Public Key cryptography and RSA

Reading

- Section 9.5

Practice Problems

9.5 1-9

9.5 (Challenge, Optional) Implement these algorithms in a programming language, and do exercises 10-12.

Notes

Public Key Cryptography

The cryptographic techniques we have seen so far are *symmetric*: The same key is used for both encryption and decryption.

This presents a problem: When two parties want to communicate with each other, they need to both know that key. But if someone is listening on the line, how will they communicate that key to each other at the beginning?

> Symmetric Key Encryption relies on the two parties having already established a shared secret key. This has to happen with some other means.

So for example if you want to talk to your bank over the internet, how could you and your bank share this secret key, when it is possible that someone is watching everything you send to each other over the internet?

Before the advent of Public Key Cryptography the answer was that you simply could not do it. You could perhaps have arranged for the bank to physically mail you some secret number via the postal service, or you could go by the bank and pick it up. Then you could use that to shared secret key to talk to each other over the internet.

But there is another way, Public Key Cryptography:

> In Public Key Cryptography there are two keys:
>
> **public key** This key is used to encrypt messages. You share this with the "world", so *everyone knows how to encrypt messages* and send them to you.
>
> **private key** This key is used to decrypt messages. You keep this key private, noone else knows about it. So *only you can decrypt messages*.

This is therefore an *asymmetric* encryption scheme.

So encryption is done via the public key, and decryption via the private key. Since encryption is always an inverse process to decryption, this requires that we have some "function" whose inverse cannot be efficiently computed.

Asymmetric encryption schemes tend to be more expensive. They are typically used at the beginning of a communication, they allow the two parties to use a public medium to share a secret key. Once they have that shared secret, they can use a faster symmetric encryption scheme.

So here is a typical way this might go (remember our two parties are called Alice and Bob):

Basic Public Key Cryptography Handshake

- Alice knows Bob's public key. She devises a new, unique, "secret key", then encodes it using Bob's key.
- Alice sends the encrypted secret key to Bob.
- Bob uses his private key to decrypt Alice's message. He now also knows that "secret key".
- Alice and Bob now both share a secret key, and can use that in future communication using a symmetric scheme.
- Anyone listening in does not know what this secret key is. They only saw its encrypted form, and without Bob's private key they cannot decrypt that message.

This is a perfectly good scheme, but it leaves something out: It would be nice if there was a way for Bob to know that he's really talking to Alice. He knows Alice's public key, maybe this can be used somehow? Indeed it can!

The key here is that decryption and encryption are trully inverse processes to each other: If D, E are the decryption and encryption algorithms respectively, then we have:

$$E(D(x)) = x$$
$$D(E(x)) = x$$

In other words, you can either first use D and then E, or first use E and then D, and in both cases you get back the original message.

Using this we can do what is known as *digital signing*

Digital Signature using Public Keys

- Alice wants to send a message x to Bob. She needs to encrypt that message, but also somehow "sign it" to identify it as comming from her.
- Alice has a public key, which is used in algorithm E_A, and a private key, used in algorithm D_A.

- Bob has a public key, which is used in algorithm E_B, and a private key, used in algorithm D_B.
- Alice uses her private key on the message, resulting in $D_A(x)$.
- Alice then uses Bob's public key to encrypt the message, resulting in $E_B(D_A(x))$.
- Alice sends this over to Bob.
- Bob can use his private key to decrypt this message, resulting in $D_A(x)$ (as $D_B(E_B(D_A(x))) = D_A(x)$).
- Bob then uses Alice's public key to compute $E_A(D_A(x)) = x$ and obtains the message x.
- Anyone listening in cannot break the message because they don't know Bob's private key.
- Bob knows the message came from Alice, because noone could have produced $D_A(x)$ without knowing Alice's private key.

To sum up: Alice performs D_A followed by E_B, Bob performs D_B followed by E_A.

The RSA algorithm

The RSA algorithm is an Public Key Encryption scheme that uses exponentiation. It is named after its three creators, Ron Rivest, Adi Shamir and Leonard Adleman. It is widely used in internet communications.

RSA system

- Choose two large prime numbers p, and q. Compute $N = pq$.
- The number $\phi(N) = (p-1)(q-1)$ is only known to those that know p and q.
- Choose encryption exponent e with $gcd(e, \phi(N)) = 1$.
- Compute decryption exponent d such that $de = 1 \bmod \phi(N)$.
- Share the public key: (N, e)
- Keep the private key: (N, d)
- To encrypt a message x: Compute $E(x) = x^e \bmod N$.
- To decrypt an encrypted message c: Compute $D(c) = c^d \bmod N$.
- d can only be computed if you know $\phi(N)$.
- $\phi(N)$ can only be known if you know how to factor N into p and q.
- This factorization cannot be done efficiently without prior knowledge of p, q, if the those numbers are large enough.

You might wanted