ממ"ן 16 – מטלה תכנותית

הקדמה

המטלה התכנותית שיצרתי היא תוכנה המאפשרת לחתום ולאמת קבצים מסוג ZIP בצורה דיגיטלית. כיום, קבצים מסוג zip אינם מאפשרים לחתום ולאמת נתונים הנשמרים בהם בצורה מאובטחת, לעומת PDF שבפורמט שלהם קיימת תמיכה בחתימה דיגיטלית המאפשרת אימות.

הפרויקט כולל בתוכו מימוש של תוכנה ושרת. התוכנה מאפשרת:

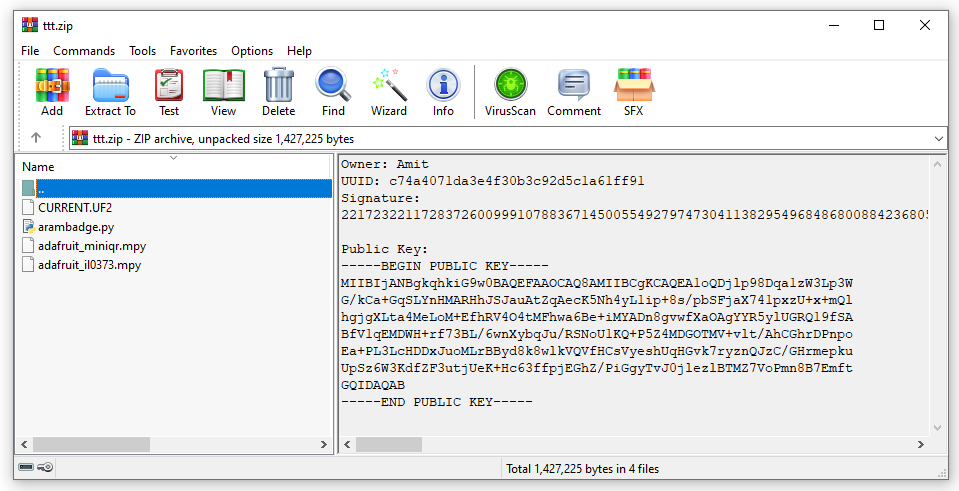
1. ליצור מפתחות RSA לחתימה על קובץ ZIP.
2. להרשם בשרת בתור הבעלים של אותם מפתחות, ובכך לקבל ממנו certificate.
3. לחתום על קובץ ZIP בעזרת certificate. נתוני החתימה ישמרו בקובץ בחלק של ה-comment.
4. לאמת קובץ ZIP חתום אשר יתקבל ממקור לא ידוע.
5. מחיקת certificate הרשום בשרת. הלקוח מספק את המפתח הפרטי שלו לשרת והשרת מוחק את ה-certificate, כך שאינו יהיה מאומת יותר.

השרת יספק לתוכנה את השירותים הבאים:

1. רישום של certificate. לקוח שולח לשרת את שמו ומפתח ציבורי, שבאמצעותו ניתן לאמת את התכנים עליהם הוא חותם, והוא נרשם בשרת כבעלים של אותו certificate.
2. אימות של certificate. כאשר לקוח מאמת קובץ ZIP, לאחר ווידוא החתימה הדיגטלית על תוכן הקובץ, הוא פונה לשרת על מנת שיוודא את זהות הבעלים (שהוא אכן הבעלים של ה-certificate).
3. מחיקת certificate מהשרת. עבור מקרה תיאורטי שבו המפתח הפרטי של מישהו נגנב, יש באפשרותו להזדהות מול השרת בעזרת המפתח הפרטי ובכך למחוק אותו מהשרת. בצורה כזו – כל קובץ שנחתם באמצעות ה-certificate של הבעלים לא יחשב לאמין יותר.

למעשה, אם עושים הקבלה לאיך שהאינטרנט עובד באמת: השרת הוא מעין CA שמנפיק תעודות דיגיטליות (רק שכאן אין אימות פיזי של האנשים, הראשון שמצהיר שהוא הבעלים של מפתח ציבורי מסוים מוגדר בתור הבעלים שלו). כאשר לקוח מעוניין לאמת קובץ ZIP (שקול לדוגמא לפנייה לאתר באמצעות TLS), אז הוא מוודא את החתימה הדיגיטלית של הקובץ, ומוודא מול השרת את תקינות התעודה – שאכן היא שייכת לבעלים הרשומים בה.

להלן תמונה של קובץ ZIP הפתוח בתוכנת WinRar אשר נחתם באמצעות התוכנה:



מימוש

התוכנה חותמת על קבצי ZIP באמצעות אלגוריתם RSA. משתמש התוכנה מזין את שמו (להלן Owner) ויכול ליצור מפתחות חדשים וכך התוכנה יוצרת עבורו מפתחות RSA בגודל 2048 ביט. את ה-Owner והמפתח הציבורי שנוצר המשתמש שולח לשרת. השרת מייצר מספר סידורי עבור המשתמש (להלן uuid). השילוב של owner, public key ו-uuid יוצר ביחד certificate אשר שמור בשרת ואצל הבעלים שיצר את המפתחות. הדבר היחיד שהשרת לא מחזיק בו הוא המפתח הפרטי שבאמצעותו הבעלים חותם על קבצים.

השרת שומר את כל ה-certificate-ים הרשומים ברשימה שהוא מעדכן בזמן ריצה ובנוסף מגבה אותה בזיכרון שאינו נדיף בקובץ השמור בפורמט json. התוכנה גם כן שומרת את כל ה-certificate-ים שנוצרו באותו מחשב ברשימה (כלומר, למשתמש שאינו יצר אף certificate לא שמור כלום), אשר מוצגת למשתמש וגם היא שמורה בקובץ json בתיקיית התוכנה.

כאשר בעלים של certificate מעוניין לחתום על קובץ ZIP, כל שעליו לעשות הוא לבחור את הקובץ ואת ה-Certificate שבאמצעותו הוא מעוניין לחתום. תהליך החתימה הוא לוקלי לחלוטין: התוכנה מחשבת hash מסוג sha384 על תוכן הקבצים שבתוך ה-ZIP ועל המבנה שבו הם מסודרים ולבסוף חותמת על ה-hash הזה. בסוף התהליך החתימה שמורה ב-section ה-comment של קובץ ה-zip (ראה תמונה בעמ' הקודם). החתימה הנ"ל מכילה את ה-owner, ה-uuid של ה-certificate, החתימה עצמה והמפתח הציבורי המאפשר לאמת את החתימה.

בהתאם, כאשר משתמש מעוניין לאמת חתימה על קובץ ZIP, הוא בוחר את הקובץ בתוכנה. התוכנה מחלצת מן הקובץ את ה-owner, ה-uuid, ה-signature וה-public key ומבצעת את אימות החתימה: מחושב hash מסוג sha384 בדיוק באותה הצורה שבה הוא חושב בתהליך החתימה, ומבוצע פענוח של ה-signature בעזרת המפתח הציבורי. לאחר מכן, ה-hash שחושב מושווה ל-hash המפוענח בחתימה ואם יש התאמה אז החתימה תקנית. אבל, שום דבר לא מבטיח שאף אחד לא שינה את ה-owner או ה-uuid. לכן, התוכנה פונית לשרת ומאמתת מולו את ה-certificate המכיל את ה-owner, ה-uuid וה-public key. אם אכן יש התאמה לכל הנתונים הללו, אז ה-certificate מאומת והקובץ כולו מאומת.

לבסוף, התוכנה מאפשרת לבעלים של certificate למחוק אותו (למקרה שהמפתח הפרטי נחשף ומישהו מזדהה לבעלים). במצב כזה הבעלים צריך לבחור את ה-certificate בתוכנה ולבקש מהשרת להסיר אותו. התוכנה תשלח לשרת את כל הפרטים של ה-certificate כולל ה-private key. השרת יאמת את הנתונים ויוודא שה-private key מתאים ל-public key המופיע ב-certificate. במידה והוא מתאים, ה-certificate יוסר מהשרת וכל משתמש אשר ינסה לאמת קובץ שנחתם עם אותו certificate יקבל דיווח מהשרת שה-certificate אינו תקין.

התקשורת בין השרת ללקוחות מתבצעת באמצעות פרוטוקול TLS 1.3. הדבר מבטיח שאף אחד לא יוכל להתחזות לשרת ולזייף certificate-ים. כיוון שהפרוטוקול דורש אימות של ה-certificate של השרת (certificate אמיתי, לא כזה שהתוכנה יוצרת) אז יצרתי self-signed certificate בעזרת openssl והתוכנה פשוט "סומכת" על ה-certificate שיצרתי. כמובן שבמציאות השרת היה צריך להרשם מול CA שיאמת אותו. ה-certificate של ה-ca הועתק לקבצי התוכנה והיא טוענת אותו כאשר מתבצע חיבור מול השרת, אשר דורש אימות ע"י ה-certificate של ה-ca.

מודולים ומחלקות

התוכנה והשרת נכתבו בשפת Python בגרסא 3. התוכנה והשרת הם שתי יחידות נפרדות אשר חולקות חלק מן המודולים ששניהם זקוקים להם.

מודולים המשותפים לתוכנה ולשרת:

* messages – מודול המגדיר את פרוטוקול התקשורת בין התוכנה לשרת ואת פורמט ההודעות המועברות בין השרת ללקוחות. משתמש בספריה construct המאפשרת לבצע serialization.
* rsa – מודול האחראי על ביצוע חתימה ואימות באמצעות RSA. נעזר בספריה Crypto, רק בשביל לפרסר קבצים בפורמט PEM.

מודולים השייכים לתוכנה:

* communication – מודול האחראי לכל התקשורת המתבצעת מול השרת. הוא בונה את ההודעות בעזרת המודול messages ושולח אותן לשרת, ובנוסף מקבל את התגובה מהשרת ומפרסר אותה.
* certificate – מכיל מחלקה אשר מייצגת certificate. אובייקט במחלקה את ה-owner, המפתח (ציבורי+פרטי או רק ציבורי), וה-uuid. נעזר במודול rsa כדי לבצע חתימה ואימות בעזרת המפתחות. נעזר בספריה Crypto בשביל ליצור מפתחות RSA.
* signed\_zip – מחלקה אשר עוטפת קובץ מסוג zip ומאפשרת לחתום / לוודא אותם. המחלקה מאפשרת לחשב hash המייצג את תוכן ה-zip והמבנה שלו ובנוסף לשמור ולפרסר certificate המופיע ב-section ה-comment שלו.
* ZipSigner – המודול הראשי של התוכנית. מכיל את ממשק המשתמש אשר נבנה באמצעות הספריה PySimpleGui.

מודול השייכים לשרת:

* request\_handler - מודול האחראי על טיפול הבקשות מהמשתמש. הוא מפרסר את הבקשה, מזהה את סוגה, ומטפל בה. נעזר במודול messages.
* client\_handler – מודול האחראי על הטיפול בלקוחות. הוא מקבל את הודעות הלקוח, מוודא שהן תקינות, מעביר אותן לטיפול ב-request\_handler ומחזיר ללקוח את התשובה המתאימה.
* server – מודול המממש שרת TLS. הוא יוצר את החיבור עם כל לקוח ויוצר עבורו process לטיפול בו.
* main – המודול הראשי שאיתו מריצים את השרת.

התקנה ראשונית

התקנת השרת:

1. אם עדיין לא מותקן פייתון בגרסא 3 (נבדק על גרסא 3.8.2), אז צריך כמובן להתקין.
2. להתקין את ה-requirements של השרת (כל הפקודות מתבצעות בתיקייה הראשית):

* pip install -r common/requirements.txt
* pip install -r server/requirements.txt

1. (אופציונאלי) להתאים את הפורט של השרת בקובץ server.py בשורה 10. (דיפולטי: 12345)
2. אם לא קיימים certificate-ים בתייקיה server/certificates (או אם רוצים ליצור חדשים):
   1. יש לוודא את קיומם של התוכנות make ו-openssl.
      * ב-windows: ניתן להתקין make בעזרת choco: choco install make
      * Openssl הותקן אצלי אוטומטית עם git. מאמין שלא אמורה להיות בעיה.
   2. בתיקייה server/certificates יש לבצע את הפקודה make generate.
   3. יש להעתיק את הקובץ server/certificates/ca.crt לתיקייה ZipSigner.
3. להריץ את השרת בעזרת: python -m server.main

התקנת התוכנה:

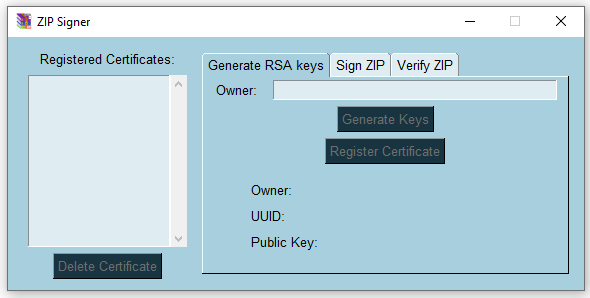
1. אם עדיין לא מותקן פייתון בגרסא 3 (נבדק על גרסא 3.8.2), אז צריך כמובן להתקין.
2. צריך להתקין tkinter (ב-Windows מתקינים אותו עם ה-installer של python, ב-linux עם apt).
3. להתקין את ה-requirements של התוכנה (כל הפקודות מתבצעות בתיקייה הראשית):

* pip install -r common/requirements.txt
* pip install -r ZipSigner/requirements.txt

1. לוודא שה-certificate של ה-ca קיים ב-ZipSigner/ca.crt.
2. להתאים את ה-IP וה-PORT של השרת בקובץ communication.py בשורות 9,10.
3. להריץ את התוכנה בעזרת: python -m ZipSigner.ZipSigner

מדריך שימוש

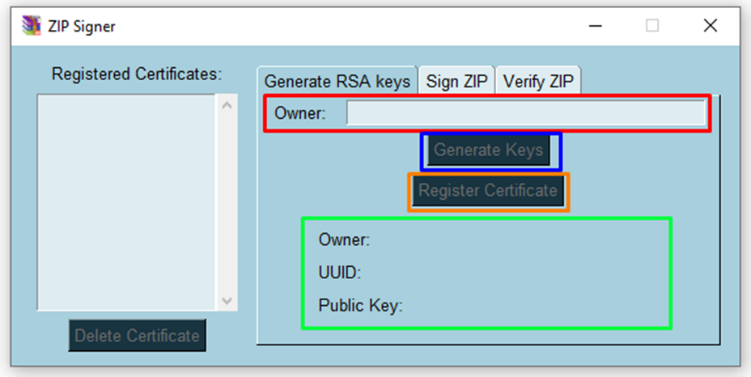
בהרצת התוכנה, ייפתח החלון הבא:



התוכנה מכילה 3 חלונות, בכל חלון ניתן לבצע פעולה שונה בעזרת התכונה.

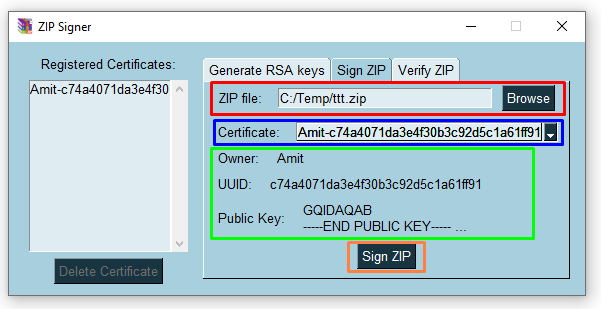
בחלון Generate RSA keys ניתן לייצר מפתחות RSA חדשים ולרשום אותם בשרת בתור certificate:

1. בתיבת ה-Input בצבע אדום יש להכניס את הבעלים של ה-certificate.
2. לאחר מכן, יש ללחוץ על הכפתור הכחול בשביל לייצר את המפתחות.
3. לאחר ייצור המפתחות, יש ללחוץ על הכפתור הכתום בשביל להרשם בשרת בתור הבעלים של אותם מפתחות (רק המפתח הציבורי למעשה) ולקבל ממנו certificate.
4. לבסוף, פרטי ה-certificate יופיעו באיזור הירוק שכולל owner, uuid, ו-public key. יתר על כן, ניתן יהיה לראות את ה-certificate מתווסף לרשימה מצד שמאל.

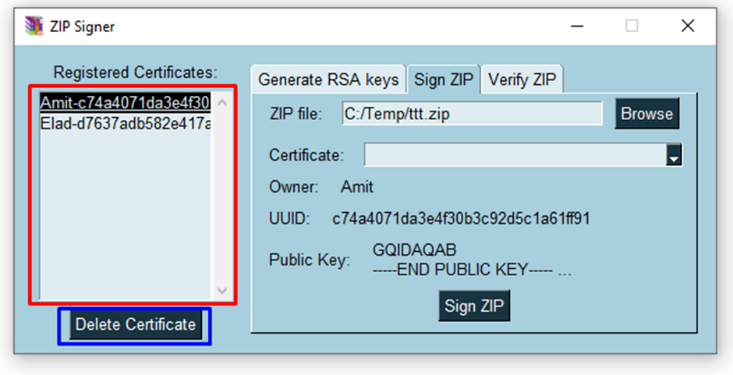


בחלון Sign ZIP ניתן לחתום על קובץ zip בעזרת certificate שכבר נרשם בשרת ע"י אותו מחשב:

1. לבחירת קובץ ה-zip, יש ללחוץ על הכפתור Browse בתיבה האדומה בשרטוט.
2. לאחר מכן, יש לבחור מהרשימה בכחול את ה-certificate שבאמצעותו נחתום.
3. פרטי ה-certificate יופיעו באיזור הירוק שכולל owner, uuid, ו-public key.
4. לבסוף, בשביל לחתום על ה-zip בעזרת ה-certificate יש ללחוץ על הכפתור הכתום.

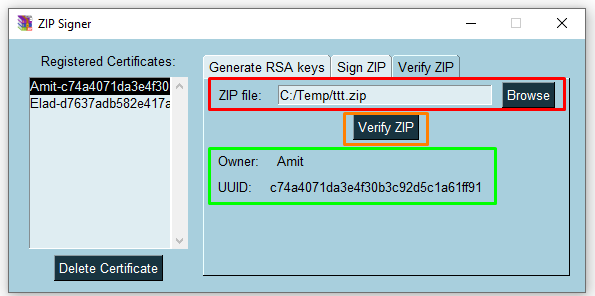


בכל חלון בתוכנה מופיע מצד שמאל (אדום) רשימה של כל ה-certificate-ים אשר נחתמו בשרת ע"י אותו מחשב. ה-certificate-ים רשומים בפורמט “owner-uuid” על מנת לזהות אותם. ניתן לבחור certificate מהרשימה ולאחר מכן ללחוץ על הכפתור הכחול בשביל להסיר אותו מהשרת.



בחלון Verify ZIP ניתן לאמת קובץ zip אשר נחתם ע"י מישהו אחר (ה-certificate שבעזרתו חתמו על הקובץ צריך להיות רשום בשרת):

1. לבחירת קובץ ה-zip, יש ללחוץ על הכפתור Browse בתיבה האדומה בשרטוט.
2. לאחר מכן, בשביל לאמת את ה-zip יש ללחוץ על הכפתור הכתום.
3. אם החתימה של ה-zip תקינה, תופיע הודעת הצלחה ופרטי ה-certificate יופיעו באיזור הירוק שכולל owner, uuid, ו-public key.
4. אם החתימה לא תקינה, בין אם ה-Signature הרשום בקובץ אינו תואם את תוכנו ובין אם ה-certificate שבאמצעותו חתמו על הקובץ אינו תקין, תופיע הודעת שגיאה מתאימה.



דוגמאות הרצה

את הקוד השלם של הפרויקט ניתן למצוא ב-github: <https://github.com/amit-sides/SignedZip>

לנוחיותך, אצרף אותו גם פה וגם בהגשה עצמה: 

אני מאמין שההסברים מעלה מספיק מובנים כדי שתבין איך להריץ ולהשתמש בתוכנה, אבל בשביל שתשתמש בתוכנה לשם מטרה שהיא באמת נועדה אליה, הכנתי בעצמי קובץ ZIP וחתמתי עליו באמצעות ה-certificate שלי. אתה אומנם לעולם לא תגלה את המפתח הפרטי שלי, אבל אם תשים את הקובץ certificates.json, המוצרף מטה, בתיקייה ZipSigner/server (בשביל לדמות כאילו שנינו מתקשרים עם אותו שרת, צריך את זה בשביל שה-certificate שלי יהיה רשום בשרת שלך) אז תוכל לאמת שאני אכן חתמתי על הקובץ mmns.zip, שגם הוא מצורף מטה:



