

## Exercice Scientifique et technique

### Vaultwarden – Gestionnaire de mot de passe

**BENTEFRIT Mounir**

# SOMMAIRE

---

## PROBLÉMATIQUE

Comment l'architecture cryptographique de Vaultwarden permet-elle de sécuriser les données sensibles des utilisateurs, et comment se compare-t-elle aux solutions open source similaires ?

## PARTIE 1 : Architecture et fonctionnement de Vaultwarden

## PARTIE 2 : Analyse des mécanismes cryptographiques


## PARTIE 3 : Comparaison de Vaultwarden avec KeePass et Psono

## Références



## Vaultwarden

### Qu'est-ce que Vaultwarden ?

- Gestionnaire de mots de passe open-source
- Alternative libre à Bitwarden   
    *↪ Certifié ISO 27001*
- Auto-hébergeable

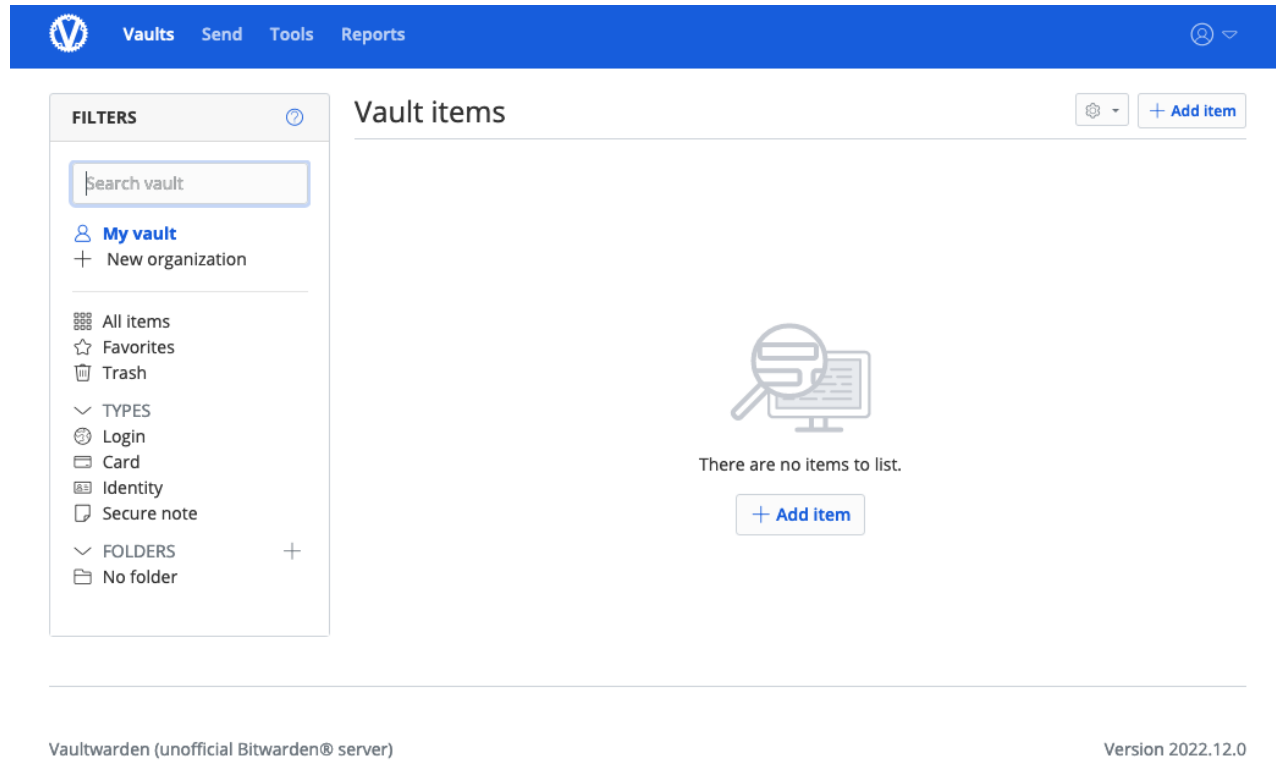
### Pour quel type de public ?

- Particuliers
- Entreprises
- Communautés de self-hosting / homelab

Conçu pour les personnes qui veulent gérer leurs propres données sans dépendre d'un service cloud tiers\*.

*\*Exemples de services cloud : AWS, OVHCloud, Digital Ocean, Google Cloud Platform*

## Vaultwarden



Interface web de Vaultwarden

Que peut-on stocker dans son compte Vaultwarden ?

### Des éléments

- Des identifiants
- Des notes sécurisées
- Des identités → Ex: Nom, prénom, adresse, email
- Des cartes bancaires

### Des dossiers

### Des organisations

- Des membres → Ex: Développeur, Administrateur IT
- Des groupes → Ex: DevOps, Développement, Marketing
- Des collections
- Des « events logs » → Enregistrent toutes les actions réalisées dans l'organisation
  - Connexion d'un utilisateur, modification d'un item, ...
- Des politiques de sécurité

### Sécurité

*Mécanismes qui sont mises en place pour protéger ces données vulnérables :*

- **Chiffrement de bout en bout**
- **Le Privacy By Design (PbD)**
- **Le « zero-knowledge »**
- **Des audits de sécurité par des tiers parties au moins une fois par an**
  - « *The Applied Cryptography Group* » à l'institut de Sécurité de l'Information du Département d'informatique de l'ETH Zurich
  - **Cure53, Fracture Labs, IOActive, Paragon Initiative Enterprises (PIE), Insight Risk Consulting (IRC), Mandiant**
  - **Dernière faille trouvée : Malicious Key Rotation Attack (Medium risk)**

*Comment est-ce que les données sont chiffrées et quels algorithmes sont utilisés derrière ?*

## Chiffrement



Voyons plus en détails les primitives cryptographiques utilisées dans l'architecture de chiffrement de Vaultwarden

Vu au semestre 2 dans le module « Cybersécurité » avec M. Badis

### Dérivation de clé

PBKDF2

Standardisé

Dans la norme PKCS #5 et la RFC 2898

DK = PBKDF2(PRF, Password, Salt, round, dkLen)

Clé dérivée      Fonction pseudo-aléatoire (Ex: HMAC-SHA256)      Nombre d'itérations      Longueur de la clé dérivée

```
// 1. Master password pour déverrouiller le coffre
$password = "MasterPassword123!";
$salt = "mounir.bentefrit@edu.esiee.fr";

/* 2. PBKDF2 : dérivation de la Master Key */
$masterKey = hash_pbkdf2("sha256", $password, $salt, 600000, 32, true);

echo "Clé dérivée PBKDF2 : " . bin2hex($masterKey) . "\n";
```

Clé dérivée PBKDF2 : 4b10a12a2702b76091b1659b44dda794355b35efeb5677133be7d4fccb97cc04

### Expansion de clé

HKDF

Standardisé dans la RFC 5869

Repose sur le principe *Extract-then-Expand*<sup>1</sup>

1 - Phase Extract : PRK=HMAC(salt, IKM)

2 - Phase Expand : OKM=HKDF-Expand(PRK, info, L)

```
/* 3. HKDF : Extension de la master key */
$stretchedKey = hash_hkdf("sha256", $masterKey, 32, "vault-encryption", "");

echo "Clé étendue HKDF : " . bin2hex($stretchedKey) . "\n";
```

Clé étendue de la master key :

55bd61c45e057708aa66b8451e9bb44d49653a355c574ecbdf26f327d6d79788

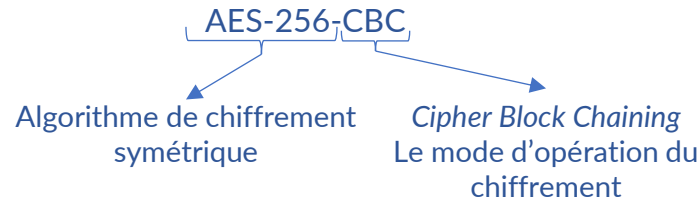
<sup>1</sup>Krawczyk, H., "Cryptographic Extraction and Key Derivation: The HKDF Scheme", IACR ePrint 2010/264

## Chiffrement



### Algorithme de chiffrement

Vu au semestre 2 dans le module « Cybersécurité » avec M. Badis



Standard : FIPS-197

$2^{256}$  possibilités de générations de clé  
 $\approx 1.1579209 \times 10^{77}$  clés

```

/* Chiffrement du mot de passe */
$cipher = openssl_encrypt($data, "aes-256-cbc", $encryptionKey, OPENSSL_RAW_DATA, $iv);
/* Déchiffrement du mot de passe */
$plain = openssl_decrypt($cipher, "aes-256-cbc", $encryptionKey, OPENSSL_RAW_DATA, $iv);

echo "Cipher généré : " . bin2hex($cipher) . "<br><br>";
echo "Mot de passe déchiffré : " . $plain . "<br>";
    
```

Cipher généré : c0b0cd937472c203d0701cf208d1dc1bec4e1bfb4f85672fec0a889999bb0fd7

Mot de passe déchiffré : MotDePasseGoogle

### Intégrité des données

HMAC-SHA256

#### Protocoles utilisant HMAC :

- TLS / SSL -> HMAC-SHA<sub>y</sub> (y = 1 ou 256)
- IPSec -> HMAC-SHA<sub>x</sub>
- JWT (JSON Web Token) -> HMAC-SHA<sub>x</sub> (x = 256 ou 384 ou 512)

```

$mac = hash_hmac("sha256", $iv . $cipher, $macKey, true);
echo "MAC : " . bin2hex($mac);
    
```

MAC : 6aa5c036e9fae257b5a2ddd69d90c23c7c4436e80b315e3369b631559e33cff1

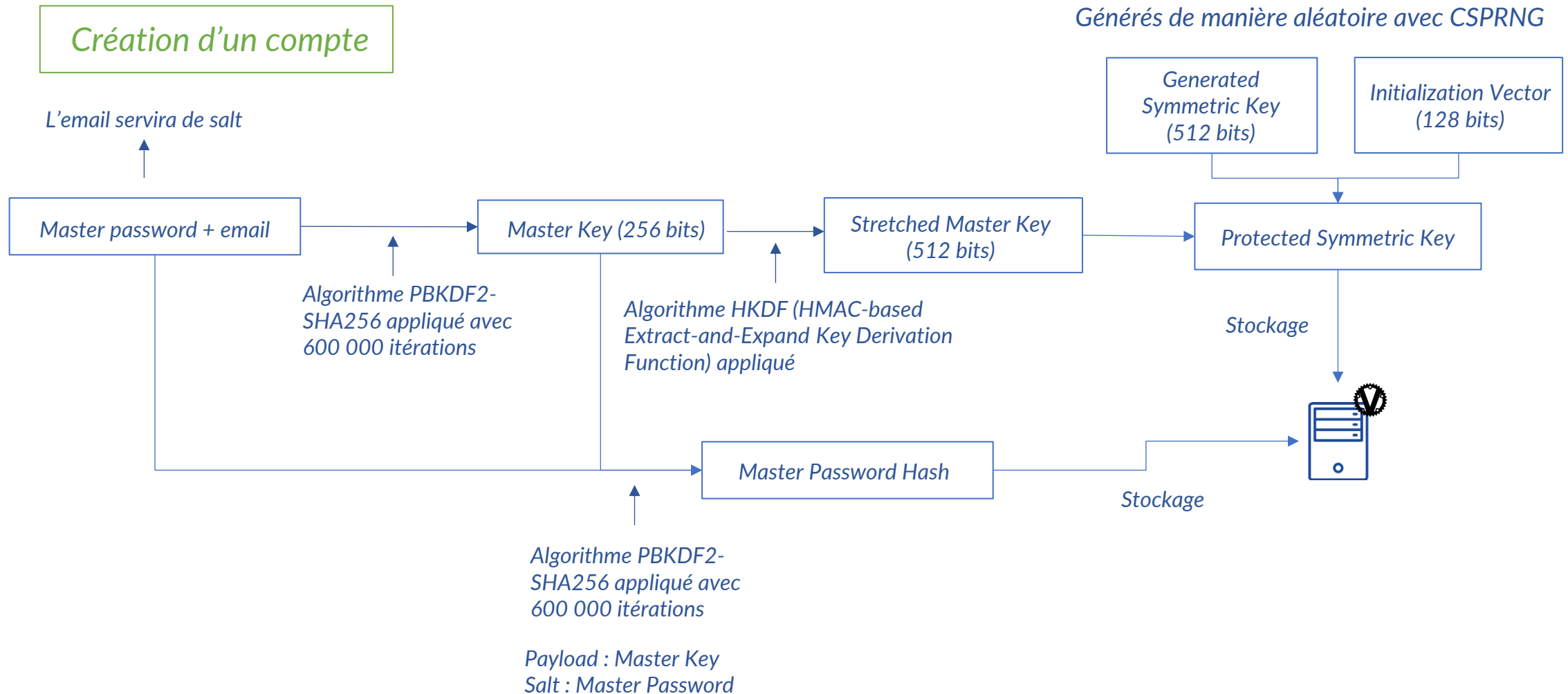


# Chiffrement



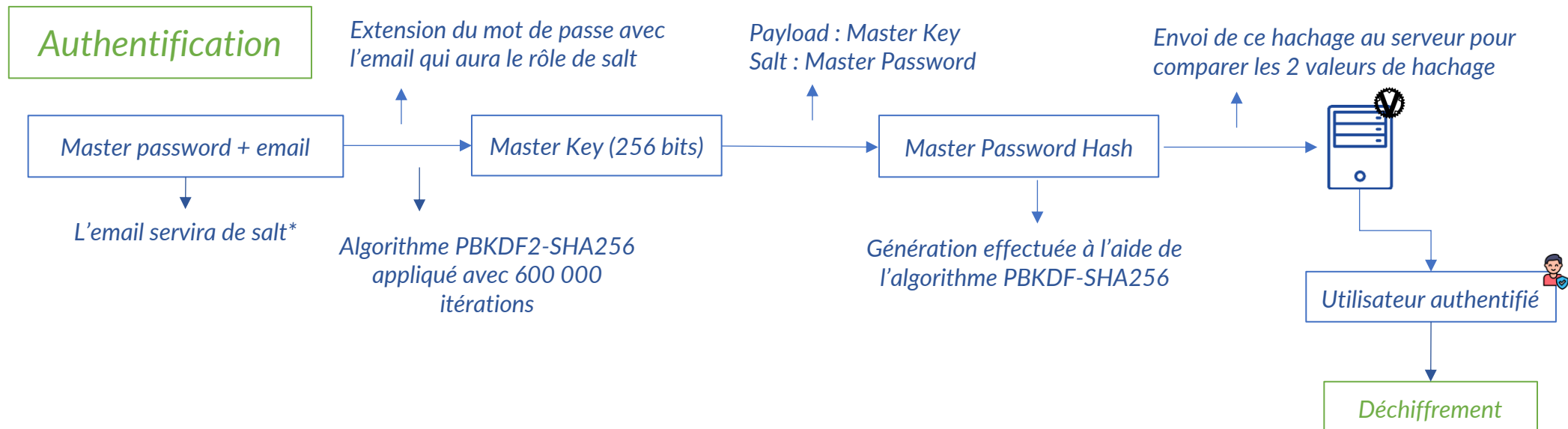
Le chiffrement repose entièrement sur le master password et effectué localement côté client

## Création d'un compte

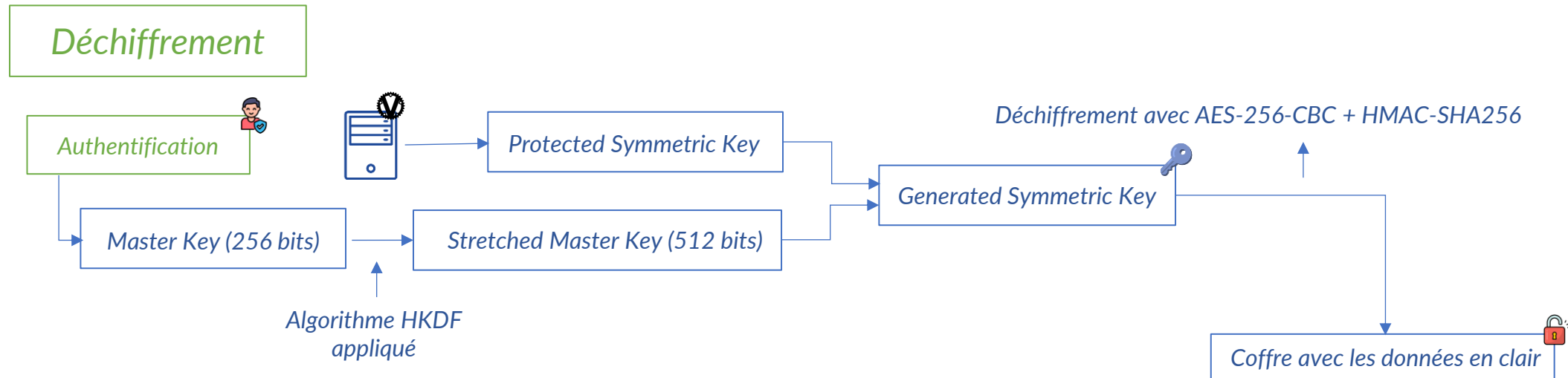




## Chiffrement






## Chiffrement



## Comparaison

Comparons à présent les primitives cryptographiques de Vaultwarden avec celles de deux autres gestionnaires de mot de passe open source

Solution	Architecture	Chiffrement symétrique	Fonction de dérivation	Chiffrement asymétrique	Intégrité
Vaultwarden 	- Architecture client-serveur - Zero-knowledge	AES-256-CBC	PBKDF2-SHA256 Argon2id	RSA	HMAC-SHA256
KeePass 	- Gestionnaire local - Données chiffrées dans un fichier .kdbx	AES-256 Twofish ChaCha20	Argon2 AES-KDF	Inexistant	HMAC-SHA256
Psono 	- Architecture client-serveur	Salsa20 / ChaCha20	Scrypt Argon2	Curve25519	Poly1305

Primitives cryptographiques

**FIN DE LA PRÉSENTATION**

---

**Merci de votre attention**



# Références

## Documentation spécifique à Bitwarden

- Bitwarden Security Whitepaper : <https://bitwarden.com/help/bitwarden-security-white-paper/>
- Bitwarden Cryptography Report (2025) : [https://bitwarden.com/assets/Kki4W785JIPOdFj6EeWB5/dbf51066c1041aa90dc503ca0c911194/2025\\_Bitwarden\\_Cryptography\\_Report.pdf](https://bitwarden.com/assets/Kki4W785JIPOdFj6EeWB5/dbf51066c1041aa90dc503ca0c911194/2025_Bitwarden_Cryptography_Report.pdf)

## Normes et standards de dérivation de clés (KDF)

- RFC 2898 (PBKDF - Password-Based Cryptography Specification) : <https://www.ietf.org/rfc/rfc2898.txt>
- RFC 5869 (HKDF - HMAC-based Extract-and-Expand Key Derivation Function) : <https://www.ietf.org/rfc/rfc5869.txt>
- RFC 7914 (Scrypt - Password-Based Key Derivation Function) : <https://www.ietf.org/rfc/rfc7914.txt>
- RFC 9106 (Argon2 Memory-Hard Function for Password Hashing and Proof-of-Work Applications) : <https://www.ietf.org/rfc/rfc9106.txt>
- Krawczyk, H., "Cryptographic Extraction and Key Derivation: The HKDF Scheme", IACR ePrint 2010/264 : <https://eprint.iacr.org/2010/264>

## Algorithmes de chiffrement et primitives cryptographiques

- Twofish – Schneier B., Kelsey J., Whiting D., Wagner D., Hall C., Ferguson N., Twofish: A 128-Bit Block Cipher, 1998 : <https://www.schneier.com/wp-content/uploads/2016/02/paper-twofish-paper.pdf>
- Salsa20 – Bernstein D. J., The Salsa20 Family of Stream Ciphers, 2008 : <https://cr.yp.to/snuffle/salsafamily-20071225.pdf>
- ChaCha20 – Bernstein D. J., ChaCha, a Variant of Salsa20, 2008 : <https://cr.yp.to/chacha/chacha-20080128.pdf>
- Poly1305 – Bernstein D. J., The Poly1305-AES Message-Authentication Code, 2005 : <https://cr.yp.to/mac/poly1305-20050329.pdf>

## Travaux académiques sur la cryptographie et la sécurité

- Boneh D., *Twenty Years of Attacks on the RSA Cryptosystem* : <https://crypto.stanford.edu/~dabo/papers/RSA-survey.pdf>