10. Symmetric Cryptosystems

First we formulate a general (deterministic) cryptosystem. Let M denote the set of all possible message chunks, let C denote the corresponding set of all cipher-text chunks. Let P be the set of all possible public keys and S the set of all possible secret keys. A general cryptosystem can be thought of as a pair of maps: the encryption map $E : M \times P \to C$ and the decryption map $D : C \times S \to M$. In addition we need an assignment map $a : P \to S$. The condition these maps must satisfy is that for each $p \in P$ the map $D_{a(p)} \circ E_p : M \to M$ must be identity. In a symmetric-key system the map a is the identity; in other words there is a single key space $K = P = S$ for encryption as well as decryption. In modern use such systems are used in contexts where the operations must be very rapid. Thus it is also customary to assume that M and C are of roughly the same size; for simplicity let us say $V = M = C$. Thus a symmetric-key system can be modelled as $E : V \times K \to V$ and $D : V \times K \to V$. It follows then that for each key k we obtain a permutation of the set V. Thus one naive approach would be to take K to be the collection of all permutations of V. However, we need to ensure that K is somewhat smaller than M as it would be rather pointless to exchange large length keys in order to exchange short messages! Secondly it is also clear that permutations that are of small order are risky as these patterns could be spotted. More generally, it is also a bad idea for K to be a subgroup of the permutation group as group theoretic relations between different keys could be used to attack the cryptosystem. A symmetric key cryptosystem thus attempts to obtain a "good" selection K of permutations of the set V.

To summarise, a symmetric-key cryptosystem consists of a map $K \to \mathrm{Perm}(V)$ from a set K of size less than the size of V to the permutation group of V.

10.1. **Types of operations.**

All block cipher operations are actually made up of conceptually simpler building blocks. Usually, the elements of V are represented as strings of a fixed length n out of an alphabet A, while elements of K are also represented as strings of some length k in the same alphabet ($k < n$).

There is one obvious way to obtain a permutation of V and that is to apply a permutation in Perm_n to permute the elements of the string. Such an operation is called a "Transposition" (which is confusing for group theorists familiar with the notion of *transpositions* as elements of the permutation group that interchange two elements!). Another obvious way to obtain a permutation on V is apply a permutation in $\mathrm{Perm}(A)$. Such an operation is called a "Substitution" since it substitutes one letter of the alphabet with another. It is clear that Substitutions and Transpositions commute with one another and form the subgroup $\mathrm{Perm}_n \times \mathrm{Perm}(A)$ of $\mathrm{Perm}(V)$. Thus, a cryptanalyst (or "code cracker") could try to solve the problem of finding the Substitution and Transposition independently, thus weakening the cryptosystem. Thus we need another operation that "mixes" the Substitutions with the Transpositions; appropriately this is known as "Mixing". One method used for mixing is "Polyalphabetic" Substitution; different substitutions rules are applied to different portions of the string; in addition a Transposition of these different portions can also be performed. A different procedure is break the string into "words" of m "letters" in the alphabet and directly find a nice permutation that performs "Word substitution" (using a "code book" for example).

Now only some of these operations need to depend on the key and the operations can be repeated in multiple "rounds" since the new collection is not commutative anymore (because of Mixing).

10.2. **Modes of enciphering.** An actual message is typically made up of many message blocks. One simple way to use the symmetric encryption would be to encrypt each block individually using E as above. The disadvantage of this is that after a suitable interval an attacker with access to plaintext-ciphertext combinations can set up a dictionary. Thus it is beneficial to consider other modes of functioning. The simplest method described above (encrypting block by block) is called Electronic Code-Book since it functions effectively like a (very large) code book.

In another mode, an additional "key"-like entity called Initialisation Vector is used which consists of one bit for each message block. This bit is added in to the key (in some specified way) to affect the encryption performed. This way the different blocks correspond to different keys and so the size of the dictionary required for a dictionary attack is much larger *even if* the initialisation vector is well-known. This method is known as Cipher Block Chaining, since the message cannot be decrypted unless the blocks are arranged in the sequence in which they were encrypted.

Other modes which depend on "feedback" from the output of the previous encryption block can also be used to chain the encryption procedure.

In addition one can construct ciphers with longer keys by constructing of chain of distinct ciphers each of which uses a sub-key. The two notable variants are Double encryption (usually performed with different algorithms) and Triple encryption (usually performed with the same algorithm but one of the three encryptions is actually a "decryption" operation (i. e. uses D rather than E).

10.3. **Cryptanalytic methods.** All that the above discussion gives us is various thumb rules about the options available in order to construct symmetric-key systems and certain guidelines about what to avoid (group, elements of small order, dictionary attack and so on). The actual construction of a symmetric-key "confusion function" is a mathematically non-unique "Black Art". Various ciphers with names like as DES (Data Encryption Standard), FEAL (Fast Encryption ALgorithm), IDEA, SAFER, RC5, Rijndael, Blowfish, Serpent and others have been proposed. A mathematician may balk at the question of choosing a "best" one from all these choices. Unlike public-key systems that are based on known hard problems in Theoretical Computer Science and Mathematics, the strength of symmetric-key systems is based on a lack of knowledge; if you can't figure out a way to crack the system then it must be secure. Thus it is imperative to know what techniques *can* be used to crack such systems. It is then possible to estimate how successful a given method will be based on currently available computing time and storage space.

While most cryptanalytic methods in "real-life" scenarios are based on partial information, it is best to examine a symmetric system as one would a "black box". The following tests come to mind:

Entropy of output: How random is the output given general input? One can measure the extent to which the output contains redundancies (as measured by Shannon's theory of information) for different inputs and keys.

Diffusion of input variations: Is the effect of small changes in the input localised? One can examine a large number of input pairs which differ by the same small change; is the change in the resulting output (for a fixed key) in the same place or does it "diffuse"?

Diffusion of key variations: Is the effect of small changes in the key localised? One can examine (over a large number of inputs) how two keys that differ by a small amount cause variation in the output. Is this difference "diffuse"?

Weak inputs and keys: Are there some simple inputs which have a lot of symmetry that produce output from which the key can be computed?

In addition, by examining the actual algorithm used one can further attempt to examine the following aspects:

Symmetries: The cryptographic operations should "break symmetry". If not then it may be possible to show that each key has an "inverse" key and such an inverse may be easily computable.

Small order: The cryptographic functions may be such that iterating them may cause the resulting permutation to be one which is easily recognised.

Partial Reverse-engineering: Is it possible to build simple black-boxes (ones that are "crackable") that give output that has considerable overlap with the given black-box?