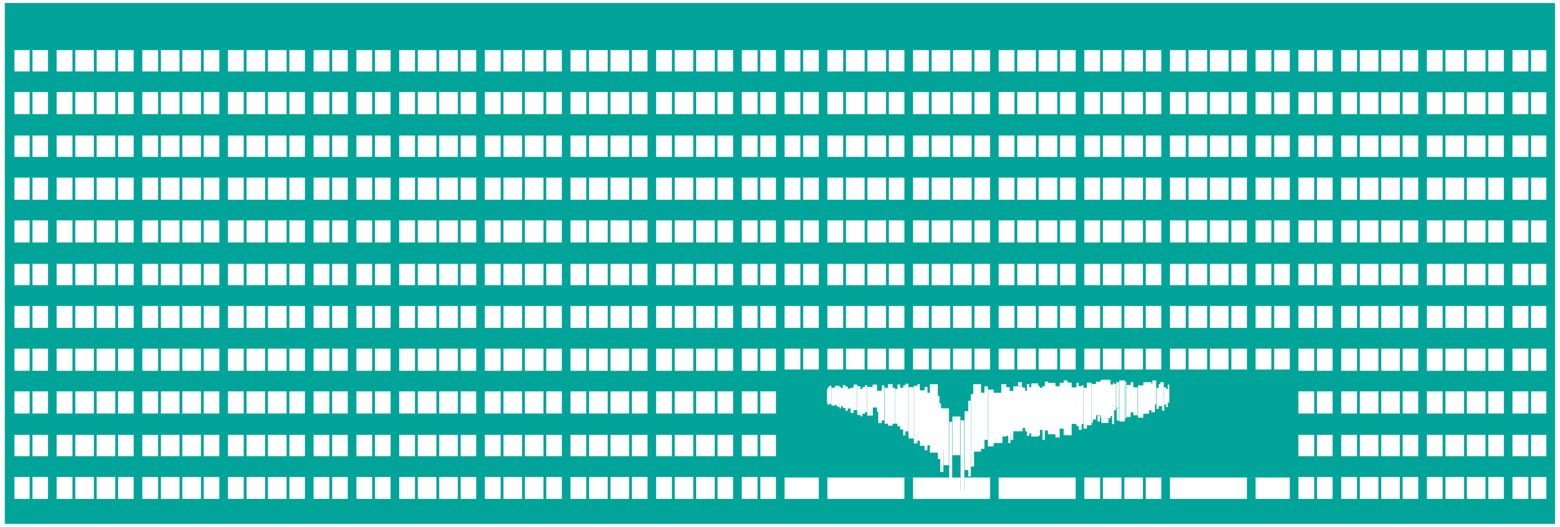
VŠB TECHNICKÁ UNIVERZITA OSTRAVA

VSB TECHNICAL UNIVERSITY OF OSTRAVA



www.vsb.cz

Android and Cryptography

Michal Krumnikl

Cryptography

- Cryptography is the practice and study of **hiding information**. Cryptography is the study of mathematical techniques related to aspects of **information security**. Increasingly used to protect information
- **Goals**
  - Confidentiality
  - Data integrity
  - Authentication
  - Availability
  - Non-repudiation

Cryptographic Criteria

- Cryptographic methods should be evaluated with respect to various criteria
  - **Level of security**
  - **Functionality**
  - **Methods of operation**
  - **Performance**
  - **Ease of implementation**

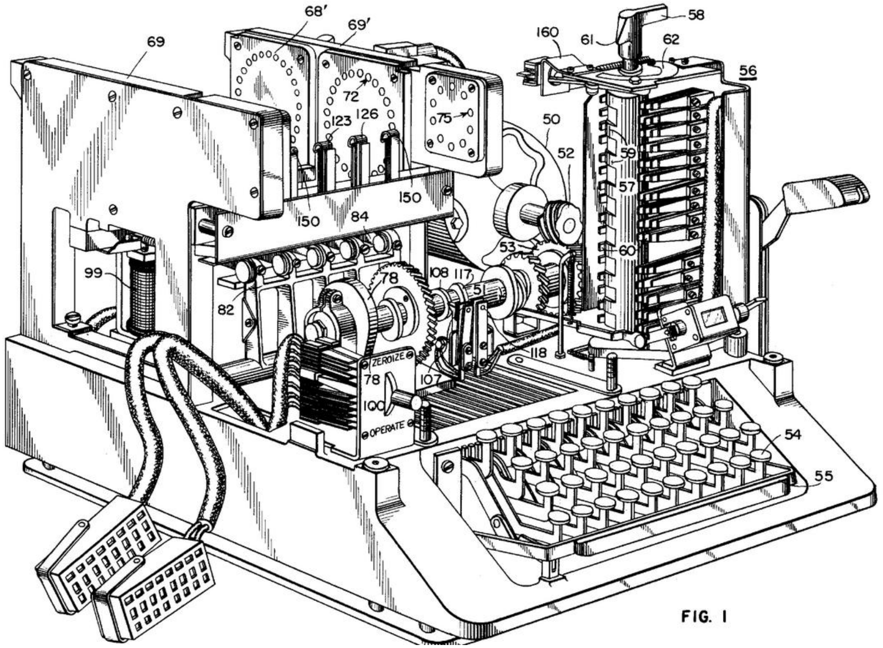


FIG. 1

SIGABA is described in U.S. Patent 6,175,625, filed in 1944 but not issued until 2001.

Basic Terms

Plaintext – A message in its natural format readable by an attacker

Ciphertext – Message altered to be unreadable by anyone except recipients

Key – Sequence that controls the operation and behavior of the algorithm

Keyspace – Total number of possible values of keys in a crypto algorithm

Cryptosystem – Algorithm, key, and key management functions

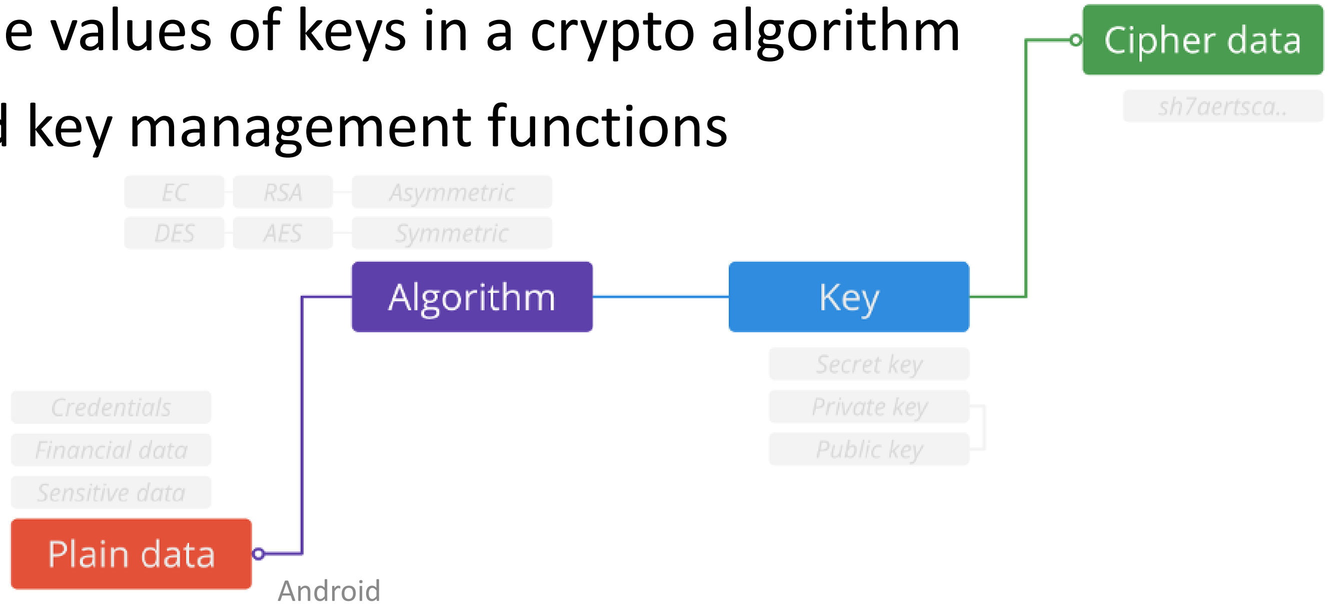
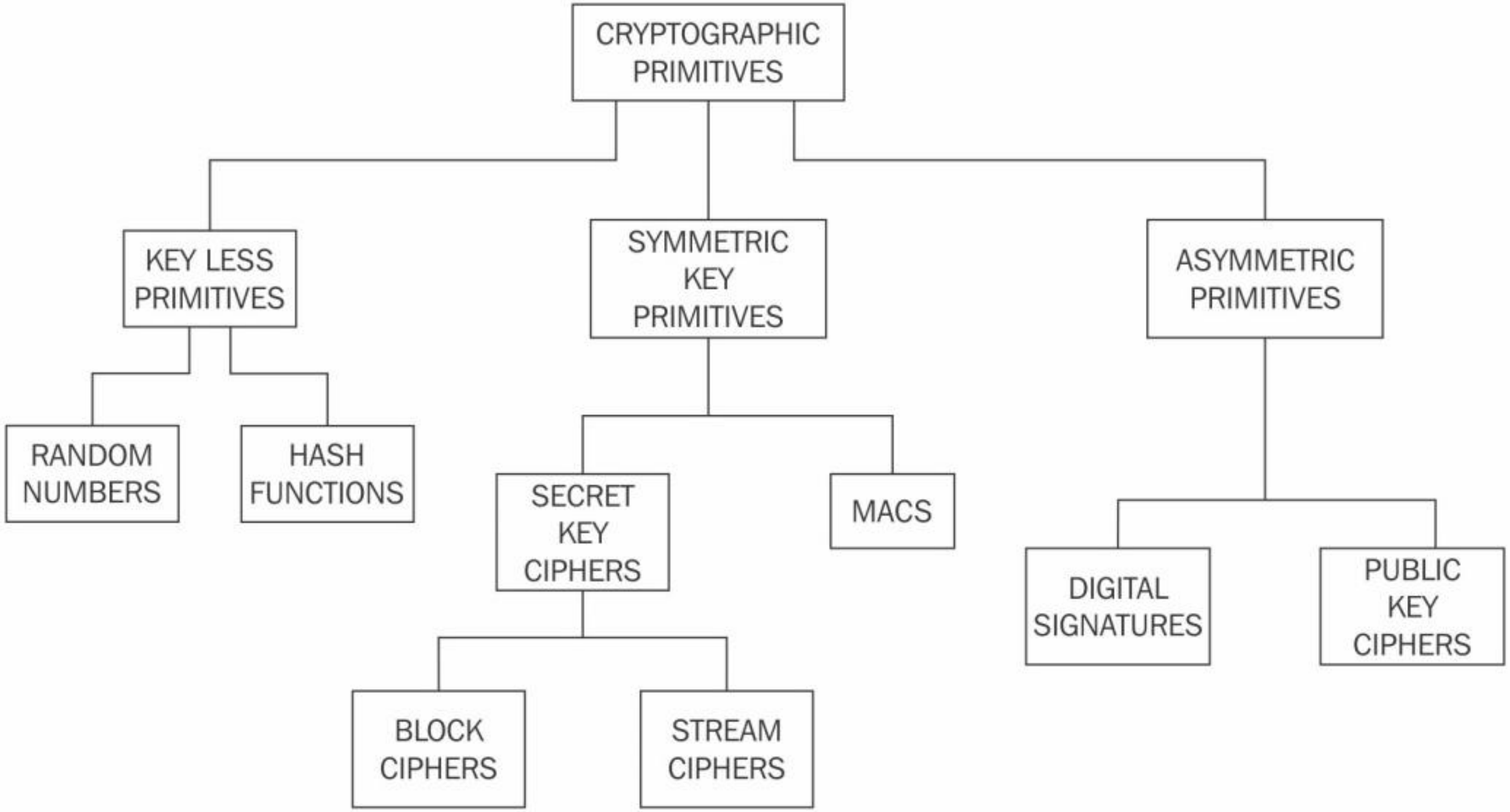
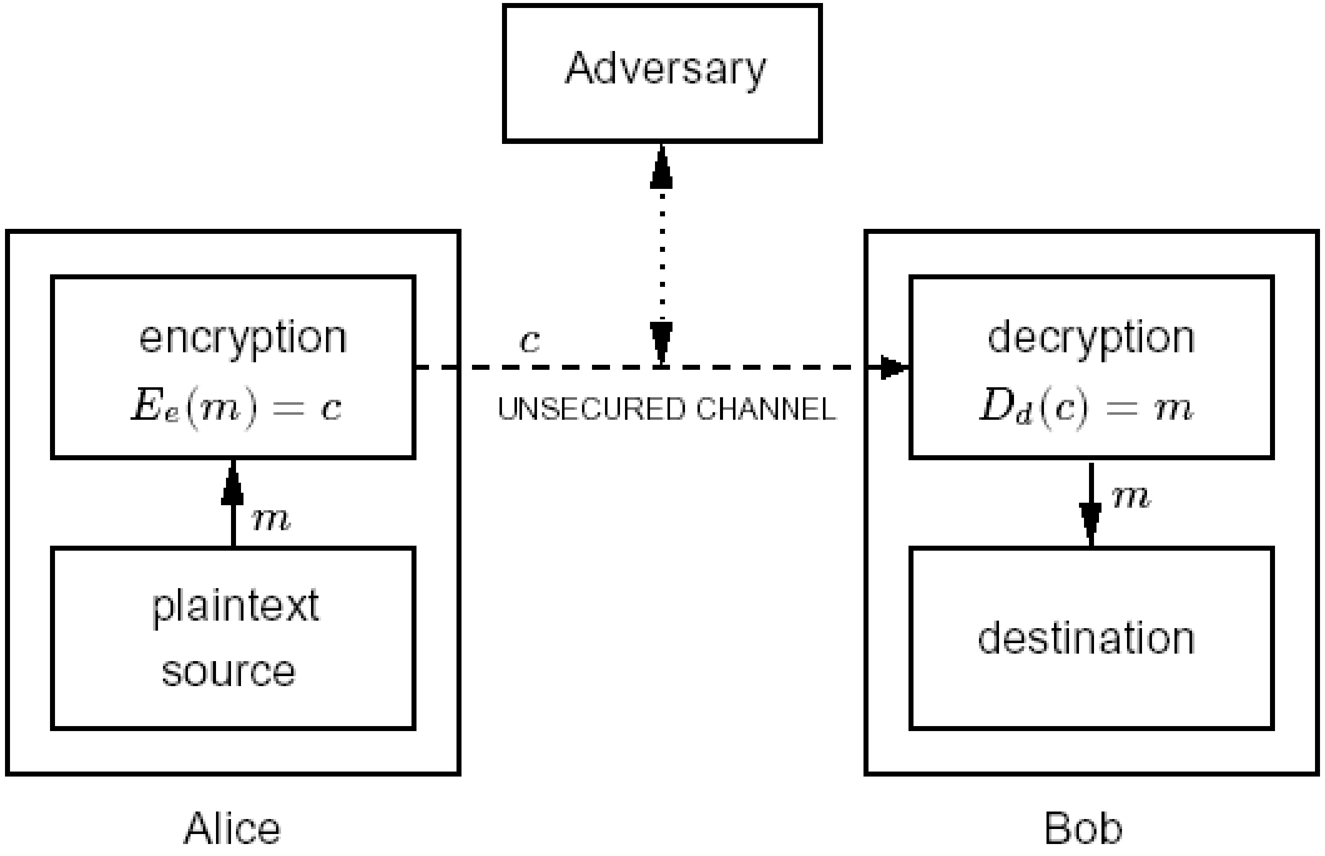


Image source: https://proandroiddev.com/secure-data-in-android-encryption-7eda33e68f58

Basic Cryptographic Primitives



Two-party Communication



Java Implementations

- Android ***javax.crypto*** package
- Defines **classes and interfaces** for various **cryptographic operations**.
- Class is *Cipher* is used to encrypt and decrypt data and streams
  - *CipherInputStream*
  - *CipherOutputStream*
- *SealedObject* - encrypt an arbitrary serializable Java object.

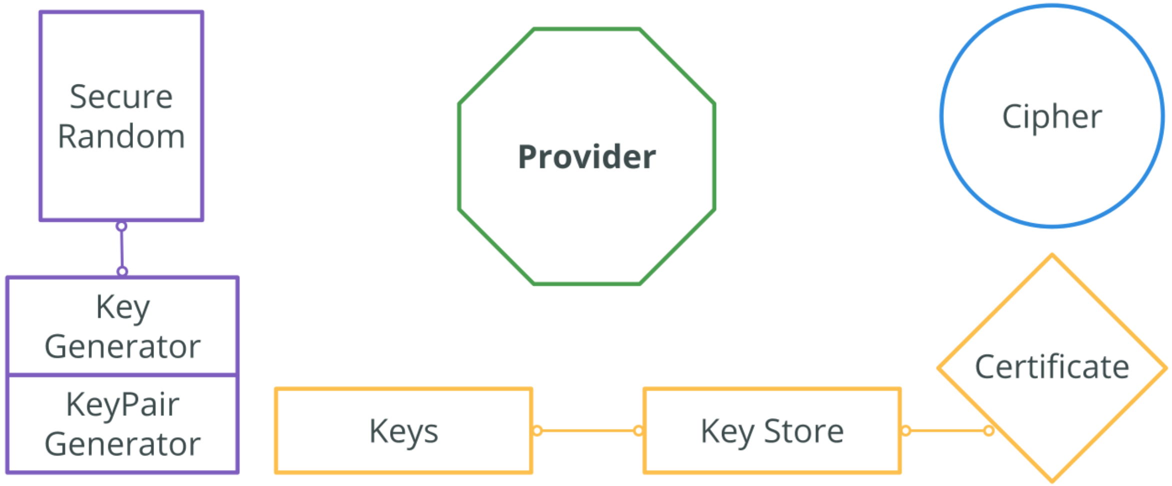


Image source: https://proandroiddev.com/secure-data-in-android-encryption-in-android-part-1-e5fd150e316f

Cipher

- Performs **encryption and decryption** of byte arrays.
- Cipher is provider-based, call the static *getInstance()* factory method.
- The arguments to this method are a string that describes the type of encryption desired.
- To specify the desired **type of encryption**, you can simply specify the name of an encryption algorithm, such as "DES". Or you can specify a **three-part name** that includes the encryption algorithm, the algorithm operating mode, and the padding scheme.
  - "DES/CBC/PKCS5Padding„
  - https://developer.android.com/guide/topics/security/cryptography#SupportedCipher

Key Agreement

- To use a **KeyAgreement** object, each party first calls the *init()* method and supplies a Key object of its own.
- Then, each party obtains a Key object from one of the other parties to the agreement and calls *doPhase().*
- Each party obtains an intermediate Key object as the return value of *doPhase(),* and these keys are again exchanged and passed to *doPhase().*
- After all calls to *doPhase()* have been made, each party calls *generateSecret()* to obtain an array of bytes or a *SecretKey* object for a named algorithm type. All parties obtain the same bytes or *SecretKey* from this method.

Android Key Store

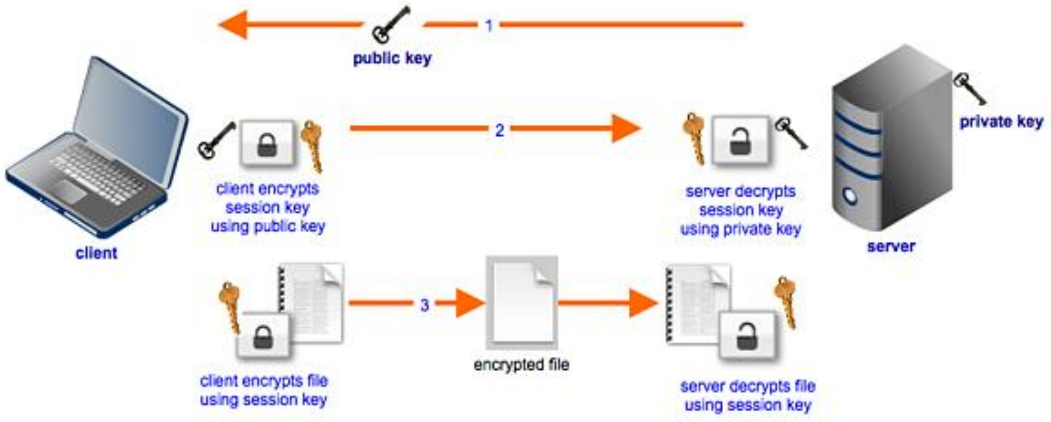
- The Android Key Store system lets you store cryptographic keys in a **container to make it more difficult to extract from the device**.
- Once keys are in the key store, they can be used for cryptographic operations with the **key material remaining non-exportable**.
- Moreover, it offers facilities to restrict when and how keys can be used, such as requiring user authentication for key use or restricting keys to be used only in certain cryptographic modes.
- If device manufacture supports **Trusted Execution Environment(TEE)**, your keys will be saved there (the most secure option);
- **Keys material is never exposed** - work with key references.

Symmetric and Asymmetric Algorithms

- **Symmetric ciphers** are the oldest and most used cryptographic ciphers.
  - **Key that deciphers the ciphertext is the same** as (or can be easily derived from) the key enciphers the clear text.
- **Asymmetric cipher** uses **two keys**
  - One key that is kept secret and known to only one person (the private key) and another key that is public and available to everyone (the public key).
  - The two keys are mathematically interrelated, but it's impossible to derive one key from the other.

Hybrid Algorithms

- Combines strengths of both methods
- **Asymmetric distributes symmetric key**
  - Also known as a session key
- **Symmetric provides bulk encryption**
- Example:
  - SSL negotiates a hybrid method

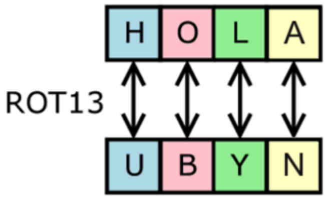


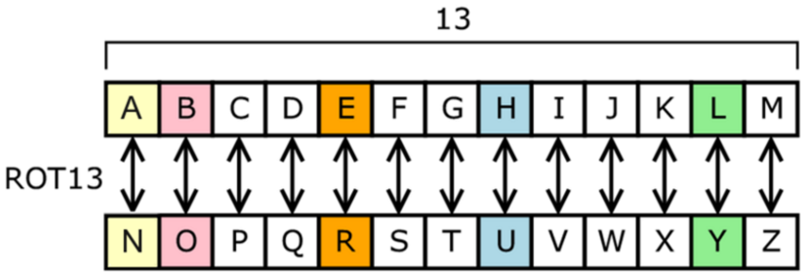
Symmetric Algorithms

- Symmetric-key encryption can use either **stream ciphers** or **block ciphers**.
  - **Stream ciphers** encrypt the digits (typically bytes) of a message one at a time. Mixes plaintext with key stream. Good for real-time services.
  - **Block ciphers** take a number of bits and encrypt them as a single unit, padding the plaintext so that it is a multiple of the block size. Uses substitution and transposition.
- Examples of **popular symmetric algorithms**
  - **Twofish, Serpent, AES (Rijndael), Blowfish, CAST5, RC4, 3DES, Skipjack, Safer+/++ (Bluetooth), IDEA**

History

- **Caesar cipher**
  - According to Suetonius, Caesar simply replaced each letter in a message with the letter that is three places further down the alphabet.
  - **Substitution cipher**

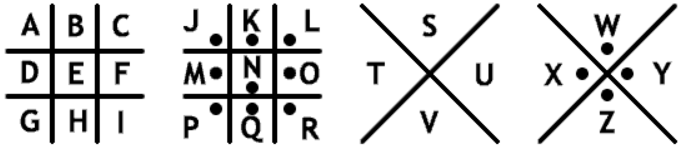


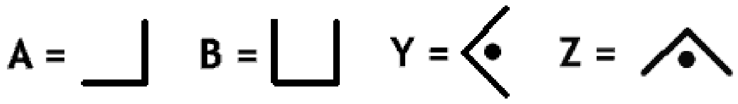


- **Kama-sutra cipher**
  - 45. art on the list is mlecchita-vikalpa, the art of secret writing to conceal the details of the liaisons.

History

- **Pigpen Cipher**
    - used by Freemasons in the 18th Century to keep their records private. The cipher substitutes each letter for a symbol





- **Playfair cipher**
    - Invented in 1854 by Charles Wheatstone.
      The technique encrypts pairs of letters (digraphs)

H	E	S	L	O
A	B	C	D	F
G	I	K	M	N
P	Q	R	T	U
V	W	X	Y	Z

History

- **Rail Fence Cipher**
  - Also called a zigzag cipher is a form of transposition cipher that derives its name from the way in which it is encoded.

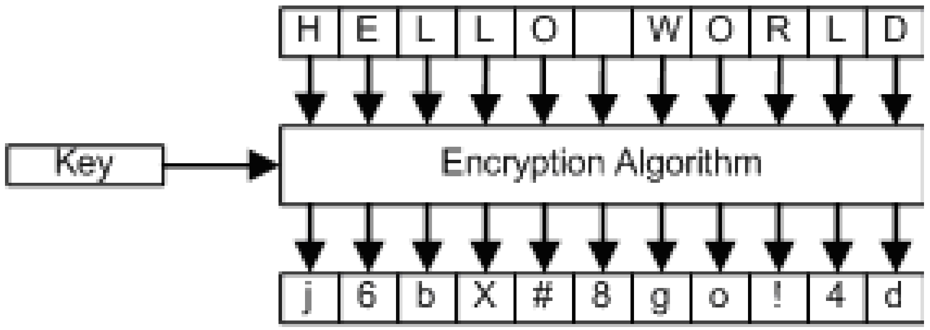
T*H*I*S*I*S*T*H*E*

***H*E*L*L*O*M*S*G*!**

THHEILSLIOSMTSHGE!

Modern Block Ciphers

- **Message is divided to n-bit block**
- **Cipher has same length as Message**
- Len (Msg) >> Len(Key)
- Key must be long (dictionary attack)
- Key must be changed periodically
- Used in all modes
- Simple implementation



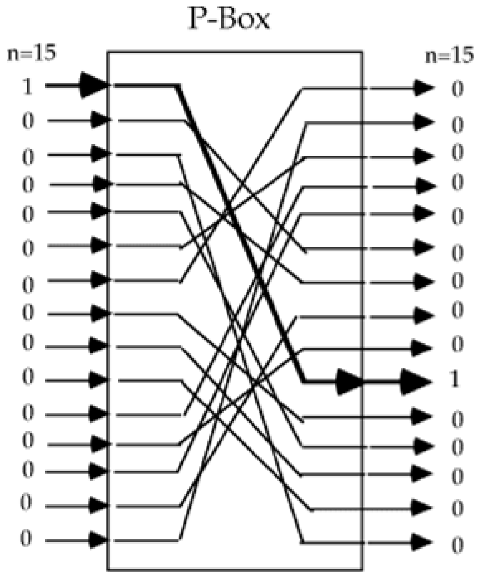
Modern Block Ciphers

- **Substitution box**
- Permutation box

S_5		Middle 4 bits of input															
		0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**	**1000**	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
Outer bits	**00**	0010	1100	0100	0001	0111	1010	1011	0110	1000	0101	0011	1111	1101	0000	1110	1001
	01	1110	1011	0010	1100	0100	0111	1101	0001	0101	0000	1111	1010	0011	1001	1000	0110
	10	0100	0010	0001	1011	1010	1101	0111	1000	1111	1001	1100	0101	0110	0011	0000	1110
	11	1011	1000	1100	0111	0001	1110	0010	1101	0110	1111	0000	1001	1010	0100	0101	0011

Modern Block Ciphers

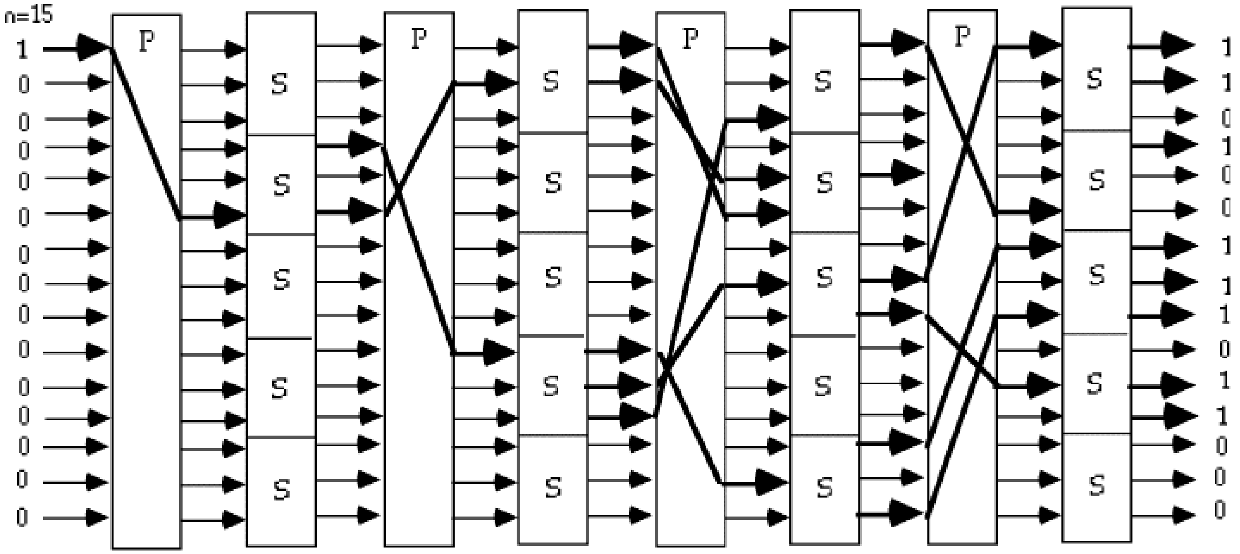
- Substitution box
- **Permutation box**



Modern Block Ciphers

- Substitution box
- Permutation box

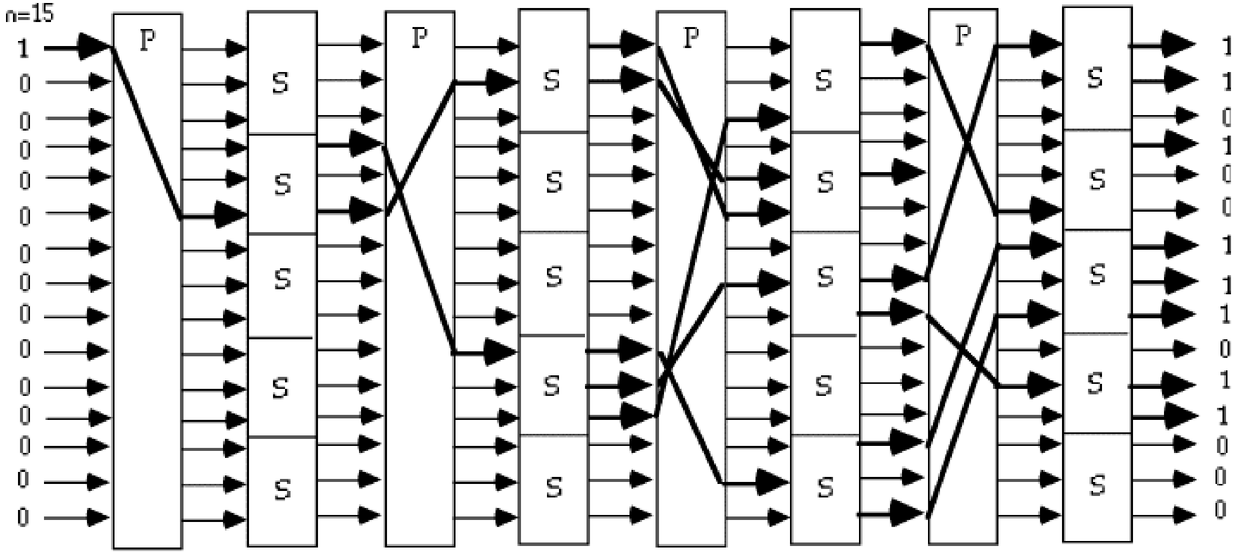
- **Avalanche effect**

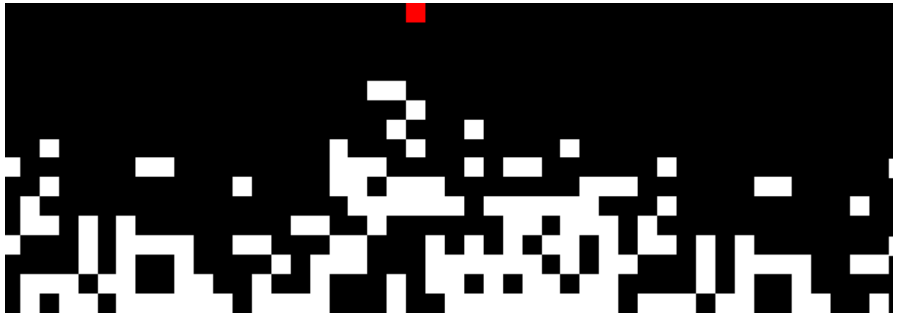


Modern Block Ciphers

- Substitution box
- Permutation box

- **Avalanche effect**
- **Completeness**



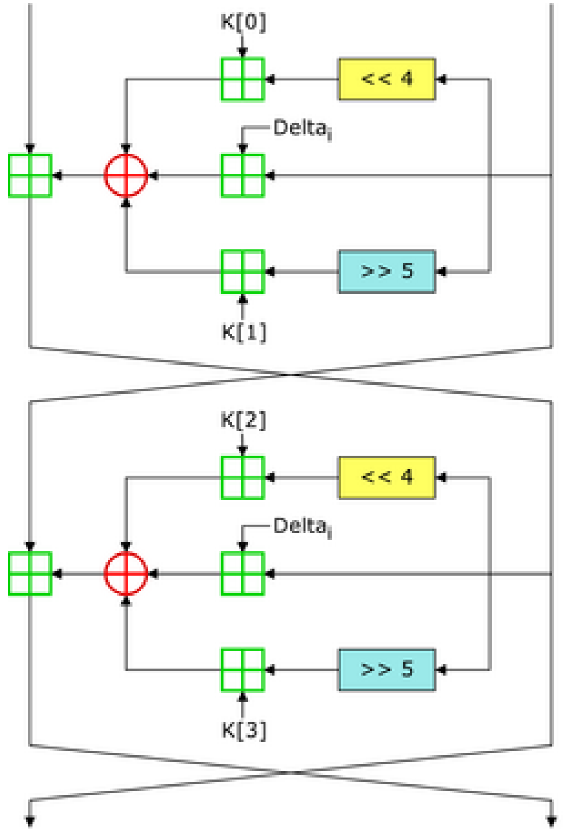


TEA – Tiny Encryption Algorithm

- **Block cipher** notable for its simplicity of description and implementation.
- Operates on **64-bit blocks** and uses a **128-bit key**
- Different multiples of a magic constant are used to prevent simple attacks based on the symmetry of the rounds.
- Each key is equivalent to three others, which means that the effective key size is only 126 bits.
- **TEA is especially bad as a cryptographic hash.**
  (see Hacking Microsoft's Xbox game console)
  - https://hackaday.com/2018/11/19/how-the-xbox-was-hacked/

TEA – Tiny Encryption Algorithm

```c
void encrypt (unsigned long* v, unsigned long* k) {
    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;
}
```

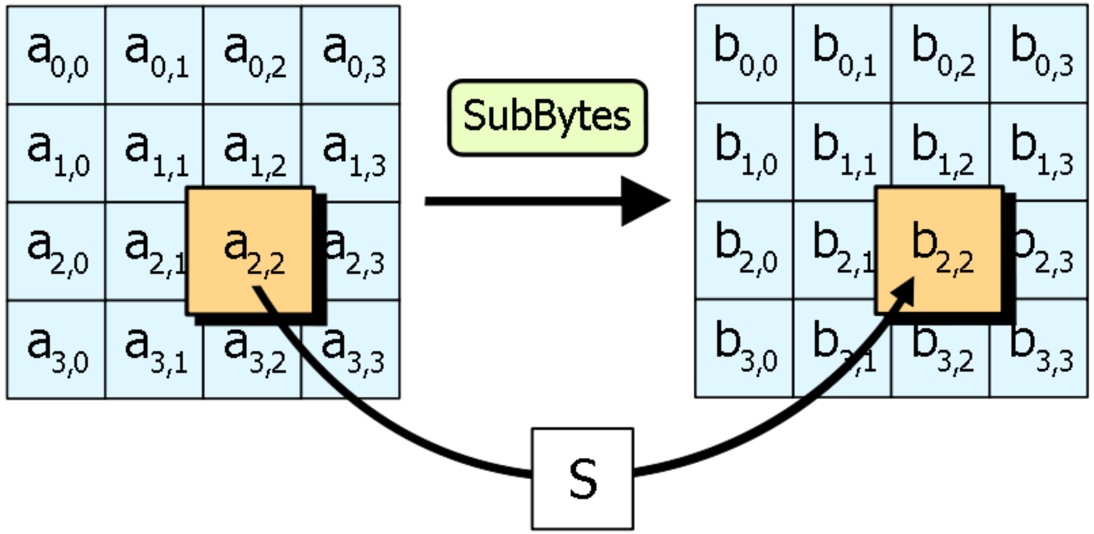


AES - Advanced Encryption Standard

- Design principle **Substitution permutation network**
- **Fixed block size of 128 bits** and a **key size of 128, 192, or 256 bits**
- AES has 10 rounds for 128-bit keys, 12 rounds for 192-bit keys, and 14 rounds for 256-bit keys
- National Institute of Standards and Technology (NIST) as U.S. FIPS PUB 197 (FIPS 197) on November 26, 2001

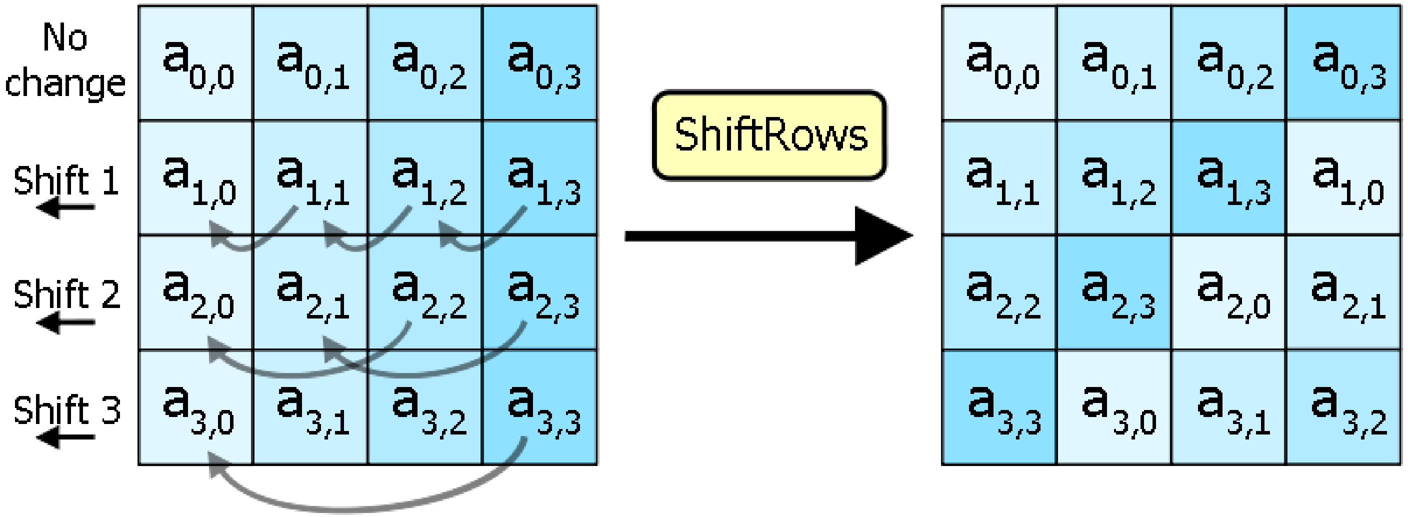
AES - Advanced Encryption Standard

- KeyExpansion
- Initial Round
  - AddRoundKey
- Rounds
  - **SubBytes**
  - ShiftRows
  - MixColumns
  - AddRoundKey
- Final Round (no MixColumns)
  - SubBytes
  - ShiftRows
  - AddRoundKey



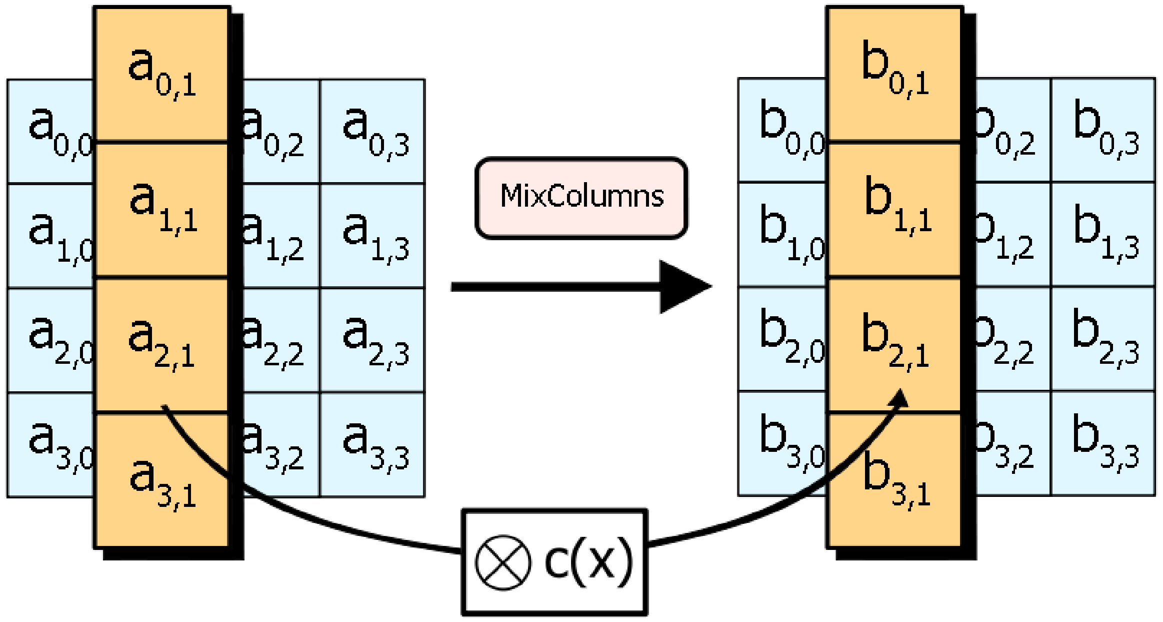
AES - Advanced Encryption Standard

- KeyExpansion
- Initial Round
  - AddRoundKey
- Rounds
  - SubBytes
  - **ShiftRows**
  - MixColumns
  - AddRoundKey
- Final Round (no MixColumns)
  - SubBytes
  - ShiftRows
  - AddRoundKey



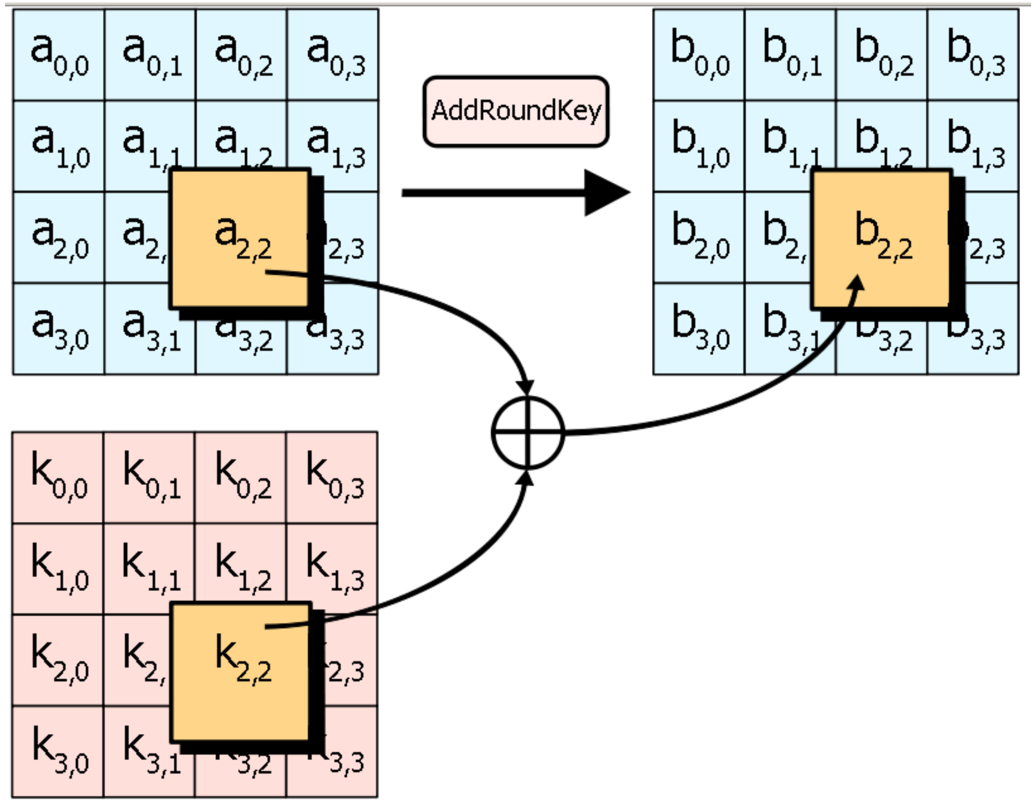
AES - Advanced Encryption Standard

- KeyExpansion
- Initial Round
  - AddRoundKey
- Rounds
  - SubBytes
  - ShiftRows
  - **MixColumns**
  - AddRoundKey
- Final Round (no MixColumns)
  - SubBytes
  - ShiftRows
  - AddRoundKey



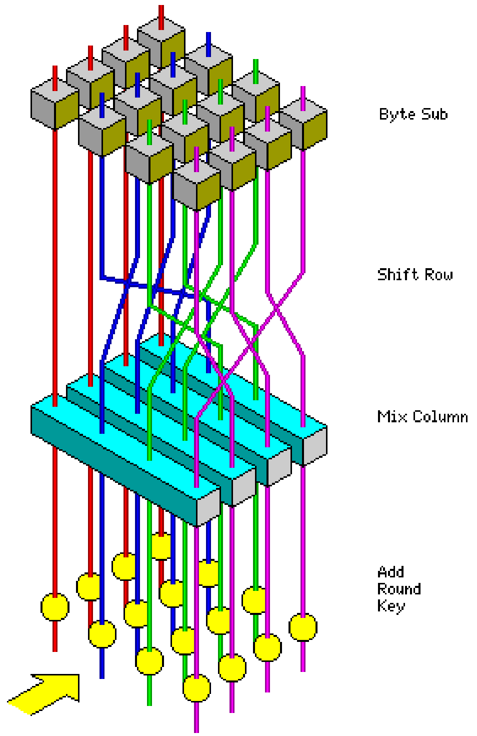
AES - Advanced Encryption Standard

- KeyExpansion
- Initial Round
  - AddRoundKey
- Rounds
  - SubBytes
  - ShiftRows
  - MixColumns
  - **AddRoundKey**
- Final Round (no MixColumns)
  - SubBytes
  - ShiftRows
  - AddRoundKey



AES - Advanced Encryption Standard

- KeyExpansion
- Initial Round
  - AddRoundKey
- Rounds
  - **SubBytes**
  - **ShiftRows**
  - **MixColumns**
  - **AddRoundKey**
- Final Round (no MixColumns)
  - SubBytes
  - ShiftRows
  - AddRoundKey



AES - Advanced Encryption Standard

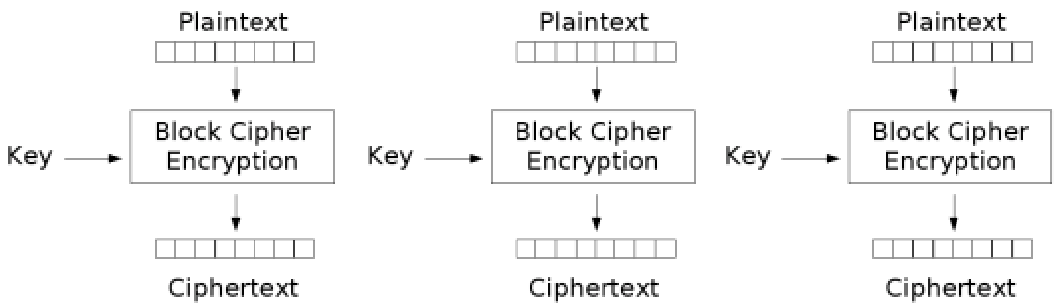
```
byte[] plaintext = ...;

KeyGenerator keygen = KeyGenerator.getInstance("AES");
keygen.init(256);
SecretKey key = keygen.generateKey();

Cipher cipher = Cipher.getInstance("AES/ECB/NoPadding");
cipher.init(Cipher.ENCRYPT_MODE, key);

byte[] ciphertext = cipher.doFinal(plaintext);
byte[] iv = cipher.getIV();
```

Modes and Paddings

- **Modes**
  - Describes how to repeatedly apply a cipher’s single-block operation to securely transform amounts of data larger than a block.
- **Padding**
  - Block cipher works on units of a fixed size (known as a block size), but messages come in a variety of lengths. So some modes (namely ECB and CBC) require that the final block be padded before encryption.

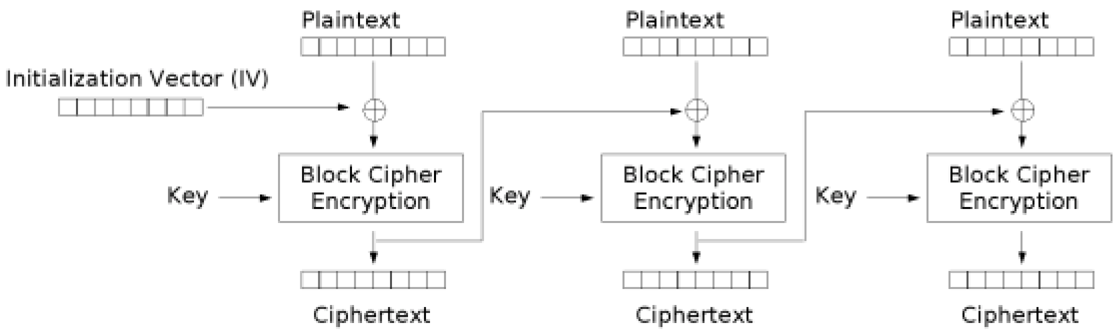
Modes of Operation

ECB (Electronic codebook)



Electronic Codebook (ECB) mode encryption

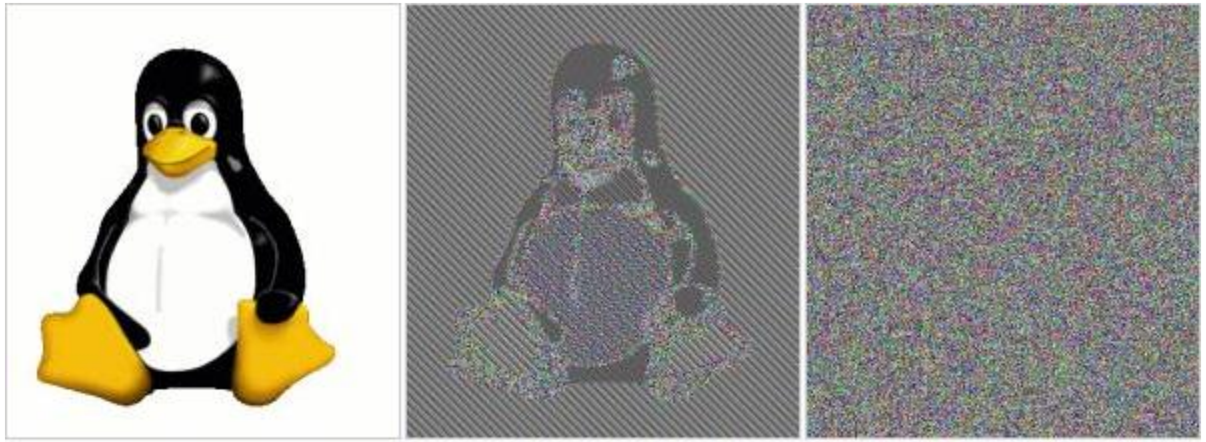
CBC (Cipher-block Chaining)



Cipher Block Chaining (CBC) mode encryption

Modes of Operation

- **ECB encryption mode should not be used.**
- ECB-encrypted ciphertext can **leak information** about the plaintext
- Adobe password database leak
  https://nakedsecurity.sophos.com/2013/11/04/anatomy-of-a-password-disaster-adobes-giant-sized-cryptographic-blunder/



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

Valid Cipher Modes and Paddings

- **Valid modes and ciphers**
  - https://developer.android.com/guide/topics/security/cryptography#SupportedCipher

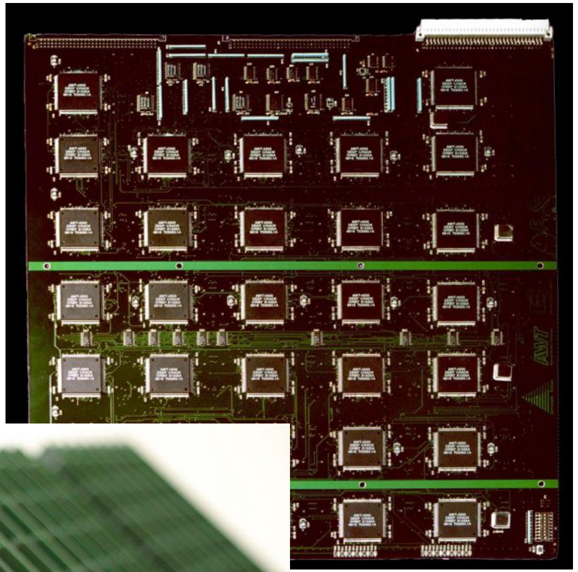
```
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
cipher.init(Cipher.ENCRYPT_MODE, key);

byte[] ciphertext = cipher.doFinal(plaintext);
byte[] iv = cipher.getIV();
```

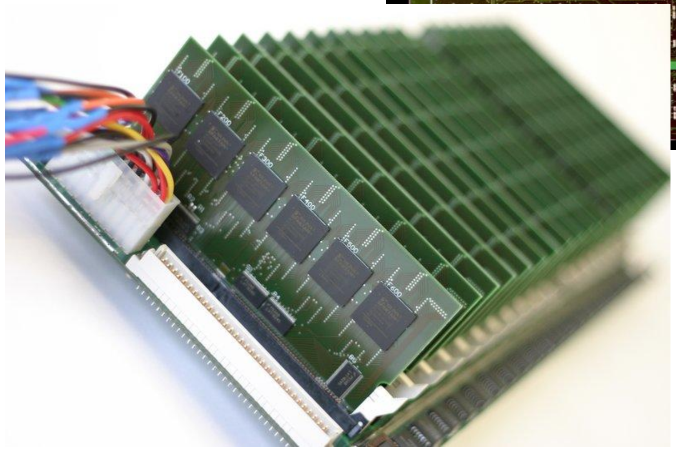
DES – Data Encryption Standard

- Symmetric-key algorithm that uses a **56-bit key**.
- Official Federal Information Processing Standard (FIPS) for the United States in 1976
- The block size is **64 bits**

Unsecure nowadays - hardware crackers

- 1998 - custom DES-cracker
- 2006 - COPACOBANA, FPGA machine
- Today - http://www.picocomputing.com/

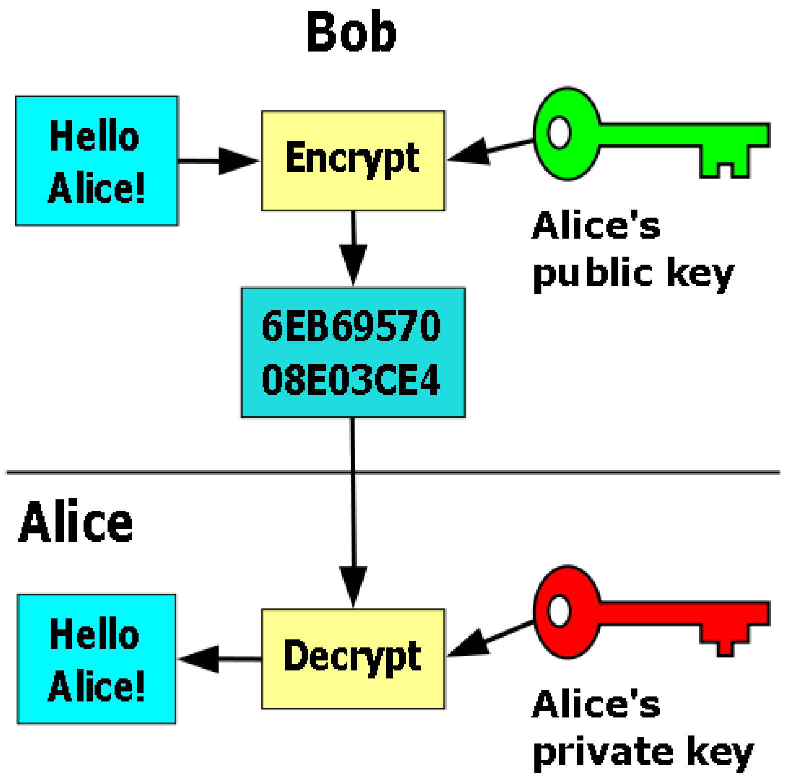


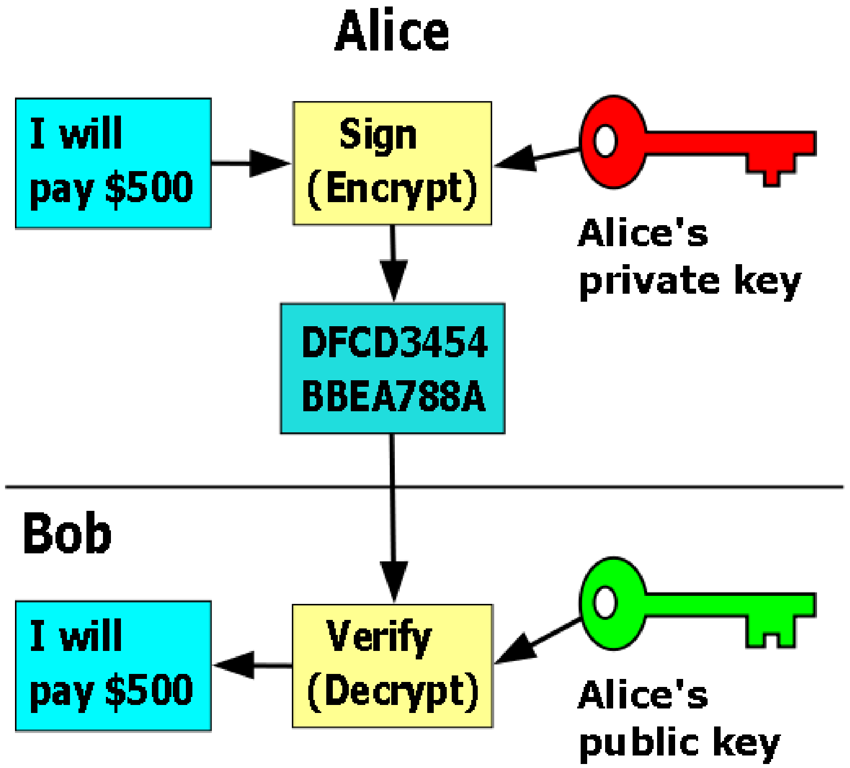


Asymmetric algorithms

- Public-key cryptography
  - Requires two separate keys, one of which is **secret (or private)** and one of which is **public**.
- No need for secure key transfer
- Key length is not so important
- Public key is accessible
- Examples of algoritms
  - Diffie-Hellman Key Exchange, ElGamal, eliptic curves, RSA, YAK, DSS

Encryption and Signing





RSA Operation

- Conditions:
  - It is possible to find *e,d,n* , so M^{ed} mod n=M
  - It is simple to calculate M^e a C^d
  - It is impossible to find *d* if we know *{e,n}*
- Algorithm :
  - Choose primes *p, q* (private values)
  - Calculate n=p*q (public value)
  - Public key *e* not dividable by $\Phi(n)$, $NSD(e, \Phi(n))=1$, $1 < e < \Phi(n)$ (public value)
  - Private key $d \equiv e\text{-}1 \mod \Phi(n)$
  - Destroy *p, q*

RSA Example

- p = 61, q = 53
- n = p*q = 61 * 53 = 3233
- Φ(n) = (p-1)(q-1) = (61-1)(53-1)=3120
- e > 1, not dividable by 3120 – e.g. *e*=17
- d, d ≡ e-1 mod Φ(n)
  - d = 2753
  - 17 * 2753 = 1 + 15*3120
- **Public key (n=3233, e=17)**
- **Private key (n=3233, d=2753)**

- **Encrypt** m = 123
  - 123^{17} mod 3233 = 855
  - c = 855
- **Decrypt** c = 855
  - 855^{2753} mod 3233 = 123
  - m = 123

RSA Key Generation

```
// Generate key pair for 2048-bit RSA encryption and decryption
Key publicKey = null;
Key privateKey = null;
try {
    KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
    kpg.initialize(2048);
    KeyPair kp = kpg.genKeyPair();
    publicKey = kp.getPublic();
    privateKey = kp.getPrivate();
} catch (Exception e) {
    Log.e("Crypto", "RSA key pair error");
}
```

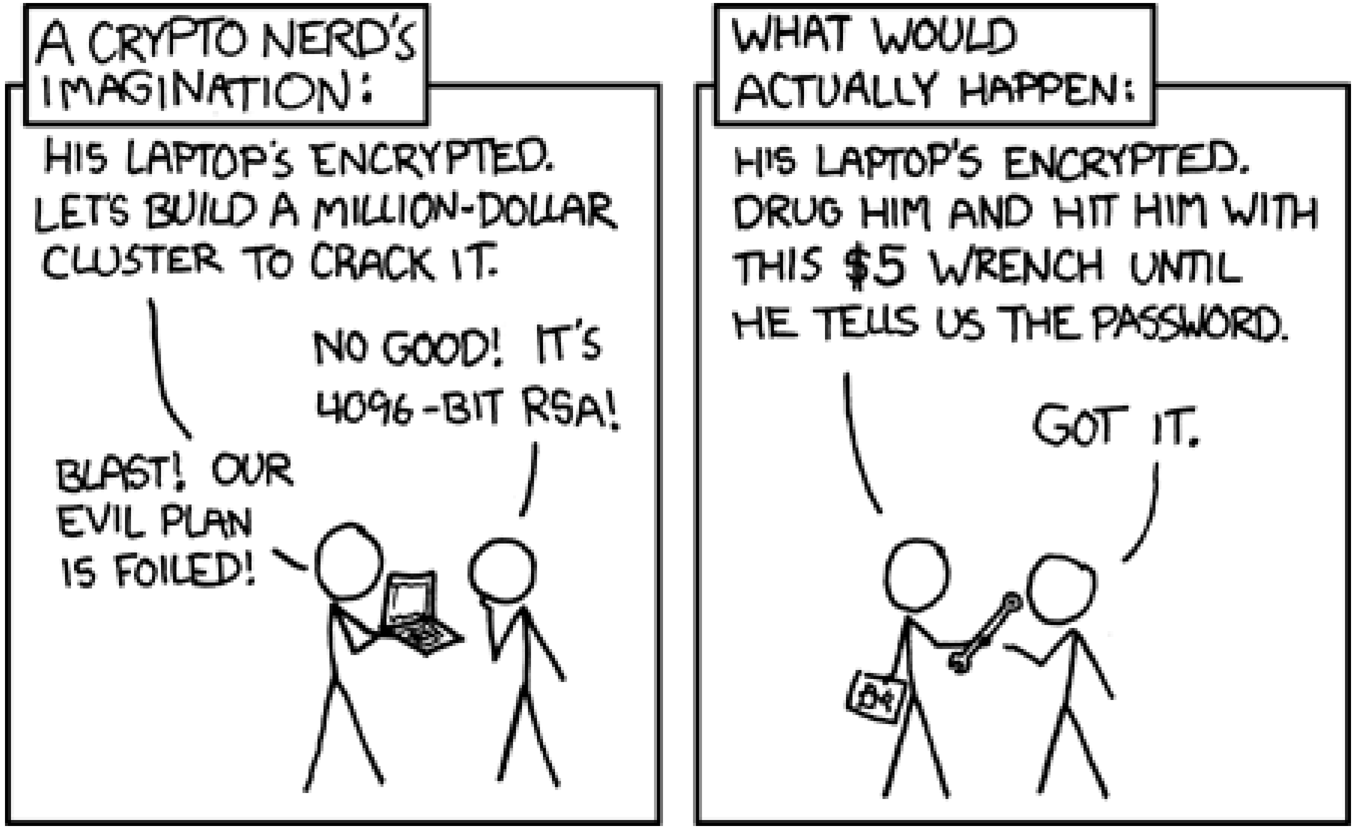
RSA Encryption

```
// Encode the original data with the RSA private key
byte[] encodedBytes = null;
try {
    Cipher c = Cipher.getInstance("RSA");
    c.init(Cipher.ENCRYPT_MODE, privateKey);
    encodedBytes = c.doFinal(targetString.getBytes());
} catch (Exception e) {
    Log.e("Crypto", "RSA encryption error");
}
```

RSA Decryption

```
// Decode the encoded data with the RSA public key
byte[] decodedBytes = null;
try {
    Cipher c = Cipher.getInstance("RSA");
    c.init(Cipher.DECRYPT_MODE, publicKey);
    decodedBytes = c.doFinal(encodedBytes);
} catch (Exception e) {
    Log.e("Crypto", "RSA decryption error");
}
```

Security by xkcd.com



References

Handbook of Applied Cryptography - http://www.cacr.math.uwaterloo.ca/hac/

Shamir's Secret Sharing Scheme - http://point-at-infinity.org/ssss/

Kryptografie a počítačová bezpečnost - http://www.cs.vsb.cz/ochodkova/courses/kpb/

Thank you for your attention

Mgr. Ing. Michal Krumnikl, Ph.D.

+420 597 325 867

michal.krumnikl@vsb.cz

www.vsb.cz