**Convert hex to base64**

The string:

49276d206b696c6c696e6720796f757220627261696e206c696b65206120706f69736f6e6f7573206d757368726f6f6d

Should produce:

SSdtIGtpbGxpbmcgeW91ciBicmFpbiBsaWtlIGEgcG9pc29ub3VzIG11c2hyb29t

So go ahead and make that happen. You'll need to use this code for the rest of the exercises.

**Cryptopals Rule**

Always operate on raw bytes, never on encoded strings. Only use hex and base64 for pretty-printing.

### Fixed XOR

Write a function that takes two equal-length buffers and produces their XOR combination.

If your function works properly, then when you feed it the string:

1c0111001f010100061a024b53535009181c

... after hex decoding, and when XOR'd against:

686974207468652062756c6c277320657965

... should produce:

746865206b696420646f6e277420706c6179

### Single-byte XOR cipher

The hex encoded string:

1b37373331363f78151b7f2b783431333d78397828372d363c78373e783a393b3736

... has been XOR'd against a single character. Find the key, decrypt the message.

You can do this by hand. But don't: write code to do it for you.

How? Devise some method for "scoring" a piece of English plaintext. Character frequency is a good metric. Evaluate each output and choose the one with the best score.

### Achievement Unlocked

You now have our permission to make "ETAOIN SHRDLU" jokes on Twitter.

### Detect single-character XOR

One of the 60-character strings in [this file](http://cryptopals.com/static/challenge-data/4.txt) has been encrypted by single-character XOR.

Find it.

(Your code from #3 should help.)

### Implement repeating-key XOR

Here is the opening stanza of an important work of the English language:

Burning 'em, if you ain't quick and nimble

I go crazy when I hear a cymbal

Encrypt it, under the key "ICE", using repeating-key XOR.

In repeating-key XOR, you'll sequentially apply each byte of the key; the first byte of plaintext will be XOR'd against I, the next C, the next E, then I again for the 4th byte, and so on.

It should come out to:

0b3637272a2b2e63622c2e69692a23693a2a3c6324202d623d63343c2a26226324272765272

a282b2f20430a652e2c652a3124333a653e2b2027630c692b20283165286326302e27282f

Encrypt a bunch of stuff using your repeating-key XOR function. Encrypt your mail. Encrypt your password file. Your .sig file. Get a feel for it. I promise, we aren't wasting your time with this.

### Break repeating-key XOR

### It is officially on, now.

This challenge isn't conceptually hard, but it involves actual error-prone coding. The other challenges in this set are there to bring you up to speed. This one is there to **qualify** you. If you can do this one, you're probably just fine up to Set 6.

[There's a file here.](http://cryptopals.com/static/challenge-data/6.txt) It's been base64'd after being encrypted with repeating-key XOR.

Decrypt it.

Here's how:

1. Let KEYSIZE be the guessed length of the key; try values from 2 to (say) 40.
2. Write a function to compute the edit distance/Hamming distance between two strings. *The Hamming distance is just the number of differing bits.* The distance between:

this is a test

and

wokka wokka!!!

is **37.** *Make sure your code agrees before you proceed.*

1. For each KEYSIZE, take the *first* KEYSIZE worth of bytes, and the *second* KEYSIZE worth of bytes, and find the edit distance between them. Normalize this result by dividing by KEYSIZE.
2. The KEYSIZE with the smallest normalized edit distance is probably the key. You could proceed perhaps with the smallest 2-3 KEYSIZE values. Or take 4 KEYSIZE blocks instead of 2 and average the distances.
3. Now that you probably know the KEYSIZE: break the ciphertext into blocks of KEYSIZE length.
4. Now transpose the blocks: make a block that is the first byte of every block, and a block that is the second byte of every block, and so on.
5. Solve each block as if it was single-character XOR. You already have code to do this.
6. For each block, the single-byte XOR key that produces the best looking histogram is the repeating-key XOR key byte for that block. Put them together and you have the key.

This code is going to turn out to be surprisingly useful later on. Breaking repeating-key XOR ("Vigenere") statistically is obviously an academic exercise, a "Crypto 101" thing. But more people "know how" to break it than can *actually break it*, and a similar technique breaks something much more important.

### AES in ECB mode

The Base64-encoded content [in this file](http://cryptopals.com/static/challenge-data/7.txt) has been encrypted via AES-128 in ECB mode under the key

"YELLOW SUBMARINE".

(case-sensitive, without the quotes; exactly 16 characters; I like "YELLOW SUBMARINE" because it's exactly 16 bytes long, and now you do too).

Decrypt it. You know the key, after all.

Easiest way: use OpenSSL::Cipher and give it AES-128-ECB as the cipher.

### Do this with code.

You can obviously decrypt this using the OpenSSL command-line tool, but we're having you get ECB working in code for a reason. You'll need it *a lot* later on, and not just for attacking ECB.

### Detect AES in ECB mode

[In this file](http://cryptopals.com/static/challenge-data/8.txt) are a bunch of hex-encoded ciphertexts.

One of them has been encrypted with ECB.

Detect it.

Remember that the problem with ECB is that it is stateless and deterministic; the same 16 byte plaintext block will always produce the same 16 byte ciphertext.