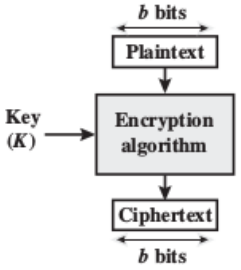
Block Ciphers

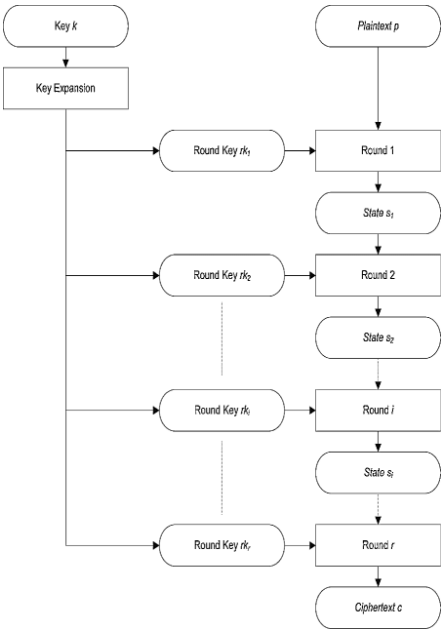
- A block cipher focuses on a simpler primitive that processes fixed-length messages (a **block**)



- The generalisation to a cipher proper (which deals with messages of arbitrary length) is handled by a standardised construction (**mode of operation**).
- Security-wise, the primitive is expected to be indistinguishable from a random permutation (when k is sampled randomly) — a (keyed indexed) **pseudo-random permutation** (**PRP**).

Design of block ciphers

- Two distinguished desirable properties [*C. Shannon* 1949]:
  - **Diffusion** — the influence of changes in the plaintext should be spread throughout the ciphertext;
  - **Confusion** — relationship between the ciphertext and the encryption key should be as complex as possible.
- Most modern designs are iterated ciphers, where a bijective **round** transformation is applied repeatedly with distinct *round-keys* (derived from the main cipher key).



Block vs. Stream Ciphers

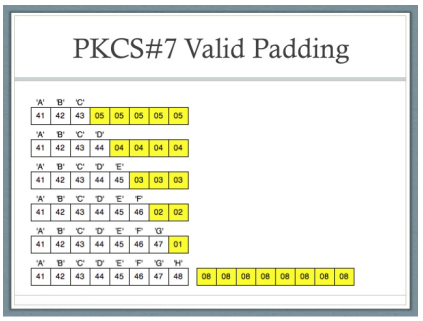
- Different “processing unit” (block vs. bit/byte);
- Use of block ciphers depend on some *mode of operation*;
- Stream ciphers do not promote diffusion;
- Block cipher inverse operation (decryption) is harder to obtain;
- Stream ciphers are typically faster than block ciphers;

Basic Modes of Operation

- A block cipher primitive by itself can only process fixed length bitstrings (blocks).
- A block cipher **mode of operation** describes how the primitive is used to encrypt/decrypt arbitrary messages.
- Depending on the applications, we can choose among several standard modes:
  - *Electronic Code Book* (**ECB**)
  - *Cipher Block Chaining* (**CBC**)
  - *Counter Mode* (**CTR**)
  - …
- Each mode of operation comes equipped with its own security analysis: assumptions; security proofs; etc.

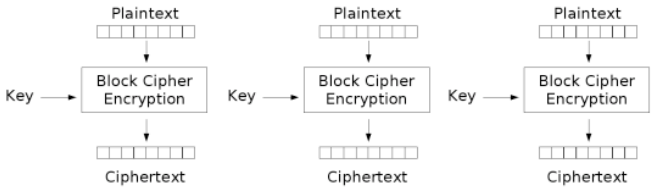
Padding

- Certain uses of block ciphers (modes of operation) require the size of the plaintext to be a multiple of the block size.
- This requirement is usually solved by adopting a **padding** method: a reversible transformation that fills the last block of the plaintext.
  - Several standard constructions available
    - bit vs. byte oriented;
    - deterministic vs. radomised;
    - etc.
  - E.g. PKCS7 padding

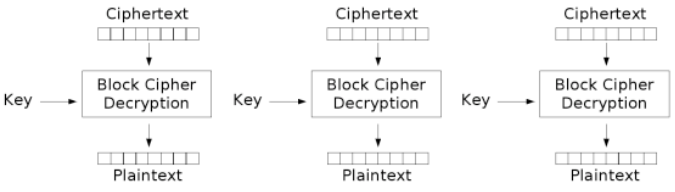


- [remark: *Ciphertext Stealing* is an alternative strategy for handling messages of arbitrary size].

Electronic Code Book (ECB)

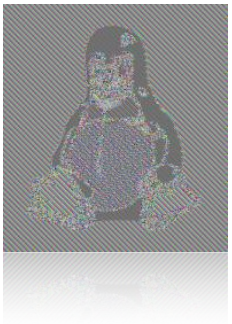


Electronic Codebook (ECB) mode encryption

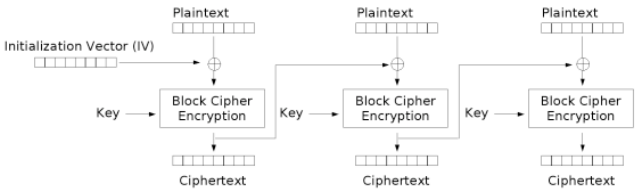


Electronic Codebook (ECB) mode decryption

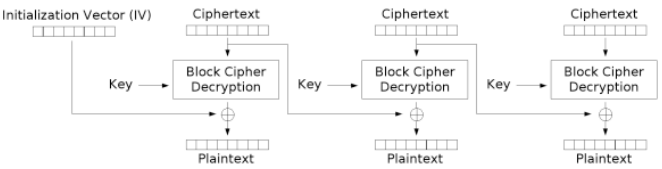
- Requires Padding.
- Repetition of blocks is detectable - **code book attack**;
- …leading to the leakage of patterns of the plaintext.
- **It should only be used to encrypt single-block messages.**
- Vulnerable to repetition/replacement attacks.



Cipher Block Chaining (CBC)



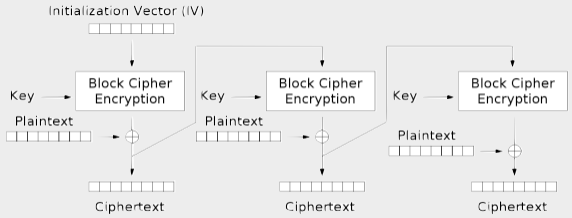
Cipher Block Chaining (CBC) mode encryption



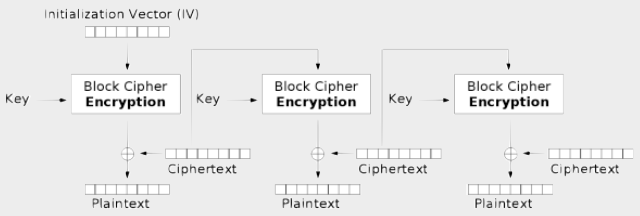
Cipher Block Chaining (CBC) mode decryption

- Each plaintext block is masked by the previous ciphertext
- …which induces a dependency on all previous plaintext blocks.
- A random **Initialisation Vector** (**IV**) shall be used to initiate the process.
- As with Nonces, the IV doesn't need to be kept secret.
- The swapping of one bit of the ciphertext would impact the corresponding block plus a single bit on the next-block.
- The last block of the ciphertext can be used as a Message Authentication Code (CBC-MAC).
    - In such use, a fixed IV shall adopted;
    - Secure for fixed-length messages (but simple fixes can be adopted to make it suitable for arbitrary length messages).

Cipher FeedBack mode (CFB)



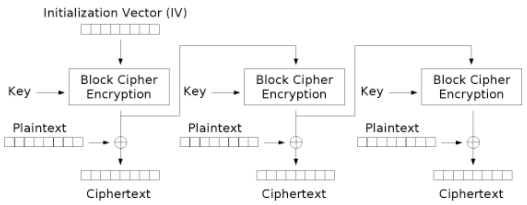
Cipher Feedback (CFB) mode encryption



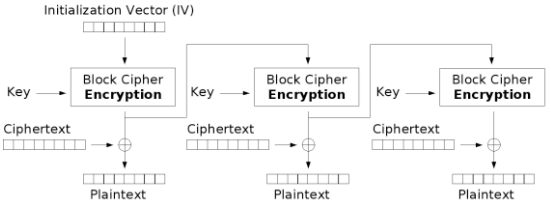
Cipher Feedback (CFB) mode decryption

- Implements a self-synchronising stream cipher.
- Does not requires padding.
- IV should be a Nonce.
- Notice that block cipher primitive is always used in “encrypt” mode.
- Keystream depends on the key, IV, and all the previous plaintext.
- Number of bits in the feedback is variable (CFBn).
  - Between 1 and the block size;
  - Feedback transfers the n most significant bits to the least significant bits (with a shift of the remaining bits);
  - Number of bits in the feedback impact the resynchronisation time.

Output FeedBack mode (OFB)



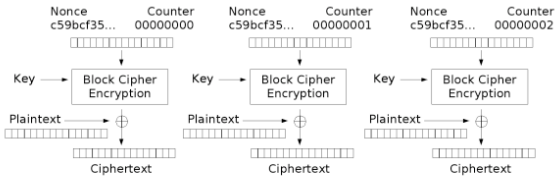
Output Feedback (OFB) mode encryption



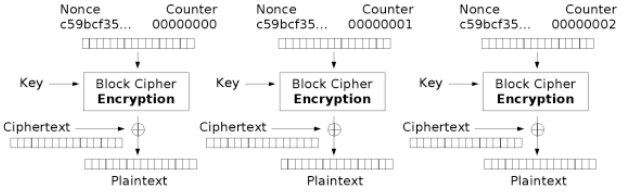
Output Feedback (OFB) mode decryption

- Implements a synchronous stream cipher (in output-feedback mode).
  - The block cipher is used as the *next-state function*;
  - The *output-function* is the identity function.
- Does not requires padding.
- IV should be a Nonce.
- Block cipher primitive is only used in “encrypt” mode.
- Keystream doesn’t depends on the message — is obtained by iterating the block cipher on the IV.
  - It can be expanded prior to the reception of the messsage;
  - but needs to be computed sequentially.
- The swapping of one bit of the ciphertext only impacts a single bit.

CounTeR mode (CTR)



Counter (CTR) mode encryption



Counter (CTR) mode decryption

- Implements a synchronous stream cipher (in counter mode).
  - The *next-state function* is a simple increment;
  - The block cipher is used as the *output-function*.
- Does not requires padding.
- IV should be a Nonce.
- Block cipher primitive is only used in “encrypt” mode.
- Keystream doesn’t depends on the message.
  - It can be expanded prior to the reception of the messsage;
  - and each block of the keystream can be processed independently (e.g. in parallel).
- The swapping of one bit of the ciphertext only impacts a single bit.

Authenticated Encryption

- A recent trend has been the adoption of block cipher modes of operation which provides both **confidentiality** and authenticity (**integrity**) of data.
- Usually supports also "*Associated Data*” — data which is not encrypted but is attached to the integrity guarantee (e.g. metadata).
- Some examples:
  - **EAX** (Encrypt-then-Mac-then-Translate)
  - **CCM** (Counter with CBC-MAC)
  - **GCM** (Galois/Counter Mode)
  - **OCB** (Offset CodeBook)

Advanced Encryption Standard (AES)

- NIST call for block cipher algorithm that would replace Data Encryption Standard (DES) — (announcement:1997, submissions: 1998, decision 2001).
- Requirements:
  - Blocks and keys with, at least, 128 bit;
  - Faster and more secure than TripleDES;
  - Detailed specification and design rational;
  - Suitable for software implementations (32bit architectures).
- Finalists:
  - **MARS** — complex, fast, high security margin
  - **RC6** — very simple, very fast, small security margin
  - **Rijndael** — clean design, fast, good security margin (**WINNER**)
  - **Serpent** — slow, clean, very high security margin
  - **Twofish** — complex, very fast, high security margin

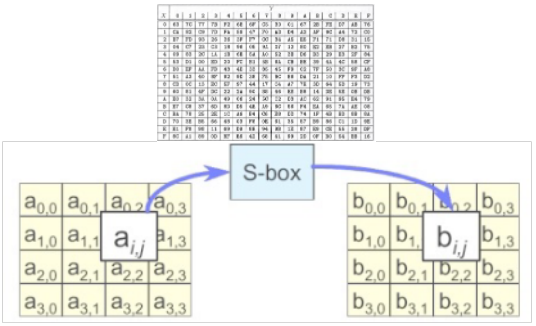
AES (RijnDael) description

- Block size of 128 bit (16 byte), organised as a 4x4 byte matrix;
- Iterated block cipher. Number of rounds: 10, 12 or 14 (depending on the key size);
- Supports keys of 128, 192 and 256 bit. KeyExpansion algorithm expands it into required 128 bit round keys;
- Detailed mathematical justification for specification details;
- Round processing steps:
  - SubBytes
  - ShiftRows
  - MixColumns
  - AddKey

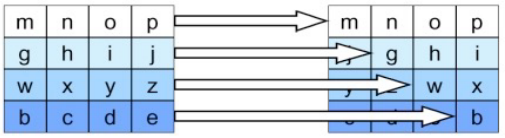
  (with corresponding *inverted* variants and subtle teaks to facilitate implementation)
- **Believed** to be resistant to all *known* attacks!

AES round

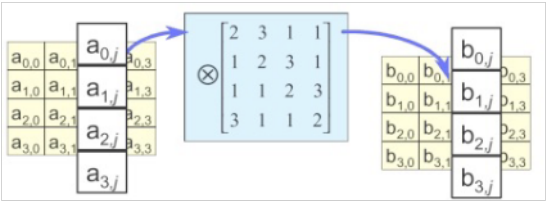
- ByteSub
  - A single S-box
  - Highly non-linear



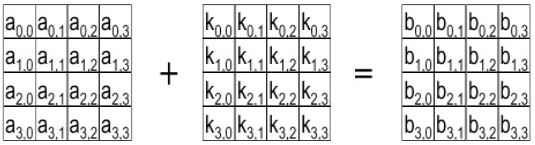
- ShiftRow
  - Rotate rows



- MixColumn
  - Interaction with ShiftRow promotes high diffusion in multiple rounds



- KeyAddition



One-Way Functions

One-Way Functions

- Security of cryptographic techniques often translates into establishing that a given function is **one-way.**

  A function $f:A \rightarrow B$ is **one-way** when, given some element $x:A$, it is *easy* to compute $f(x):B$, but giving any $y:B$ in the range of f, is very *difficult* to find some $x:A$ such that $y=f(x)$ (a *pseudo-inverse* of y).

- Notions of "easy" and "difficult" must be understood in the context of complexity theory, depending on the existence (or not) of *polynomial-time algorithms* for their calculation.

  Existence of one-way functions ⇒ P≠NP

  (obs.: but is indeed *a stronger assumption*)

- From a theoretical point of view, the existence of OW functions is usually identified as the assumption that makes symmetric cryptography feasible.

Cryptographic Hash functions

- A paradigmatic example of one-way functions are cryptographic hash functions.
- A **cryptographic hash function** is a one-way function that maps arbitrary length bit strings into a fixed-length *n*-bit range.

$$H : \{0,1\}^* \rightarrow \{0,1\}^n$$

- Looking at the cardinalities of domain/codomain, we clearly have that every hash function is not injective — **collisions** always exist! (that is, distinct domain points m_1 and m_2 such that $H(m_1)=H(m_2)$).
    - …but, being OW functions, finding collusions should be difficult.
- Their use is prevalent in cryptography.

Defining properties

- The requirements of hash functions are usually expressed by the following hierarchy of properties:
    1. (**First) pre-image resistant**: given a hash value *h*, it should be unfeasible to obtain a message *m* such that $H(m)=h$.
    2. **Second pre-image resistant**: given a message m_1, it should be infeasible to obtain a message m_2 distinct from m_1 such that $H(m_2)=H(m_1)$.
    3. **Collision resistant**: it is unfeasible to find distinct messages m_1 and m_2 such that $H(m1)=H(m_2)$.
- The first simply asks for one-wayness of the hash function H.
- But when designing/choosing hash functions, we look at the stronger property of *collision resistance* (cannot find any collision).
- However, the security of some applications only ask for one of the a weaker properties…

Birthday paradox

- A standard result in probability theory tell us that we need reasonably sized codomain to achieve collision resistance.

  How many people need to meet to make it more likely than unlikely *that at least 2 share the same birthday?*

- …it would be enough to check around *sqrt(366)* random elements!
- If we consider typical hash codomain sizes ranging on 128 and 512 bits
- …we conclude that the *birthday attack* would require around 2^{64} and 2^{256} random picks to find a collision.

Examples

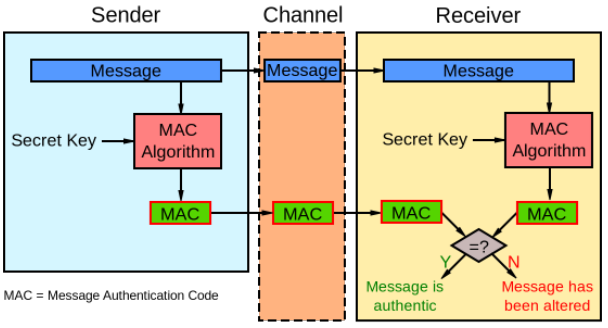
- Most common cryptographic hash functions:
  - **MD5** — codomain size: 128 bit. **Already broken!**
  - **SHA-1** (Standard Hash Algorithm) — codomain size: 160 bit. **Already broken!**
  - **SHA-2** — codomain sizes: ~~224~~, 256, 384 and 512 bits. Adopt the same design of SHA-1, with enhanced security.
  - **SHA-3** — codomain sizes: ~~224~~, 256, 384 and 512 bits. Follow a different design (sponge construction).
  - **SHAKE-128/256** — eXtendable Output Function (XOF). The same design of SHA-3.

Applications of Hash functions

- Password storage/verification;
- Building block of other cryptographic components:
  - Message Authentication Codes (MAC);
  - Key-Derivation Functions (KDF);
  - Secure Pseudo-Random Number Generators;
  - Commitment-schemes;
  - Block ciphers;
  - …

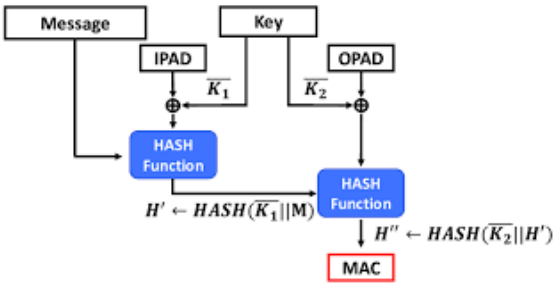
Message Authentication Codes (MAC)

- Hash functions, per se, do not ensure integrity of data — a malicious user might recompute its value for the modified data.
- A **Message Authentication Code** (**MAC**) should be used — it relies on a secret key that is known only to the legitimate parties (sender and receiver).
  - sender computes a **authentication tag** (from message and key);
  - receiver recomputes the tag, and compares it with the received one.
- Informally, it can be though as a Keyed-Hash function (Hash function with a secret key).
- Examples: `Poly1305`; `CMAC`.



HMAC

- A MAC can be easily constructed from any Hash function.
- A standard construction is the Hash-based Message Authentication Code (HMAC).
  - It performs two-passes on the hash function, to account for possible weakness of the used functions (length-extension attacks).
- Examples: HMAC-SHA256; etc.
- Remark: the SHA-3 design is not vulnerable to length-extension attacks. Hence, the HMAC nested construction is redundant for the SHA-3 family of hash functions — its enough to use $H(K||M)$.



Key-Derivation Functions (KDF)

- Key-Derivation Functions (KDF) allow the derivation of cryptographic keys from other (secret) material.
- KDFs are used in a variety of situations:
  - derive keys from weak secrets (passwords/passphrases);
  - derive keys for different cryptographic purposes from a single "*master secret*" (key diversification);
  - derive keys of different length from the ones provided;
- Sometimes, it accepts also non-secret inputs (`other-info`), allowing to bind the secret to application-specific data.

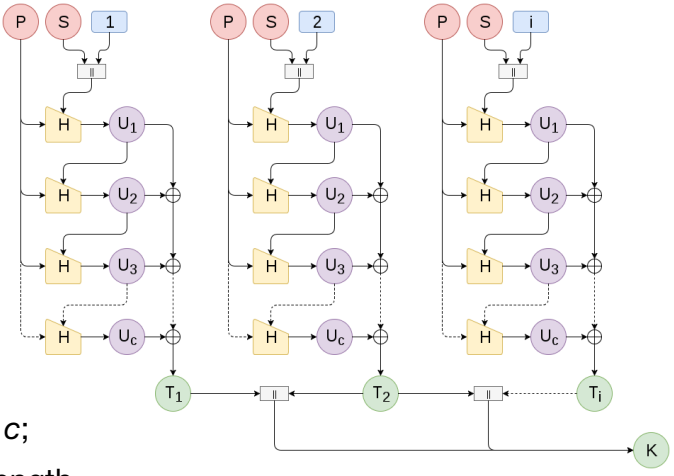
Password-Based KDF

- Passwords/Passphrases cannot be used directly as cryptographic keys
- …but we can derive cryptographic keys from those passwords (through a KDF).
- Main problem is that a KDF doesn't solve their inherent weaknesses:
  - low entropy;
  - vulnerable to *dictionary attacks*.
- Despite all their drawbacks, passwords remain a prevalent resource in information security (user authentication; keystore protection; etc.)

Password-Based KDF

- Passwords/Passphrases cannot be used directly as cryptographic keys
- …but we can derive cryptographic keys from those passwords (through a KDF).
- Main problem is that a KDF doesn't solve their inherent weaknesses:
  - low entropy;
  - vulnerable to *dictionary attacks*.
- Despite all their drawbacks, passwords remain a prevalent resource in information security (user authentication; keystore protection; etc.)

- It is recommended that a KDF, to be used with weak secrets, implement countermeasures that minimise the problems mentioned above:
  - use of a **salt** — a random number that randomises the output of the OW-function. It helps defending against attacks that use pre-computed tables (e.g. rainbow tables);
  - be **resource intensive** (CPU time and/or memory) to make dictionary attacks more difficult.
- Example: PBKDF2
  - Parametric on some MAC *H;*
  - uses a salt *S* and a iteration count *c*;
  - produces a secret of some given length.



Password protection

- Certain applications, and specifically *password protection*, recommend stricter counter-measures to dictionary attacks.
  - account for the possibility of attackers build dedicated password-cracking machines;
  - go beyond computational intensive: *memory-hard* designs.
- Some algorithms:
  - **bcrypt** (1999) — support adjustable cost;
  - **SCrypt** (2008) — combines PBKDF2 with a special construction (ROMix) making it resistant to hardware brute-force attacks;
  - **Argon2** (2015) — winner of the Password Hashing Competition.

Security balance

- …but in the end, security will always depend on the entropy of the passphrase!

Estimated cost of hardware to crack a password in 1 year.

KDF	6 letters	8 letters	8 chars	10 chars	40-char text
DES CRYPT	< $1	< $1	< $1	< $1	< $1
MD5	< $1	< $1	< $1	$1.1k	$1
MD5 CRYPT	< $1	< $1	$130	$1.1M	$1.4k
PBKDF2 (100 ms)	< $1	< $1	$18k	$160M	$200k
bcrypt (95 ms)	< $1	$4	$130k	$1.2B	$1.5M
scrypt (64 ms)	< $1	$150	$4.8M	$43B	$52M
PBKDF2 (5.0 s)	< $1	$29	$920k	$8.3B	$10M
bcrypt (3.0 s)	< $1	$130	$4.3M	$39B	$47M
scrypt (3.8 s)	$900	$610k	$19B	$175T	$210B

Key-Management

Cryptographic Keys

- A critical factor in the security of cryptographic techniques is the quality of the keys that are used.

  …always use Cryptographic Secure Random-Number Generators!

- Their shape (and specifically, their size) depend on the specific technique. For symmetric cryptography, keys are usually random bit strings (sizes ranging between 128 and 256 bits).
- Extreme care must be taken when handling cryptographic keys (storage; backup; disposal; etc.)

Key handling

- As a rule of thumb:
  - the *key lifetime* **(cryptoperiod)** shall be as short as possible,
  - and inversely proportional to their use and the criticality of the protected data.
- Keys are classified as
  - **Long-term** keys (aka **static** keys);
  - **Short-term** keys (aka **ephemeral** keys).
- As we will see, symmetric cryptography favours short-term keys.
- E.g. the establishment of a secure-channel should rely on session keys:

  A **session key** is a single-use symmetric key used for safeguarding communication during a specific interaction (a **session**).
- On the programming level, cryptographic APIs typically handle keys as an *abstract datatype*.
  - allow strict control on the available operations to manipulate keys;
  - support specific safeguards (e.g. clear the memory after use).

Key distribution

- The pre-distribution of keys is the major challenge in using symmetric crypto.
- They typically depend on secure-channels, which are costly.
- Notice the circularity:

  Cryptography can be used to establish a secure channel between two parties, but itself depends on the existence of a secure channel to distribute the secret key.
- Remembering the recommendation to use session keys makes the problem worse!
- We will see that asymmetric cryptography offers more attractive solutions, but is instructive to consider how the use of symmetric cryptography can obviate the problem.

Key distribution protocols

- In a community with N users, the number of keys each member needs to store to allow secure communication between any pair of members is (N $*$ (N - 1))/2.
  - This is clearly not feasible in a community of considerable size (e.g. internet).
- **Key Distribution Protocols** make use of a "trust network" to distribute the keys between concerned parties.
  - E.g.: a Trusted Third-Party (TTP) shares a key with each member;
  - The TTP generates and distributes a session-key between any pair of agents on demand.
- But these protocols are still not suitable for open communities (such as the Internet).

Guidelines for key-management

- Keys shall not be used for multiple purposes (use instead KDFs to derive different keys);
- In security-critical operations, key-handling (generation; storage; use; etc.) should be restricted to **Hardware Security Modules** (**HSM**).
- Keys should be deleted when compromised or expired (but different applications can adopt different policies)
- If key-backup mechanism is needed, a threshold *secret-sharing scheme* should be used.

Cryptographic Key-Management Systems (CKMS)

- A CKMS consists of a set of policies, components and devices that are used to protect, manage and distribute cryptographic keys.
- A CKMS covers all states a key passes between its generation and its destruction — **key lifecycle**