- Compression
  1. Fractal Compression
  2. Model Compression
- Cryptography
  1. Definitions and Primitives
  2. Protocols
     (a) Digital Signatures
     (b) Key Exchange
     (c) Authentication
     (d) Some Other Protocols
  3. Symmetric (private-key) Algorithms
     (a) Symmetric Ciphers
        i. Block Ciphers
     (b) DES (Data Encryption Standard)

1 Data Compression

To finish the section of the class on data compression, we discuss fractal compression and model compression before moving on to cryptography, our next topic.

1.1 Fractal Compression

To understand fractal compression, we begin by defining a fixed point and then show how they are used in fractal compression.

1.1.1 Fixed Points

Definition 1 (fixed point) *A function $f(x)$ has a fixed point x_f if $x_f = f(x_f)$.*

For example, if $f(x) = ax + b$, then $x_f = \frac{b}{1-a}$.

Unfortunately, computing a fixed point of a function is not always so easy. However, even if $x = f(x)$ is too complicated to solve, or if $f(x)$ is given to us as a black box, we can try finding x_f iteratively as follows:

guess an initial x_0
repeat
  $x_i \leftarrow f(x_{i-1})$
until x_i within some tolerance of $f(x_{i-1})$.

For example, if we set $a = .5$ and $b = 1$, our previous example becomes $f(x) = .5x + 1$. The iterative algorithm above converges to the fixed point $x = 2$ as shown:

$$\begin{aligned} x_0 &= 1 \\ x_1 &= .5x_0 + 1 = 1.5 \\ x_2 &= .5x_1 + 1 = 1.75 \\ x_3 &= .5x_2 + 1 = 1.875 \\ x_4 &= .5x_3 + 1 = 1.9375 \\ x_5 &= .5x_4 + 1 = 1.96875 \\ x_6 &= .5x_5 + 1 = 1.984375 \end{aligned}$$

In practice fixed points cannot always be found iteratively. However, for the functions used in fractal compression, they can. In addition, for the functions used in fractal compression, the iteration is guaranteed to converge to the correct fixed point ... that is, the fixed point corresponding to the encoded image.

1.1.2 Idea Behind Fractal Compression

The image x_f we want to transmit (or, more typically, a lossy version of it) is the fixed point of some function $f(x)$ which maps images to images. The encoder determines and transmits the function f and the decoder iteratively solves for the fixed point x_f, which is the image sent.

For example, we could use a function f in which each pixel was a linear combination of other pixels. In practice this is prohibitively expensive — the size of the function transmitted may well be larger than the original image.

Instead, we use fractals. For each block of the image, we try to find some larger portion of the image which when rotated and scaled is the same as the image in the smaller block. These transformations together form the function f which is ultimately transmitted. Figure 1 shows a possible step in computing the function f. The f' represents one transformation which rotates the larger light gray oval on the right by 45 degrees and scales it and gives it a different color in order to represent the smaller, darker oval on the left.

In practice not many iterations are needed to converge, within an acceptable tolerance, to the desired fixed point.

1.2 Model Compression

The assumption behind model compression is that all the data you want to compress is of one form. For example, your data might consist of a large number of faces. In that case, an efficient compression algorithm would be to create one "standard" face (in some sense this

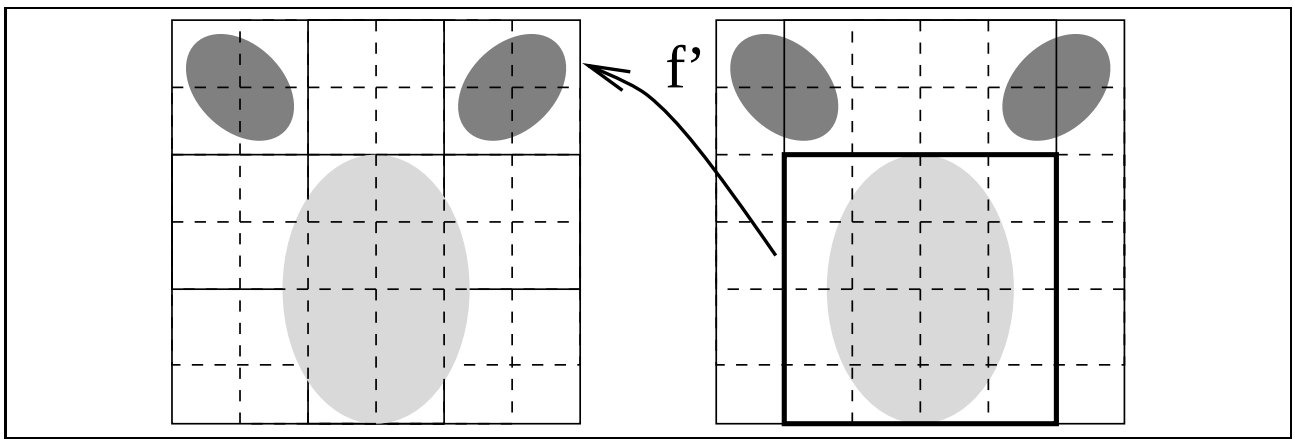

Figure 1: Example of a potential step in determining part of the function f

face should be an average of all the faces in your data set) and then to code the difference between each face in your data set and that standard face.

Where applicable, model compression is a very efficient way to compress data.

2 Cryptography

Like compression, this is a large field in its own right. In this class we focus on the algorithmic aspects of the field, paying little or no attention to other issues such as politics.

2.1 Definitions and Primitives

We begin with a series of basic definitions, building up to some primitives.

2.1.1 Definitions

Cryptography General term referring to the field.

Cryptology The mathematics behind cryptography.

Encryption Encoding. Sometimes this is used as a general term.

Cryptanalysis The breaking of codes.

Steganography The hiding of messages. This is an emerging field which studies the hiding of messages inside messages. A simple example is the hiding of a watermark in a piece of paper.

Cipher A method or algorithm for encrypting and decrypting.

A few more terms are best described with the aid of Figure 2. Referring to Figure 2, a symmetric algorithm is one where Key1 is the same as Key2. An asymmetric algorithm is one in which Key1 and Key2 are different.

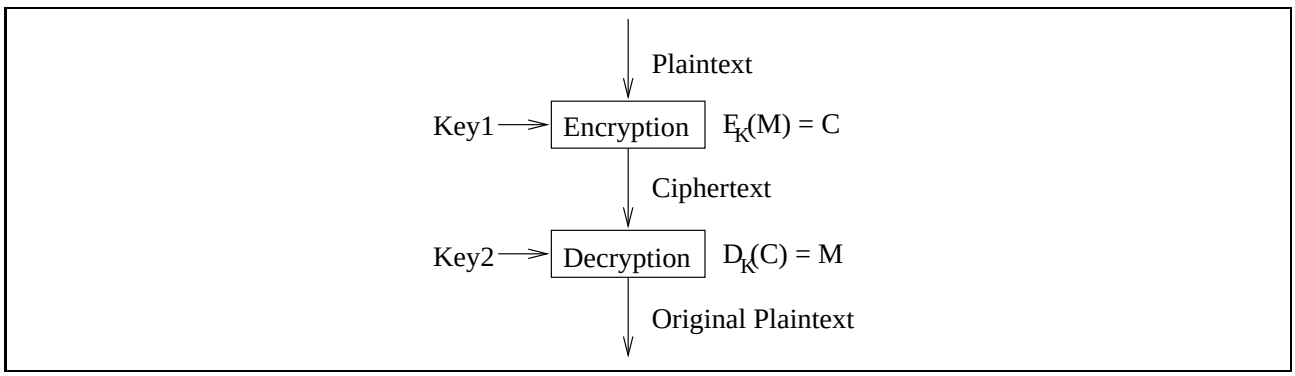

Figure 2: Flowchart showing the encryption and decryption of a message.

Next we list some commonly used names ... each name represents a specific role.

Alice initiates a message or protocol.

Bob second participant.

Carol third participant.

Trent trusted middleman.

Eve eavesdropper.

Mallory malicious active attacker.

Finally we define security by asking ourselves what it means to be secure. There are two definitions.

Truly Secure An algorithm is truly secure if encrypted messages messages cannot be decoded without the key.

In 1943 Shannon proved that for something to be truly secure, the key must be as long as the message. Informally, the intuition is that something is not secure if the message contains more information than the key.

An example of a secure algorithm is the *one-time-pad* in which the encoder and decoder both have the same random key which is the same length as the message. The encoder `XOR`s the message with the key, the decoder then decodes the message by `XOR`ing what he receives with the same random key. Each random key can only be used once.

Security Based on Computational Cost Here something is secure if it is "infeasible" the break — meaning that no (probabilistic) polynomial time algorithm can decode a message.

Notice that if $\mathcal{P} = \mathcal{NP}$, nothing is secure based on computational cost.

2.1.2 Primitives

Here we define several primitives, giving examples of most.

2.1.2.1 One-way Functions A one-way function $y = f(x)$ is a function where it is easy to compute y from x, but "hard" to compute x from y. One-way functions can be further classified based on the meaning of the word "hard".

Strong A strong one-way function f is one where for most y, no single polynomial time algorithm can compute x (i.e. $1 - \frac{1}{|x|^k}$).

Weak A weak one-way function f is one where for a reasonable number of y no single polynomial time algorithm can compute x (i.e. $\frac{1}{|x|^k}$).

Some examples of one-way functions are the following:

Factoring $y = u \cdot v$, where u and v are primes. If u and v are primes, it is hard to generate them from y.

Discrete Log $y = g^x \bmod p$, where p is a prime and g is a generator (i.e. $g^1, g^2, g^3, \ldots$ generate all i such that $0 \leq i < p$. This is believed to be about as hard as factoring.

DES with a fixed message $y = \mathrm{DES}_x(M)$.

Note that these are only conjectured to be one-way functions. Neither factoring nor discrete log are known to be in $\mathcal{NP}$.

2.1.2.2 One-way Trapdoor Functions This is a one-way function with a "trapdoor". The trapdoor is a key that makes it easy to invert the function.

An example is RSA, where the function is $y = x^e \bmod N$, where $N = p \cdot q$. p,q, and e are prime, p and q are trapdoors (only one is needed). So, in public-key algorithms, $f(x)$ is the public key (e.g. e, N), whereas the trapdoor is the private key (e.g. p or q).

2.1.2.3 One-way Hashing Functions A one-way hashing function is written $y = H(x)$. H is a many-to-1 function, so y is a fixed length. Calculating y from x is easy since it consists of just a function evaluation. Given some y, calculating any x such that $y = H(x)$ is "hard".

2.2 Protocols

Here we describe a number of basic protocols, showing possible implementations. We note that in general a protocol which can be implemented using a symmetric cryptosystem can also be implemented using an asymmetric cryptosystem — a primary difference is that the former may require a trusted middleman.

2.2.1 Digital Signatures

Digital signatures can be used to:

- Convince a recipient of the author of a message.
- Prove authorship.

- Make tampering with a signed message without invalidating the signature difficult.

They can be implemented both using symmetric and asymmetric cryptosystems as illustrated below.

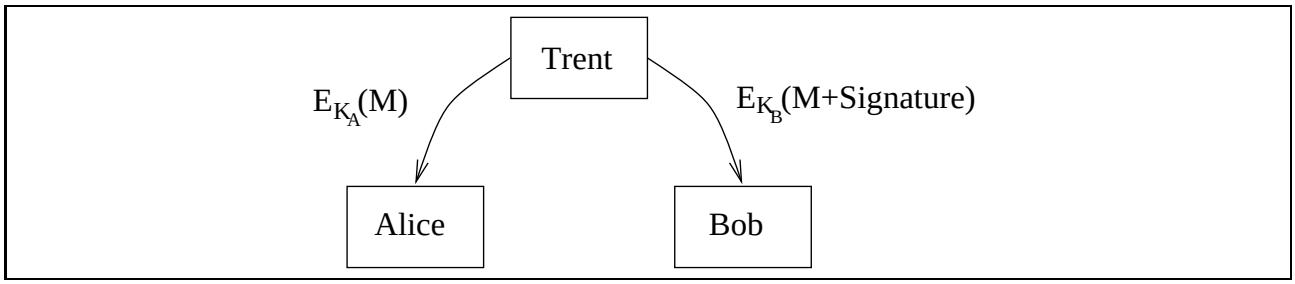

Figure 3: Flowchart showing an implementation of digital signatures using a symmetric cryptosystem.

Using a symmetric Cyptosystem: In Figure 3 Alice and Trent share the private key K_A and Bob and Trent share the private key K_B. Trent (a trusted middleman) receives a message from Alice which he can decrypt using K_A. Because he can decrypt it using K_A, he knows Alice sent the message. He now encrypts the decoded message plus a signature saying that Alice really sent this message using K_B. He sends this new version to Bob, who can now decode it using the key K_B.

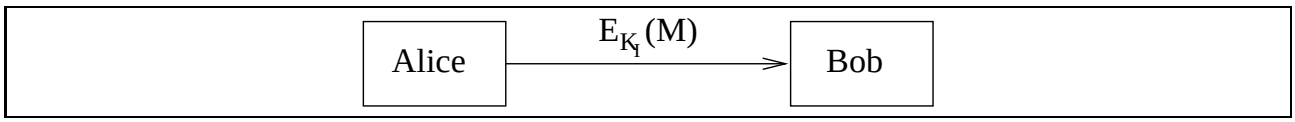

Figure 4: Flowchart showing an implementation of digital signatures using an asymmetric cryptosystem.

Using an asymmetric Cyptosystem: In Figure 4 K_I is Alice's private key. Bob decrypts the message using her public key. The act of encrypting the message with a private key that only Alice knows represents her signature on the message.

The following is a more efficient implementation using an asymmetric cryptosystem. In Figure 5 $H(M)$ is a one-way hash of M. Upon receiving M, Bob can verify that it

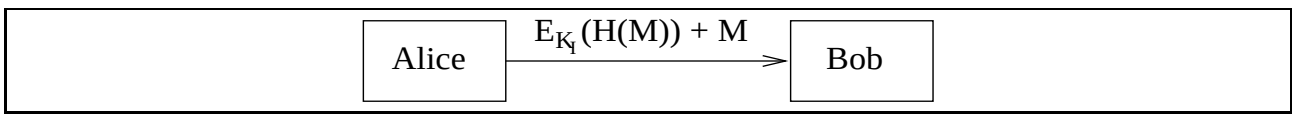

Figure 5: Flowchart showing a more efficient implementation of digital signatures using an asymmetric cryptosystem.

indeed come from Alice by decrypting $H(M)$ using Alice's public key K_I and checking that M does in fact hash to the value encrypted.

2.2.2 Key Exchange

These are protocols for exchanging a key between two people. Again, they can be implemented both with symmetric and asymmetric cryptosystems as follows:

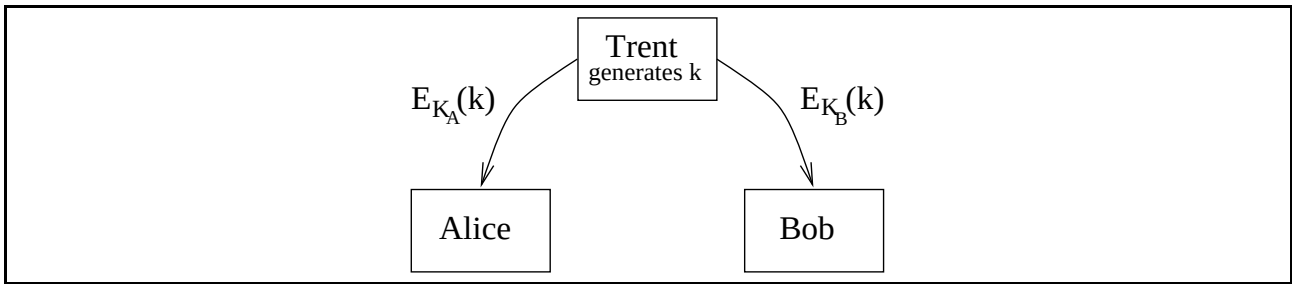

Figure 6: Flowchart showing an implementation of key exchange using a symmetric cryptosystem.

Using a symmetric Cyptosystem: In Figure 6, Alice and Trent share the private key K_A and Bob and Trent share the private key K_B. Trent generates a key k, then encodes k with K_A to send to Alice and encodes k with K_B to send to Bob.

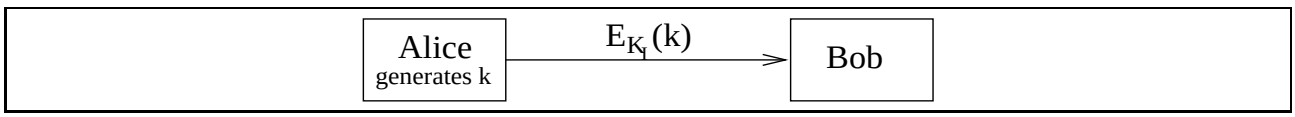

Figure 7: Flowchart showing an implementation of key exchange using an asymmetric cryptosystem.

Using an asymmetric Cyptosystem: In Figure 7 K_I is Bob's public key. Alice generates a key k, which she encrypts with Bob's public key. Bob decrypts the message using his private key.

2.2.3 Authentication

These protocols are often used in password programs (e.g. the Unix `passwd` function) where a host needs to authenticate that a user is who they say they are.

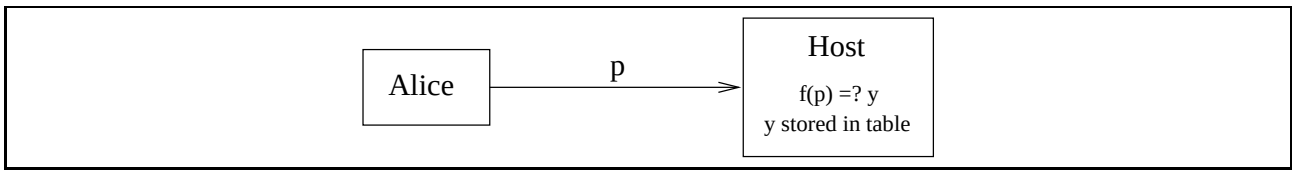

Figure 8: Flowchart showing a basic implementation of an authentication protocol.

Figure 8 shows a basic implementation of an authentication protocol. Alice sends p to the host. The host computes $f(p)$. The host already has a table which has y and checks to see that $f(p) = y$. f is a one-way function.

To make it less likely that Mallory will be able to pretend to be Alice to the host, sequences of passwords can be used. Sequences of passwords are used by having Alice give the host a random number R. The host computes $x_0 = R, x_1 = f(R), x_2 = f(f(R)), \ldots x_n = f^n(R)$ and gives all but x_n to Alice. The host keeps x_n. Now, the first time Alice logs in, she uses x_{n-1} as the password. The host computes $f(x_{n-1})$ and compares it to the x_n which it has. If the two are equivalent, Alice has been authenticated. Now the host replaces x_n with x_{n-1}. The next time Alice logs in, she uses x_{n-2}. Because each password is only used once by Alice, and f is a one-way function, sequences of passwords are more secure.

2.2.4 Some Other Protocols

Some other protocols which are not discussed here are:

- Secret sharing
- Timestamping services
- Zero-knowledge proofs
- Blind-signatures
- Key-escrow
- Secure-elections
- Digital Cash

2.3 Symmetric (private-key) Algorithms

We discuss block ciphers and then briefly introduce the well known and widely used block cipher DES. We begin discussion of how DES works, saving much for the next lecture.

2.3.1 Symmetric Ciphers (Block Ciphers)

Block ciphers divide the message to be encrypted into blocks (e.g. 64 bits each) and then convert one block at a time. Hence:

$$C_i = f(k, M_i) \qquad\qquad M_i = f'(k, C_i)$$

Notice that if $M_i = M_j$, $C_i = C_j$. This is undesirable because if someone (Eve or Mallory) notices the same block of code going by multiple times, they know that this means identical blocks were coded. Hence they've gained some information about the original message.

There are various ways to avoid this. For example, something called *cipher block chaining (CBC)* does the following:

$$C_i = f(i, C_{i-1} \oplus M_i)$$

Unfortunately, this could still cause problems across messages. Note that if two messages begin the same way, their encodings will also begin the same way (but once the two messages

differ, their encodings will be different from then on). Because many messages that people want to send begin with standard headers (e.g. specifying the type of file being sent), this means someone could gain information about the message being sent by creating a table of all the encrypted headers that they have seen.

One way to fix this problem is to attach a random block to the front of every message.

2.3.2 DES (Data Encryption Standard)

DES was designed by IBM for NIST (the National Institute of Standards and Technology) with "consultation" from the NSA in 1975.

DES uses 64-bit blocks and 56-bit keys. It is still used heavily, for example by banks when they encrypt the PIN that you type in at an ATM machine. However, DES is no longer very safe — supercomputers can crack it in a few hours. Nevertheless, because it is so common, we describe how it works here, beginning by describing some of the computing components used in DES.

Specifically, we describe some bit manipulations:

Substitution A fixed mapping. For example:

$$\begin{aligned} 000 &\rightarrow 011 \\ 001 &\rightarrow 101 \\ 010 &\rightarrow 000 \\ 011 &\rightarrow 001 \\ 100 &\rightarrow 100 \\ &\vdots \end{aligned}$$

This is often stored as a table.

Permutation A rearrangement of order. For example:

$$b_0 b_1 b_2 b_3 b_4 \rightarrow b_2 b_4 b_3 b_0 b_1$$

This is also often stored as a table.

Expansion Permutation A rearrangement of order that may repeat some bits of the input in the output. For example:

$$b_0 b_1 b_2 b_3 \rightarrow b_2 b_1 b_2 b_0 b_3 b_0$$