Kryptografi

Hashing & Kryptering

Serversideprogrammering III - Skoleassignment

1 Sprint 1: Hashing

1.1 Øvelse 1: Hashing metode kategorier

Kategoriser de 16 hash-metoder i tre kategorier: **Slow**, **Fast**, og **Message Authentication**.

#	Hashing-metode	Slow	Fast	Message Auth
1	bcrypt	X		
2	scrypt	X		
3	Argon2 (i/d/id)	X		
4	PBKDF2	X		
5	~~MD5~~		~~X~~	
6	~~SHA-1~~		~~X~~	
7	SHA-2 (SHA-256/512)		X	
8	SHA-3		X	
9	BLAKE2		X	
10	~~CRC32~~		~~X~~	
11	HMAC-SHA256			X
12	~~HMAC-SHA1~~			~~X~~
13	HMAC-SHA512			X
14	~~HMAC-MD5~~			~~X~~
15	CMAC			X
16	GMAC			X

Markeret med ~~strikethrough~~: Metoder der IKKE længere bør anvendes (obsolete/deprecated)

1.2 Øvelse 2: Hashing metode anvendelsesområder

#	Hashing-metode	Brug
1	bcrypt	Password-hashing i web-applikationer. Adaptiv cost factor gør brute-force langsom.
2	scrypt	Password-hashing med memory-hardness. Bruges i cryptocurrency (Litecoin).
3	Argon2 (i/d/id)	**Bedste valg** til password-hashing (vinder PHC 2015). Argon2id anbefales.
4	PBKDF2	Key derivation fra passwords, Wi-Fi WPA2, disk-kryptering. **Bruges i vores email-hashing.**
5	~~MD5~~	**FORÆLDET** - Kun legacy checksums. ALDRIG til sikkerhed.
6	~~SHA-1~~	**FORÆLDET** - Legacy Git commits. Kollisioner demonstreret 2017.
7	SHA-2	Digital signatures, TLS/SSL, blockchain, fil-integritet. **Industristandard.**
8	SHA-3	Backup til SHA-2, fremtidssikring, høj-sikkerhedsapplikationer.
9	BLAKE2	Hurtig fil-hashing, KDF, erstatning for MD5/SHA-1.
10	~~CRC32~~	**IKKE kryptografisk** - kun checksums for fejldetektering.
11	HMAC-SHA256	API authentication, JWT tokens, fil-integritet. **Bruges i vores app.**
12	~~HMAC-SHA1~~	**FORÆLDET** - Legacy systemer. Brug HMAC-SHA256.
13	HMAC-SHA512	Ekstra sikkerhed hvor længere output ønskes. TLS.
14	~~HMAC-MD5~~	**FORÆLDET** - ALDRIG bruge. MD5 er kryptografisk brudt.
15	CMAC	Block cipher-baseret MAC. AES-CMAC i sikkerhedsprotokoller.
16	GMAC	Galois MAC - del af AES-GCM. TLS 1.3, høj performance.

1.3 Salt vs Pepper

	Salt	Pepper
Formål	Sikrer unikke hashes pr. bruger	Tilføjer ekstra hemmelighed
Område	Unik per bruger	Typisk global for alle passwords
Gemmes?	Ja, i databasen	Nej, gemmes sikkert udenfor (.env)
Beskytter mod	Rainbow tables, duplikat hashes	Database læk uden pepper

1.4 Vores Implementation

Hashing-metoder brugt i projektet:

Metode	Anvendelse	Fil
PBKDF2	Email-hashing med salt + pepper	`lib/hashing.ts`
bcrypt	Password-hashing	`app/actions/auth.ts`
HMAC-SHA256	Fil-integritet verifikation	`lib/hashing.ts`
SHA-256	Generel hashing	`lib/hashing.ts`

Email Hashing (PBKDF2):

```
Input: user@example.com
Salt: Per-user (32 chars, gemt i DB)
Pepper: Global (gemt i .env, ALDRIG i DB)
Iterations: 100,000
Output: 64 char hex string

Formula: PBKDF2(email, salt+pepper, 100000, 32, sha256)
```

Password Hashing (bcrypt):

```
Input: userPassword123
Salt: Auto-generated (included in hash)
Cost Factor: 10
Output: 60 char bcrypt string
```

File Integrity (HMAC-SHA256):

```
Input: File content (Buffer)
Key: Random per-file (IKKE hardcoded)
Output: 64 char hex signature

Verificering: Genberegn HMAC og sammenlign
Match: "No contamination detected"
Mismatch: "Contaminated"
```

2 Sprint 2: Kryptering

2.1 Øvelse 1: Krypterings kategorier

	AES	RSA	Hybrid
Type	Symmetrisk	Asymmetrisk	Hybrid (begge)
Nøgle	Én delt hemmelig nøgle	Nøglepar: public + private	AES-nøgle krypteret med RSA
Hastighed	Meget hurtig	Langsom (1000x)	Hurtig for data, langsom for nøgle
Sikkerhed	Meget sikker (256-bit)	Sikker (2048+ bit)	Kombinerer begge
Brug	Fil-kryptering, VPN, TLS	Signering, nøgleudveksling	TLS/HTTPS, sikker fil-overførsel
Nøglestørrelse	128/192/256 bit	2048/3072/4096 bit	AES: 256, RSA: 2048+
Fordele	Hurtig, hardware-support	Løser nøgledistribution	Bedste fra begge
Ulemper	Nøgledistribution svært	Langsom, små data	Kompleks implementering

2.2 Øvelse 2: Krypterings begreber

#	Begreb	Brug
1	IV	Tilfældig værdi med krypteringsnøgle. Sikrer forskellig ciphertext hver gang. SKAL være unik.
2	Operation modes	Hvordan block cipher håndterer data. ECB (usikker), **CBC** (bruges her), CTR, GCM.
3	Block cipher	Algoritme på faste blokstørrelser (AES: 128-bit). Data opdeles i blokke.
4	IV (CBC)	Første blok XOR'es med IV, efterfølgende med forrige krypterede blok.
5	Nonce	"Number used ONCE" - må ALDRIG genbruges med samme nøgle. Bruges i GCM/CTR.

2.3 Hybrid Encryption Flow

AFSENDER (Third-party app):

1. Hent RSA public key fra server
2. Generer tilfældig AES-nøgle (256 bit)
3. Generer tilfældig IV (16 bytes)
4. Krypter fil med AES-CBC(nøgle, IV)
5. Beregn HMAC-SHA256 af krypteret data
6. Krypter AES-nøgle med RSA public key
7. Send: krypteret_fil + krypteret_nøgle + IV + HMAC

MODTAGER (Vores web app):

1. Verificer HMAC for integritet
2. Dekrypter AES-nøgle med RSA private key
3. Dekrypter fil med AES-CBC(nøgle, IV)
4. Gem dekrypteret fil i Files/uploads
5. Generer ny HMAC for lagret fil
6. Gem metadata i fildatabase

2.4 Implementation Komponenter

Komponent	Fil	Beskrivelse
EncryptionService	`lib/encryption.ts`	AES-256-CBC, RSA-2048, HMAC-SHA256
KeyManager	`lib/keys.ts`	RSA key pair generering og storage
Public Key API	`/api/external/public-key`	Endpoint for third-party
Upload API	`/api/external/upload`	Modtager krypterede filer
Third-Party Sender	`third-party-sender/`	Ekstern app der sender

Test Resultater: Valid Upload: PASS | Tampered HMAC: PASS | Invalid RSA: PASS