Cryptanalysis of TS-Hash*

Vişan Ionuţ

341C4

1. Intorducere

TS-Hash poate fi descris concis prin pseudocodul C-style următor:

```
word TSHash(bitstring m) {
        word poly[2] = {p0, p1};
        word s = s0;
        for (i = 0; i < m.size(); i++) {
                while ((s & 1) == 0) s >>= 1;
                s = (s >> 1) ^ poly[m[i]];
                }
        return s;
}
```

În esență, funcția de hash se bazează pe un registru de deplasare cu feedback liniar (LFSR), cu polinomul de feedback selectat de bitul mesaj dintre două polinoame publice p0, p1. Bitul mesaj este consumat doar atunci când polinomul de feedback este necesar, adică atunci când LFSR generează valoarea 1, ceea ce creează sursa de neliniaritate.

LFSR (Linear Feedback Shift Register) este un tip de registru de deplasare utilizat în teoria calculatoarelor și în criptografie. Este un element important în proiectarea anumitor funcții de hash, generatori de numere pseudo-aleatoare și alte aplicații în domeniul securității informatice.

2. Formulări ale TS-Hash

2.1. Formulare algebrică a TS-Hash

Reformularea algebrică a schemei presupune tratarea registrului de deplasare cu feedback liniar (LFSR) cu unul dintre polinoame ca înmulțire cu elemental g=X în câmpul finit F2n≈F2[X]/(p0(X)).

Ordinul bitilor este inversat: coeficienții monomilor de cel mai mic grad corespund celor mai semnificativi biți într-un cuvânt de stare. În această reformulare, schimbarea polinomului constă în adăugarea elementului h=p1(x)−p0(X)∈F2n la starea după pas. Comutarea este controlată de bitul mesaj m și un bit în reprezentarea lui g·s, care poate fi întotdeauna exprimat ca Tr(α·s) pentru un anumit α∈F2n, unde

$$\mathsf{Tr} : \mathbb{F}_{2^n} \to \mathbb{F}_2 : z \mapsto \sum_{i=0}^{n-1} z^{2^i}$$

este funcția de urmă în câmpul finit.

Definim functia pasului TS-Hash ca:

$$F : \mathbb{F}_{2^n} \times \mathbb{F}_2 \to \mathbb{F}_{2^n} : (s, m) \mapsto g \cdot s + h \cdot m \cdot \mathsf{Tr}(\alpha \cdot s)$$

Unde g, h, α∈F2n sunt fixe astfel încât Tr(α·g−1·h)=0. Spunem că o stare s este controlată dacă Tr(α·g−1·s)=1.

TS-Hash extins $H(m) : \mathbb{F}_2^N \to \mathbb{F}_2^n$ este definită ca starea finală sN, unde starea inițială $s_0 \in \mathbb{F}_2^n$ este o constantă publică, iar stările intermediare sunt date de

$$s_i = F(s_{i-1}, m_i), \quad 1 \le i \le N.$$

TS-Hash comprimat (original) $\hat{H}(\hat{m}) : \mathbb{F}_2^M \to \mathbb{F}_2^n$ este definită ca sN, unde starea inițială $s_0 \in \mathbb{F}_2^n$ este o constantă publică, j0=1, valorile intermediare pentru 1≤i≤N sunt date de

$$s_i = F(s_{i-1}, \hat{m}_{j_{i-1}}),$$

$$j_i = \begin{cases} j_{i-1}, \\ j_{i-1} + 1, \end{cases}$$ unde, primul caz corespunde lui si cand nu este controlat și al doilea caz pentru si cand este controlat.

Și N este cel mai mic întreg astfel încât jN=M.

2.2. Expresia algebrică explicită a TS-Hash

Folosind formularea algebrică a unui pas TS-Hash, putem acum deriva o expresie algebrică a stării de ieșire în funcție de biții de mesaj extinși.

Fie N un număr întreg pozitiv și fie $s_N = H(m)\ for\ m = (m_1, \ldots, m_N) \in \mathbb{F}_2^N$.

Atunci:

$$s_N = s_0 g^N + \sum_{\substack{I \subseteq \{1,\ldots,N\} \\ I \neq \emptyset}} \underbrace{\left(\prod_{i \in I} m_i\right)}_{monomial\ in\ m} \cdot \underbrace{h g^{N - \max(I)}}_{constant} \cdot \underbrace{\mathsf{Tr}(\alpha s_0 g^{\min(I)-1}) \cdot \prod_{j=2}^{|I|} \mathsf{Tr}(\alpha h g^{i_j - i_{j-1} - 1})}_{control\ bits}$$

Acest rezultat indică o extrem de redusă densitate a formei normale algebrice a stării de ieșire în funcție de biții extinși de mesaj.

3.Atacuri asupra TS-Hash

Un mesaj format doar din biți zero, m=0^N, poate fi folosit ca o preimagine pentru (anumite) valori ale funcției de hash, alegând o valoare potrivită pentru N, ceea ce echivalează cu calculul unui logaritm discret în F2n.

Ideea acestui atac poate fi generalizată pentru a devia de la un mesaj format doar din zero. Dacă setăm mi=1 într-o stare controlată si, obținem o nouă valoare

$$s_i' = s_i + h = s_0 g^i + h,$$

astfel încât noua stare finală devine

$$s_N' = (s_0 g^i + h) g^{N-i} = s_0 g^N + h g^{N-i}.$$

Acest lucru se generalizează natural la mai mulți biți de mesaj setați la 1.

3.1. High-probability differentials

Probabilitatea este luată peste toate valorile posibile ale unei anumite stări intermediare (de exemplu, sN−i pentru un i mic), care se presupune că sunt distribuite uniform peste prefixele de mesaj suficient de lungi.

Pentru orice $\hat{m} \in \mathbb{F}_2^M$ se respect $\hat{H}(\hat{m}) + \hat{H}(\hat{m} + (0, \ldots, 0, 1)) = h$ cu probabilitate 1.

Pentru penultimul bit, situația variază în funcție de h.

Dacă Tr(αh)=0, atunci, pentru orice $\hat{m} \in \mathbb{F}_2^M$ se respectă

$$\hat{H}(\hat{m}) + \hat{H}(\hat{m} + (0, \ldots, 0, 1, 0)) = gh$$

cu probabilitatea ½.

Pentru al treilea bit de la sfârșit, o constrângere suplimentară asupra lui*h*h duce la un rezultat similar:

Dacă Tr(αh)=Tr(αgh)=0, atunci, pentru orice $\hat{m} \in \mathbb{F}_2^M$ se respectă

$$\hat{H}(\hat{m}) + \hat{H}(\hat{m} + (0, \ldots, 0, 1, 0, 0)) = g^2 h$$

cu probabilitatea ¼.

Această idee poate fi ușor generalizată pentru inversări mai îndepărtate de biți și diferențe în rezultate (în funcție de h).

3.2. Preimage attack on short messages

Se presupune existența unui mesaj (comprimat) de lungime ℓ care produce valoarea hash dorită. Este important de menționat că aceasta diferă de atacul de a doua preimagine, unde primul mesaj este dat adversarului. Deoarece lungimea mesajului depinde de reprezentare (comprimată/extinsă), această secțiune funcționează cu ambele reprezentări.

Presupunem că lungimea N a mesajului extins este cunoscută. Atunci, putem reprezenta mulțimea necunoscută de exponenți ei prin coeficienții binari necunoscuți λi ∈ F2:

$$H(m) = s_0 g^N + h \cdot \left(\sum_{i=0}^{N} \lambda_i g^i \right),$$

Aceasta oferă un sistem de ecuații liniare peste F2 cu n ecuații pentru variabilele λi ∈ F2.

Extinderea atacului poate fi combinată cu o căutare exhaustivă a unui prefix (sau sufix) al mesajului. Presupunem că verificăm exhaustiv primii p biți ai mesajului comprimat. Putem apoi aplica atacul algebric liniar asupra mesajului de ℓ−p biți, presupunând o reprezentare extinsă de 2(ℓ−p) biți. Sistemul de rang n furnizează atunci 2^(2ℓ−2p−n) soluții candidat, totalizând 2^(2ℓ−p−n) soluții când se iau în considerare cele 2^p prefixe ghicite.

Atacul se aplică natural la preimaginile mesajelor mai lungi cu un prefix și/sau sufix cunoscut, astfel încât partea din mijloc necunoscută să fie de lungime mică.

3.3. Generalized birthday general preimage attack

Examinăm atacul general de preimagine pentru valori arbitrare ale digestiilor hash. Această analiză se referă exclusiv la reprezentarea extinsă. Se reamintește ecuația polinomială pentru TS-Hash:

$$H(m) = s_0 g^N + h(g^{N-e_1} + g^{N-e_1-e_2} + g^{N-e_1-e_2-e_3} + \ldots + g^{N-e_1-\ldots-e_k}),$$

care se aplică atunci când sterile si sunt controlate. Deși nu putem asigura că stările alese sunt controlate, putem spera că acest lucru se întâmplă aleator. Fiecare bit este controlat cu o probabilitate de 1/2, astfel încât pentru k monoame avem nevoie de aproximativ 2^k încercări "aleatoare" pentru a reuși.

Problema este reprezentată acum de obținerea polinomului (H(m)−s0 *g^N)/h sub forma unui polinom foarte rar în g, cu coeficienți binari, unde gradul polinomului trebuie să fie și el limitat.

Avându-se g,t∈F2n, găsiți un polinom q(x)∈F2[x] de grad redus și cu puțini coeficienți nenuli, astfel încât q(g)=t.

Observați că un polinom dat poate fi testat eficient (pentru a avea toți biții necesari controlați) prin calculul doar al stărilor relevante: un astfel de salt necesită calcularea unei puteri a lui g folosind exponențiere rapidă. Acest lucru este în contrast cu evaluarea directă a tuturor celor N stări ale TS-Hash, care poate fi lentă pentru mesaje lungi (ce pot avea un grad posibil ridicat al polinomului q).

Se consideră lista L0={g^0,g^1,…,g^(2^(t+1)−1)}. Utilizând o tabelă hash pe cele mai puțin semnificative t biți (să zicem), putem găsi toate perechile (i, j) astfel încât gi+gj să aibă t cei mai puțin semnificativi biți egali cu zero.

Chiar dacă obținem un polinom care se evaluează la zero la g, această abordare poate fi ușor adaptată pentru a face suma să fie egală cu orice valoare prestabilită în loc de zero.

Utilizrea unui t=0 permite găsirea unei coliziuni pentru funcția TS-Hash. Această problemă este echivalentă cu găsirea unui multiplu cu greutate redusă al polinomului minimal al lui g, care este p0(X).

3.4. Prefix/suffix linearization technique

Vom încerca acum să linearizăm funcția TS-hash folosind teorema inițială:

$$s_N = s_0 g^N + \sum_{\substack{I \subseteq \{1,\dots,N\} \\ I \neq \emptyset}} \underbrace{\left(\prod_{i \in I} m_i\right)}_{\text{monomial in } m} \cdot \underbrace{hg^{N-\max(I)}}_{\text{constant}} \cdot \underbrace{\mathsf{Tr}(\alpha s_0 g^{\min(I)-1}) \cdot \prod_{j=2}^{|I|} \mathsf{Tr}(\alpha h g^{i_j - i_{j-1}-1})}_{\text{control bit}}.$$

Scopul este, pentru monoamele m^I cu |I|≥2, să forțăm biții de control corespunzători să fie egali cu zero. Acest lucru poate fi realizat prin fixarea anumitor biți de mesaj la zero. Apoi, sN va deveni o funcție liniară a celorlalți biți de mesaj (cei ne-fixați). Bitul de control total pentru fiecare monom constă din două părți.

Să luăm în considerare un monom de grad cel puțin 2. Bitul său de control este egal cu zero în special atunci cand

$$\prod_{j=2}^{|I|} \mathsf{Tr}(\alpha h g^{i_j - i_{j-1}-1}) = 0$$

Prin urmare, putem alege pași între variabile pentru a asigura această condiție.

A doua contribuție la bitul de control este termenul

$$\mathsf{Tr}(\alpha s_0 g^{\min(I)-1})$$

Depinde doar de cel mai mic index al unei variabile implicate în monom. Deoarece ne propunem să permitem (și să impunem) doar monoame liniare, acest termen trebuie să fie egal cu 1. În caz contrar, biții activi aleși ar deveni pur și simplu inactivi. Pentru a menține toți cei k biți activi aleși, trebuie să fie adevărat:

$$\mathsf{Tr}(\alpha s_0 g^{\ell+is-1}) = 1 \quad \forall i\{0,\ldots,k-1\}$$

3.5. Linearization with quadratic constraints

Abordarea directă constă în simpla rezolvare a sistemului de constrângeri pătratice și ecuații liniare rezultate din egalarea lui sM cu valoarea țintă a digestiei (digest value). Structura simplă a sistemului de constrângeri poate potențial să conducă la metode eficiente.

O tehnică destul de directă este de a genera o soluție aleatoare pentru partea liniară a sistemului (care este subdefinit pentru M>n) și să sperăm că vom satisface constrângerile pătratice întâmplător. De fapt, fiecare constrângere pătratică poate fi așteptată să fie îndeplinită cu o probabilitate de 3/4 datorită formei sale, ceea ce duce la o probabilitate de (3/4)^{M-1} ca soluția parțială să fie corectă. Deci, este mai eficient să alegeți M=n și să alegeți aleator preimaginea țintă făcând pași inversi aleatorii de la ținta originală.

4.Discuție

TS-Hash este o instanță a așa-numitelor funcții de dispersie Cayley.

Dat fiind bijecțiile liniare

$$T_0, T_1 : \mathbb{F}_2^n \to \mathbb{F}_2^n,$$

o bijecție neliniară

$$S : \mathbb{F}_2^n \to \mathbb{F}_2^n$$

și starea inițială

$$s_0 \in \mathbb{F}_2^n,$$

o funcție de dispersie h poate fi definită astfel încât pentru un mesaj

$$m = (m_1, m_2, \ldots, m_M) \in \mathbb{F}_2^M,$$

aceasta este dată de:

$$h(m) = T_{m_M} \circ S \circ \ldots \circ T_{m_2} \circ S \circ T_{m_1} \circ S(s_0).$$

În particular, TS-Hash este dată de funcția "shift" astfel:

$$S(s||0) = S(0||s), S(s||1) = (s||1).$$

Iar T0,T1sunt înmulțiri cu

$$X \text{ in } \mathbb{F}_2[X]/(p_0(X))$$

Și

$$\mathbb{F}_2[X]/(p_1(X)).$$

Această structură poate fi în principiu sigură, deoarece funcțiile moderne de dispersie pot fi privite ca un exemplu al acestei structuri (cu injectarea de mesaje de un singur bit). Cu toate acestea, acest lucru necesită o hartă de amestecare neliniară S suficient de puternică, ceea ce nu este cazul pentru TS-Hash. În plus, există multă interacțiune între funcțiile S,T0,T1: funcțiile S și T0"aproape" comută, iar T1 este egală cu T0 cu o adiție constantă.

5. Contramăsuri

Trunchierea ieșirii

Funcțiile de dispersie tipice returnează doar o parte din starea finală ("stoarcerea"). Trunchierea funcției TS-Hash ar putea duce la creșterea securității împotriva unora dintre atacuri, pentru o lungime fixă a ieșirii: atacul meet-in-the-middle necesită o coliziune pe întreaga stare, astfel încât complexitatea sa O(2^(n/2)) pentru o stare de n biți poate fi mai rău decât o căutare generică exhaustivă a preimaginii pentru o dimensiune de ieșire mai mică decât n/2. Cu toate acestea, aceasta nu ar afecta atacul de mesaj scurt și atacul de aniversare generalizată.

Completarea externă a mesajului

O idee naturală pentru protejarea funcției hash împotriva atacurilor la mesaje scurte este să o umpleți cu biți ficși. Cu toate acestea, o umplere cu prefix nu adauga nici o securitate, deoarece ajustează doar starea de pornire. O umplere cu sufix este la fel de inutilă în sine, dar poate fi mai utilă în combinație cu trunchierea.

Completarea internă a mesajului

O altă contramăsură care vizează creșterea confuziei prin absorbția biților de mesaj este să se injecteze o umplere internă (biți constanți) între biții consecutivi ai mesajului. Similar umplerii externe, biții alternanți 0-1 sunt o alegere bună. Cu alte cuvinte, înlocuiți funcția de pas Tmi◦ S cu

$$P_{\ell'} \circ T_{m_i} \circ S = (T_1 \circ S \circ T_0 \circ S) \circ (T_1 \circ S \circ T_0 \circ S) \circ \ldots \circ T_{m_i} \circ S$$

6. Avantaje si Dezavantaje ale TS-Hash

Avantaje:

Eficiență la nivel hardware: TS-Hash este proiectată să fie o funcție hash ușoară, ceea ce înseamnă că poate fi implementată eficient la nivel hardware pe dispozitive cu resurse limitate. Aceasta o face potrivită pentru utilizarea în sisteme încorporate sau în dispozitive cu putere redusă.

Simplicitatea implementării: Structura simplă a TS-Hash, bazată pe un registru de deplasare cu feedback liniar (LFSR), face ca implementarea să fie relativ simplă și ușor de înțeles. Aceasta poate duce la o dezvoltare rapidă și la o mai mică probabilitate de introducere a erorilor în implementare.

Rezistență la anumite clase de atacuri: Chiar dacă prezentarea a evidențiat potențiale vulnerabilități ale TS-Hash, este important să menționăm că funcțiile de hash pot rezista unui set specific de atacuri. TS-Hash are încă valori teoretice și de proiectare care pot fi apreciate în anumite contexte de utilizare.

Dezavantaje:

Vulnerabilitate la anumite atacuri criptanalitice: Conform analizei propuse, TS-Hash pare să fie vulnerabil la anumite atacuri, cum ar fi atacul de preimagine pentru mesaje scurte și diferențele cu probabilitate ridicată. Aceste vulnerabilități ar putea pune în pericol securitatea funcției de hash în anumite scenarii.

Complexitatea în implementarea contra-măsurilor: Contramăsurile propuse pentru a face față la potențialele vulnerabilități, cum ar fi trunchierea sau umplerea mesajelor, pot adăuga complexitate la implementare. Gestionarea cu atenție a acestor contramăsuri este esențială pentru a menține securitatea funcției de hash.

7. Implementare

```python
def ts_hash(message, p0, p1, s0):
    def lfsr_update(s, m, poly):
        """
        Actualizează starea LFSR în funcție de un bit de mesaj și un set de polinoame de feedback.

        Parameters:
        - s: Starea curentă a LFSR.
        - m: Bitul de mesaj (0 sau 1) pentru a actualiza starea LFSR.
        - poly: Lista cu două polinoame de feedback.

        Returns:
        - Starea actualizată a LFSR.
        """
        # Se execută shift la dreapta până când se ajunge la un bit setat
        while (s & 1) == 0:
            s >>= 1
        # Se actualizează starea LFSR folosind polinomul de feedback corespunzător bitului de mesaj
        return (s >> 1) ^ poly[m]

    # Se inițializează polinoamele de feedback
    poly = [p0, p1]
    # Se inițializează starea LFSR
    s = s0

    # Se parcurge fiecare bit din mesaj pentru a actualiza starea LFSR
    for bit in message:
        s = lfsr_update(s, bit, poly)

    # Se returnează rezultatul final al funcției hash
    return s
```

```
# Exemplu de utilizare
p0 = 0b1011
p1 = 0b1101
s0 = 0b0101
message = [1, 0, 1, 0]  # Exemplu de mesaj

hash_result = ts_hash(message, p0, p1, s0)
print(f'TS-Hash pentru mesajul {message}: {hash_result}')
```

```
TS-Hash pentru mesajul [1, 0, 1, 0]: 10
```

Implementarea constă într-o funcție ts_hash care aplică algoritmul TS-Hash pe un mesaj dat folosind un registru cu decalaj cu feedback liniar (LFSR). Funcția acceptă polinoamele de feedback (p0, p1), starea inițială a LFSR (s0) și un mesaj sub formă de șir binar (message). Rezultatul este starea finală a LFSR, reprezentând rezultatul funcției TS-Hash. Există, de asemenea, o funcție internă lfsr_update care realizează actualizarea LFSR pentru un singur pas. Un exemplu de utilizare este furnizat pentru claritate.

Definirea Funcției Interne lfsr_update

Funcția primește trei parametri: starea curentă a LFSR (s), un bit de mesaj (m), și o listă cu două polinoame de feedback (poly).

Execută un shift la dreapta în starea LFSR până când ajunge la un bit setat.

Actualizează starea LFSR folosind polinomul de feedback corespunzător bitului de mesaj.

Returnează starea actualizată a LFSR.

Inițializarea Variabilelor

Se inițializează variabila poly cu cele două polinoame de feedback p0 și p1.

Se inițializează variabila s cu starea inițială a LFSR s0.

Parcurgerea Bit cu Bit a Mesajului

Se parcurge fiecare bit din mesajul de intrare message.

La fiecare pas, se apelează funcția lfsr_update pentru a actualiza starea LFSR.

Returnarea Rezultatului Final

După ce s-au parcurs toți biții din mesaj, rezultatul final al funcției hash este starea finală a LFSR.

Exemplu de Utilizare

Se definește un exemplu de mesaj (message), polinoamele de feedback (p0 și p1), și starea inițială a LFSR (s0).

Se apelează funcția ts_hash cu aceste parametri.

Rezultatul final, care reprezintă valoarea de hash pentru mesajul dat, este afișat.

Pentru mesajul [1, 0, 1, 0]:

Inițial, s = 0b0101, poly = [0b1011, 0b1101].

La primul bit, se face un shift la dreapta, apoi se actualizează starea LFSR folosind polinomul p1.

La al doilea bit, se face un shift la dreapta, apoi se actualizează starea LFSR folosind polinomul p0.

Procesul se repetă pentru toți biții mesajului.Rezultatul final este starea LFSR după ce s-au procesat toți biții din mesaj.

8. Alte metode folosite

SHA-256 (Secure Hash Algorithm 256-bit):

Descriere: Face parte din familia SHA-2 și este larg utilizată datorită proprietăților sale puternice de securitate. Produce o valoare hash de 256 de biți (32 de octeți), în mod obișnuit reprezentată sub formă de număr hexadecimal.

Aplicații: Folosită în diverse aplicații și protocoale de securitate, inclusiv TLS, PGP, SSH, Bitcoin și altele.

SHA-3 (Secure Hash Algorithm 3):

Descriere: Ultimul membru al familiei Secure Hash Algorithm, proiectat să ofere o alternativă la SHA-2. Folosește o structură internă diferită cunoscută sub numele de construcția Keccak sponge.

Aplicații: Furnizează o opțiune de rezervă și alternativă la SHA-2, utilizată în situații în care diversitatea în algoritmi criptografici este dorită.

BLAKE2:

Descriere: Funcție de hash criptografic optimizată pentru viteză și simplitate. BLAKE2 este adesea mai rapidă decât multe funcții de hash existente, inclusiv MD5, SHA-1, SHA-2 și SHA-3, menținând în același timp proprietăți puternice de securitate.

Aplicații: Utilizată în diverse aplicații și protocoale unde eficiența este crucială, cum ar fi gestionarea pachetelor software, sistemele de control al versiunilor și altele.

Skein:

Descriere: Unul dintre finaliștii în competiția funcțiilor de hash NIST. Skein este proiectată să fie rapidă, sigură și flexibilă, permițând utilizatorilor să aleagă dimensiunea hash-ului și să ajusteze parametrii.

Aplicații: Deși nu este adoptată la scară largă ca unele alte funcții de hash, Skein a fost luată în considerare în aplicații în care flexibilitatea și proprietățile sale de securitate sunt avantajoase.

Whirlpool:

Descriere: Whirlpool este o funcție de hash criptografică cu o dimensiune a rezultatului mai mare (512 biți) în comparație cu familia SHA-2. Folosește o construcție bazată pe cifru de bloc.

Aplicații: Utilizată în unele protocoale și aplicații de securitate care beneficiază de dimensiunea sa mai mare a rezultatului hash.

Keccak:

Descriere: Keccak este funcția de hash subiacentă aleasă pentru SHA-3. Folosește o construcție de tip burete și este cunoscută pentru simplitatea și flexibilitatea sa.

Aplicații: SHA-3, bazată pe Keccak, este utilizată în scenarii în care este benefic să existe o funcție de hash distinctă față de SHA-2, furnizând diversitate în primitivele criptografice.

9. TS-Hash vs. SHA-256

Eficiența la Resurse Limitate:

TS-Hash: Conceput pentru eficiență în implementări cu resurse limitate, cum ar fi dispozitivele IoT sau cele cu capacități reduse de calcul.

SHA-256: Poate fi intensivă din punct de vedere computațional, ceea ce o poate face mai puțin potrivită pentru dispozitive cu resurse limitate.

Lărgimea Hash-ului:

TS-Hash: Nu este standardizat și are o lărgime de hash specifică implementării.

SHA-256: Oferă un hash de 256 biți, oferind un spațiu de adresare mult mai mare pentru valori hash unice.

Recunoaștere și Adaptare:

TS-Hash: Poate să nu beneficieze de aceeași recunoaștere și adopție ca algoritmii standard, cum ar fi SHA-256.

SHA-256: Este o funcție de hash larg acceptată, cu o istorie lungă de utilizare în aplicații critice pentru securitate.

Rezistența la Atacuri Criptografice:

TS-Hash: Necunoscut în ce măsură rezistă la diverse tipuri de atacuri, întrucât depinde de implementarea specifică și de parametrii aleși.

SHA-256: Este bine studiată și cunoscută pentru rezistența sa la o gamă largă de atacuri criptografice, cum ar fi coliziunile și preimaginea.

10. TS-Hash vs. BLAKE2

Eficiența la Resurse Limitate:

TS-Hash: Optimizat pentru implementări cu resurse limitate, precum dispozitivele IoT sau cele cu capacități reduse.

BLAKE2: De asemenea, eficient din punct de vedere al resurselor, dar poate avea un avantaj în performanță datorită optimizărilor.

Lărgimea Hash-ului:

TS-Hash: Are o lărgime de hash specifică implementării și nu este standardizat.

BLAKE2: Furnizează hash-uri cu o lărgime variabilă, inclusiv variante cu lărgimi precum BLAKE2b (512 biți) și BLAKE2s (256 biți).

Recunoaștere și Adaptare:

TS-Hash: Poate să nu beneficieze de aceeași recunoaștere ca algoritmii standard.

BLAKE2: A fost adoptat în diverse proiecte și biblioteci datorită vitezei sale și a securității bune.

Rezistența la Atacuri Criptografice:

TS-Hash: Nivelul său de rezistență la atacuri poate varia și depinde de implementare.

BLAKE2: Considerat sigur și rezistent la o serie de atacuri, având în vedere că face parte din competiții și standarde de securitate.

Simplitatea Implementării:

TS-Hash: Concepția sa simplificată poate facilita implementarea și înțelegerea.

BLAKE2: Este optimizat pentru simplitate și poate fi mai ușor de implementat și integrat în anumite scenarii.

11. Concluzie

TS-Hash este o funcție de dispersie ușoară, bazată pe un registru de decalare cu feedback liniar (LFSR). Algoritmul este proiectat să fie eficient din punct de vedere al resurselor și să ofere o performanță ridicată în implementările cu restricții, cum ar fi cele pe dispozitive cu resurse limitate.

TS-Hash se remarcă prin simplitatea și eficiența sa, fiind ușor de implementat pe dispozitive cu resurse limitate.

În implementări practice, TS-Hash are potențialul de a furniza o dispersie rapidă a datelor, ceea ce poate fi crucial în anumite aplicații.

Analizele criptografice prezentate evidențiază diverse vulnerabilități ale TS-Hash, inclusiv atacuri de preimagine și diferențiale cu probabilitate mare.

Algoritmul TS-Hash pare să fie mai susceptibil la anumite tipuri de atacuri, în special pentru mesaje scurte.

TS-Hash este o opțiune atractivă pentru scenarii cu resurse limitate, datorită simplității și eficienței sale. Cu toate acestea, este important să fie luate în considerare vulnerabilitățile sale criptografice semnalate. Utilizarea acestui algoritm poate fi justificată în anumite contexte, dar implementatorii trebuie să fie conștienți de riscuri și să ia măsuri suplimentare de securitate, sau să considere alternative mai robuste în funcție de cerințele specifice ale aplicației.

12. Referinte

1. Laila El Aimani. A new approach for finding low-weight polynomial multiples. IACR Cryptol. ePrint Arch., 2021:586, 2021.

2. Itay Bookstein and Boaz Tsaban. TS-Hash: a lightweight cryptographic hash family based on Galois LFSRs. Cryptology ePrint Archive, Paper 2023/179, 2023. https://eprint.iacr.org/2023/179.

3. Itay Bookstein and Boaz Tsaban. TS-Hash: a lightweight cryptographic hash family based on Galois LFSRs. Mathematical Cryptology, 3(1):96–106, Aug. 2023.