digital whisper
Technologic Papers

מלח, פלפל ותבלינים נוספים – מבוא לפונקציות HASH

מאת בר טופליאן

הקדמה

מעטים הקורסים במסגרת תואר ראשון במדעי המחשב שלא מתארים/מציינים/זורקים לחלל האוויר את המושג 'פונקציות Hash'. נדמה שלכל אחד יש שם אחר לתאר את אותו העקרון: אחד ישתמש במילה 'תמצות', השני יעדיף 'גיבוב', השלישי 'ערבול' וכן הלאה. מרוב שמות נהיה "סלט" וכמו כותרת המאמר, אי אפשר לשכוח בסלט טוב מלח ופלפל.

למרות שקל מאוד להתפתות ולהתייחס לפונקציה זו מכאן והלאה בשם האלטרנטיבי, 'פונקציית טחינה'. ובכך להמשיך את הנרטיב, נרצה להמשיך את הכיוון הכללי בו השתמש ד"ר אור דונקלמן במאמרו[1] ("פונקציות תמצות קריפטוגרפיות ותחרות ה-SHA3") ומעתה נשתמש בשם 'פונקציית תמצות'.

פונקציות תמצות

בצורה הגסה ביותר, פונקציית תמצות היא לא אחרת מאשר מכונת קלט-פלט המקבלת קלט באורך חופשי וממירה אותו לפלט באורך קבוע. על גבי פונקציה זו ניתן לבנות עולם ומלואו, וכמעט בלתי אפשרי למצוא מערכת כלשהי שאינה מממשת עקרון זה לפחות פעם אחת. מבני נתונים, אלגוריתמים, חיפושים, הצפנות, חתימות ועוד רבים מהמונחים שנשמעים באוזני כל סטודנט לתואר ראשון במדעי המחשב, עושים שימוש בפונקציות תמצות, גם כשהדבר נעשה מאחורי הקלעים.
נתאר דוגמה לפונקציה (לא ממולצת לשימוש) שכזו, נגדיר מכונת קלט-פלט המקבלת מחרוזת כלשהי המורכבת ממילים ומחזירה את מספר האותיות במחרוזת (כולל רווחים):

$$f: string \rightarrow number$$

קל לראות את אופי הפונקציה, למשל:

$$f(Shadows\ on\ the\ hills\ sketch\ the\ trees\ and\ the\ daffodills) = 56$$

מצד שני מתקיים:

$$f(For\ they\ could\ not\ love\ you\ but\ still\ your\ love\ was\ true) = 56$$

אמנם שני המשפטים שייכים לאותו שיר ("Vincent", Don McLean) אך המשפטים אינם זהים. כל לוגיקה שתבנה על ידי שימוש בפונקציה זו תקצה לאותם המשפטים את אותו ערך תמצות. וכפי שנתאר בהמשך, התנהגות זו אינה רצויה בהיבט קריפטוגרפי.

פונקציות תמצות קריפטוגרפיות

פונקציות אלו, בדומה לפונקציות התמצות הרגילות (שאינן קריפטוגרפיות), מקיימות את אותה הבטחה של קבלת קלט באורך חופשי והמרתו לפלט באורך קבוע, אך בליבן הן יקיימו שלוש דרישות בסיסיות:

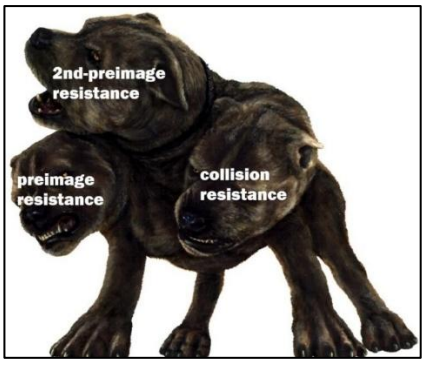
- **collision resistance** – פונקציית התמצות תהיה קשה מבחינה חישובית למציאת שני קלטים שונים הממופים לפלט זהה. כלומר קושי למצוא $input_1$ ו-$input_2$ כך שמתקיים:

$$(input_1 \neq input_2) \wedge \big(f(input_1) = f(input_2)\big)$$

- **preimage resistance** – בהינתן פלט כלשהו $output_1$ יהיה קשה מבחינה חישובית למצוא את ערך הקלט המקורי $input_1$ המקיים: $f(input_1) = output_1$.
- **2nd-preimage resistance** – בהינתן קלט כלשהו $input_1$ יהיה קשה מבחינה חישובית למצוא קלט אחר $input_2$ כך ששני הקלטים ממופים לאותו פלט.

דרישות אלו מקנות לפונקציית התמצות בטיחות רבה יותר בהשוואה לפונקציות התמצות הרגילות. דבר שאפשר את הטמעתן ולמעשה, במקרים רבים, להוות "קורה תומכת" בממנגנוני אבטחה. ד"ר דונקלמן מתאר זאת בצורה הטובה ביותר:

> התפישה הרווחת בנוגע לפונקציות תמצות קריפטוגרפיות בטוחות היא שהן מייצרות ערכי תמצית שנראים אקראיים למדי. תכונה זו, יחד עם יצירת הפלט באורך קבוע, הפכה את פונקציות התמצות הקריפטוגרפיות לפופלריות מאוד: הן בשימוש בחתימות אלקטרוניות (כאשר הנוהג הוא לחתום על תמצית של ההודעה, ולא על ההודעה המקורית), בקבצי סיסמאות (כאשר נשמר בקובץ ערך התמצית של הסיסמא, מה שמונע קריאת הסיסמא מהקובץ), בגזירת מפתחות קריפטוגרפיים לשימוש [...] ובעוד שורה ארוכה של יישומי אבטחת מידע. לפיכך, יש רבים המדמים את פונקציות התמצות הקריפטוגרפיות לאולר השיווצרי הקריפטוגרפי - כלי שיכול לעשות הכל[1].



[איור 1: שלושת הדרישות הבסיסיות עבור פונקציות תמצות קריפטוגרפיות, מקור[2]]

חשיבות פונקציות תמצות קריפטוגרפיות

במערכות אבטחה, פונקציית תמצות קריפטוגרפית היא מעמסה נוספת עליה יצטרך התוקף להתגבר. באופן כללי, חוזקה של מערכת אבטחה נקבע על ידי חוזקה של החולייה החלשה ביותר או רצף החוליות החלשות ביותר שמאפשר חדירה למערכת. מנגנון אבטחה משומן, בעל עשרות הגנות מתקדמות, הצפנות פוסט-קוואנטיות ואפס סובלנות לניסיונות אימות שגויים, לא יחשב כחזק במידה ושרת האימות שלו יהיה פרוץ לכל.

הדבר הראשון שעולה בראש בהקשר של שמירת סיסמאות הוא כנראה האלמנט החשוב ביותר: **אל תשמרו את הסיסמאות שלכם כטקסט פשוט**. בעת הרשמה לשירות מקוון כלשהו, בפרט לשירותי בנקאות דיגיטאליים, יש להגדיר שם משתמש וסיסמה. על פרטי המשתמש להשמר במסד הנתונים של הבנק או נותן השירות. בעת נסיון התחברות למערכת, תתבצע השוואה של הפרטים המוזנים כמול הפרטים השמורים ובמידה וישנה התאמה, תאושר ההתחברות למערכת.

הבנקים יודעים שלשמור במסד הנתונים את הצמד המורכב משם משתמש + סיסמה זה רעיון רע ולא בטוח. במידה ויצליח תוקף להשיג את ידו על תוכן מסד הנתונים, יתגלו למולו פרטי כלל המשתמשים ורק ישאר לו להתחבר אחד-אחד ולהעביר את הכסף לחשבון המבוזר שלו.

לשם כך, במסד הנתונים לא ישמרו הסיסמאות בצורתן הטבעית (כטקסט פשוט). אלא במקומן ישמרו ערכים המחושבים מהסוד (הסיסמה), באופן המקיים קשר חד-כיווני בין הסוד לתוצאת החישוב. עולה השאלה: מדוע לא להצפין את הסיסמאות במפתח סודי שרק הבנק מכיר?

הרי גם במידה ותוקף יצליח לגלות את ערך הסיסמה המוצפנת, ללא המפתח המתאים לא יהיה יכול לגלות את הסוד.

לכך נענה בציון הנקודות הבאות:

- לבנק אין צורך אמיתי לדעת את הסיסמה, רק לאמת אותה.
- במידה ותוקף הצליח לשבור את ההצפנה או אפילו להשיג את הסיסמה בדרך כלשהי, כל הסיסמאות נחשפות בבת אחת.

כאן נכנסת לשימוש פונקציית התמצות הקריפטוגרפית. בזמן יצירת המשתמש, המערכת תפעיל את פונקציית התמצות על הסיסמה ותשמור במסד הנתונים את הערך המוחזר. באופן דומה, בעת התחברות למערכת, תתבצע הפעלה של פונקציה זו על הסיסמה המוזנת. וכאשר ערך התמצות המחושב זהה לערך השמור במסד הנתונים עבור שם המשתמש הנתון, תתאפשר כניסה למערכת.

נניח כי במערכת בנקאות ממוחשבת כלשהי, ישנה פרצת אבטחה המאפשרת לתוקף לחלץ את כל המידע השמור במסד הנתונים המאחסן את פרטי התחברות המשתמשים.

במקרה זה (ובהנחה שאין חוליות חלשות אחרות), פונקציית התמצות בה נעשה שימוש לחישוב ערך התמצות מהסוד, היא זו שתקבע את חוזק המערכת. נחזור לפונקציית התמצות שציינו קודם לכן, הראנו כי ניתן בקלות למצוא שני משפטים (במקרה זה שתי סיסמאות) שונות שערך הגיבוב שלהן יהיה זהה. שימוש בפונקציית התמצות שלעיל במערכת תאפשר לתוקף למצוא/ליצור לכל ערך גיבוב במערכת סיסמה מתאימה וכך להתחבר בזה אחר זה לכל אחד מהמשתמשים ולגנוב את כספו.

שימוש בפונקציית תמצות קריפטוגרפית (למשל SHA-256, פונקציית גיבוב קריפטוגרפית שמייצרת פלט בגודל 256 סיביות, נחשבת מהירה מאוד), המקיימת את שלוש התכונות שלעיל. תמנע מהתוקף למצוא סיסמאות לערכי גיבוב ידועים (של המשתמשים) וכך לא יהיה יכול לגנוב את כספם.

שימוש בסיסמאות נפוצות

סיסמאות נפוצות, כפי שהשם מרמז, הן נפוצות. אלמנט האבטחה נעלם ברגע הראשון בו הסיסמה היא: 12345, password, admin וכיוצא בזה. ניתן לצאת מנקודת ההנחה כי כל תוקף פוטנציאלי ינסה לפחות כמה סיסמאות שכאלו במידה והוא מנסה לתקוף אדם מסוים כלשהו. אין צורך לשבור קיר אם המפתח לדלת נמצא מתחת לשטיח.

[איור 2: קודן בעל מקשים שחוקים, מקור[3]]

נשוב למערכת הבנק שתוארה קודם לכן. כעת במקומה של פונקציית התמצות החלשה תעמוד פונקציית תמצות קריפטוגרפית (למשל SHA-256). כידוע, לתוקף גישה מלאה למסד הנתונים המאחסן את פרטי התחברות המשתמשים. התוקף, שהבין שנעשה שימוש בפונקציית תמצות קריפטוגרפית, ממקד את מאמציו אל ניתוח סטטיסטי של המידע שהשיג. הוא מבחין כי קיים ערך גיבוב מסויים שנמצא אצל לא מעט משתמשים, אחרי ספירה הוא מגלה כי ערך זה הוא הנפוץ ביותר מבין כלל ערכי הגיבוב. התוקף יכול להסיק כי הסיסמה הנפוצה ביותר ככל הנראה תהיה המקור של ערך הגיבוב הנפוץ ביותר.

בדיקה מהירה ברשת מגלה כי הסיסמה הנפוצה ביותר (כאשר אין דרישות לאורך או תווים מסוימים) היא '123456'[4]. הפעלה של פונקציית התמצות הקריפטוגרפית SHA-256 על סיסמה זו תחזיר את הערך הבא: d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c928.

באותו הרגע צבר התוקף יכולת לפרוץ לכל המשתמשים שערך הגיבוב של סיסמתם הוא כמצויין לעיל, הרי הוא יודע את סיסמתם המקורית (טרם הגיבוב). באופן דומה יהיה יכול תוקף להשתמש בערכי תמצות מחושבים מראש עבור סיסמאות נפוצות את ולחפש התאמות במסד הנתונים (אליו יש לו גישה), התקפה זו ידועה בשם: Rainbow Table Attack.

מלח (Salt)

בדיוק כדי למנוע התקפות בהן ניתן להסיק סיסמה על סמך ניתוח סטטיסטי של ערכי גיבוב נשתמש במלח. מלח הוא ערך אקראי ויחודי למשתמש (לא סודי בהכרח) אותו מוסיפים לקלט (במקרה זה לסיסמה) לפני שמפעילים עליו פונקציית תמצות. שימוש במלח מאפשר לשתי סיסמאות זהות לקבל ערכי גיבוב שונים, כך גם סיסמה זהה אצל כמה משתמשים, ערך הגיבוב יהיה שונה.

Username	Salt value	String to be hashed	Hashed value = SHA256 (Password + Salt value)
user1	D;%yL9TS:5PalS/d	**password123**D;%yL9TS:5PalS/d	9C9B913EB1B6254F4737CE947EFD16F16E916F9D6EE5C1102A2002E48D4C88BD
user2	)<,-<U(jLezy4j>*	**password123**)<,-<U(jLezy4j>*	6058B4EB46BD6487298B59440EC8E70EAE482239FF2B4E7CA69950DFBD5532F2

[טבלה 1: שתי סיסמאות זהות שערך התמצות שלהן שונה (שימוש במלח), מקור[5]]

בעת יצירת המשתמש, המערכת תיצור ערך מלח אקראי, תוסיף אותו (שרשור) לסיסמה, לאחר מכן תפעיל את פונקציית התמצות על הסיסמה ותשמור במסד הנתונים את הערך המוחזר. באופן דומה, בעת התחברות למערכת, תתבצע הפעלה של פונקציה זו על הסיסמה המוזנת יחד עם ערך המלח השמור עבור המשתמש המסוים (את ערך המלח ניתן לשמור במסד הנתונים כפי שהוא, הוא אינו סודי). ותנאי הכניסה למערכת נשאר זהה.

חשוב שערך המלח יהיה אקראי ולא יחזור על עצמו. הרי מתכונות פונקציית התמצות הקריפטוגרפית, לא סביר שלשתי סיסמאות שונות יחושב ערך תמצות זהה (בפרט כאשר מתווסף ערך מלח אקראי). כלומר במידה ונמצאו שני ערכי תמצות שווים, כמעט בוודאות מדובר באותה הסיסמה ואותו ערך המלח. שימוש חד-פעמי בערך המלח משמר את אקראיות ערך הגיבוב עבור סיסמאות זהות.

באופן זהה כפי שידוע עבור אורכי סיסמאות: ככל שהסיסמה ארוכה יותר, מרחב המדגם גדול יותר. ולכן יהיה קשה יותר לבצע חיפוש ממצה של הסיסמה. בנוסף לכך, מרחב מדגם גדול, מקטין את ההסתברות לכך שערכים אקראים יחזרו על עצמם.

פלפל (Pepper)

בדומה למלח, פלפל הוא ערך המתווסף לקלט לפני שמפעילים עליו פונקציית תמצות. אך בשונה ממנו, הוא סודי, לא בהכרח אקראי, גלובאלי (כמעט תמיד אינו ייחודי למשתמש) ואינו נשמר במסד הנתונים (לרוב נשמר במודול אבטחת חומרה, HSM).

בעת יצירת המשתמש, המערכת תאחזר את ערך הפלפל הסודי, תוסיף אותו (שרשור) לסיסמה, לאחר מכן תפעיל את פונקציית התמצות על הסיסמה ותשמור במסד הנתונים את הערך המוחזר. באופן דומה, בעת התחברות למערכת, תתבצע הפעלה של פונקציה זו על הסיסמה המוזנת יחד עם ערך הפלפל. ותנאי הכניסה למערכת נשאר זהה.

ניתן לראות כי עבור שתי סיסמאות זהות יחושב ערך תמצות אחד זהה, למרות השימוש בפלפל. ומכאן המערכת תשאר פגיעה לניתוח סטטיסטי. לעומת זאת לא רצוי לבטל בהינף יד את חשיבות הפלפל. שכן שימוש בו מייתר את יכולת התוקף לבצע התקפה על פי ערכי תמצות מחושבים מראש (כפי שמוזכר קודם לכן), מפני חוסר ידיעת הפלפל.

תבלינים נוספים (Bcrypt)

בשונה ממלח ופלפל אותם הצגנו קודם לכן, חלק זה רוצה להציג פונקציית תמצות ייעודית לסיסמאות, שלטעמנו היא כה אלגנטית ומעניינת עד שחבל שלא להזכירה ולתארה בכלליות. הכותרת ממשיכה את הכיוון הכללי של המאמר, אף שחלק זה יכול היה להקרא גם "מנה עיקרית", על פי שמו של המנגנון המרכזי הטמון בו, *Blowfish* (נַפּוּחִיתִיִים, מכונה בסלנג "אבו נפחא"[6]).

הפונקציה bcrypt הינה פונקציית תמצות קריפטוגרפית איטית הייעודית לתמצות סיסמאות, שבמרכזה עומד מנגנון קריפטוגרפי בשם Eksblowfish ("expensive key schedule Blowfish") המבוסס על אלגוריתם ההצפנה Blowfish. הפונקציה נועדה להיות כבדה מבחינת דרישות עיבוד, לשם כך היא מבצעת מספר רב של חישובים סדרתיים. תכונה זו יעילה וחיונית להתמודדות עם סוגי מתקפות שונים, ובפרט מתקפות חיפוש ממצה *(*Brute-Force*)*, גם כאשר התוקף מחזיק במשאבי חישוב משמעותיים.

אלגוריתם ההצפנה Blowfish, מתבלט בכך ששגרת הכנת המפתח שלו היא פעולה יקרה מבחינה חישובית. אלגוריתם זה מתחיל מסט של תתי-מפתחות במצב התחלתי קבוע (סטנדרטי), מבצע הצפנת בלוק בעזרת מפתח חלקי (חלק מהמפתח), ומשתמש בתוצאה זו על מנת להחליף חלק מתתי-המפתחות. לאחר מכן האלגוריתם משתמש בתת-המפתחות המעודכנים (לאחר ההחלפה) בכדי להצפין חלק נוסף מהמפתח. שוב ושוב תוצאת ההצפנה משמשת לצורך החלפת תתי-מפתחות נוספים. תהליך זה נמשך באופן איטרטיבי, כאשר בכל שלב מתבצעת הצפנת בלוק, עד שכל תתי-המפתחות הוחלפו.

המנגנון קריפטוגרפי Eksblowfish, המבוסס על אלגוריתם ההצפנה Blowfish שתואר לעיל, מרחיב רעיון זה. תהליך הכנת המפתח באלגוריתם זה מתחיל בצורה מותאמת של שגרת הכנת המפתח באלגוריתם Blowfish, שבה נעשה שימוש הן במלח והן בסיסמה על מנת לאתחל את כל תתי-המפתחות. לאחר שלב האתחול מתבצעת סדרת סבבי הכנת מפתח נוספים, שבהם מיושמת שגרת הכנת המפתח הסטנדרטית של אלגוריתם Blowfish.

כאשר בכל סבב נעשה שימוש לסירוגין במלח או בסיסמה כמפתח, ותוצאת כל סבב משמשת כמצב התחלתי לסבב הבא. באופן זה מתבצע מעין שרשור של תהליך הכנת המפתח, כך שכל סבב מתחיל ממצב תתי-המפתחות שהתקבל בסבב הקודם.

מנגנון הכנת המפתח (הן בEksblowfish והן ב-Blowfish) מחזיר שני פרמטרים: *P*, מערך של תתי-מפתחות ו-*S*, מערך של קופסאות החלפה (***Substitution-Boxes***). קופסאות ההחלפה שלעיל משמשות לקביעת סדר החלפה של סיביות על פי תבנית מסוימת (קרא בהרחבה[7]) ונעשה בהן שימוש כחלק ממנגנון ההצפנה שיתואר בהמשך.

לאחר פעולת שגרת הכנת המפתח, מתבצעת הצפנה כ-64 פעמים באמצעות שגרת ההצפנה של אלגוריתם Blowfish הסטנדרטי (במצב ECB) של המחרוזת הבאה: *OrpheanBeholderScryDoubt*. בכל אחת מאיטרציות ההצפנה, הקלט אל שגרת ההצפנה יהיה תוצאת ההצפנה מהסבב הקודם, בנוסף לתתי-המפתחות וקופסאות ההחלפה שהתקבלו קודם לכן בתחילת האלגוריתם.

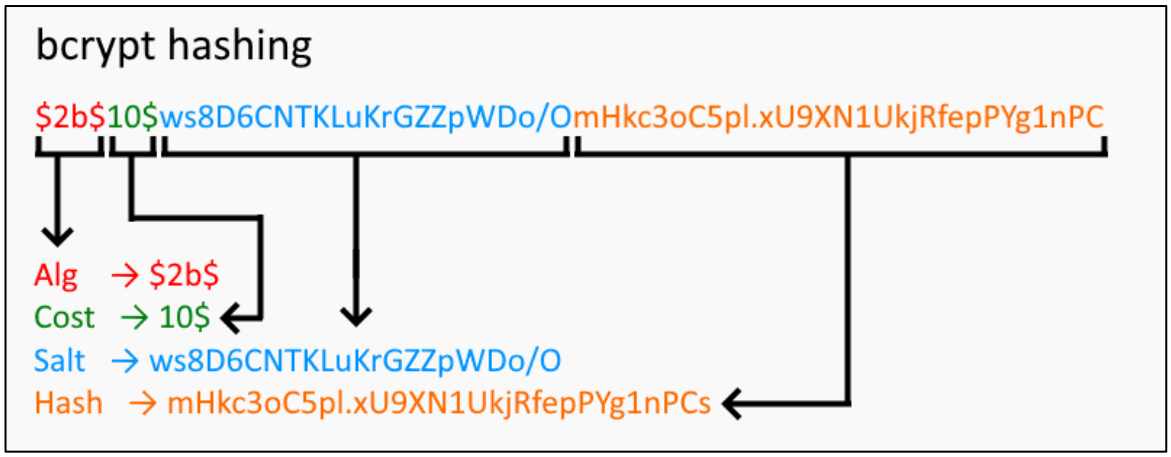
בסיום האלגוריתם, מוחזרת מחרוזת הפלט המורכבת מערך המוצפן, ערך המלח והמחיר (מוסבר בהמשך).

```
bcrypt (cost, salt, pwd)
    state ← EksBlowfishSetup (cost, salt, key)
    ctext ← "OrpheanBeholderScryDoubt"
    repeat (64)
        ctext ← EncryptECB (state, ctext)
    return Concatenate (cost, salt, ctext)
```

[איור 3: פונקציית Bcrypt, מקור[8]]

החלק החשוב ביותר באלגוריתם, זה שמגדיר את הפונקציה כ-"איטית", הוא המחיר (Cost). הזכרנו קודם לכן כי בזמן שגרת הכנת המפתח, לאחר אתחול תתי-המפתחות, מתבצעת סדרת סבבי הכנת מפתח על פי השגרה הסטנדרטית של אלגוריתם Blowfish. מספר הסבבים האלו נקבע כ- 2^{cost} כאשר המחיר הוא פרמטר המתקבל עם הקריאה לפונקציית התמצות. ככל שהמחיר גדול יותר, יתבצעו (באופן מעריכי) יותר תתי-סבבים בשגרת הכנת המפתח. קל לראות כי ללא תתי-סבבי הכנת המפתח, שגרת הכנת המפתח תהא זהה למעשה לשגרת הכנת המפתח באלגוריתם Blowfish הסטנדרטי.

מבחינת עקרונות קריפטוגרפיים טהורים, מנגנון זה אינו בהכרח חזק יותר משגרת הכנת המפתח של אלגוריתם ההצפנה Blowfish הסטנדרטי. האלגוריתם אינו מוסיף חוזק קריפטוגרפי תאורטי, אלא קושי חישובי מכוון. עם זאת יתרונו המרכזי טמון בכך שמספר הסבבים ניתן לקביעה ולשליטה, ולכן ניתן להפוך את תהליך הכנת המפתח לאיטי כרצוננו. תכונה זו מקשה משמעותית על ניסיונות התקפה, ובפרט על מתקפות מבוססות חיפוש ממצה.



[*איור 4:* פלט לדוגמה, פונקציית Bcrypt, מקור[9]]

סיכום

המאמר נכתב ברמת פירוט מצומצמת יחסית, מטרתו מעולם לא הייתה להעמיק ולצלול לפרטי פעולתן של פונקציות תמצות. ספרים שלמים נכתבו בנושא, ומאמר זה אינו מתיימר להוות תמצית שלהם. לפני מספר שבועות ישבתי עם חבר מתכנת לשיחה, והצגתי בפניו שרת אבטחה ואימות[10] שמימשתי כחלק ממטלה בקורס אבטחת מידע באוניברסיטה. במהלך השיחה התחלתי להסביר על הדו"ח הנלווה למטלה, וכיצד הוא מתאר את ההבדלים בין שיטות תמצות שונות, לצד מנגנוני הגנה מגוונים, כפי שהם באים לידי ביטוי בזמני חישוב ובקושי שבפריצה.

מהר מאוד גיליתי שלמרות שמדובר בבוגר תואר ראשון במדעי המחשב ואף במתכנת פעיל וחריף, מי שלא העמיק מעט בתחום פונקציות התמצות והגנות קריפטוגרפיות או פשוט לא לקח קורסים רלוונטים, ככל הנראה אינו מכיר מושגים בסיסיים כגון מלח ופלפל.

אשמח אם מאמר זה הצליח, ולו במעט, להנגיש את היופי שבפונקציות תמצות, ולהמחיש עד כמה הן רחוקות מלהיות שטחיות או טריוויאליות. קל מאוד להתייחס לשימוש בפונקציות אלו כאל סוג של קסם, מכונות קלט-פלט וְתו לָא. למעשה, קל מאוד לקבל כל עיקרון וכל מנגנון כמובן מאליו. אך הרגע שבו אנו מפסיקים לקבל רעיונות אלו כמובנים מאליהם, הוא הרגע שבו עולם חדש נגלה לנגד עינינו.

על המחבר

בר טופליאן, בוגר תואר ראשון במדעי המחשב באוניברסיטה הפתוחה. אוהב לקרוא, ללמוד, לכתוב ולהסביר, כחלק מהדרך לצבור ולהעמיק ידע. בעל תשוקה לאבטחת מידע, למתמטיקה ומה שבניהן (הכנת פיצה נאופוליטנית, משום מה...).

מקורות מידע

- https://www.digitalwhisper.co.il/files/Zines/0x15/DW21-2-CryptoHashFunction.pdf
- https://harrypotter.fandom.com/wiki/Three-headed_dog
- https://www.schneier.com/blog/archives/2009/07/information_lea_1.html
- https://nordpass.com/most-common-passwords-list
- https://en.wikipedia.org/wiki/Salt_(cryptography)
- https://he.wikipedia.org/wiki/נפוחיתיים
- https://en.wikipedia.org/wiki/S-box
- https://auth0.com/blog/hashing-in-action-understanding-bcrypt
- https://stytch.com/blog/what-is-password-hashing
- https://github.com/b1pb0p/authentication-security-server