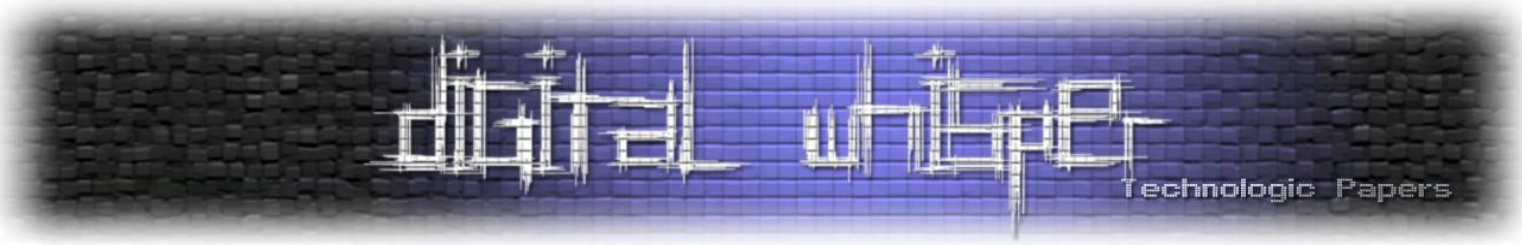


Certificate Transparency as Communication Channel

מאת עומר מדן

הקדמה

במאמר זה אדגים כיצד ניתן להסתיר מידע במפתחות ציבוריים, ולהשתמש בדרך זו על מנת לתקשר בין גורמים שונים, ללא תקשורת ישירה בין הצדדים. אסביר על Certificate Transparency ועל sigstore, המשמש מנגנון שמירה על תוצרים של בנייה (לדוגמא docker image). לבסוף, אציג שיטות להסתרת מידע ב-RSA public key, ואסביר כיצד חברות חושפות פרטים פנימיים על החברה דרך אותו מנגנון.

Certificate Transparency

למי שלא מכיר, Certificate Transparency הוא מנגנון ששומר על כל החתימות של אתרים שהופקו על ידי ה-Certificate Authority השונים – לדוגמא Let's Encrypt או DigiCert. המנגנון הזה בנוי בצורה דומה מאוד ל-blockchain, ככה שבעצם כל רשומה לא יכולה להימחק לעולם. המנגנון הזה מבוסס על עצי Merkle, שהם בעצם מבנה נתונים שבנוי ככה שאפשר יהיה לוודא חתימות של כל עלה בעץ, נשים לב גם שאי אפשר למחוק מידע מהמבנה נתונים הזה לאחר שנכנס אליו.

כל חתימה שמישהו יוצר לאתר, בערך מאז 2013, נשמרת במנגנון הזה. המידע שנשמר הוא המידע המלא של החתימה - כלומר מי חתם, מה חתם, מתי חתם ואיזה CA חתם. אם נכנס לאתר crt.sh, נוכל לחפש את החתימות של כל אתר/דומיין שהוא בעולם כיום, וגם היסטורית אחורה.

הבעיה הגדולה של המנגנון שמירת חתימות האלו, היא שהוא חושף כל בעל אתר ל-info leak קליל של כל ה-subdomains שלו. חברות רבות לא שמות לב לעניין, אבל זה דיי משמעותי, בלי שום DNS enumeration אפשר להשיג רשימה כמעט של כל subdomain של החברה.

המנגנון עצמו מוגדר היטב ב-RFC 9162, שבו מתואר כיצד האובייקטים של כל חתימה נשמרים ואיזה API נחשף למשתמש. לדוגמה – דפדפן יכול לחפש באובייקטים רשומים על מנת לוודא שהחתימה היא נכונה. ה-API שנחשף ציבורי לגמרי, אפשר אפילו לגשת לקרוא את המידע מה-API מדומיין של גוגל ישירות.

בנוסף לחתימות של אתרים, פרויקט sigstore המוצא על ידי כמה חברות על מנת לבנות כלים נגד supply chain attacks. לדוגמא, כדי לוודא שלא נכנס malware ב-docker image שהורדנו. הפרויקט משתמש בכלי שנקרא cosign שחותם קבצים על ידי מנגנון דומה של חתימות של אתרים, ומשתמש גם ב- Certificate Transparency משלו בשם rekor. כל קובץ שמישהו חותם נרשם ב-rekor על ידי API פתוח לכל.

חתימות

התייחסתי מספר פעמים לחתימות, אבל מה זה בדיוק אומר?

חתימה מוגדרת בפורמט x.509 וכוללת פרטים רבים, כמו מי חתם, מתי בוצעה החתימה, מהו ה-public key, וכן אפשרות להוסיף שדות נוספים כ-extensions. כאשר אנחנו משתמשים ב-SSL או בגרסאות חדשות יותר של TLS, אנחנו למעשה יוצרים חתימה כזו, שבדרך כלל מיוצגת כקובץ PEM, אם כי קיימים גם פורמטים נוספים. ניתן לראות את החתימה של כל אתר, למשל, על-ידי לחיצה על סמל המנעול (הירוק) ליד שם האתר בדפדפן.

המידע הזה מועבר למחשב שלנו בזמן תהליך ה-handshake, ומאפשר להגן על גולשים על-ידי זיהוי אתרים עם חתימה שפג תוקפה או שאינה תקינה. בנוסף, כל חתימה כוללת גם חתימות נוספות של ה- Certificate Authority שחתם עליה, למשל Let's Encrypt, כך שבפועל מתקבלת שרשרת של חתימות. כל המידע הזה נרשם בסופו של דבר ב-Certificate Transparency, למעט מקרים שבהם מידע מסוים, כמו חלק מ- x.509 extensions, לא בהכרח נכלל.

אוקי... מה אתה רוצה?

עד עכשיו, כל מה שאנחנו רואים זה מנגנון בסיסי שמגן על גולשים באינטרנט , בכך שמוודא שהם נכנסים לאתר לגיטימי, ולא גולשים לאיזה אתר facebook מומצא. נשים לב שכל ה-APIs של המנגנונים האלו הם ציבוריים לחלוטין, ללא כל אימות כלשהו. בנוסף, כל ה-domains של המנגנונים האלו הם דומיינים שיהיה מאתגר מאוד לחסום אותם בארגון, הרי הם משמשים את הכלים שמגנים עלינו, או שהם פשוט Google או Cloudflare.

בבסיס המנגנון שאסביר כאן נמצא טריק די מטופש, אבל יש לו משמעות גדולה להמשך – אנחנו פשוט מחביאים מידע בתוך ה-Public key של החתימה. ה-Public key הזה נמצא בכל מקום ברשימות הללו, אם נמצא דרך להכניס לשם מידע מוסתר, נוכל לדחוף מידע ישירות ל-Certificate Transparency, ומישהו אחר יוכל לקרוא אותו, מבלי שיש בין שניהם תקשורת ישירה.

איך מחביאים מידע ב-Public Key

אינני איזה מומחה הצפנות דגול, וגם במתמטיקה אני לא כזה טוב, אבל באלגוריתם ההצפנה RSA בוחרים שני מספרים ראשוניים נורא גדולים, ואז כופלים אותם ביחד כדי לייצר את ה-public key. התוצאה היא מספר נורא גדול שבסוף נכתב ב-certificate. אז מה שנעשה בעצם זה נמצא מספר אחד ראשוני, ונחפש מספר ראשוני נוסף, שהכפל של שניהם בסוף יביא לנו תוצאה שבה המידע שאנחנו רוצים להסתיר מתקיים בתוך המספר הסופי.

לדוגמה, אנחנו רוצים לשים את הערך ascii של A שהוא 65 בתוצאה הסופית. נחפש שני מספרים שתוצאת המכפלה שלהם יוצאת מספר שמכיל את הסטרינג "65", כמו 100**65**1233. זה אולי מרגיש לכם מטופש, אבל זה עובד. בהתחלה חשבתי שהחיפוש עצמו ייקח מלא זמן, אבל בערך תוך דקה זה מוצא מספרים ראשוניים שמתאימים להודעות של 11 bytes בערך.

אם ככה, יש לנו public key שהוא נשלט על ידינו והערך שלו תקין, אבל מכיל גם את המידע שאנחנו רוצים להסתיר. חשוב לציין שהמידע לא באמת מוסתר, אם מסתכלים על החתימה רואים שרשום שם המספר שלנו, אבל צריך לדעת איפה הוא אמור להיות. נראה בהמשך שאפשר להצפין את המידע בצורה שלא יהיה אפשרי לקרוא אותו.

אופציה א' - העברת מידע דרך domains

ברגע שיש לנו את היכולת ליצור חתימה שמכילה מידע שנשלט על ידינו, אפשר להתחיל במשימה עצמה. השלב הראשון הוא רכישת דומיין שלא באמת אכפת לנו ממנו, כאשר כל המטרה שלו היא לשמש כערוץ להעלאת מידע ל-Certificate Transparency. לאחר רכישת הדומיין, נגדיר את ה-DNS שלו כך שיצביע על מכונה בענן, אשר תשמש אותנו לצורך שלב ה-domain validation של Let's Encrypt.

אני, למשל, התקנתי OpenBSD, שמגיע עם כלי בשם acme-client, אבל כמובן שניתן להשתמש גם בכלים או שיטות אחרות.

בשלב הבא בניתי חתימה שבה רשמתי הודעה כלשהי, העליתי אותה למכונה, וביקשתי מהכלי ליצור עבורה תעודה. בפועל, Let's Encrypt בדקו מול המכונה שלי שהשליטה בדומיין תקינה (ולכן היה צורך בהגדרת ה-DNS), ולאחר מכן חתמו על החתימה בעצמם. התוצאה היא שתעודה שמכילה את המידע שהוספתי נרשמת ונשמרת לצמיתות ב-Certificate Transparency. כשלעצמו זה לא נשמע מעניין במיוחד, אבל החלק המשמעותי כאן הוא דווקא הצד הקורא. כפי שצוין קודם, Certificate Transparency מספק API שמאפשר לחפש ולקרוא חתימות. לאחר קצת עבודה עם ה-API (שהוא לא הכי נוח), הגעתי למצב שבו ניתן לקרוא מהצד השני את ההודעה שהוטמעה בחתימה.

מה המשמעות של זה בפועל? הקורא לעולם לא ניגש לדומיין שלנו ישירות, אלא קורא את המידע דרך דומיין של גוגל (או ספק CT גדול אחר). מאחר שכמעט אף כלי אבטחה לא יחסום דומיינים של גוגל, מתקבל ערוץ תקשורת שקשה מאוד לחסום בכניסה לארגון. עקרונית גם הכיוון ההפוך אפשרי: אם מישהו משיג שליטה על תוכנה שמנפיקה חתימות, למשל בתוך K8S, הוא יכול לסמן החוצה שהוא הצליח להשתלט על המערכת.

כדי למצוא את החתימה הרלוונטית, ניתן להשתמש בשרתי ה-Certificate Transparency שמנוהלים על-ידי חברות גדולות. מתחילים בקריאה ל-get-sth דרך ה-API כדי לקבל את גודל העץ, ולאחר מכן מבצעים קריאות get-entries עם טווח סביר. בשלב הזה צריך לעבור חתימה-חתימה ולחפש את החתימה הרצויה. זה תהליך די מעצבן, אבל אם יודעים בערך מתי ההודעה נשלחה, אפשר להשתמש בחיפוש בינארי כדי להגיע יחסית מהר לאזור המתאים בעץ, ואז לעבור רשומה-רשומה.

ה-API עצמו די מתיש, אבל בסופו של דבר מדובר בתקשורת מאוד מוסתרת: כמעט שום כלי אבטחה לא יתריע על קריאות ל-Cloudflare או לספקי CT גדולים, והסיכוי שהקריאות יעברו בלי זיהוי הוא גבוה מאוד. גם הפענוח של החתימה עצמה לא לגמרי טריוויאלי, אך לבסוף ניתן לחלץ את המידע הרלוונטי — במקרה הזה ה-modulus, שבתוכו מוסתרת ההודעה.

אופציה ב' – העברת מידע דרך sigstore

sigstore מספק API פתוח לכולם, שמאפשר לחתום על קבצים, Docker images ועוד. כל מה שנדרש כדי לבצע חתימה הוא ה-hash של האובייקט שנחתם ו-public key, שני פרמטרים שאנחנו יכולים לשלוט בהם, כפי שכבר ראינו.

כאן אנחנו משתמשים בטריק נוסף. קיימת תוכנה בשם magic-wormhole, שמאפשרת לשני מחשבים להתחבר זה לזה אם שני הצדדים מסכימים על passphrase משותף, למשל "שרה שרה שיר שמח". ברגע ששני הצדדים מזינים את אותו passphrase, החיבור נוצר. את הרעיון הזה אנחנו מאמצים, והופכים את ה-passphrase ל-hash, יחד עם מספר מקטע של ההודעה, לדוגמה: "שרה שרה שיר שמח 1/10", "שרה שרה שיר שמח 2/10".

באמצעות פיצול ההודעה למקטעים, ניתן לבנות סדרה של public keys, שכל אחד מהם מכיל חלק מהמידע, ולהעלות אותם בקלות יחסית ל-sigstore. עד כמה זה פשוט? בפועל כתבתי על זה תוכנת צ'אט שלמה.

בצורה הזו אנחנו מפצלים הודעה, למשל: "שמע, מחר בבוקר נפגש בחדרה... בשעה 10", למקטעים שכל אחד מהם נכנס ל-public key משלו. בצד הקורא, כל מה שנדרש הוא לקרוא את ה-public keys הרלוונטיים ולחבר מחדש את ההודעה המלאה.

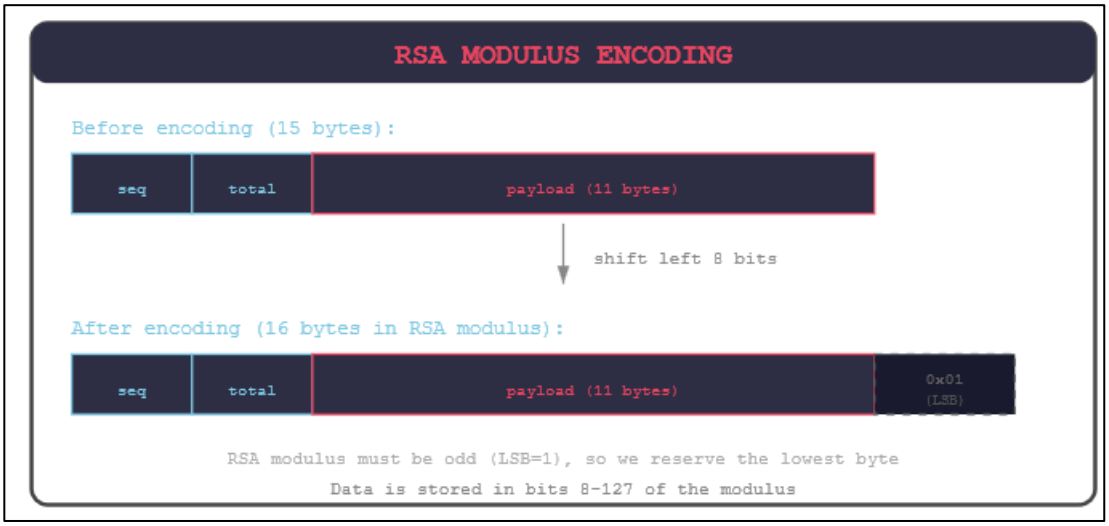
גם שלב הקריאה מ-sigstore פתוח לחלוטין לכולם, ומתבצע מול דומיין שקשה מאוד לחסום, מכיוון שארגונים רבים משתמשים ב-sigstore כחלק מכלי ההגנה שלהם. בנוסף, ההודעות עצמן מוצפנות, כך שהמידע לא נשלח בצורה גלויה, וגם גורם חיצוני שייתקל בחתימות לא יבין שמדובר בערוץ תקשורת שדורש תשומת לב.

בצד הקורא, הקריאה אפילו פשוטה יותר מאופציה א'. מאחר שאנחנו משתמשים במנגנון ה-passphrase הראשוני, כל מה שצריך לעשות הוא לחפש לפי ה-hash הרלוונטי, בהתאם למנגנון המקטעים שנבחר. ה-API של sigstore נוח יותר לשימוש, ומאפשר קריאת HTTP פשוטה מסוג retrieve עם ה-hash המבוקש, שמחזירה את החתימה. במקרה הזה, החתימה עצמה היא ה-public key, שבתוכו מוסתרת ההודעה.

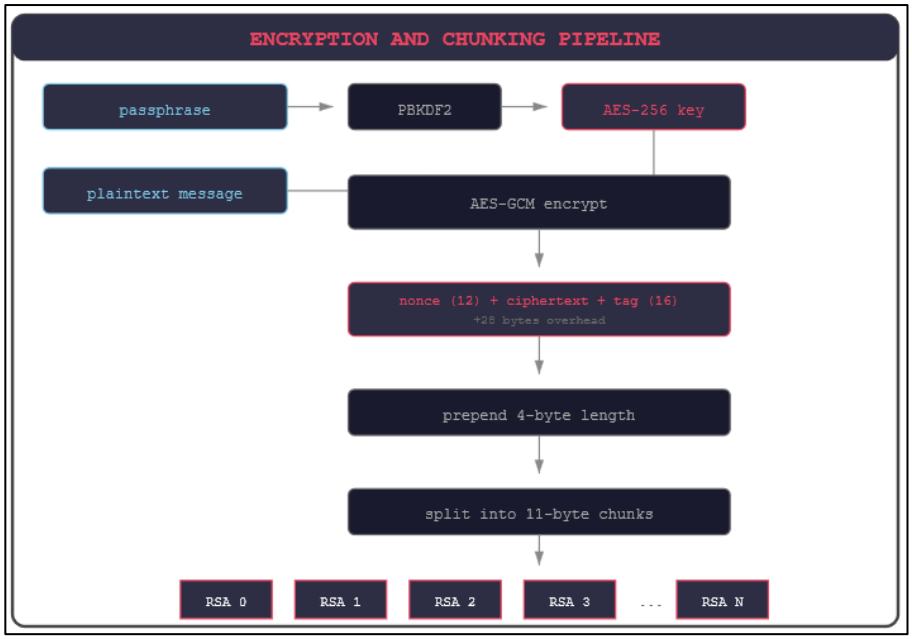
נסביר את זה טוב יותר

קודם כל אנחנו צריכים להכין את המידע שלנו להכנסה לתוך מספר ראשוני. בגלל שמספר ראשוני חייב להיות אי זוגי, אנחנו חייבים להזיז כל bit של מידע מה-LSB על מנת שלא יצא לנו מספר זוגי. נזיז בכמה ביטים ונחפש את המספרים הרלוונטיים שיוצרים את הערך הסופי שאנחנו רוצים, בשרטוט המצורף תוכלו לראות איך זה קורה.

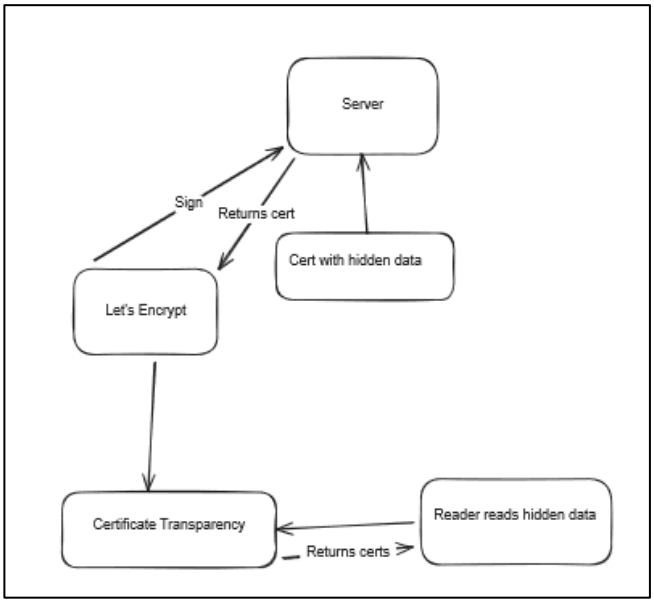
כשאנחנו שולחים הודעה שהיא יותר מביטים בודדים, אנחנו צריכים לבנות מקטעים, על מנת שנוכל להצפין כמו שצריך את המידע, לכן אנחנו צריכים לבנות header של כמה יש וכמה כבר נכתבו.



כפי שניתן לראות בציור למטה, אנחנו בונים מקטעים, שהם כולם בעצם public keys, המידע הנוסף שלהם לא רלוונטי לנו, מה שמעניין אותנו זה רק המידע שאנחנו הסתרנו.



אחרי שבנינו את ה-public keys אנחנו צריכים להשתמש באחת מהדרכים שרשמנו קודם, לדוגמא ניקח את Let's Encrypt.



מה שיפה פה זה שהמידע לא ימחק לעולם, כי ככה המודל של certificate transparency עובד.

Certificate Transparency Info Leaks

בעיה נוספת ש-Certificate Transparency גורם לחברות, ורובן לא שמות לב לנושא, זה דליפה של מידע רב על subdomains שלהם. חברות רבות משתמשות היום בחתימות אוטומטיות על ידי Let's Encrypt, ובגלל ההתנהגות שתיארנו קודם, כל חתימה נרשמת, מה שמייצר בעצם מאגר מידע עצום של חברות ושל כל החתימות שהן רשמו. לדוגמא, אם רוצים לדעת באיזו מערכת queue management איזה סטראטפ משתמש, מספיק להיכנס ל-crt.sh ולרשום את הדומיין של החברה, רוב הסיכויים שצוות ה-devops שלהם חתם subdomain עם השם-rabbitmq או kafka. אפשר לבנות מפת מידע שלמה על חברות בלי שום בעיה דרך Certificate Transparency ואף אחד לא שם לזה לב כמעט.

הדגמות

נתחיל בהדגמה ליצירת חתימה לאתר digitalwhisper.co.il שמכילה מידע "מוסתר":

```
→  secertcert git:(main) uv run secertcert.py -m "HelloWorld" -o output --mode x509 -d digitalwhisper.co.il
Mode: x509
Data size: 10 bytes
Chunks needed: 1
Payload per chunk: 11 bytes

Generating certificate 1/1... done

Generated 1 certificates in output/
Metadata saved to output/metadata.json
→  secertcert git:(main) ✗
```

אנו בעצם מייצרים חתימת X.509 לאתר, התוכנה מחפשת מספרים ראשוניים מתאימים ובונה את החתימה, התוצאה הסופית תראה כך:

```
→  output git:(main) ✗ openssl rsa -pubin -in key_0000.pem -noout -modulus | xxd
00000000: 4d6f 6475 6c75 733d 3935 3130 3933 3430  Modulus=95109340
00000010: 3046 3634 4541 3643 3134 4236 3143 3330  0F64EA6C14B61C30
00000020: 3432 3537 3536 3241 4545 3538 3246 3941  4257562AEE582F9A
00000030: 3945 3139 3644 3742 3132 3830 3344 3043  9E196D7B12803D0C
00000040: 3945 3530 4341 4245 3435 3043 3842 3541  9E50CABE450C8B5A
00000050: 3842 3330 3131 3033 3042 3946 3737 3837  8B3011030B9F7787
00000060: 4331 4333 3431 3039 3831 4530 3733 4531  C1C3410981E073E1
00000070: 4342 3245 4135 3639 3833 3431 3436 3545  CB2EA5698341465E
00000080: 3738 3435 3132 3739 4545 4434 3241 4544  78451279EED42AED
00000090: 4437 3642 3838 4533 3333 3838 4336 4539  D76B88E33388C6E9
000000a0: 4330 3442 4134 3839 3833 3431 3237 3743  C04BA4898341277C
000000b0: 3833 3833 4534 4230 3041 3333 4244 4235  8383E4B00A33BDB5
000000c0: 3433 4141 3532 3234 3536 3545 3838 3243  43AA5224565E882C
000000d0: 3339 4346 4334 4341 4638 3242 3834 3837  39CFC4CAF82B8487
000000e0: 4236 3143 3830 4243 4330 3739 4239 4130  B61C80BCC079B9A0
000000f0: 3342 4235 3430 4536 3831 3332 3736 3830  3BB540E681327680
00000100: 3041 3632 3234 4337 3232 3932 4538 4630  0A6224C72292E8F0
00000110: 3645 3834 3635 3133 4432 3738 3936 4645  6E846513D27896FE
00000120: 3230 4438 4143 3136 3434 3934 4231 3832  20D8AC164494B182
00000130: 4538 4345 3645 4541 4334 3137 3846 3835  E8CE6EEAC4178F85
00000140: 3538 4342 4636 4239 3130 3944 3944 4132  58CBF6B9109D9DA2
00000150: 4646 3442 3732 3234 4432 3933 3135 4635  FF4B7224D29315F5
00000160: 4431 4143 4345 4435 3530 3537 4631 4445  D1ACCED55057F1DE
00000170: 4343 3434 3837 3341 4244 3930 3943 3943  CC44873ABD909C9C
00000180: 3844 3133 4135 3136 4131 3641 4442 3843  8D13A516A16ADB8C
00000190: 3134 3144 3743 4646 3832 3841 3946 3642  141D7CFF828A9F6B
000001a0: 3739 3232 3032 3632 4141 4444 4437 4145  79220262AADDD7AE
000001b0: 3243 3043 3137 4338 3541 4345 4344 3432  2C0C17C85ACECD42
000001c0: 4341 3933 3946 4638 3939 4139 3232 3138  CA939FF899A92218
000001d0: 3236 4137 3443 4144 3139 3446 4336 3136  26A74CAD194FC616
000001e0: 4430 4143 3344 3739 3030 3030 3030 3031  D0AC3D7900000001
000001f0: 3438 3635 3643 3643 3646 3537 3646 3732  48656C6C6F576F72
00000200: 3643 3634 3030 3031 0a                   6C640001.
```

בסוף החתימה אפשר לראות את המספרים 48656C6C6F576F726C64 שהם בעצם הASCII של "HelloWorld".

עכשיו נדגים כתיבה וקריאה פשוטה מ-rekor ללא שימוש בכמה מקטעים:

```
➜  cert-data-pass git:(main) ✗ uv run rekor-write -m "HelloWorld" -c "123-dag-maluach"

╭────────── Rekor Steganography ──────────╮
│ Rekor Transparency Log Writer           │
│ Embedding hidden data in Sigstore Rekor │
╰─────────────────────────────────────────╯
Artifact content: 123-dag-maluach-2026-01-31
Artifact hash (rendezvous): sha256:577dc4f3dd992b4ecf127df8249076c5360efaff027f730e03154bf7af1bb37d
Generating RSA-2048 key with 128 bits of hidden data...
Searching for RSA key with hidden data in modulus...
Target: 128 bits of hidden data in lower bits of n
Generating base prime q...
Searching for prime p with correct lower bits...
Found matching key after 126 attempts!
Uploading to Rekor at https://rekor.sigstore.dev...

╭───────── Rekor Upload ─────────╮
│ Entry Uploaded Successfully    │
╰────────────────────────────────╯
                         Entry Details
┏━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┓
┃ Property         ┃ Value                                        ┃
┡━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┩
│ UUID             │ 108e9186e8c5677a6d01c8c29120670dc1f0564c...  │
│ Log Index        │ 884906514                                    │
│ Integrated Time  │ 2026-01-31T21:08:18                          │
│ Artifact Hash    │ sha256:577dc4f3dd992b4ecf127df8249076c5...   │
│ Channel ID       │ 123-dag-maluach                              │
│ Hidden Data Bits │ 128                                          │
│ Hidden Message   │ HelloWorld                                   │
└──────────────────┴──────────────────────────────────────────────┘

To retrieve this entry:
  rekor-cli get --uuid 108e9186e8c5677a6d01c8c29120670dc1f0564ca573376f4f267cdbcbccf51ab4cf056b96fb2e88
  curl https://rekor.sigstore.dev/api/v1/log/entries/108e9186e8c5677a6d01c8c29120670dc1f0564ca573376f4f267cdbcbccf51ab4cf056b96fb2e88

Or search by channel:
  uv run python -m rekor.rekor_reader -c "123-dag-maluach"
```

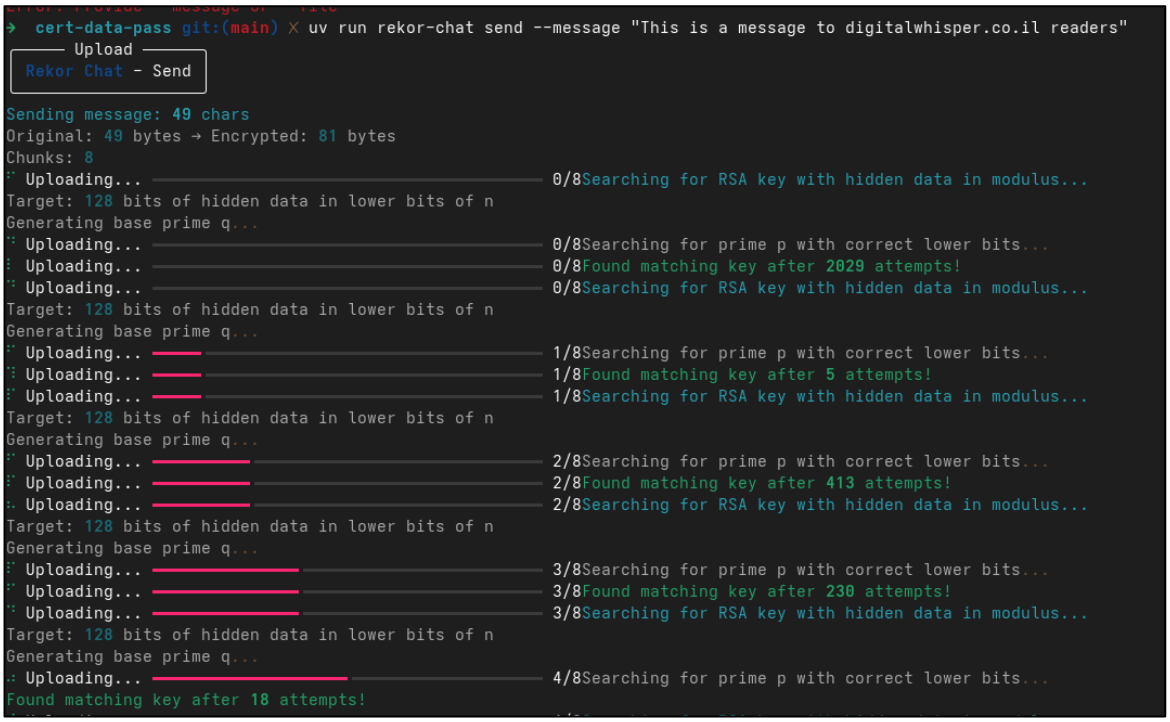
אנחנו מתחילים בלחפש מספרים ראשוניים מתאימים, ורואים שהכלי מצא אחרי 126 ניסיונות, זה בעיקר תלוי בהגרלה שלו בהתחלה, אחרי זה הוא משתמש ב-API של rekor כדי לשמור את המידע, ונותן לנו לגשת בחזרה למידע עם curl. מאחר וזה JSON לא כזה מעניין, נתעלם ממנו כרגע.

כדי לקרוא, אנחנו משתמשים באותה "סיסמה" שבחרנו עם הדג מלוח.

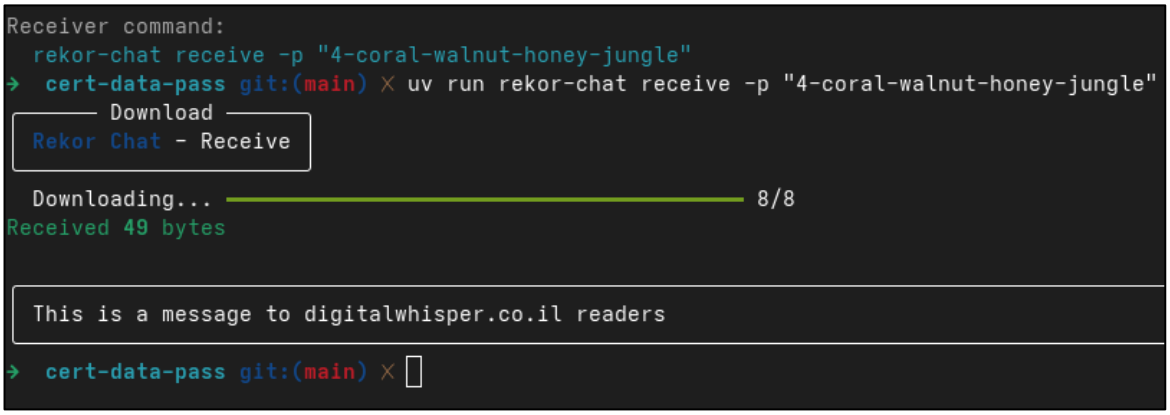
```
  uv run python -m rekor.rekor_reader -c "123-dag-maluach"
➜  cert-data-pass git:(main) ✗ uv run python -m rekor.rekor_reader -c "123-dag-maluach"
<frozen runpy>:128: RuntimeWarning: 'rekor.rekor_reader' found in sys.modules after import
╭────────── Rekor Steganography ──────────╮
│ Rekor Transparency Log Reader           │
│ Extracting hidden data from Sigstore Rekor │
╰─────────────────────────────────────────╯
Channel: 123-dag-maluach
Searching for artifact hash: sha256:577dc4f3dd992b4ecf127df8249076c5...
Found 1 entries


╭──────────────────────────────────────────────────────────────────────
│  UUID                 108e9186e8c5677a6d01c8c29120670dc1f0564ca573376f4f...
│  Log Index            884906514
│  Time                 2026-01-31T21:08:18
│  Artifact Hash        sha256:577dc4f3dd992b4ecf127df8249076c5...
│  Data Bits            128
│  Hidden Data (hex)    48656c6c6f576f726c64000000000001
╰──────────────────────────────────────────────────────────────────────

Tip: Use -m flag to output only the hidden message
Tip: Use -j flag for JSON output
```

הכלי מחפש על פי ה"סיסמה" שנתנו, ומוציא את ההודעה, אפשר לראות ב-Hidden Data את ה-"HelloWorld" שלנו שוב.

לסיום נדגים כתיבה ארוכה יותר, שדורשת כתיבה מרובה ל-rekor:



כאן אנחנו רואים שהכלי צריך להעלות כמה חתימות, וגם מחפש כמה וכמה פעמים, לפעמים הוא מוצא מאוד מהר (5 ניסיונות) ולפעמים כמעט אחרי 2000 ניסיונות, הוא רושם את כולם ל-rekor כמו שהדגמנו בפעם הקודמת.



אנחנו מבקשים ממנו לקרוא בחזרה, הכלי צריך לחפש את כל 8 החתימות, אבל לבסוף בונה מחדש את ההודעה שלנו.

סיכום

הדוגמאות והטכניקות שהוצגו לאורך הפוסט ממחישות עד כמה הדרכים לשימוש במנגנוני חתימה הן מגוונות ולעיתים גם מפתיעות. בפועל, הצלחתי להטמיע מידע מורכב מאוד, עד כדי הסתרה של VM שלם, בתוך חתימות לגיטימיות לחלוטין. המשמעות היא שניתן להעביר פקודות או מידע רגיש דרך תעודות וחתימות, מבלי שהצד הקורא ייגש ישירות ליעד חשוד, ולעיתים אף דרך דומיינים שנתפסים כ"בטוחים" בעיני רוב כלי האבטחה.

המסקנה המרכזית מהפוסט היא שחתימות, Certificate Transparency ושירותים כמו sigstore אינם רק מנגנוני הגנה, אלא גם תשתיות חזקות להעברת מידע. שימוש יצירתי בהן יכול לעקוף הנחות בסיסיות שמערכות אבטחה נשענות עליהן, ולהדגיש עד כמה חשוב לא רק לסמוך על "מי מדבר", אלא גם להבין איך ולשם מה התשתית מנוצלת.

על המחבר

שמי עומר מדן, עבדתי בכמה חברות סייבר, ולפני כן בתחום ה-embedded הרבה שנים.

מקורות מידע

- https://github.com/latedeployment/secertcert – כלי שכתבתי להדגמה בנושא
- https://datatracker.ietf.org/doc/rfc9162/ - RFC של certificate transparency 2
- https://certificate.transparency.dev/ - האתר של certificate transparency
- https://www.sigstore.dev/ - האתר של sigstore
- https://crt.sh/ - אתר לחיפוש חתימות
- https://latedeployment.github.io/posts/encrypting-and-chunking-data-in-rsa-keys/ - בלוג שכתבתי שמסביר איך המידע מוצפן בפועל