Recap

- Semantic security relates to confidentiality but not to integrity.
- To get to authenticated encryption, but together a secure encryption function and a secure MAC (in the right way: *EtM*).
  - Encrypt plaintext with encryption key and nonce.
  - Compute MAC of that. *Does the MAC need to be seeded with both a key and nonce?*

Block cipher

A **block cipher** takes a fixed-size input (e.g., 128-bit) and a fixed-size key (e.g., 128-bit) and produces an output that is the same size as the input. Another approach to **confidentiality**.

As opposed with to with stream ciphers, for a block cipher, the decryption function is the inverse of the encryption function (i.e., $D = E^{-1}$).

We'll define the block cipher on a **pseudorandom permutation** (PRP), starting with the idea of a random permutation.

	PRF	**PRP**	**PRG**	**Hash**
Input	Any	Fixed-size	Fixed-size	Any
Output	Fixed-size	Fied-size	Any	Fixed-size
Has key?	Yes	Yes	Yes	No
Invertible?	No	With key	No	Depends
Collisions?	Yes (can't find)	No	No	Yes (can't find)

PRF, PRP, and PRG are all related in the sense that *if you give me any one of them, I can use it to build the other two.*

Designing a PRP

The hard property to achieve is that it's a complicated function, but one that's invertible with the key. The easy properties are that it's non-linear ("confusion"), that every input bit affects every output bit ("diffusion"), and that it depends on the key.

There are a few ways to build PRPs.

Feistel structure

Define a simple kind of permutation: take the input and divide it in half, labeling the left half L and the right half R. Define a function f and compute $output = R||[f(R, key) \oplus L]$. In other words, the right half gets shifted to the left half, and the left and right half are combined using f and the XOR operator.

This is useful because we know the inverse: $output^{-1} = [f(L, key) \oplus R]||L$, where L and R are the left and right halves of the output of the encryption step (i.e., just reverse the structure). Also nice because f can be any hairy function.

- **Permutation?** Yes.
- **Pseudorandom?** No. The left half of the output is always equal to the right half of the input, deterministically.

To achieve pseudorandomness, we can chain this process multiple times, so that the values that are untouched in the first iteration are touched in the second iteration and so forth. (To invert, we need to invert the same number of times.)

Theorem 0.1. *If f is a PRF, then a four-round Feistel structure is a PRP.*

In practice, we often use a weaker f, but with more rounds. As an example, we have **Data Encryption Standard** (DES), issued in 1977, which used a 16-round Feistel with 64-bit blocks and a 56-bit key.

(DES has stood up amazingly well over time. When DES came out, there was a lot of concern that it was designed as deliberately weak because the NSA was involved in its creation. For example, the 56-bit key size was quite random and, these days, can roughly be brute-forced. There was also concern about the choice of the f-function because the design team refused to explain why they designed it the way they did (i.e., how they picked their "s-boxes"). We now know that the NSA wanted a 48-bit key, and 56-bit was a compromise. With the s-boxes, the IBM team had figured out a new technique called differential cryptanalysis and their s-boxes were resistant to this new attack. The NSA demanded that the IBM team not tell anyone about differential cryptanalysis because they wanted to spy on others around the world.)

Advanced Encryption Standard (AES)

The new standard for encryption. Has 128-, 192- and 256-bit versions (for both block and key sizes). We're going to talk about the 128-bit version.

AES does not use a Feistel structure, but does operate over ten rounds. Represent 128-bits (16-bytes) using a 4-by-4 square grid.

One round is equivalent to four steps:

1. **Non-linear step**: take each byte in the grid and run it through a particular function f that runs from 8-bits to 8-bits and has a permutation. Implemented with a lookup table.

2. **Shift step**: take the ith row and shift it right by i steps.

3. **Linear-mix step**: Multiply each column by M, again looked up in the standards document.

4. **Key-addition step**: XOR with the key, which is another 4-by-4 grid of bytes. Each round uses a different key, which is computed in a straightforward manner based on the original key.

To invert, take the inverse of each step, and in the reverse order. For example, step (4) is inverted by XORing again, step (3) is inverted by multiplying by M^{-1}, step (2) is inverted by shifting left, and step (1) is inverted using f^{-1}.

Arbitrarily-sized messages

The messages we want to encrypt won't always be 128-bits in length. We need a way to encrypt arbitrarily-sized messages. To do so, we'll need:

1. **Padding**: this is actually quite difficult because the padding operation needs to be invertible (i.e., the decryptor needs to be able to figure out where the padding is). One approach: append with $10*$. (If your block ends in $10*$, you append this anyway.)

2. **Multi-block message**: "cipher modes" are the various ways to handle multi-block messages, most of which have a three-letter acronym.

   - **ECB mode**: start with a multi-block message, p_0, p_1, etc. Encrypt p_0 with our key k to get c_0, encrypt p_1 with the same key to get c_1, etc. However, this approach is **not** semantically secure, the intuition being that if you have the same plaintext block in here twice, the resulting cipher text will be the same. This gives the adversary the ability to defeat the semantic security: adversary gives message (a, a), and then message (a, b), and can identify which output block maps to a and which maps to b.
   - **Cipher Block Chaining (CBC) mode**: use an initialization vector (IV), chosen at random. XOR p_0 with the IV, encrypt the result with the key k to produce c_0. Then XOR p_1 with c_0, etc. The cipher text will be one block larger than the plaintext because of this IV.
   - **Counter (CTR) mode**: generally agree upon as the best scheme to use. Based on the following rule: $c_i = E(k, messageid||counter) \oplus p_i$. Looks a lot like a stream cipher, because it is. Why do we do this? Why do these things even exist? Two answers: (1) historical, in that counter mode being a "good idea" is relatively recent; (2) it might be better to base a stream cipher on a block cipher than on a PRF, because it will likely be more efficient and we're more confident that no one will break ABS or that the implementation will be secure than we are for HMAC-SHA256.

There was a long time in the field before the idea of authenticated encryption was well formalized. The belief was that stream ciphers were more likely to be subject to issues than CBC mode, etc. Today, we have a better idea of how these things work and both are valid options.