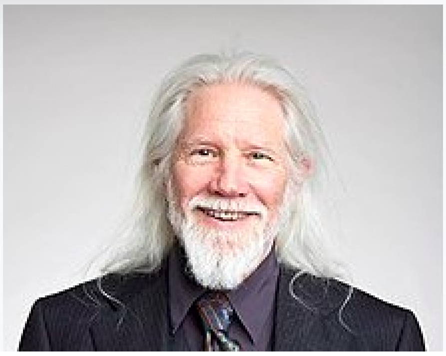
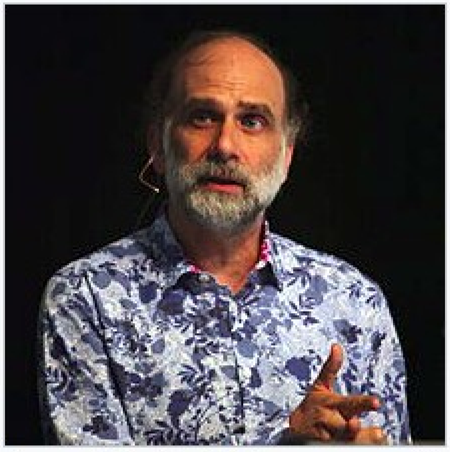
Encryption

Whitfield Diffie



Martin Hellman



Bruce Schneier 2013

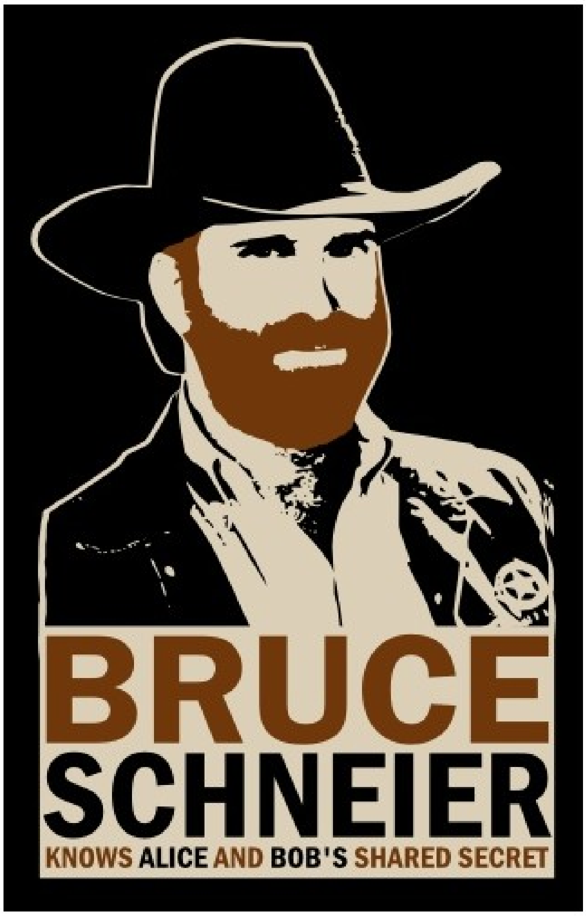
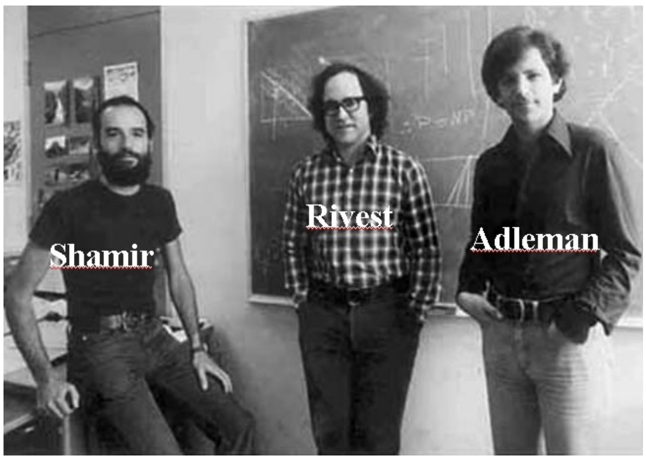


Figure 1: LOL'ed? - you are too deep in!



Symmetric Encryption

Block Ciphers

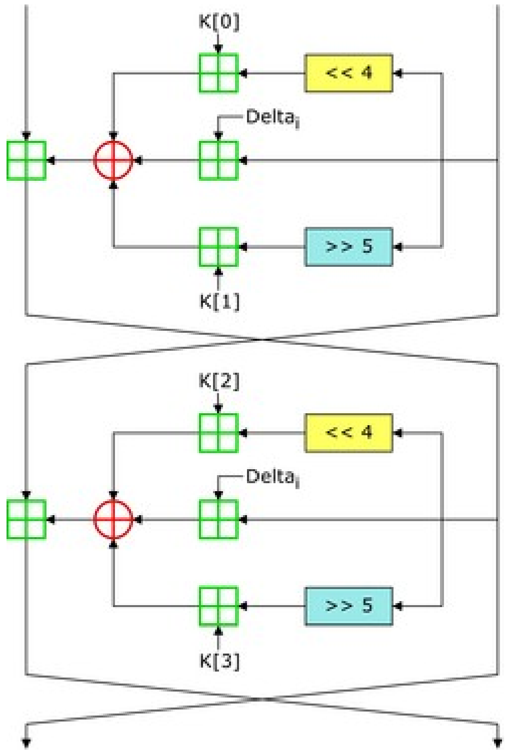


Figure 2: TEA (feistel structure)

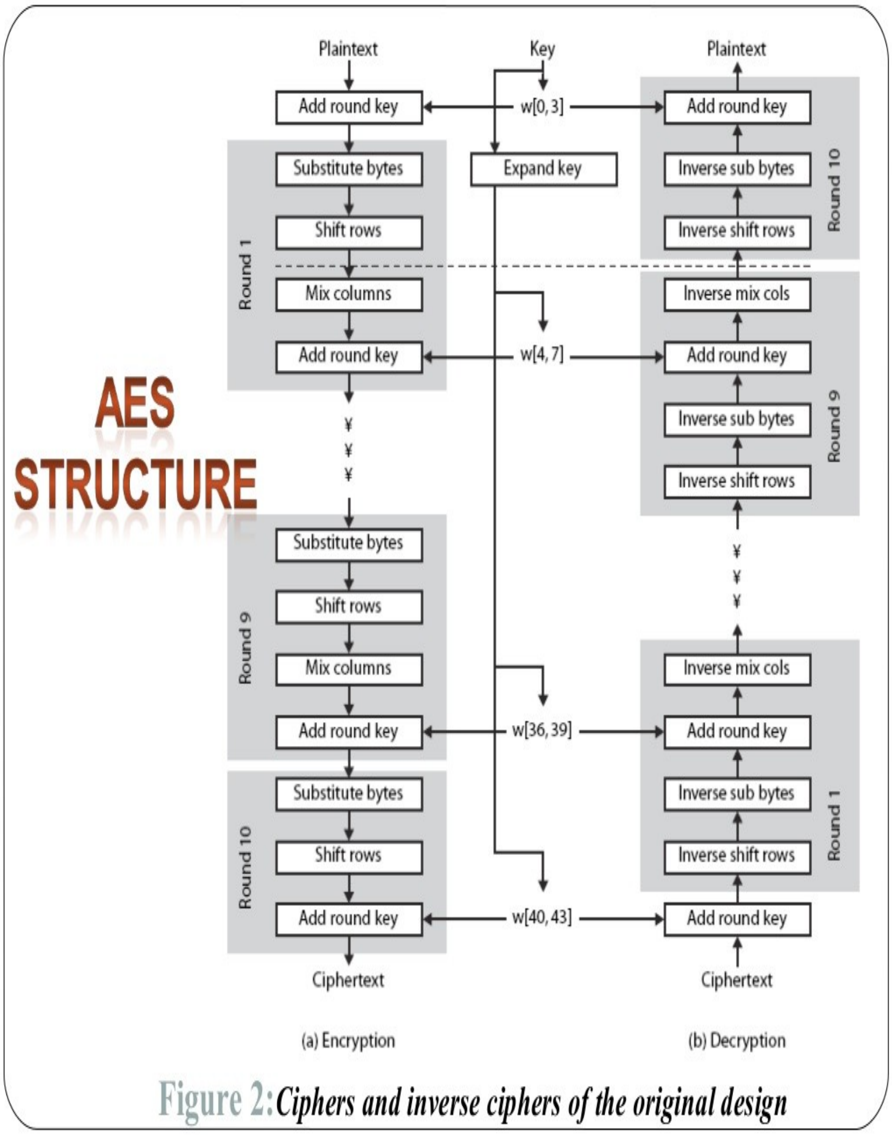
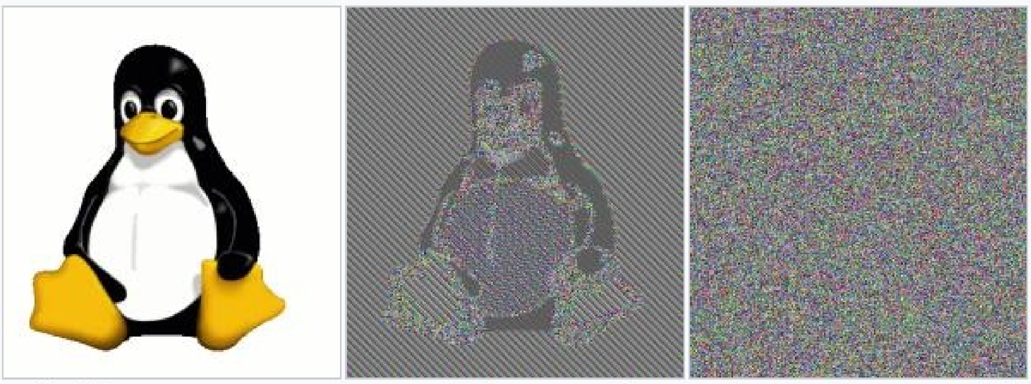


Figure 3: AES (substitution-permutation network)

```c
void encrypt(unsigned long v[2], unsigned long k[4]) {
    unsigned long v0 = v[0], v1 = v[1], sum = 0, i;           /* set up */
    unsigned long delta = 0x9E3779B9;                         /* a key schedule constant */
    unsigned long k0 = k[0], k1 = k[1], k2 = k[2], k3 = k[3]; /* cache key */
    for (i = 0; i<32; i++) {                                  /* basic cycle start */
        sum += delta;
        v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);   /* end cycle */
    }
    v[0] = v0; v[1] = v1;
}

void decrypt(unsigned long v[2], unsigned long k[4]) {
    unsigned long v0 = v[0], v1 = v[1], sum = 0xC6EF3720, i;  /* set up; sum is 32*delta */
    unsigned long delta = 0x9E3779B9;                         /* a key schedule constant */
    unsigned long k0 = k[0], k1 = k[1], k2 = k[2], k3 = k[3]; /* cache key */
    for(i = 0; i<32; i++) {                                   /* basic cycle start */
        v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
        v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        sum -= delta;                                         /* end cycle */
    }
    v[0] = v0; v[1] = v1;
}
```

Figure 4: TEA - encryption and decryption functions

Block Cipher Modes



Original image | Encrypted using ECB mode | Modes other than ECB result in pseudo-randomness

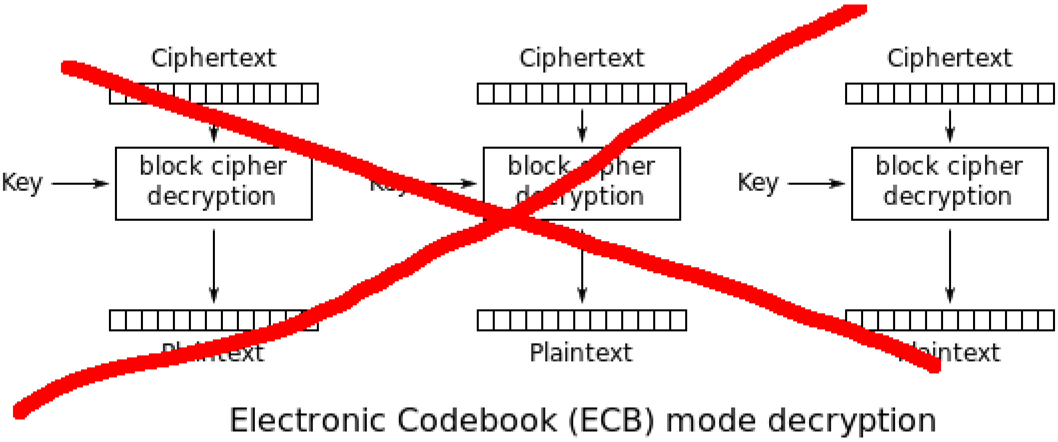


Figure 5: only to scare little children

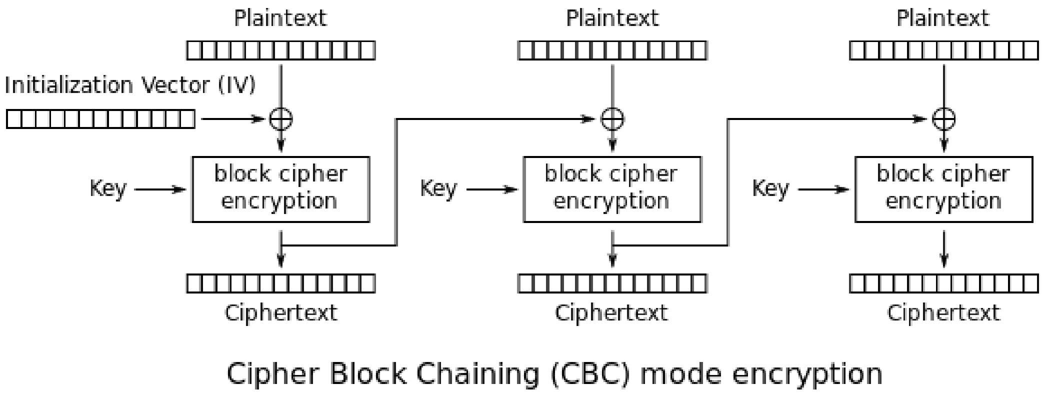


Figure 6: mimimi - I can't run in parallel

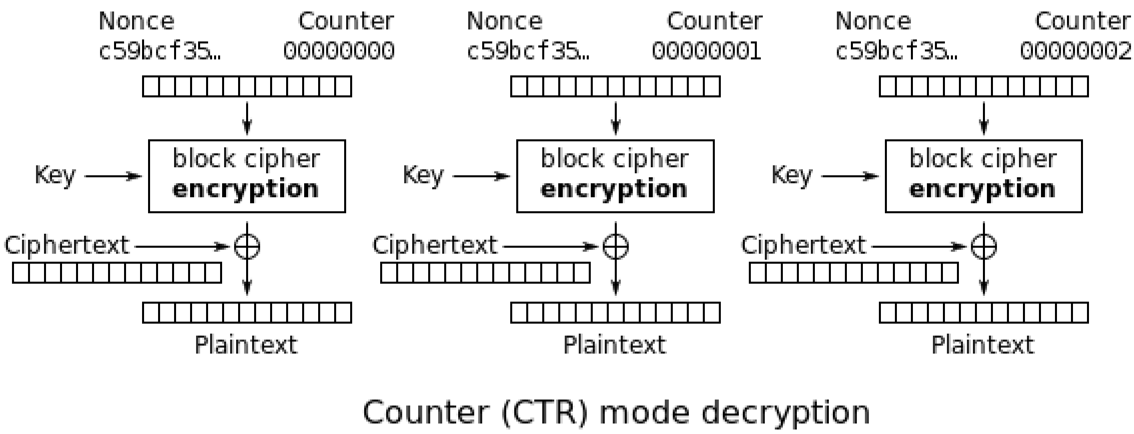


Figure 7: everything will be fine

Stream Ciphers

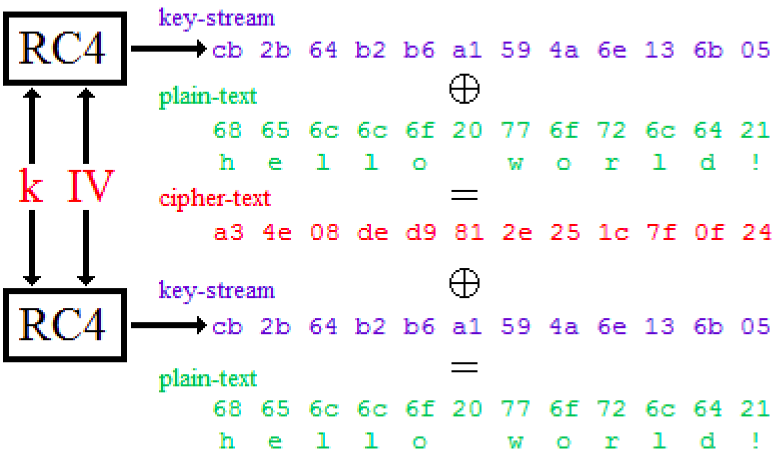


Figure 8: networks sending bytes - sender & receiver generate same key-stream

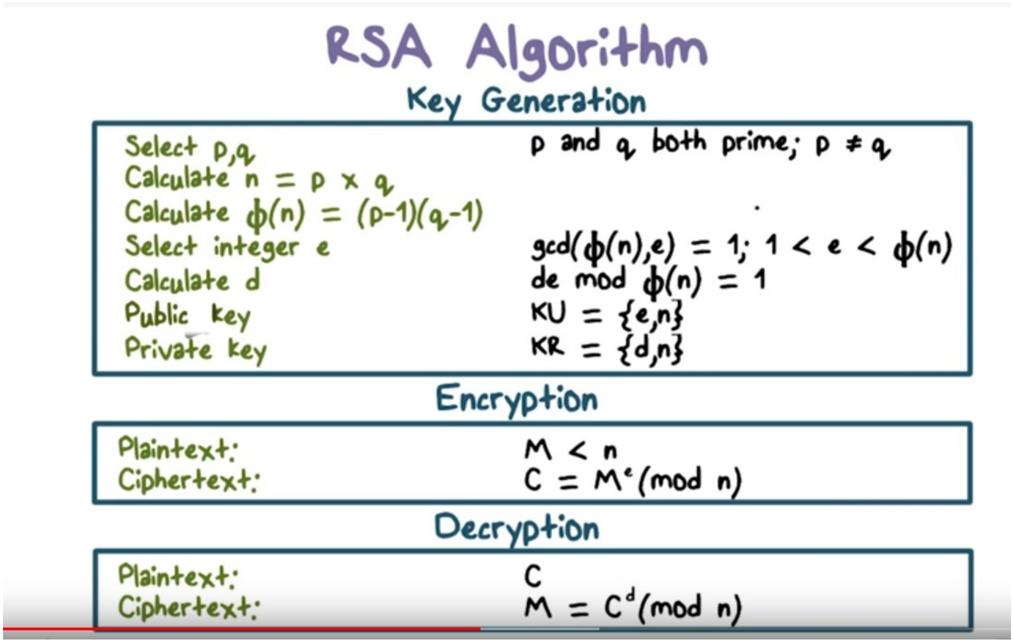
Hashing

```
#include <stddef.h>
#include <stdint.h>

uint32_t crc32(const char *s,size_t n) {
	uint32_t crc=0xFFFFFFFF;

	for(size_t i=0;i<n;i++) {
		char ch=s[i];
		for(size_t j=0;j<8;j++) {
			uint32_t b=(ch^crc)&1;
			crc>>=1;
			if(b) crc=crc^0xEDB88320;
			ch>>=1;
		}
	}

	return ~crc;
}
```

Figure 9: self-explaining: $g(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$.

Asymmetric Ciphers

RSA



Calculation of Modulus And Totient

Lets choose two primes: $p = 11$ and $q = 13$. Hence the modulus is $n = p \times q = 143$. The totient of n $\phi(n) = (p-1) \cdot (q-1) = 120$.

Key Generation

For the public key, a random prime number that has a greatest common divisor (gcd) of 1 with $\phi(n)$ and is less than $\phi(n)$ is chosen. Let's choose 7 (note: both 3 and 5 do not have a gcd of 1 with $\phi(n)$. So $e = 7$, and to determine d, the secret key, we need to find the inverse of 7 with $\phi(n)$. This can be done very easily and quickly with the *Extended Euclidean Algorithm*, and hence $d = 103$. This can be easily verified: $e \cdot d = 1 \bmod \phi(n)$ and $7 \cdot 103 = 721 = 1 \bmod 120$.

Encryption/Decryption

Lets choose our plaintext message, m to be 9:

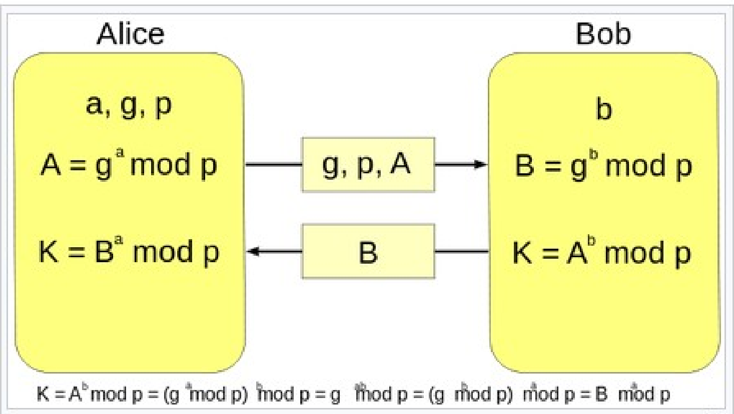
Encryption:

$$m^e \bmod n = 9^7 \bmod 143 = 48 = c$$

Decryption:

$$c^d \bmod n = 48^{103} \bmod 143 = 9 = m$$

Diffie-Hellman



Prinzip des Diffie-Hellman-Merkle-Schlüsselaustauschs
a: privater Schlüssel von Alice
b: privater Schlüssel von Bob
p: öffentlich bekannte Primzahl
g: öffentlich bekannte natürliche Zahl kleiner als p
A: öffentlicher Schlüssel von Alice
B: öffentlicher Schlüssel von Bob
K: geheimer Sitzungs-Schlüssel für Alice und Bob

P=11, G=7,

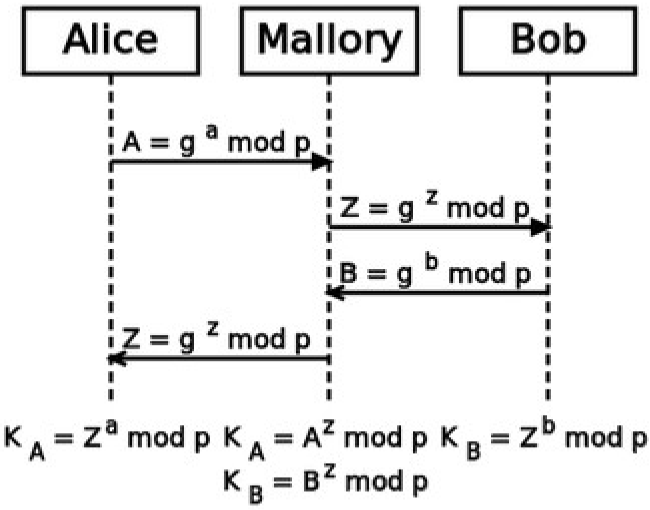


Figure 10: Man-In-The-Middle

Certificate Authority

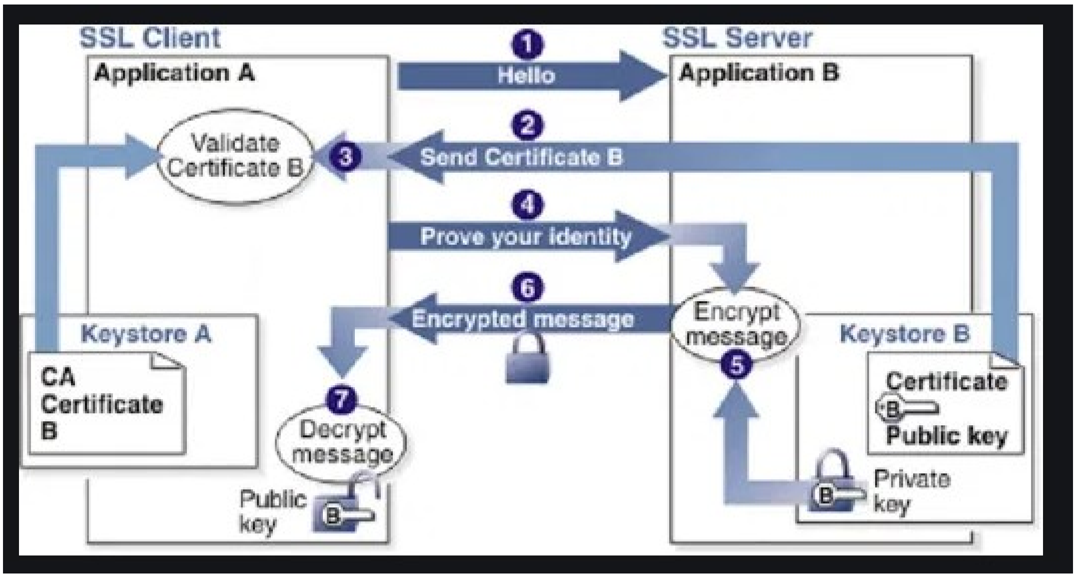
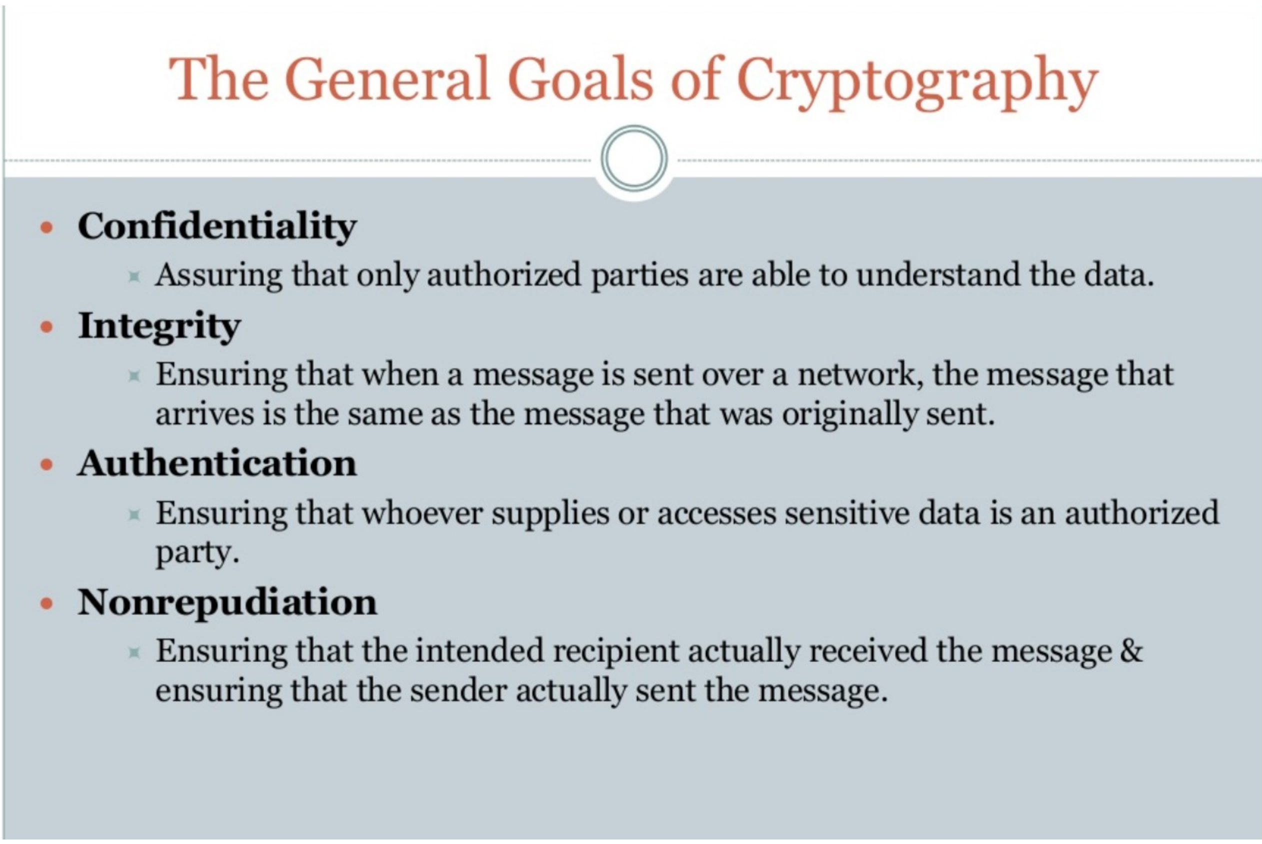


Figure 11: Bob is Bob and Alice is Alice, trust me on this one

Goals of Cryptography



All images are from a web-site named google they were all on page 1 (how-to not cite image sources)