Teoria kryptograficzna do projektu:
funkcje skrótu, hasła, sól/pieprz, ataki oraz SRP (PAKE)

21 stycznia 2026

Streszczenie

Dokument stanowi teoretyczne uzupełnienie projektu, w którym porównywane są trzy podejścia do uwierzytelniania: (1) przechowywanie skrótów haseł z użyciem szybkiej funkcji skrótu (np. MD5) wraz z omówieniem kolizji, (2) wzmocnienie przechowywania haseł przez zastosowanie soli (salt) i pieprzu (pepper) oraz analiza odporności na ataki offline, (3) logowanie oparte o SRP (Secure Remote Password) — protokół typu PAKE, w którym hasło nie jest przesyłane do serwera. Opis obejmuje podstawy matematyczne, model zagrożeń oraz praktyczne wnioski.

Spis treści

1 Model zagrożeń (dlaczego w ogóle to robimy)

Bezpieczeństwo haseł zależy od tego, *przed czym* chronimy system. W kontekście projektu kluczowe są dwa scenariusze:

- **Atak online:** napastnik zgaduje hasła poprzez serwer (API/logowanie). Ograniczają go: opóźnienia, limity prób, blokady kont, CAPTCHA, monitoring itp.
- **Atak offline:** napastnik zdobywa bazę (wyciek DB) i testuje hasła lokalnie z maksymalną wydajnością (CPU/GPU/FPGA). To najbardziej niebezpieczny wariant dla przechowywania haseł.

W projekcie pojawiają się: funkcja hashująca, kolizje, sól/pieprz, rainbow tables, brute-force/słownik oraz SRP. Ważne są też pojęcia: **entropia hasła** (ile zgadywań potrzeba) oraz **koszt jednej próby** (ile prób/s).

2 Funkcja skrótu (hash): definicja i intuicja

2.1 Definicja formalna

Kryptograficzna funkcja skrótu to funkcja

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

która mapuje dane dowolnej długości na wynik stałej długości n bitów (np. MD5: $n = 128$, SHA-256: $n = 256$). Jest deterministyczna: to samo wejście $\Rightarrow$ ten sam skrót. Powinna też spełniać efekt lawiny: mała zmiana wejścia powinna powodować „chaotyczną" zmianę wyniku.

2.2 Trzy klasyczne własności bezpieczeństwa

Dla idealnej funkcji skrótu o n bitach:

1. **Odporność na preimage:** dla danego y trudno znaleźć m takie, że $\mathsf{H}(m) = y$. W modelu idealnym koszt $\approx 2^n$ prób (średnio 2^{n-1}).
2. **Odporność na drugi preimage:** mając m trudno znaleźć $m' \neq m$ z $\mathsf{H}(m') = \mathsf{H}(m)$. Idealnie koszt $\approx 2^n$.
3. **Odporność na kolizje:** trudno znaleźć dowolne $m \neq m'$ takie, że $\mathsf{H}(m) = \mathsf{H}(m')$. Idealnie koszt $\approx 2^{n/2}$ (paradoks urodzinowy).

2.3 Paradoks urodzinowy i kolizje

Ponieważ hash ma skończoną liczbę wyników (2^n), kolizje *muszą* istnieć. Dla q losowych prób:

$$\Pr[\text{kolizja}] \approx 1 - e^{-\frac{q(q-1)}{2 \cdot 2^n}}.$$

Granica „urodzinowa" mówi, że kolizji oczekuje się przy $q \approx 2^{n/2}$.

2.4 Jak działa hash „od środka” (konstrukcja iterowana)

Wiele klasycznych hashy (MD5, SHA-1, SHA-256) ma konstrukcję iterowaną (styl Merkle–Damgård): wiadomość jest dopełniana (padding), dzielona na bloki i przetwarzana funkcją kompresji:

$$h_0 = IV, \quad h_{i+1} = f(h_i, m_i), \quad \mathsf{H}(m) = h_t.$$

To wyjaśnia m.in. deterministyczność i stałą długość wyniku.

2.5 Dlaczego MD5 jest złe w kontekście projektu

Dwie rzeczy:

- **Kolizje:** MD5 jest kryptograficznie złamane kolizyjnie (praktycznie da się konstruować kolizje szybciej niż 2^{64}), więc nie nadaje się do mechanizmów, gdzie „różne dane nie mogą dać tego samego skrótu”.
- **Szybkość:** nawet „dobry” szybki hash jest zły do haseł, bo umożliwia gigantyczną liczbę prób offline na sekundę.

3 Hasła: entropia i koszt zgadywania

3.1 Brute-force w modelu idealnym

Jeśli hasło jest równomiernie losowane z przestrzeni rozmiaru N, średnia liczba prób brute-force to $N/2$. Dla tempa prób R (prób/s):

$$T \approx \frac{N}{2R}.$$

Entropię (dla haseł faktycznie losowych) można przybliżać:

$$H \approx \log_2 N.$$

3.2 Słownik i prawdziwe życie

W realnych systemach hasła nie są losowe. Użytkownicy wybierają „łatwe” wzorce, więc ataki słownikowe (wordlist + mutacje) są bardzo skuteczne. To powód, dla którego w projekcie pokazuje się crackowanie przez listę haseł.

3.3 Online vs offline

Atak online jest ograniczony (limity, blokady), a offline jest ograniczony tylko sprzętem atakującego. To dlatego przechowywanie haseł musi być odporne na offline: należy **podnieść koszt pojedynczej próby**.

4 Sól (salt) i pieprz (pepper)

4.1 Sól: definicja i efekt

Sól s to losowa wartość generowana dla każdego użytkownika. Zamiast $\mathsf{H}(P)$ przechowuje się np.:

$$h = \mathsf{H}(P \parallel s).$$

Zalety:

- identyczne hasła różnych użytkowników dają różne hashe,
- utrudnia użycie prekomputowanych tabel (rainbow tables) „raz dla wszystkich”,
- wymusza liczenie ataku per użytkownik/per sól.

4.2 Rainbow tables: time–memory tradeoff (intuicja)

Rainbow tables to kompromis czas-pamięć: zamiast trzymać wszystkie pary (hasło, hash), trzyma się łańcuchy:

$$p_0 \xrightarrow{\mathsf{H}} h_0 \xrightarrow{R} p_1 \xrightarrow{\mathsf{H}} h_1 \xrightarrow{R} \ldots \to p_t,$$

i zapisuje tylko początek i koniec. Sól niszczy ten mechanizm, bo zmienia problem dla każdej soli.

4.3 Pieprz: sekret serwera

Pieprz p to sekret serwera, nieprzechowywany razem z bazą hashy. Przykład:

$$h = \mathsf{H}(P \parallel s \parallel p).$$

Jeśli wycieknie sama baza, ale nie pieprz, atak offline jest utrudniony. Jeśli wycieknie też pieprz, wracamy do klasycznego problemu (szybki hash = szybkie zgadywanie).

4.4 Dodatkowy element (spójny z projektem): HMAC jako „pieprz w standardzie”

Zamiast dopisywania pieprzu „na końcu”, często używa się kluczowanego hasha:

$$h = \mathrm{HMAC}_k(\mathrm{KDF}(P, s)),$$

gdzie k jest sekretem serwera. To uporządkowany wariant pieprzu jako klucza.

5 Dlaczego do haseł używa się KDF, a nie MD5/SHA

5.1 Key stretching i memory hardness

Funkcje do haseł (KDF): **PBKDF2**, **bcrypt**, **scrypt**, **Argon2**. Celem jest, aby każda próba hasła była kosztowna (czasowo i/lub pamięciowo), co podnosi koszt ataku offline. Zarys PBKDF2:

$$\mathrm{DK} = \mathrm{PBKDF2}(\mathrm{PRF}, P, s, c, \ell),$$

gdzie c to liczba iteracji. Argon2id i scrypt są dodatkowo pamięciożerne (utrudnia masowe łamanie na GPU).

5.2 Porównanie podejść

Podejście	Co jest w bazie	Rainbow tables	Atak offline
Szybki hash (MD5)	$\mathsf{H}(P)$	łatwe	bardzo szybki
Sól + szybki hash	$\mathsf{H}(P \parallel s), s$	trudne	nadal szybki
Sól + pieprz + szybki hash	$\mathsf{H}(P \parallel s \parallel p), s$	trudne	zależy od wycieku p
KDF + sól	$\mathrm{KDF}(P, s), s$	trudne	celowo wolny
SRP (PAKE)	verifier v i salt s	n/d	zależy od hasła

6 SRP (Secure Remote Password) — teoria i matematyka

6.1 PAKE: idea

PAKE to protokół, który:

- uwierzytelnia na bazie hasła,
- uzgadnia klucz sesji,
- nie przesyła hasła,
- uniemożliwia podsłuchującemu łatwą weryfikację zgadywań offline *na podstawie samej transmisji.*

6.2 Parametry SRP

SRP działa w grupie modulo dużej liczby pierwszej N z generatorem g. Używa funkcji skrótu H (np. SHA-256) oraz stałej:

$$k = \mathsf{H}(N \parallel g).$$

Bezpieczne są duże parametry (np. 2048 bitów lub więcej).

6.3 Rejestracja (co zapisujemy w DB)

Dla identyfikatora I i hasła P:

$$x = \mathsf{H}(s \parallel \mathsf{H}(I \parallel \text{":"} \parallel P)), \qquad v = g^x \bmod N.$$

Do bazy zapisuje się (I, s, v). Hasło P nie jest przechowywane ani wysyłane.

6.4 Logowanie: wiadomości A i B

Klient losuje a i wysyła:

$$A = g^a \bmod N.$$

Serwer losuje b i odsyła:

$$B = (k \cdot v + g^b) \bmod N,$$

oraz sól s. Następnie obie strony liczą parametr:

$$u = \mathsf{H}(A \parallel B),$$

który wiąże sesję z konkretnymi wartościami A i B.

6.5 Wspólny sekret sesji i dowody M/HAMK

Klient wylicza:

$$S_c = (B - k \cdot g^x)^{(a+u \cdot x)} \bmod N,$$

serwer:

$$S_s = (A \cdot v^u)^b \bmod N.$$

W poprawnym protokole $S_c = S_s = S$, a klucz sesji:

$$K = \mathsf{H}(S).$$

Dowód klienta (M). Klient wysyła wartość M (proof), zależną od (A, B, s, K). Dla SRP-6a często spotyka się:

$$M = \mathsf{H}\Big(\mathsf{H}(N) \oplus \mathsf{H}(g) \ \parallel \ \mathsf{H}(I) \ \parallel \ s \ \parallel \ A \ \parallel \ B \ \parallel \ K\Big).$$

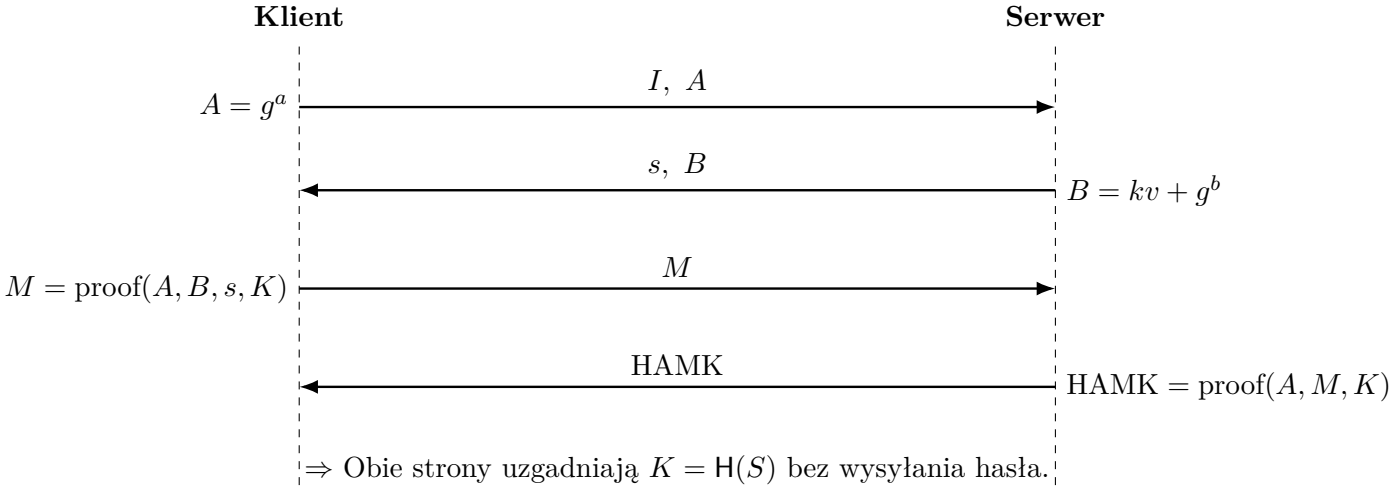
Serwer weryfikuje M używając własnego K.

Dowód serwera (HAMK). Jeśli M jest poprawne, serwer odsyła:

$$\text{HAMK} = \mathsf{H}(A \parallel M \parallel K),$$

co pozwala klientowi zweryfikować, że serwer też zna K (wzajemne uwierzytelnienie).

6.6 Schemat wiadomości SRP



6.7 Co SRP daje (i czego nie daje)

Daje:

- hasło nie jest przesyłane,
- podsłuchujący nie ma prostego „hasha do porównania” z samej transmisji,
- klient weryfikuje serwer przez HAMK.

Nie jest magią: Jeśli napastnik ukradnie z bazy (s, v), może próbować zgadywać hasło offline, licząc v dla kolejnych haseł i porównując. Dlatego nadal ważne są: jakość haseł, polityka haseł, menedżery haseł oraz 2FA/MFA.

7 Dobre praktyki (żeby teoria trzymała się wdrożenia)

- **Rate limiting / blokady** (przeciw online).
- **Stałoczasowe porównania** (przeciw timing attacks).
- **Jednolite komunikaty błędów** (przeciw enumeracji kont).
- **TLS** (ochrona metadanych, cookies, dodatkowa warstwa).
- **MFA/2FA** (zmniejsza skutki wycieków haseł).

8 Podsumowanie spójne z projektem

Projekt można czytać jako „ścieżkę dojrzewania”:

1. **MD5** pokazuje, że szybkie i/ lub kolizyjne funkcje skrótu nie nadają się do haseł.
2. **Sól + pieprz** poprawia sytuację (rainbow tables tracą sens, wyciek bazy mniej groźny bez pieprzu), ale szybki hash nadal pozwala na bardzo szybkie ataki offline.
3. **SRP** przenosi nacisk z „wysyłania/porównywania hasła” na „dowód wiedzy o haśle” i uzgadnia klucz sesji, co eliminuje przesyłanie hasła oraz daje wzajemne potwierdzenie (M/HAMK).