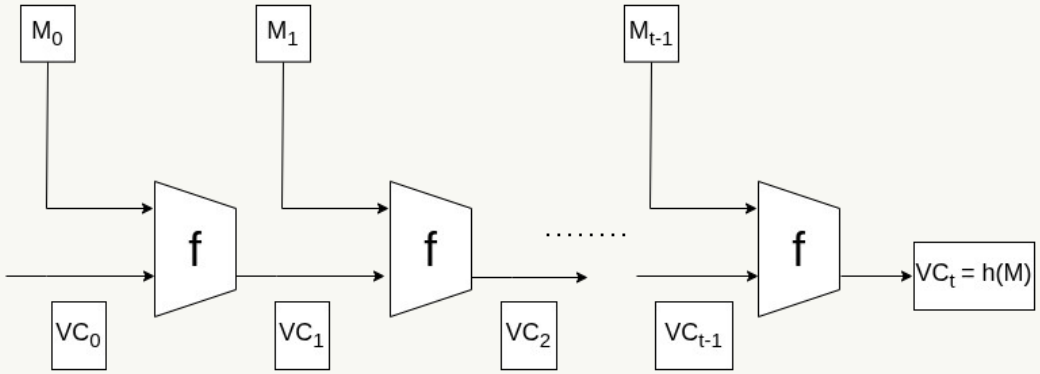
Fonctions de hachage - Motivations

- Transformer un message de longueur arbitraire en une empreinte de taille fixe.
- Détection d'altérations : changement minime du message ⇒ empreinte très différente.
- Vérification d'intégrité pour stockage/transmission.
- Accélérer signatures : on signe l'empreinte au lieu du document complet.
- Protection des mots de passe (stockage d'empreintes, salage).
- MACs (hachage + clé) pour authentifier messages.
- Dérivation/diversification de clés, engagements, PRNGs, structures vérifiables (arbres de Merkle).

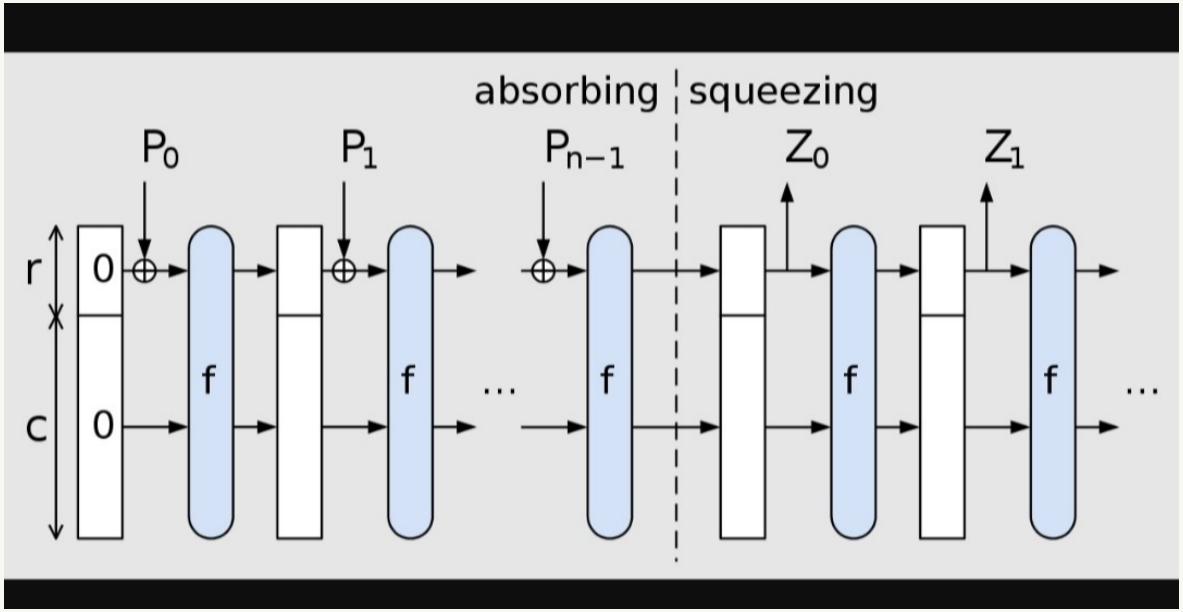
Construction des fonctions de hachage - Extenseur Merkle-Damgård : idée clé

- Définir une primitive opérant sur un domaine de taille fixe puis étendre ce domaine.
- **Extenseur Merkle-Damgard** : construction itérative fondée sur une fonction de compression f : f({0,1}^r × {0,1}^n) → {0,1}^n.
- **Padding standard** (bit `1`, zéros, longueur en fin) pour rendre le message multiple de la taille de bloc.
- **Découpage en blocs Mi** puis itération : V_C0 = IV ; V_Ci = f(V_Ci−1, Mi−1) ; h(M) = V_Ct.
- **Propriété** : sécurité de h hérite de celle de la compression f (mais attention aux attaques structurelles).
- **Faiblesses connues** : extension de longueur, multi-collisions



Extenseur « éponge » : principe

- **État interne** de taille b = r + c (r = rate, c = capacity).
- **Fonction de permutation p** : {0,1}^b → {0,1}^b (bijective, pseudo-aléatoire).
- Deux phases : **absorption** (XOR du bloc sur la partie rate + application de p) puis **squeezing** (**extraction** r bits de la partie rate, éventuellement répéter p pour plus de sortie).
- **Avantages** : sortie de longueur variable, résistance à l'extension de longueur si bien paramétré, conception *proche d'un oracle aléatoire.*



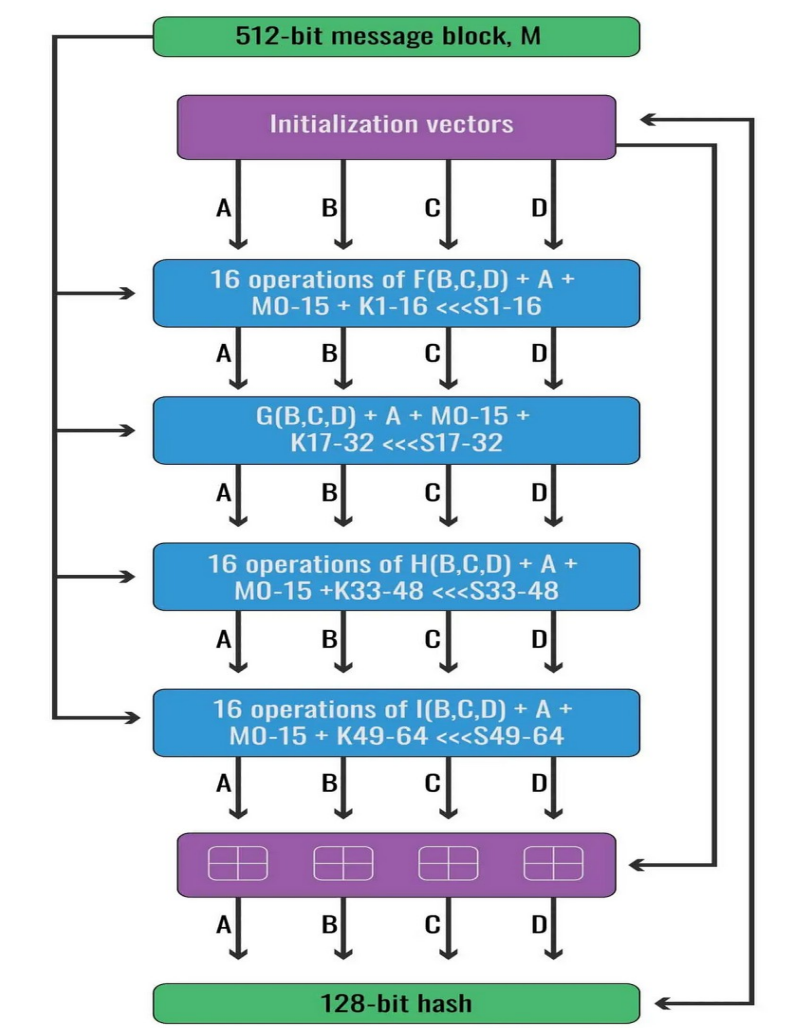
- **capacity c → sécurité** : typiquement la résistance est liée à c/2 pour collisions.
- **rate r → performance** : plus r → plus rapide mais capacity diminue.
- **Utilisée par SHA-3 / Keccak**

MD5 - Construction

- Condensé de 128 bits ; opérant sur mots de 32 bits ; blocs de 512 bits.
- Suit la construction Merkle-Damgård ; table de constantes T[i] basées sur sin(i).
- Opérations : additions modulo 2^32, rotations à gauche, fonctions non linéaires F,G,H,I, 64 itérations (4 tours).

ETAPES

- Padding : `1` + k zéros jusqu'à (L+1+k) ≡ 448 (mod 512), puis longueur L codée sur 64 bits (little-endian).
- Initialisation : A0, B0, C0, D0 (valeurs constantes).
- Pour chaque bloc de 512 bits : extraire 16 mots X0..X15, exécuter 64 itérations (4 tours) : A ← B + ROTL_sj(A + F/G/H/I(...) + X + T[j]) ; permutations (A,B,C,D) ← (D,A,B,C).
- Final : ajout des registres initiaux → concaténation An ‖ Bn ‖ Cn ‖ Dn (little-endian).
- MD5 cassé pour collisions, vulnérable aux attaques chose-prefix et extension de longueur

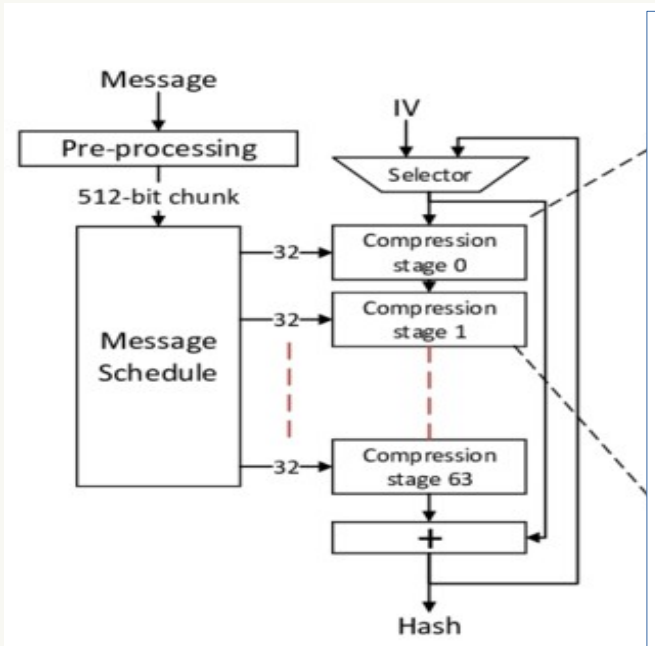


SHA-256 : caractéristiques générales

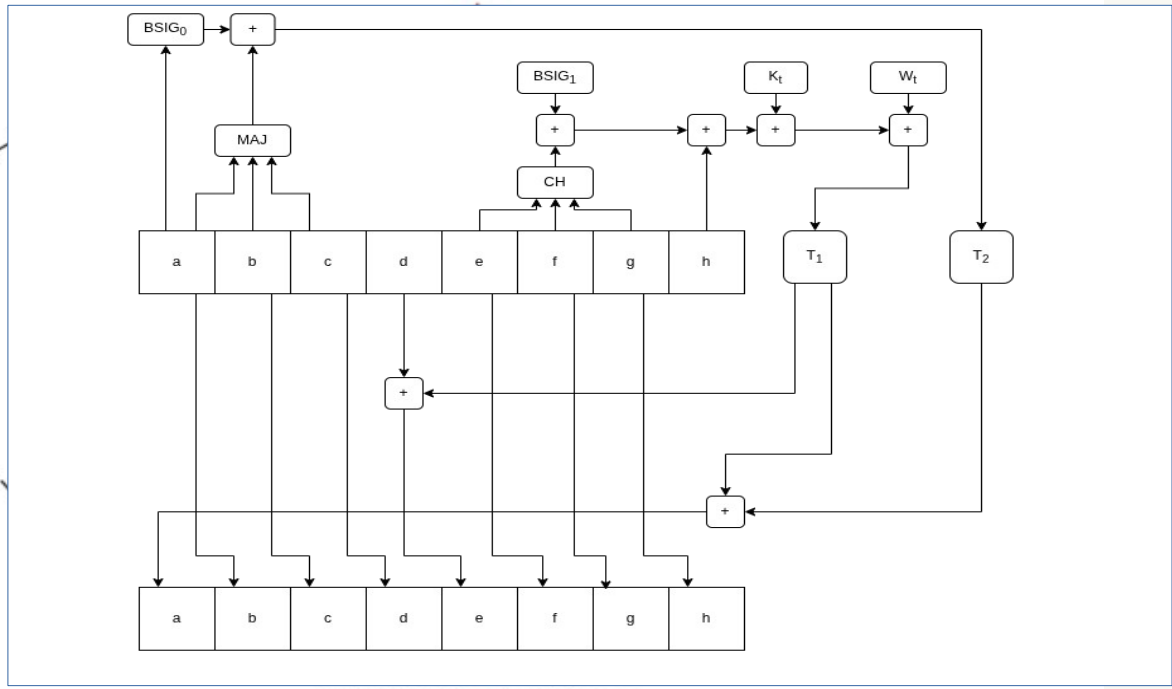
- Membre de la famille SHA-2 ; condensé de 256 bits ; mots de 32 bits ; blocs de 512 bits.
- Construction : Merkle-Damgård ; tableau de 64 constantes K0..K63 (premiers 32 bits des parties fractionnaires des racines cubiques des premiers nombres premiers).
- Opérations clés : CH, MAJ, BSIG0/1 (rotations), SSIG0/1 (petites σ), additions modulo 2^32.
- Padding : bit `1`, K zéros pour atteindre (L+1+K) ≡ 448 (mod 512), puis longueur L en 64 bits big-endian.
- Initialisation : 8 mots VC0[0..7] (constantes issues des racines carrées).
- Pour chaque bloc Mi : construire W0..W63

  Wt = σ1(Wt−2)+Wt−7+σ0(Wt−15)+Wt−16 mod 2^32)

  initier a..h = VC(i−1), puis 64 itérations ; mise à jour des registres et finaliser en ajoutant a..h à VC(i−1).
- Haché final : concaténation des 8 mots VCn[0..7] (big-endian).



(a) SHA-256 Function



(b) Compression Stages