Cryptography: Asymmetric vs Symmetric

the key difference (no pun intended) between asymmetric and symmetric cryptography:

Is the keys used and their purpose. Symmetric uses a private key, which encrypts as well as decrypts.

Asymmetric attempts to solve the risk of having one key by using two keys; a private and public key.

Symmetric Encryption Function:

\[ P = D(K_s, E(K_s, P)) \]

Asymmetric Encryption Functions:

\[ P = D(K_s, E(K_p, P)) \text{ and } P = D(K_p, E(K_s, P)) \]

where:

\[ P = \text{plaintext} \]
\[ K_s = \text{secret key} \]
\[ K_p = \text{public key} \]
\[ D() = \text{decryption} \]
\[ E() = \text{encryption} \]

In general, asymmetric encryption allows two people to send each other messages without risking their private key because only information about the public key can be obtained from the intercepted message.
Key Management Issues:

Verification of identity for owners of a public key can be tough. It's easy to pretend to be someone else, especially when it's not a face to face interaction.

Web of Trust (WoT):

WoT offers some alleviation to the identity issue by entrusting a third party entity to host public keys. Others can then add credibility to your hosted public key by digitally signing it.

How Asymmetric Cryptography Generally Works: (slide 15)

Given \( P = D(Ks,E(Kp,P)) \) and \( P = D(Kp,E(Ks,P)) \), \( E() \) and \( D() \) are mathematical, one-way inverse functions. That is \( E(x) = y \rightarrow D(y) = x \)

Given \( y \), \( x \) is extremely difficult to compute and can take a computer a large amount of time to determine.

As an example, when using a 128-bit AES cipher the number of possible keys is \( 2 \) raised to the 128th power, or \( 3.4 \times 10^{38} \). Brute-forcing becomes out of the question, especially if it's a 256-bit key.

Moore's Law (slide 16):

The law essentially stats that computer's processing speeds double every couple of years (there's an upper limit to this due to the laws of physics). Because of the increase in processing power, older encryptions will fail against modern computers.

Prime Number Generation (slide 17):

Using large prime numbers strengthens the cryptographic equation since prime factorization of large numbers takes a lot of time and processing power.
However, finding large prime numbers can be difficult as well. Proving a 27-digit number isn't factorable requires quite a bit of primality testing.

Fermat's Primality Test (slide 19):

One method of primality testing is:

\[ a^{(P-1)} = 1 \mod P, \text{ where } P = \text{prime number and } 0 < a < P \]

\[ a^{(P-1)} \mod P \neq 1 \rightarrow P \text{ is not prime} \]

However, be weary of Carmichael numbers, which pass Fermat's test but are not actually prime.

Primality Tests (slides 25):

Primality tests come in two varieties:

- Deterministic:
  
  Deterministic tests determine with absolute certainty whether a number is prime or not but are very slow in comparison to probabilistic tests.
  
  Deterministic tests include:
  
  - Lucas-Lehmer Test
  
  - Elliptic Curve Primality Proving

- Probabilistic:

  Probabilistic tests raise the probability of a number in question being prime but do not prove it's primality. The trade off is that they are much quicker than deterministic tests.

Some Asymmetric Implementations (slide 31):

- Diffie-Hellman Key Exchange
- RSA
- ElGamal
PGP

Diffie-Hellman Key Exchange (slide 32):

Diffie-Hellman (D-H) key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure communication network. These two parties can contain any number of participants.

D-H Algorithm (slide 33):

\[ y = g^x \mod P \]

where:

\[ P = \text{publicly agreed prime} \]
\[ g = \text{a primitive root modulo } P \]
\[ x, y = \text{either a public or private key depending on the phase} \]

D-H Weaknesses (slide 41):

- New keys must be generated for every group of users.
- Cannot be used to sign certificates
- Issues when implemented in OpenSSL

RSA (slide 42):

RSA method came shortly after the D-H method and is the second encryption method to implement the use of public key cryptography.

RSA differs from D-H in that it uses asymmetric algorithms instead of the symmetric algorithms found in the D-H method.

One public key can be shared with as many people as needed, so there's no need to manage multiple secret keys.

RSA Algorithm (slide 43):

Encryption: \[ c = m^e \mod n \]

Decryption: \[ m = c^d \mod n \]

where:
n,e = public key

d = private key

m = plaintext

c = ciphertext

Pseudo-randomly generate two distinct prime numbers P and Q with similar bit length but are relatively not too close to each other such that:

|P - Q| < 10^10 or their product is easily factorized by Fermat's method.

n = P * Q

Compute Phi(n)

Choose e such that e and Phi(n) are co-prime and e is the public key exponent.

Co-prime numbers have a gcd of 1 (Ex. 6 and 35).

Find d where d * e = 1 mod Phi(n), d is the private key exponent.

RSA Weaknesses:

- Timing attacks (Listening to the CPU)
- Possibly generating too small of an e compared to the modulus.
- Chinese Remainder Theorem that basically will determine a number n that when divided by some given divisors leaves given remainders.

Extras:

- For crypto exercises go to http://cryptopals.com

- UO crypto classes:
  - CIS 410
  - MATH 458