Os algoritmos de hash são fundamentais para segurança da informação, criptografia e integridade de dados. Abaixo estão os principais algoritmos de hash, seguidos de uma explicação sobre **SHAKE**, **cSHAKE**, **TupleHash** e **Argon2**:

**🔐 Principais algoritmos de hash criptográfico**

| **Algoritmo** | **Saída típica** | **Características** |
| --- | --- | --- |
| **MD5** | 128 bits | Muito rápido, mas quebrado (vulnerável a colisões). Não é mais considerado seguro. |
| **SHA-1** | 160 bits | Mais seguro que o MD5, mas também quebrado. Ataques de colisão práticos existem. |
| **SHA-2** | 224, 256, 384, 512 bits | Família segura e amplamente usada (SHA-256, SHA-512). Padrão em muitas aplicações. |
| **SHA-3** | 224, 256, 384, 512 bits | Baseado em Keccak. Segurança moderna e estrutura diferente da SHA-2. |
| **BLAKE2** | 256 ou 512 bits | Muito rápido e seguro. Alternativa moderna à SHA-2/SHA-3. |
| **BLAKE3** | 256 bits | Derivado de BLAKE2. Muito rápido, com suporte a paralelismo e streaming. |
| **RIPEMD-160** | 160 bits | Menos usado hoje. Alternativa ao SHA-1, mas com menor suporte moderno. |

**🔄 SHAKE (SHA-3 Extendable Output Function)**

* **SHAKE128 e SHAKE256** são funções de saída extensível (**XOF**, *extendable-output functions*).
* Permitem gerar **uma saída de qualquer comprimento**.
* Baseadas no mesmo núcleo Keccak que o SHA-3.
* Usadas em aplicações como KDFs (*Key Derivation Functions*) e MACs, onde o tamanho da saída precisa ser flexível.

SHAKE128(M, d) => gera uma saída de d bits

SHAKE256(M, d) => saída maior e mais resistente (para mais segurança)

**🔧 cSHAKE (customizable SHAKE)**

* Uma variante de SHAKE com **personalização por domínio**, útil em contextos onde diferentes aplicações devem evitar colisões entre si.
* Introduz **dois parâmetros adicionais**:
  + N: nome da função (ex: "MAC")
  + S: string de customização (ex: "minha aplicação")

cSHAKE128(M, d, N, S)

* É determinístico e resistente a colisões mesmo com diferentes aplicações ou propósitos.

**🧩 TupleHash**

* Também baseado em Keccak e SHAKE.
* Recebe **tuplas de strings** como entrada, mantendo a separação e a ordem dos elementos.
* Garante que TupleHash(["a", "bc"]) ≠ TupleHash(["ab", "c"]).
* Útil para aplicações onde a **estrutura e os limites dos dados são importantes**, como assinaturas digitais e protocolos de rede.

**🛡️ Argon2**

* Algoritmo de **derivação de chave (KDF)** e **hash de senhas** moderno e seguro.
* Vencedor do concurso [Password Hashing Competition (PHC)](https://password-hashing.net/) em 2015.
* **Projetado para ser resistente a ataques com hardware dedicado** (como GPUs, ASICs).
* Possui três variantes:
  + **Argon2d**: Resistente a ataques com side-channel.
  + **Argon2i**: Resistente a ataques com tempo e acesso à memória.
  + **Argon2id**: Combina as vantagens dos dois, é o recomendado para uso geral.

Parâmetros importantes:

* tempo: número de iterações.
* memória: quantidade de memória usada.
* paralelismo: quantas threads paralelas usar.

**✅ Resumo final**

| **Algoritmo** | **Tipo** | **Uso comum** |
| --- | --- | --- |
| SHA-2 | Hash fixo | Segurança geral, assinaturas digitais, certificados. |
| SHA-3 | Hash fixo | Alternativa mais recente e segura. |
| SHAKE | XOF | Derivação de chaves, autenticação flexível. |
| cSHAKE | XOF personalizado | Evitar colisões entre domínios. |
| TupleHash | XOF estruturado | Hash de dados compostos. |
| Argon2 | KDF | Hash seguro de senhas com resistência a ataques. |

Abaixo apresento **exemplos de uso em Python** para os algoritmos **SHAKE**, **cSHAKE**, **TupleHash** e **Argon2**. Todos utilizam bibliotecas modernas com suporte seguro:

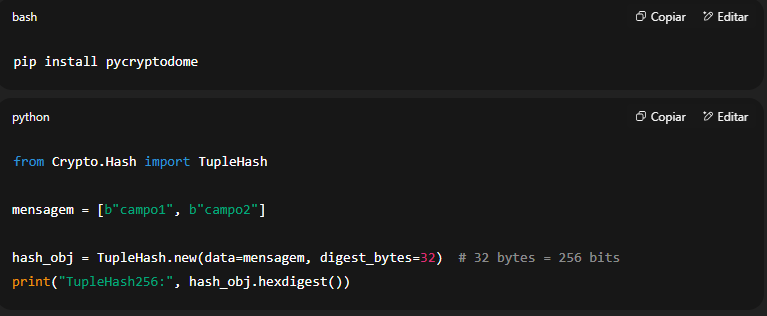
**🐍 1. SHAKE128 / SHAKE256 (Python)**

Usa o módulo hashlib (padrão da biblioteca Python):



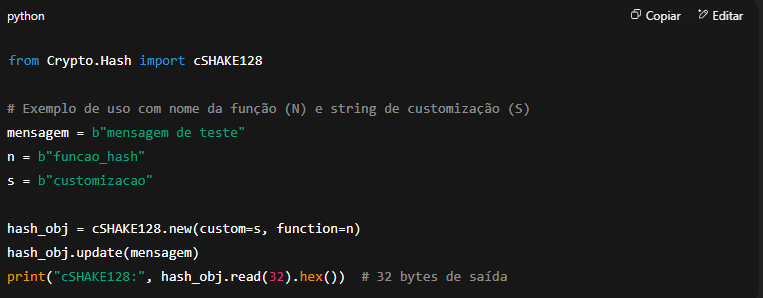
**🧩 2. TupleHash (Python)**

Usa a biblioteca externa pycryptodome com Crypto.Hash.TupleHash:



**🔧 3. cSHAKE128 / cSHAKE256 (Python)**

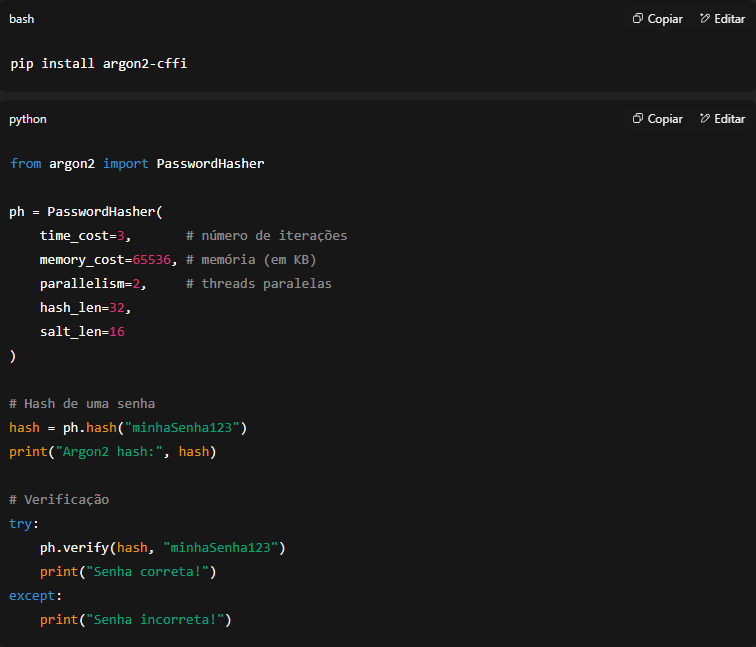
Também pela pycryptodome, módulo Crypto.Hash.cSHAKE128:



Para usar cSHAKE256, basta importar Crypto.Hash.cSHAKE256.

**🔐 4. Argon2 (Python)**

Usa a biblioteca argon2-cffi, amplamente recomendada para segurança real:



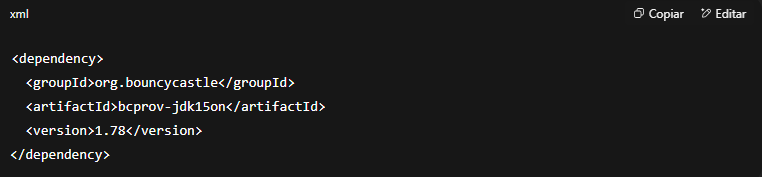
**⚠️ Observações**

* **SHAKE**, **TupleHash** e **cSHAKE** são úteis para estruturas e extensibilidade (por exemplo, protocolos criptográficos).
* **Argon2** é específico para **proteção de senhas**.
* Para aplicações reais, **sempre use bibliotecas auditadas**, como as apresentadas acima.

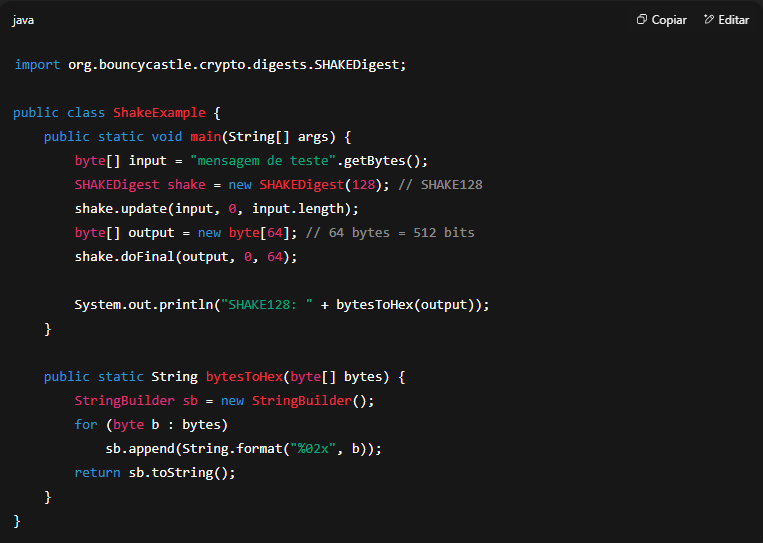
A seguir estão **exemplos em Java** para **SHAKE, cSHAKE, TupleHash** e **Argon2**, utilizando bibliotecas modernas como **Bouncy Castle** e **argon2-jvm**.

**☕ 1. SHAKE128 / SHAKE256 em Java (via Bouncy Castle)**

Primeiro, adicione a dependência do Bouncy Castle ao seu projeto (Maven):



**SHAKE128 com 512 bits de saída:**



**🧩 2. TupleHash (Java – Bouncy Castle)**

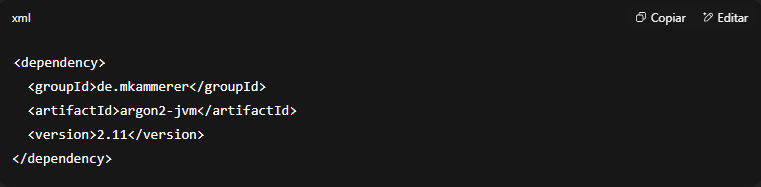


**🔧 3. cSHAKE128 (Java – Bouncy Castle)**



**🔐 4. Argon2 (Java – com de.mkammerer:argon2-jvm)**

Adicione ao pom.xml:



Exemplo de uso:

