The one missing piece for private-key encryption is a sampler of pseudorandom bits. We are now in a position to describe some common algorithms for this task.

1 Stream ciphers

Conceptually, we made a distinction between pseudorandom generators that produce a random-looking stream; and pseudorandom functions that allow chosen access, but only output some fixed number of pseudorandom bits. A *stream cipher* S fuses the two notions. It produces a pseudorandom stream, but may also take in an optional input called an initialization vector (IV for short). The IV plays the same role in a cipher as an input does in a PRF. Pseuodorandom streams initialized by the same key but different IVs should appear mutually independent.

As we explained in Lecture 2, a cipher that uses ℓ bits of key can always be broken by brute-force search over all 2^ℓ possible keys. The primary objective in cipher design is to force Eve to put in as close as possible to 2^ℓ effort. A secondary objective is the efficiency of the cipher itself. Some ciphers are optimized for specific hardware architectures.

A stream cipher consists of two deterministic algorithms: Initialization (**Init**) and update (**Next**) with the following functionalities:

- **Init**: Given the key, and possibly an input (IV), compute the initial (private) state.
- **Next**: Given a current state, compute the next item in the output stream and update the state.

Linear Feedback Shift Registers

A Linear Feedback Shift Register (LFSR) is a keyed bit stream that is not pseudorandom. It is a useful building block for certain stream ciphers. Here is a 5-bit LFSR:

State: A 5-bit register $s = s_4 s_3 s_2 s_1 s_0$.
Init: Set s to equal the 5-bit key k.
Next: Set *next* to $s_2 + s_0 \bmod 2$.
Output the rightmost bit of s.
Right shift all other bits of s.
Set the leftmost bit of s to *next*.

Figure 1 shows the sequence produced by this LFSR on key `11000`, read from right to left.

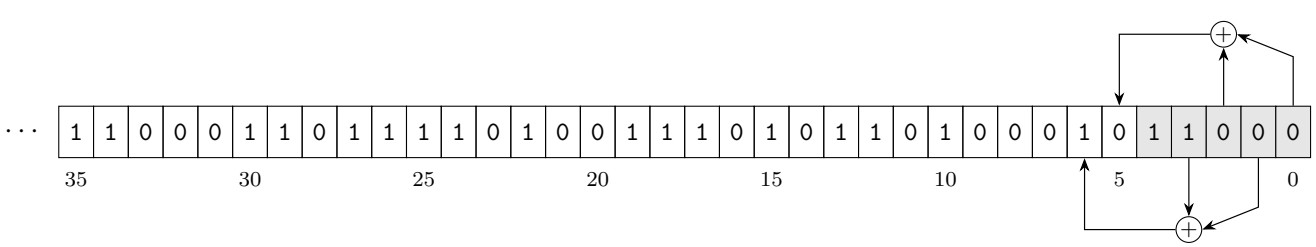

FIGURE 1: The 5-bit LFSR initialized with key `11000`.

Every bit in the output stream is unbiased. The reason is that it is a linear combination of the previous bits, which are themselves a sum (modulo 2 as always) of some key bits. (It turns out that at least one key bit must participate in the sum.) It is, however, not pseudorandom. The stream entries at positions

t, $t+2$, and $t+5$ always add up to zero. It fails to be pseudorandom for an even sillier reason: The key is exposed in the first five bits of the stream. Eve can easily recover the rest.

Nevertheless, this LFSR has one attractive property. It has a fairly long *period*. At some point the LFSR state is bound to repeat. The sequence then becomes periodic. In this example, the first repetition occurs at positions 31 to 35.

An ℓ-bit LFSR works in the same manner. The *next* bit is set to equal a sum of some subset S of the current state bits. (In our example $S = \{0, 2\}$.) It turns out that for every ℓ there are choices of S that result in a period of $2^\ell - 1$, as long as the key is nonzero. This is as large as the period can be: The LFSR cycles through all possible $2^\ell - 1$ nonzero states before revisiting the key.

The Trivium cipher

Trivium is built from three interacting Feedback Shift Registers. See Figure 2.

State:	A 92-bit register s,	a 83-bit register t,	and a 110-bit register u.
Init:	$s \leftarrow k0^{12}$,	$t \leftarrow x0^3$,	$u \leftarrow 0^{107}1^3$.
Next:	$next_s \leftarrow s_{24} + u_0 + u_{45} + u_1u_2$,	$next_t \leftarrow t_6 + s_0 + s_{27} + s_1s_2$,	$next_u \leftarrow u_{24} + t_0 + t_{15} + t_1t_2$,
	output $s_0 + s_{27} + t_0 + t_{15} + u_0 + u_{45}$		
	Right shift s,	right shift t,	right shift u.
	$s_{91} \leftarrow next_s$,	$t_{82} \leftarrow next_t$,	$u_{109} \leftarrow next_u$.

FIGURE 2: The Trivium cipher. The key k and initialization vector x are each 80-bit long. The register bits are ordered right-to-left, e.g., $s = s_{91} \cdots s_0$. The first 1152 output bits are discarded.

Two design choices here are crucial for the security of Trivium. First, the output combines the state of all three registers. This makes it challenging to reconstruct the state from the output as in LFSRs. Second, the update rule contains the nonlinear terms s_1s_2, t_1t_2, and u_1u_2. If these terms were removed, the output bits would be linear combinations of the key bits. The key would then be recoverable from the output using modular linear algebra.

2 Pseudorandom permutations

The encryption schemes we have seen so far are all based on the same idea. Alice masks the message with a pseudorandom shift. To decrypt Bob undoes the shift.

There is another paradigm for building encryption. Alice scrambles the message by applying some transformation P to it. The transformation appears random to Eve. To decrypt, Bob inverts the transformation. See Figure 3.

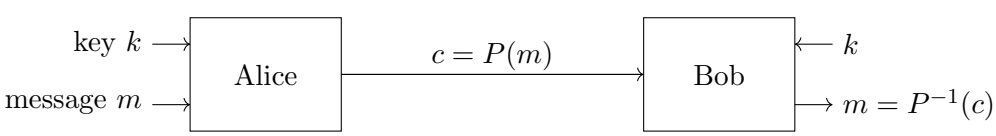

FIGURE 3: Private-key encryption with a pseudorandom generator G.

Functions are in general not invertible. Some outputs might arise from different inputs. Others might not occur at all. A function in which every output is taken exactly once is called a bijection. In a bijection, the domain and the range must have the same cardinality. By now you know that it is convenient to represent domains and ranges by bit strings of some fixed length, e.g., $n = 128$. A *permutation* is a bijection whose domain and range are the same set.

An example permutation on 3-bit strings ($n = 3$) is shown in Table 1(a). The inverse of p is the unique permutation that takes $y = p(x)$ back to x (Table 1(b)).

TABLE 1: (a) A permutation p of 3-bit strings. (b) The inverse permutation p^{-1}.

x	000	001	010	011	100	101	110	111
$p(x)$	101	011	100	111	110	001	000	010

y	000	001	010	011	100	101	110	111
$p^{-1}(y)$	110	101	111	001	010	000	100	011

A *random permutation* R is one chosen equally likely among all possible ones. How many permutations on n bit strings are there? A permutation is just a function. It is described by listing its values in order. The values at different inputs, however, must be distinct. When $n = 3$, $p(\texttt{000})$ can be chosen in 8 possible ways, $p(\texttt{001})$ in 7 remaining ways, and so on. The last value $p(\texttt{111})$ is determined—the only missing item from the list. There are $8 \cdot 7 \cdots 1 = 8!$ permutations of 3-bit strings. In general, there are $2^n!$ permutations on n-bit strings.

A random permutation can be chosen in precisely this manner: First sample $R(\texttt{000})$. Then sample a candidate value for $R(\texttt{001})$. If it collides, repeat. Keep sampling each output until it is distinct from all previous ones.

A pseudorandom permutation (PRP) P is a keyed permutation that is indistinguishable from a random permutation to Eve. As for a pseudorandom function, Eve can ask to see values of P or R at inputs of her choosing. She will never observe the same value as long as she issues distinct queries. With knowledge of the key, Alice and Bob can both compute P and invert it, i.e., compute P^{-1}.

The scheme in Figure 3 is not secure for multiple messages. Once the key is fixed, encryption is deterministic. Here is a randomized variant that is CPA secure:

The Pseudorandom Permutation Pad (PRP Pad):

Shared key: The key k used to set up PRP P.

Encryption: To encrypt a r-bit message m, Alice samples a random pad x. She sends $(x, P(m + x))$ as the ciphertext.

Decryption: Upon receiving ciphertext pair (x, c), Bob recovers $P^{-1}(c) + x$ as the message.

In the PRF pad, the message is masked by the output of the PRF evaluated on the random pointer x. In the PRP pad, x is a random pad that masks the message.

To understand the security of the PRP pad, suppose Eve gets to observe the encryptions of some two messages, e.g. $m = \texttt{010}$ and $m^* = \texttt{100}$. These might look like

$$(x, P(m + x)) = (\texttt{011}, P(\texttt{010} + \texttt{011})) = (\texttt{011}, P(\texttt{001}))$$
$$(x^*, P(m^* + x^*)) = (\texttt{001}, P(\texttt{100} + \texttt{001})) = (\texttt{001}, P(\texttt{101})).$$

Eve is looking at the two random pads, together with two evaluations of P at two inputs that depend on the pads in some known way. As long as the two inputs are distinct, as in this example, the observed values $P(\texttt{001})$ and $P(\texttt{101})$ are indistinguishable from two distinct random values. All she appears to be seeing are random pads, followed by distinct random values.

If, by chance, the two inputs to P happened to collide, Eve would have been able to deduce something of interest about the messages m and m^* (specifically, that $m + m^*$ equals $x + x^*$). For PRPs whose inputs are hundreds of bits long, such collisions are extremely unlikely to happen.

3 Modes of encryption

One technical drawback of using PRFs and PRPs for encryption is that the length of their outputs is a priori bounded. A message m to be sent might be many times longer. A natural strategy is to divide m into

blocks $m_1, m_2, \ldots$ and encrypt each block separately. The resulting encryptions will look like independent random strings to Eve. This strategy is secure. There is, however, a price to pay in efficiency. Every time a block is encrypted, Alice has to send an associated random pointer or pad.

The Electronic Counter Block (ECB) mode of encryption foregoes the pads altogether. $m = (m_1, m_2, \ldots)$ encrypts as $c = (P(m_1), P(m_2), \ldots)$. The scheme in Figure 3 is applied separately to each message block. As expected, the outcome is not secure. Figure 4 is a palpable illustration of this insecurity.

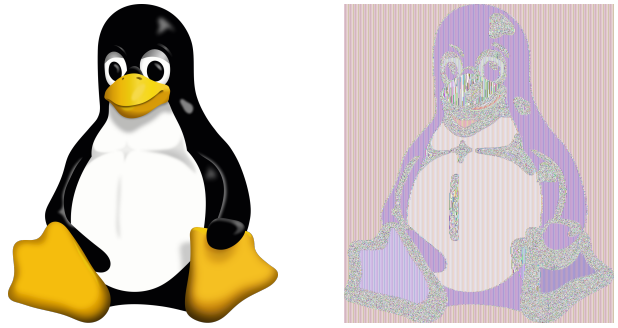

FIGURE 4: An image and its encryption under ECB mode. Monochromatic blocks in the plaintext image produce identical ciphertext blocks. (From *Wikipedia*.)

Counter (CTR) mode is a varation on the PRF pad. Alice chooses a single random pointer x for the whole message m. She then encrypts m as the sequence

$$x, F(x) + m_1, F(x + \underline{1}) + m_2, F(x + \underline{2}) + m_3, \ldots$$

Here $\underline{c}$ stands for the binary representation of c. CTR mode is CPA-secure: Unless some two inputs to F collide in encryptions of multiple messages, what Eve observes is indistinguishable from independent random bits. One advantage of CTR mode is that it is parallelizable. The blocks can be encrypted independently.

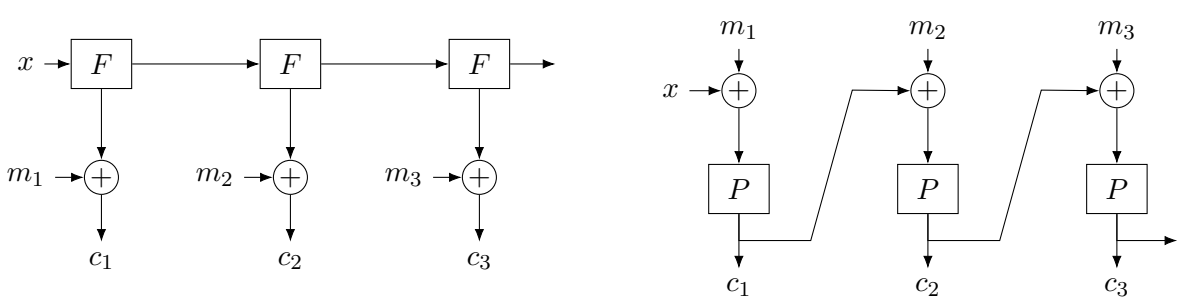

FIGURE 5: (a) OFB and (b) CBC modes of encryption.

Output Feedback (OFB) mode (Figure 5(a)) is an alternative to CTR mode. It is built from a PRF F whose input and output lengths are identical. The encryption of m is the sequence

$$x, F(x) + m_1, F(F(x)) + m_2, F(F(F(x))) + m_3, \ldots$$

It is also CPA-secure. The reason is that nested evaluations of the PRF F on a random input are unlikely to collide.

Chained Block Counter (CBC) mode (Figure 5(b)) is the analogous extension of the PRP pad. The encryption of m is the sequence

$$x, P(x + m_1), P(P(x + m_1) + m_2), P(P(P(x + m_1) + m_2) + m_3), \ldots$$

where P is now a PRP. CBC mode is also CPA-secure. While encryption is not parallelizable, decryption is. Ciphertext $(x, c_1, c_2, c_3, \ldots)$ decrypts to

$$P^{-1}(c_1) + x, P^{-1}(c_2) + c_1, P^{-1}(c_3) + c_2, \ldots$$

4 Block ciphers

Implementations of pseudorandom permutations are called block ciphers. One approach for building block ciphers is to interleave two simple steps:

- **Confusion (substitution).** The state is divided into small blocks. A fixed mapping called an S-box is then applied to each block.
- **Diffusion (permutation).** The bits are dispersed among different blocks.

The substitution cipher from Lecture 1 is an example of a confusion step. One reason it was easy to crack is that confusion is localized. Each ciphertext symbol appears random, but there are noticeable dependencies among them. The diffusion step aims to break these dependencies.

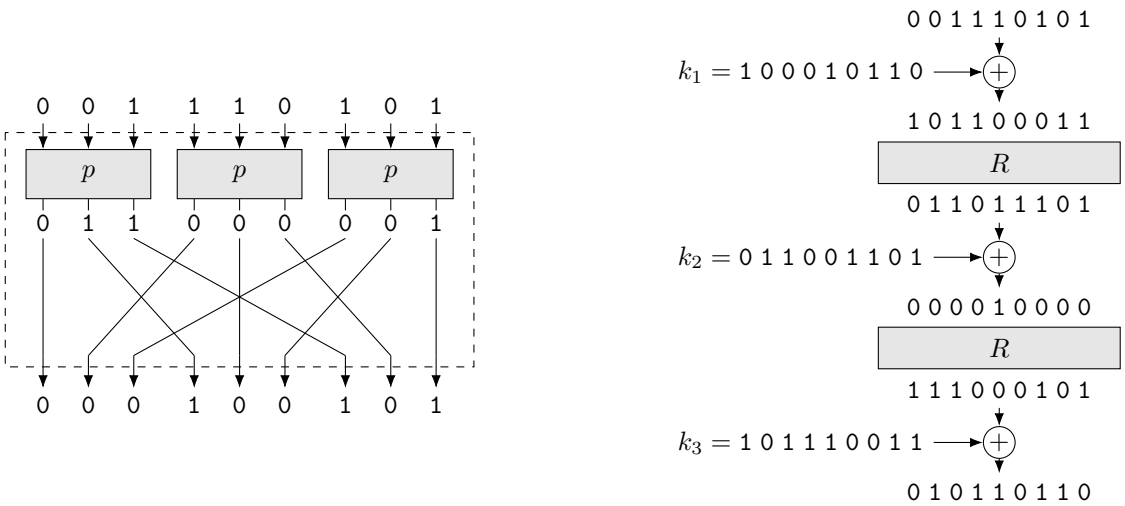

FIGURE 6: (a) A round R of a substitution-permutation network. (b) Two rounds interleaved with key mixing.

Figure 6(a) shows a round consisting of a substitution step followed by a permutation step. The S-boxes are instantiated with permutation p in Table 1(a).

Substitution-permutation networks

In a substitution-permutation network (SPN), multiple rounds are sandwiched between *key mixing* steps. Each key mixing step adds the current state to a portion of the key. When the S-boxes are invertible, the SPN can be inverted as a whole. In the reverse direction, each permutation is replaced by its inverse.

It is reasonable to expect that as the number of rounds increases, the SPN becomes more pseudorandom. The cipher however becomes less efficient. Not only does the computation grow linearly in the number of rounds, but so does the length of the key. To save on key length, some constructions introduce dependencies among the round keys k_1, k_2, k_3. The round keys are derived as sub-keys of the "master" key to the PRP.

The Digital Encryption Standard (DES)

DES is a block cipher that takes in an $\ell = 56$-bit key and permutes $n = 64$-bit strings. It is viewed as a marvel of engineering by applied cryptographers. No practical attacks are known that avoid exhaustive search over all 2^{56} possible keys.

The building block of DES is a substitution-permutation round R on 32 bits with a 48-bit subkey. The 32-bit input is stretched into 48 bits, mixed with the key, and then divided into eight blocks on 6 bits.

Each block is passed through an S-box that produces 4 output bits. The $4 \cdot 8 = 32$ output bits are then diffused.

The DES S-boxes turn 6-bit inputs into 4-bit outputs. They are not invertible: Every output arises from four different inputs. The DES round R is in fact not a permutation.

The *Feistel transform* turns a possibly non-invertible length-preserving function like R into a permutation P. P operates on twice as many bits as R—64 in this case. The input x is divided into a left half x_0 and a right half x_1. The Feistel permutation applies the PRP pad to x_0 using x_1 as the mask, and then swaps the entries:

$$P(x_0, x_1) = (x_1, x_0 + R(x_1)).$$

To invert P, the steps are carried out in reverse:

$$P^{-1}(y_0, y_1) = (y_1 + R(y_0), y_0).$$

A Feistel round by itself is not pseudorandom, even if R happens to be a truly random function. Unlike a random function, P copies the right half of its input. It is known that at least three Feistel rounds are necessary for P to have a chance of being pseudorandom.

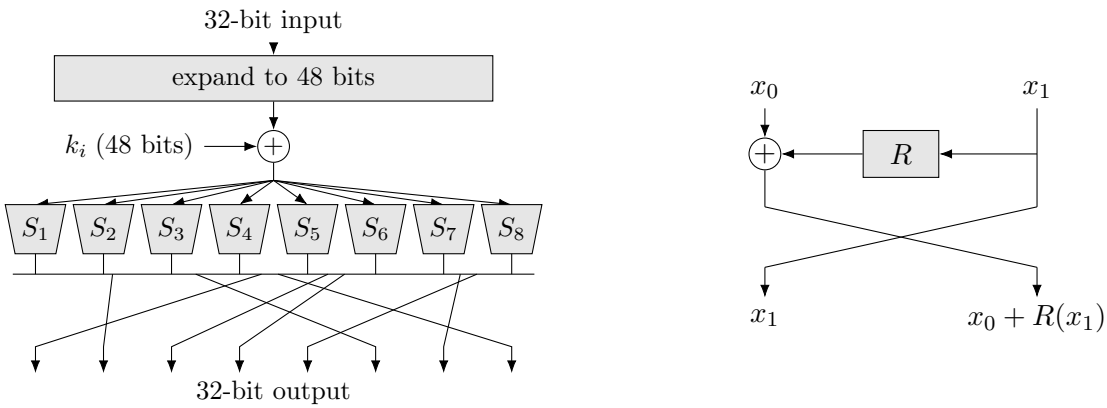

FIGURE 7: (a) A schematic of the DES round function R. (b) The Feistel transform.

DES consists of sixteen Feistel transforms applied to the same substitution-permutation round R. The sixteen 48-bit round keys are all derived from the same 56-bit master key.

The main drawback of DES is its short key length. When it was introduced in the 1970s, exhaustive search over all 2^{56} keys was infeasible. By the late 1990s it could be carried out in less than a day.

The Advanced Encryption Standard (AES)

AES is a substitution-permutation network designed to operate on 128-bit inputs. AES supports keys of length 128, 192, and 256 bits. We describe AES-128, which uses a 128-bit key and consists of 10 rounds. The 128-bit input is viewed as sixteen bytes $x_0, \ldots, x_{15}$. AES arranges these bytes, in order, down the columns of a 4×4 table called the *state* (see Figure 8, left).

Each round of AES consists of the following operations:

SubBytes: confusion. AES has an S-box that permutes 8-bit strings. This S-box is applied to each state byte x_i separately. The permutation is nonlinear. This is the main source of confusion in AES. Apart from the S-box, the transformations applied to the state are linear.

ShiftRows and MixColumns: diffusion. These operations disperse these local changes throughout the state. ShiftRows cyclically shifts the bytes in each row by different offsets (0, 1, 2, and 3). Bytes that began in the same column are moved into different columns. MixColumns then replaces every column by a mixture of all four bytes in that column. The actual calculation is a linear transformation on the 32 bits in

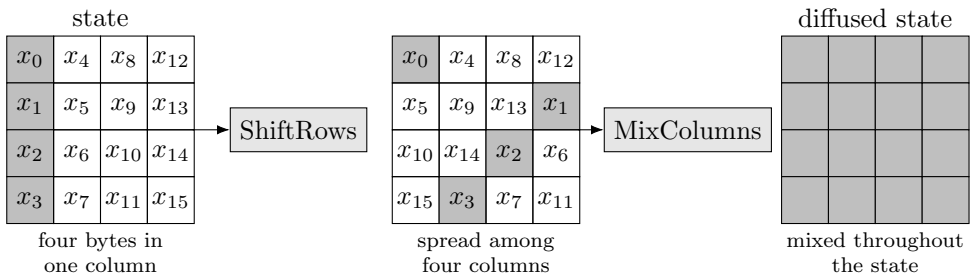

FIGURE 8: Diffusion in AES.

the column. It is chosen so that every output byte is affected by all four input bytes. The two operations complement each other. MixColumns spreads a change among the four bytes of one column. ShiftRows moves those bytes into different columns. In the following round, MixColumns spreads them again. After only a small number of rounds, changing one input byte affects all sixteen state components.

AddRoundKey. The 128-bit state is added to a 128-bit round key. As in the substitution-permutation network of Figure 6, the round keys are not independent. They are all derived from the 128-bit master key.

The AES algorithm. Initially, the state is populated by the input and AddRoundKey is applied. Nine rounds of SubBytes, ShiftRows, MixColumns, and AddRoundKey follow. The tenth round is the same except that MixColumns is omitted.

AES is believed to be extremely secure. No practical key-recovery attacks on full AES-128 are known.

ChaCha20

ChaCha20 is a *stream* cipher built from a different kind of pseudorandom function. Like AES, it achieves confusion and diffusion by repeatedly applying simple operations to a state. Unlike AES, it has no S-boxes. Its basic operations are addition modulo 2^{32}, addition modulo 2, and bit rotation. Addition modulo 2^{32} is a *nonlinear* operation modulo 2. The repeated additions, rotations, and mixing of different state words disperse dependencies throughout the state.

The state consists of sixteen 32-bit registers. Four contain fixed constants, eight contain the 256-bit key, one contains a 32-bit block counter, and three contain a 96-bit IV x. ChaCha20 applies 20 rounds to this state. The same key therefore gives rise to a different pseudorandom block for each choice of IV and counter.

There is one complication. Each step is invertible, and so is the whole 20-round transformation. If its final state was revealed, Eve could run the cipher backwards and recover the initial state, including the key. ChaCha20 therefore adds the initial state, word by word modulo 2^{32}, to the state obtained after 20 rounds. This final *feed-forward* step destroys the obvious invertibility. The resulting 512-bit value is used as a pseudorandom stream block, i.e., the output of a PRF at x.

5 Cryptanalysis of ciphers (optional)

The ciphers we saw are built by iterating several rounds of simple transformations. A single round by itself is insecure. A shallow cipher cannot be secure. We might expect that the security should grow dramatically as more iterations are added.

A common way to establish confidence in iterated cipher constructions is by cryptanalysis of variants with a reduced number of rounds. Ciphers with fewer rounds are easier to reason about and analyze for potential structural flaws. There are two systematic approaches to this.

Linear cryptanalysis

All ciphers are built from linear and non-linear components. A purely linear cipher cannot be secure. Some parity of input and output bits will be fixed. In a truly random function, all parities are unbiased.[1] The two can be distinguished.

The idea of linear cryptanalysis is to approximate the cipher by a linear one. If the approximation error is reasonable, some subset parity will not be fixed, but it will be noticeably biased. The offending parity can reliably distinguish the cipher from a random function from random input samples and their corresponding outputs.

The reason we might expect a good linear approximation is that the cipher is built mostly from linear components. In a substitution-permutation network, nonlinearities only come from the S-boxes. If those can be linearized, the whole cipher can be.

We illustrate the approach on the cipher in Figure 6. Our S-box p has some unusual biases:

- The second output equals the sum of the three inputs 6 out of 8 times, namely with probability 3/4:

$$\Pr(y_2 = x_1 + x_2 + x_3) = 3/4.$$

- The sum of the second and third outputs matches the first input 6 out of 8 times:

$$\Pr(y_2 + y_3 = x_1) = 3/4.$$

Now look at the three leftmost inputs to the two-round network in Figure 6(b). Ignore the keys for the moment. After the first round, the sum of these three inputs is correlated with the second output of the substitution, which moves to position 4 after permutation. This is the first input to the second S-box in substitution round 2. The sum of its second and third output are correlated with it. Those are at positions 5 and 6. After the permutation, these two bits move to positions 5 and 8. In summary, the two round network exhibits the approximate relation

$$X_1 + X_2 + X_3 \approx Y_5 + Y_8.$$

X and Y represent the 9-bit input and output of the cipher, respectively. Incorporating the key does not change the picture significantly. The formula updates to

$$X_1 + X_2 + X_3 + Y_5 + Y_8 \approx \text{some bit that depends on } k \text{ only.} \tag{1}$$

Relation (1) reflects a real bias in the cipher. Out of the 512 possible input-output pairs (X, Y), $X_1 + X_2 + X_3 + Y_5 + Y_8$ take one value 320 times and the other value 192 times. If the permutation were random, we would expect each value to occur about 256 times. The deviation is significant.

The idea of linear cryptanalysis is to look for approximate linear dependencies among S-boxes that propagate through the network. More rounds inhibit propagation, thwarting the effectiveness of the attack.

Differential cryptanalysis

A complementary approach is to look at how *differences* propagate through a cipher. Suppose that Eve chooses two distinct inputs X and X'. She asks Alice to evaluate the cipher on both and observes the outputs Y and Y'. If the permutation was random, Y and Y' would be distinct random values. The difference $\Delta Y = Y + Y'$ is equally likely to take any one of the $2^n - 1$ nonzero values. Differential cryptanalysis attempts to find input pairs with unusually large differences. The approach is to analyze how input differences $\Delta X = X + X'$ propagate into output differences $\Delta Y = Y + Y'$, one step at a time.

[1] This is not exactly true for a random permutation, but it is close enough.

As in linear cryptanalysis, the interesting component is the S-box. Suppose that two inputs x and x' of p differ by $\Delta x = \texttt{001}$. As x ranges over the eight possible inputs, $\Delta y = p(x) + p(x + \texttt{001})$ takes the four values $\texttt{010}, \texttt{011}, \texttt{110}, \texttt{111}$ each twice. Thus, for example, Δy equals $\texttt{010}$ with probability 1/4.

We can propagate this difference through the cipher in Figure 6. Start with input difference

$$\Delta X = \texttt{001 000 000}.$$

Only the first S-box is affected. With probability 1/4, its output difference is $\texttt{010}$. The permutation moves the nonzero bit into the first input of the second S-box in the next round, producing the state difference $\texttt{000 100 000}$. The input difference in the second S-box of round 2 is $\texttt{100}$. The corresponding output difference $\texttt{010}$ is taken two out of eight times, namely with probability 1/4. After the second round of S-boxes, the difference pattern $\Delta Y = \texttt{000 010 000}$ is therefore quite a bit more likely in the cipher than it would be in a random permutation. The last permutation round leaves ΔY invariant.

Among all 512 input pairs with difference ΔX, 32 output pairs have difference ΔY. In a random cipher, the expected number would be $512/511 \approx 1$.[2]

The analysis is oblivious to key mixing. If the same round key k is added to two states X and X', their difference remains unchanged:

$$\Delta(X + k) = (X + k) + (X' + k) = X + X' = \Delta X.$$

The idea of differential cryptanalysis is to search for input differences that propagate to particular output differences much more frequently than one would expect from a random permutation. As more rounds are added, diffusion activates more S-boxes. Predictable differential paths become less likely.

6 Summary

Cipher design is one part art and one part science. Many designs work by combining and modifying components that are insecure on their own. The confusion-diffusion paradigm is a somewhat systematic methodology for this.

Ciphers come in two flavors. Stream ciphers implement pseudorandom generators and/or pseudorandom functions. Block ciphers implement pseudorandom permutations (and their inverses). Pseudorandom permutations provide an alternative method for CPA-secure encryption.

Care should be taken when encrypting long messages with a cipher of fixed block length. Adhering to a standard mode of operation guarantees security for long messages.

[2] The denominator is 511 instead of 512 because a difference of zero is not an option for a permutation.