: at least $2^{128}$

**Crypto primitives**

- Base complicated systems on a minimal number of simple operations
- Designed to be fast, secure in wide variety of uses
- Study those primitives very intensely

**Attacks on encryption**

- Known ciphertext
  - Weakest attack
- Known plaintext (and corresponding ciphertext)
- Chosen plaintext
- Chosen ciphertext (and plaintext)
  - Strongest version: adaptive
Certificational attacks

- Good primitive claims no attack more effective than brute force
- Any break is news, even if it’s not yet practical
  - Canary in the coal mine
- E.g., $2^{126.1}$ attack against AES-128
- Also watched: attacks against simplified variants

Fundamental ignorance

- We don’t really know that any computational cryptosystem is secure
- Security proof would be tantamount to proving $P \neq NP$
- Crypto is fundamentally more uncertain than other parts of security

Relative proofs

- Prove security under an unproved assumption
- In symmetric crypto, prove a construction is secure if the primitive is
  - Often proof the looks like: if the construction is insecure, so is the primitive
- Can also prove immunity against a particular kind of attack

Random oracle paradigm

- Assume ideal model of primitives: functions selected uniformly from a large space
  - Anderson: elves in boxes
- Not theoretically sound; assumption cannot be satisfied
- But seems to be sound in practice

Pseudorandomness and distinguishers

- Claim: primitive cannot be distinguished from a truly random counterpart
  - In polynomial time with non-negligible probability
- We can build a distinguisher algorithm to exploit any weakness
- Slightly too strong for most practical primitives, but a good goal

Open standards

- How can we get good primitives?
- Open-world best practice: run competition, invite experts to propose then attack
- Run by neutral experts, e.g. US NIST
- Recent good examples: AES, SHA-3
A certain three-letter agency

- National Security Agency (NSA): has primary responsibility for “signals intelligence”
- Dual-mission tension:
  - Break the encryption of everyone in the world
  - Help US encryption not be broken by foreign powers

Stream ciphers

- Closest computational version of one-time pad
- Key (or seed) used to generate a long pseudorandom bitstream
- Closely related: cryptographic RNG

Shift register stream ciphers

- Linear-feedback shift register (LFSR): easy way to generate long pseudorandom sequence
  - But linearity allows for attack
- Several ways to add non-linearity
- Common in constrained hardware, poor security record

RC4

- Fast, simple, widely used software stream cipher
  - Previously a trade secret, also “ARCFOUR”
- Many attacks, none yet fatal to careful users (e.g. TLS)
  - Famous non-careful user: WEP
- Not recommended for new uses

Encryption ≠ integrity

- Encryption protects secrecy, not message integrity
- For constant-size encryption, changing the ciphertext just creates a different plaintext
- How will your system handle that?
- Always need to take care of integrity separately

Stream cipher mutability

- Strong example of encryption vs. integrity
- In stream cipher, flipping a ciphertext bit flips the corresponding plaintext bit, only
- Very convenient for targeted changes
Stream cipher assessment

- Currently out of fashion as a primitive in software
- Not inherently insecure
  - Other common pitfall: must not reuse key(stream)
- Currently no widely vetted primitives

Note to early readers

- This is the section of the slides most likely to change in the final version
- If class has already happened, make sure you have the latest slides for announcements

Outline

- Crypto basics
- Announcements intermission
- Block ciphers and modes of operation
- Hash functions and MACs
- Building a secure channel

Basic idea

- Encryption/decryption for a fixed sized block
- Insecure if block size is too small
  - Barely enough: 64 bits; current standard: 128
- Reversible, so must be one-to-one and onto function
Pseudorandom permutation

- Ideal model: key selects a random invertible function
- i.e., permutation (PRP) on block space
  - Note: not permutation on bits
- "Strong" PRP: distinguisher can decrypt as well as encrypt

Confusion and diffusion

- Basic design principles articulated by Shannon
- Confusion: combine elements so none can be analyzed individually
- Diffusion: spread the effect of one symbol around to others
- Iterate multiple rounds of transformation

Substitution/permutation network

- Parallel structure combining reversible elements:
- Substitution: invertible lookup table ("S-box")
- Permutation: shuffle bits

Feistel cipher

- Split block in half, operate in turn:
  \((L_{i+1}, R_{i+1}) = (R_i, L_i \oplus F(R_i, K_i))\)
- Key advantage: \(F\) need not be invertible
  - Also saves space in hardware
- Luby-Rackoff: if \(F\) is pseudo-random, 4 or more rounds gives a strong PRP

AES

- Advanced Encryption Standard: NIST contest 2001
  - Developed under the name Rijndael
- 128-bit block, 128/192/256-bit key
- Fast software implementation with lookup tables (or dedicated insns)
- Allowed by US government up to Top Secret

DES

- Data Encryption Standard: AES predecessor 1977-2005
- 64-bit block, 56-bit key
- Implementable in 70s hardware, not terribly fast in software
- Triple DES variant still used in places
Some DES history

- Developed primarily at IBM, based on an earlier cipher named “Lucifer”
- Final spec helped and “helped” by the NSA
  - Argued for smaller key size
  - S-boxes tweaked to avoid a then-secret attack
- Eventually victim to brute-force attack

DES brute force history

- 1977 est. $20m cost custom hardware
- 1993 est. $1m cost custom hardware
- 1997 distributed software break
- 1998 $250k built ASIC hardware
- 2006 $10k FPGAs
- 2012 as-a-service against MS-CHAPv2

Double encryption?

- Combine two different block ciphers?
  - Belt and suspenders
- Anderson: don’t do it
- FS&K: could do it, not a recommendation
- Maurer and Massey (J.Crypt’93): might only be as strong as first cipher

Modes of operation

- How to build a cipher for arbitrary-length data from a block cipher
- Many approaches considered
  - For some reason, most have three-letter acronyms
- More recently: properties susceptible to relative proof

ECB

- Electronic CodeBook
- Split into blocks, apply cipher to each one individually
- Leaks equalities between plaintext blocks
- Almost never suitable for general use

Do not use ECB

5271
CBC

- Cipher Block Chaining
- C_i = E_K(p_i \oplus C_{i-1})
- Probably most popular in current systems
- Plaintext changes propagate forever, ciphertext changes only one block

CBC: getting an IV

- C_0 is called the initialization vector (IV)
  - Must be known for decryption
- IV should be random-looking
  - To prevent first-block equalities from leaking (lesser version of ECB problem)
- Common approaches
  - Generate at random
  - Encrypt a nonce

Stream modes: OFB, CTR

- Output FeedBack: produce keystream by repeatedly encrypting the IV
  - Danger: collisions lead to repeated keystream
- Counter: produce from encryptions of an incrementing value
  - Recently becoming more popular: allows parallelization and random access

Outline

- Crypto basics
- Announcements intermission
- Block ciphers and modes of operation
- Hash functions and MACs
- Building a secure channel

Ideal model

- Ideal crypto hash function: pseudorandom function
  - Arbitrary input, fixed-size output
- Simplest kind of elf in box, theoretically very convenient
- But large gap with real systems: better practice is to target particular properties

Kinds of attacks

- Pre-image, “inversion”: given y, find x such that H(x) = y
- Second preimage, targeted collision: given x, H(x), find x' \neq x such that H(x') = H(x)
- (Free) collision: find x_1, x_2 such that H(x_1) = H(x_2)
Birthday paradox and attack

- There are almost certainly two people in this classroom with the same birthday.
- \( n \) people have \( \binom{n}{2} = \Theta(n^2) \) pairs.
- So only about \( \sqrt{n} \) expected for collision.
- “Birthday attack” finds collisions in any function.

Security levels

- For function with \( k \)-bit output:
  - Preimage and second preimage should have complexity \( 2^k \).
  - Collision has complexity \( 2^{k/2} \).
  - Conservative: use hash function twice as big as block cipher.
    - Though if you’re paranoid, cipher blocks can collide too.

Non-cryptographic hash functions

- The ones you probably use for hash tables.
- CRCs, checksums.
- Output too small, but also not resistant to attack.
- E.g., CRC is linear and algebraically nice.

Short hash function history

- One the way out: MD5 (128 bit).
  - Flaws known, collision-finding now routine.
- SHA(-0): first from NIST/NSA, quickly withdrawn.
  - Likely flaw discovered 3 years later.
- SHA-1: fixed SHA-0, 160-bit output.
  - Attacks with complexity around \( 2^{60} \).
  - No collisions yet publicly demonstrated.

Length extension problem

- MD5, SHA1, etc., computed left to right over blocks.
- Can sometimes compute \( H(a || b) \) in terms of \( H(a) \).
  - \( || \) means bit string concatenation.
- Makes many PRF-style constructions insecure.

SHA-2 and SHA-3

- SHA-2: evolutionary, larger, improvement of SHA-1.
  - Exists as SHA-{224, 256, 384, 512}.
  - But still has length-extension problem.
- SHA-3: chosen recently in open competition like AES.
  - Formerly known as Keccak, some standardization details pending.
  - New design, fixes length extension.
  - Too early for wide use yet.
### MAC: basic idea
- Message authentication code: similar to hash function, but with a key
- Adversary without key cannot forge MACs
- Strong definition: adversary cannot forge anything, even given chosen-message MACs on other messages

### CBC-MAC construction
- Same process as CBC encryption, but:
  - Start with IV of 0
  - Return only the last ciphertext block
- Both these conditions needed for security
- For fixed-length messages (only), as secure as the block cipher

### HMAC construction
- $H(K \parallel M)$: insecure due to length extension
  - Still not recommended: $H(M \parallel K)$, $H(K \parallel M \parallel K)$
- HMAC: $H(K \oplus a \parallel H(K \oplus b \parallel M))$
- Standard $a = 0x5c^*$, $b = 0x36^*$
- Probably most widely used MAC

### Outline
- Crypto basics
- Announcements intermission
- Block ciphers and modes of operation
- Hash functions and MACs
- Building a secure channel

### Session keys
- Don’t use your long term password, etc., directly as a key
- Instead, session key used for just one channel
- In practice, usually obtained with public-key crypto
- Separate keys for encryption and MACing

### Order of operations
- Encrypt and MAC (“in parallel”)
  - Safe only under extra assumptions on the MAC
- Encrypt then MAC
  - Has cleanest formal safety proof
- MAC then Encrypt
  - Preferred by FS&K for some practical reasons
  - Can also be secure
Authenticated encryption modes

Encrypting and MACing as separate steps is about twice as expensive as just encrypting.

"Authenticated encryption" modes do both at once:
- Recent (circa 2000) innovation, many variants
- NIST-standardized and unpatented: Galois Counter Mode (GCM)

Ordering and message numbers

Also don't want attacker to be able to replay or reorder messages.

Simple approach: prefix each message with counter.
Discard duplicate/out-of-order messages.

Padding

Adjust message size to match multiple of block size.
To be reversible, must sometimes make message longer.
E.g.: for 16-byte block, append either 1, or 2 2, or 3 3 3, up to 16 “16” bytes.

Padding oracle attack

Have to be careful that decoding of padding does not leak information.
E.g., spend same amount of time MACing and checking padding whether or not padding is right.
Remote timing attack against CBC TLS published just last year.

Don't actually reinvent the wheel

This is all implemented carefully in OpenSSL, SSH, etc.
Good to understand it, but rarely sensible to reimplement it.
You'll probably miss at least one of decades worth of attacks.

Next time

Public-key encryption protocols
More about provable security and appropriate paranoia.