my characters for examples, as is the standard for encryption lectures.

Alice writes a message, encrypting it with her *private* key. She sends the ciphertext to Bob.

Bob knows what Alice's public key is- after all, it *is* a "public" key. Bob, upon decrypting the ciphertext with Alice's public key, can verify that, since the message decrypts properly with Alice's public key, it MUST have been encrypted with her private key. That bit represents authentication.

Alice, in another example, encrypts her message with Bob's public key. Nobody can read this message without using Bob's private key to decrypt it.

That part represents privacy.

RSA involves the *combination* of the two. Encrypting a message with your own private key *and* the recipient's private key lets the recipient verify that it's you, *and* it also ensures that only the intended recipient will receive the message.

→ What's the point, though? Why do we need to encrypt messages? Who is going to read them, or, even worse- modify them?

Eve, named as she is for her presence as an eavesdropper, is tapped into your Internet connection. She might be the government, or your Internet Service Provider, or some shady guy who plugged a cable into your router. Eve can

READ information sent, and can

CHANGE information sent.

When you send info in plaintext, you make Eve's job really easy.

But when you use RSA, Eve is pretty much screwed.

Now, though, you must learn about (wo)man-in-the-middle attacks.

How will I tell you what my public key is? Using the Internet?

Laughable.

I'd be sending that info in plaintext. After all, if we already had a secure channel of communication, we wouldn't need RSA. If, hypothetically, there was an Eve who could read but not modify the stream, there wouldn't be anything to worry about- after all, you ARE sending a "public" key. The problem, though, is that Eve generally has the ability to change information.

This is the scenario I'd like you to imagine.

Bob, a naïve message-sender, sends his public key to Alice in plaintext. Along the way, that message gets replaced with Eve's public key. Whenever Alice sends Bob a message, Eve intercepts it. She can read it, because it was encrypted using her public key (unbeknownst to Alice). Eve, who knows Bob's public key, resends the information to Bob seamlessly. Using an automated process, the additional latency is not noticeable.

Eve also now has the ability to send messages as Bob! If she uses her private key to encrypt messages, Alice will think the messages came from Bob.

Man-in-the-middle attacks highlight the primary flaw of almost any sort of encryption: The two people who are communicating *MUST* use a secure channel at least once.

Still, how can you trust anyone you meet at all? If you want to speak to someone in the first place, you must have some sort of trust that they are who they say they are. If you want to speak with someone you've met in real life, you verify their identity by talking to them in person.

So, there's your solution. Exchange public keys in person. It's as simple as that.