Definition (Symmetric-Key Cryptography)**.** *Using the same secret key to encrypt and decrypt information.*

Definition (Asymmetric (or Public-Key) Cryptography)**.** *Use different keys to encrypt and decrypt, or to sign and verify.*

Public-key cryptography is great because it gives one individual the ability to encrypt but not decrypt, and vice versa. Usually, you have a secret ("private") key and a "public" key.

RSA

The most common public-key scheme is RSA (Rivest, Shamir, Adleman 1978). Keys are always generated in pairs as follows:

- Pick two large (2048 bit) random primes p and q. (Never reuse p or q.)
- Define $N = pq$. *(Useful fact: if p and q are prime and $0 < x < pq$, then $x^{(p-1)(q-1)} \bmod pq = 1$.)*
- Pick e to be any value you like, as long as it's relatively prime to $(p-1)(q-1)$.
- Find d such that $ed \mod (p-1)(q-1) = 1$. This ensures that ed is some multiple of $(p-1)(q-1)$ plus 1. *(Easy computation using a modified version of Euclid's Algorithm.)*
- Define the public key to be (e, N).
- Define the private key to be (d, N). Might also keep p, q.
- To encrypt with a public key: $RSA((e, N), x) = x^e \mod N$.
- To encrypt with a private key: $RSA((d, N), x) = x^d \mod N$. *(For RSA, encryption and decryption are the same function.)*

Theorem 1 ("It Works")**.** $RSA(public, RSA(private, x)) = x$

Proof.

$$\begin{aligned}
RSA((e,N), RSA((d,N),x)) &= (x^d \mod N)^e \mod N \\
&= (x^d)^e \mod N \\
&= x^{de} \mod N \\
&= x^{a(p-1)(q-1)+1} \mod N \text{ for } \textit{some } a \\
&= ((x^{(p-1)(q-1)} \mod N)^a x) \mod N \\
&= 1^a x \mod N \\
&= x
\end{aligned}$$

□

Open question: is it secure? The best known attack: factor N to derive p and q. Main drawback: inefficiency, as RSA is a factor of 1000 slower than symmetric algorithms (in order to encrypt, you need to exponentiate a huge number to a huge power; further, the key is very large (roughly 4000 bits)).

Thus, we will use public-key cryptography **only when we need it**.

Applications

"Your Eyes Only" Message

You can encrypt a message with Alice's public key and reveal it to everyone. But only Alice will be able to decrypt, as she is the only individual with the private key. (This is the public-key version of encrypting for confidentiality.)

Digital Signature

You can encrypt a message with your private key ("signing") and reveal it to everyone. Only you can sign your own messages, as it requires the private key. Anyone can then decrypt ("verifying") to check that you sent the message. (This is the public-key version of a MAC.)

Security Properties

The RSA we've described thus far is **not** semantically secure and **not** a secure digital signature. The former is because Mallory can encrypt 0 and 1 on her own, send 0 or 1 to us, see the cipher text that's produced, and verify her guess. The latter is because:

$$\begin{aligned} sig(x) &= x^d \mod N \\ sig(y) &= y^d \mod N \\ (x^d \mod N)(y^d \mod N) &= x^d y^d \mod N \\ &= (xy)^d \mod N \\ &= \text{signature of } xy \end{aligned}$$

Therefore, Mallory can cook up the signature on a value that she never intended to sign. Simple RSA thus fails the tests we know. In addition, small plain texts are very easy to crack based on the mathematics above.

Augmenting RSA

We can fix all of these problems with a single maneuver.

Definition (Optimal Asymmetric Encryption Padding (OAEP)). *To encrypt a message M, append b_0 zeroes to the end (in practice, can do $b_0 = 128$). Append b_1 random bits to the end. Run b_1 through a PRG G as the seed. Concatenate m and b_0, and XOR the result with the output of G. Feed the result into a cryptographic hash function H. XOR the result with the original random bits. Concatenate that result with the input to H and take the result to the d power mod N.*

To decrypt, take the inverse of each operation. In the end, the output will be $[m'][z'][r']$. It should be the case that z' is all zeroes (doctoring the cipher text will almost definitely leave at least a single one in there). If so, then we can accept m' as the original message and throw away r'.

This process imposes a limit on the size of M, as we need the output of OAEP to be a valid choice of x (i.e., $0 < x < pq$).

Why is this secure? The value we exponentiate will almost definitely be quite large, for one. In addition, the results of RSA are no longer predictable in the same way. With OAEP, we get semantic security, digital signatures, etc.

Message Size

How do we encrypt large messages?

- Use CTR mode. Actually, this doesn't work (due to the randomization).
- Use CBC mode. This works, but it's relatively inefficient.
- Better: use a hybrid scheme. Let E be a secure symmetric authenticated encryption algorithm. Generate two values $(key, nonce)$ randomly. To encrypt a value x, compute: $encrypt(x) = RSA\text{-}OAEP(rsakey, key||nonce)||E(key, nonce, x)$.
- If you want to sign a large document efficiently, there's a separate trick: $sign(x) = RSA\text{-}OAEP(rsakey, H(x))$. In other words, sign the hash instead. If x is collision-free, then our security properties hold.

Details

What's the optimal key size? We typically use 2048-bit. Primes are recommended.

As a useful performance trick, consider picking a small public exponent, like $e = 3$ (another popular choice: $e = 65537$, useful in its bit representation is of the form 100...1 and it's prime). It's not a secret and the fact that it's predictably small is not an issue. In addition, exponentiating to the third power is much more efficient than exponentiating to some random, larger value.

The “Impostor” Problem

There’s a persistent problem in these schemes: you need to know Alice’s public key in order to engage with her, but the whole point is that you want to do these operations without any interaction. In other words, Mallory could show up, claim to be Alice, and present a public key (that she claims belongs to Alice).

One solution is to make trust a transitive property. For example, if you trust Bob, you can have Bob send you something along the lines of: “Alice’s public key is [insert key here], signed Bob.” But this relies on trusting Bob. **Public-key cryptography relies on having an accurate mapping from identity to key.**

The way that this is usually solved is in one of two ways:

- **Certificate authority**: checks credentials, issues certificates (“certs”), everybody knows the CA’s public key. The CA is essentially the end of this transitive trust chain.
- **Web of trust**: A lot of people sign certificates for a lot of other people. If Alice wants to establish her identity to me, I can say “Okay Alice, who do you have certs from?” If I trust one of her signers, then I can trust Alice.