

פרויקט הגנת סייבר

Cyber Safe



מגיש: רון קנטורוביץ'

ת.ז.: 207733015

בית ספר: תיכון ליאו-בק

מקצוע: הגנת סייבר

שמות המנחים:

- שרית לולב

- אלון בר-לב

	תוכן עניינים
1	תוכן עניינים
2	מבוא
3	ארכיטקטורה
4	הצפנות
5	הרקע התאורטי
5	מושגי תקשורת
5	מושגי מערכות הפעלה
6	שפות תכנות ומושגי הנדסת תוכנה
6	מושגי הצפנה
7	מימוש
7	מבנה הדיסק
8	מבנה File Entry
10	אופן פעולת השירותים
16	פרוטוקול תקשורת
18	פרמטרים וסוגי בקשות
20	בעיות ידועות
21	התקנה ותפעול
21	התקנה
23	הרצה
26	גישה של לקוח למערכת
30	תכניות עתיד
30	פרק אישי
31	קוד פרויקט
32	נספח א' - Sequence Diagram Source

מבוא

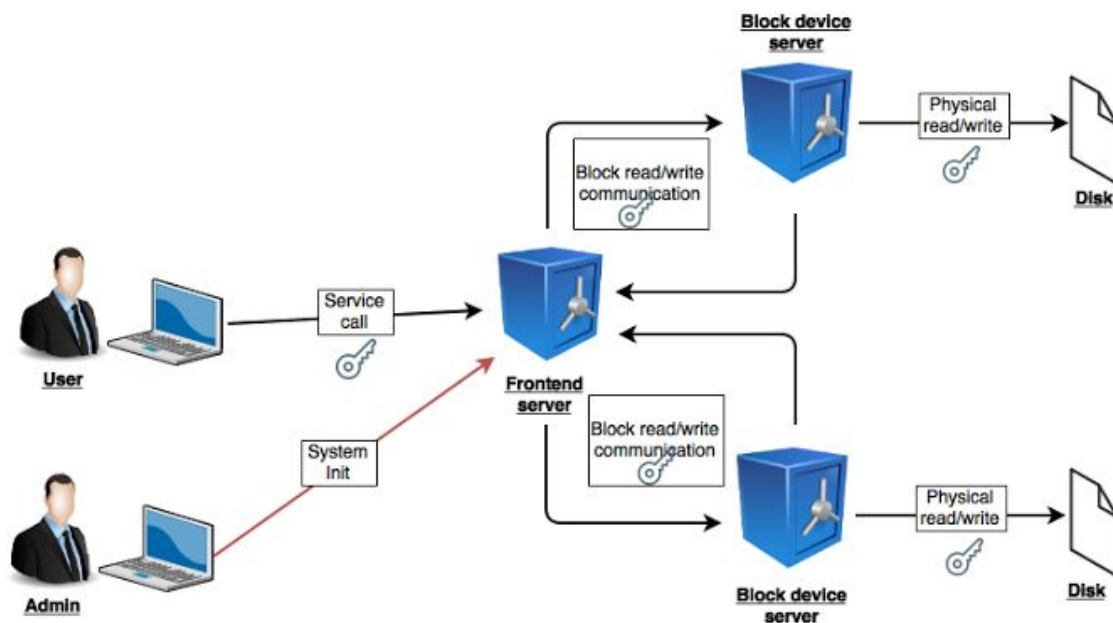
פרויקט זה נועד לספק אבטחת מידע במערכת מחשבים מרובת לקוחות. הפרויקט הוא מערכת המאפשרת למשתמשיה לאחסן קבצים במבנה נתונים דמוי-כספת באופן שאינו מאפשר לגורמים אחרים לצפות בתוכן הרגיש שבקבצים. הפרויקט הוא פרויקט כספת מבוצרת אשר מחולקת בין מספר שרתים, כאשר לכל שרת תפקיד שונה בעבודה עם תוכן הכספת ועם הצפנתו, באופן הבא:

המערכת מכילה שרת Frontend שמקבל בקשות לקוחות בפרוטוקול HTTP באופן אסינכרוני. הבקשות מטופלות על ידי השרת בעבודה אל מול מספר שרתי Block Device באופן שיתואר בהמשך מסמך זה.

לקוחות מסוגלים לקרוא למספר שירותים שונים בהתאם לצורכיהם ולהרשאותיהם:

- שירות העלאת קובץ והצפנתו במפתח משתמש
 - שירות הורדת קובץ שבבעלות המשתמש
 - שירות צפייה ברשימת קבצים שתואמים את מפתח המשתמש
 - שירות מחיקת קובץ שבבעלות המשתמש
 - שירות אתחול המערכת - ניתן לקריאה רק על ידי משתמש בעל סיסמת אדמין
- כאשר כל משתמש מספק מפתח אישי בעת קריאה לשירות, ובאמצעותו נוספת שכבת הצפנה לקבצי המשתמש.
- המערכת מלווה בממשק משתמש נוח וקל לשימוש.

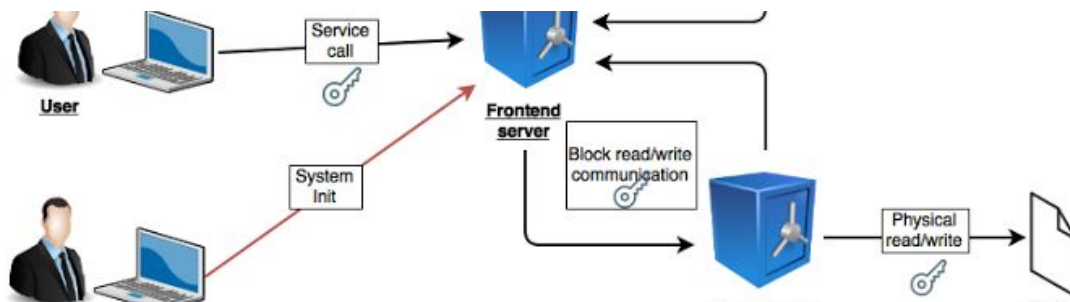
ארכיטקטורה



הקבצים הנשלחים על ידי משתמש עוברים דרך מספר שרתים, אותם ניתן לראות בדיאגרמה שלעיל, כאשר לכל שרת תפקיד שונה בהצפנה ובתהליך האבטחה של הקובץ:

- **שרת ה-Frontend** הוא הנקודה הראשונה, אליה מגיע כל קובץ וכל בקשה של המשתמש בפרוטוקול HTTP. ה-Frontend בודק את סוג הבקשה ואת הרשאות המשתמש באמצעות המפתח האישי שסיפק ומחזיר הודעת שגיאה במקרה של בקשה לא חוקית. בשירות העלאת קובץ, תוכן הקובץ מחולק לבלוקים בגודל קבוע של 4096 בתים לקראת שלב העבודה מול שרתי הבלוקים.
- **שרתי בלוקים** הם שרתים שמקבלים מה-Frontend בקשות בפרוטוקול HTTP לקריאה וכתיבה של בלוקים לדיסק פיזי שמצוי אצל כל שרת בלוקים. הבלוקים מאוחסנים בדיסק לפי מערכת קבצים (File System) שתואר בהמשך. שרתי הבלוקים מכילים מנגנון הזדהות שלא מאפשר לגורמים שאינם שרת ה-Frontend לשלוח בקשות קריאה וכתיבה.

הצפנות



לצורך אבטחת מידע גבוהה של המידע השמור בשרתי הבלוקים, פרויקט זה משתמש ב-4 שכבות של הגנה על סוגים שונים של בלוקים. שכבות ההגנה מכילות 3 הצפנות AES ושכבה אחת של אלגוריתם פיצול ביטים. סדר ההצפנות מתואר בדיאגרמה שסופקה לעיל:

- הצפנת Block Device – כל בלוק שנכתב לשרתי הבלוקים עובר הצפנה. מפתח ההצפנה נקבע מראש בקובץ הקונפיגורציה של כל שרת בלוקים, וה-IV נגזר באמצעות פונקציית hash על מקור שנקבע מראש ועל מספר הבלוק.
 - הצפנת User Key – הצפנה במפתח משתמש שכל בלוק עובר ב-Frontend לפני שהוא נשלח לשרתי הבלוקים. מפתח ההצפנה נגזר באמצעות hash על הסיסמה שסיפק המשתמש בעת גישה למערכת, וה-IV מוגרל באופן אקראי ומצורף לתחילת הבלוק לצורך פענוח עתידי. בהצפנה זו מוצפנים אך ורק בלוקים של תוכן קובץ, שכן אין משתמש שבבעלותו נמצאים הבלוקים ששייכים לתפקוד המערכת, דוגמת ה-Bitmap.
 - הצפנת Frontend – לפני שליחה של בלוק לשרתי הבלוקים, כל בלוק עובר הצפנה על ידי שרת ה-Frontend. בדומה להצפנה על ידי Block Device, מפתח ההצפנה הוא מפתח שנקבע מראש בקובץ הקונפיגורציה של השרת, וה-IV נגזר באמצעות פונקציית hash על מקור קבוע ועל מספר הבלוק.
 - אלגוריתם פיצול ביטים – אלגוריתם זה מתבצע בשרת ה-Frontend לפני שליחה של כל בלוק, ומבטיח שכל גורם חיצוני עדיין שמעוניין לקרוא את המידע השמור, יצטרך לפרוץ לכל אחד משרתי הבלוקים. כל בלוק ששרת ה-Frontend מעוניין לכתוב למערכת מפוצל באופן אקראי למספר בלוקים התואם את מספר שרתי הבלוקים. עבור כל ביט בתוכן הבלוק, המערכת מגרילה מספר אקראי שקובע לאיזה עותק של הבלוק להעתיק את הביט. בשאר העותקים שמים באותו מקום ביט 0. לאחר פיצול כל הביטים לעותקים, על כל עותק בלוק מבוצעת פעולת xor עם בלוק אקראי על מנת להגביר את האבטחה של המידע. תהליך פיצול זה מכריח לבצע פעולת xor של כל עותקי הבלוק המפוצל על מנת לקבל את הבלוק המידע המקורי.
- בעת קריאה של בלוק מהמערכת, על ה-Frontend לקרוא את כל עותקי הבלוקים מכל שרתי הבלוקים, ולאחר מכן לאחד אותם.

הרקע התאורטי

מושגי תקשורת

- [שרת](#) – תוכנה שתפקידה לרוץ באופן ממושך ולספק שירותים לתכנות לקוח באמצעות תקשורת רשתות.
- [socket](#) – נקודת קצה עבור זרם נתונים בתקשורת בין תהליכים על גבי רשת מחשבים.
- [כתובת IP](#) – מספר ייחודי לכל נקודת קצה בתקשורת רשתות שעושה שימוש בפרוטוקול התקשורת IP, שמעניק מזהה לכל נקודה כזו.
- [Port](#) – תהליך ספציפי שדרכו תכנות מסוגלות להעביר נתונים באופן ישיר. בפרויקט זה נעשה שימוש בפורטים על מנת לאפשר לשרת לנהל מספר חיבורים במקביל עם לקוח.
- [פרוטוקול HTTP](#) – פרוטוקול תקשורת שנועד לאפשר תקשורת בין שרתים לבין תכנות דפדפן והעברת בקשות ודפי HTML. בקשת HTTP מורכבת ממספר חלקים, בהם שיטת הבקשה, שדות כותרת וגוף הבקשה.
- [Cookie](#) – עוגיה היא מחרוזת המוענקה למשתמש על ידי השרת ונשלחת יחד עם בקשות עתידיות של המשתמש לצורך שמירה של נתונים חשובים יחודיים למשתמש. בפרויקט זה נעשה שימוש ב-Cookie לצורך הזדהות של הלקוח בפני שרת ה-Frontend.
- [MAC](#) – קוד אימות מסרים, זהו קוד שמיוצר באמצעות אלגוריתם בהינתן מידע באורך שרירותי. הקוד המיוצר הוא בדרך כלל באורך קצר, ושליחתו לצד המידע מאפשרת אימות מהיר ויעיל של שלמות המידע שנשלח.
- [HMAC](#) – מנגנון MAC מיוחד אשר מבוסס על פונקציית גיבוב. בנוסף לתוכן מידע, מסופק למנגנון גם מפתח, ועליו מבוצעות פעולות גיבוב ביחד עם המידע, כך שרק באמצעות המפתח המקורי ותוכן המידע המקורי ניתן לאמת את שלמות המידע ולהגיע לתוצאת HMAC זהה למקור.

מושגי מערכות הפעלה

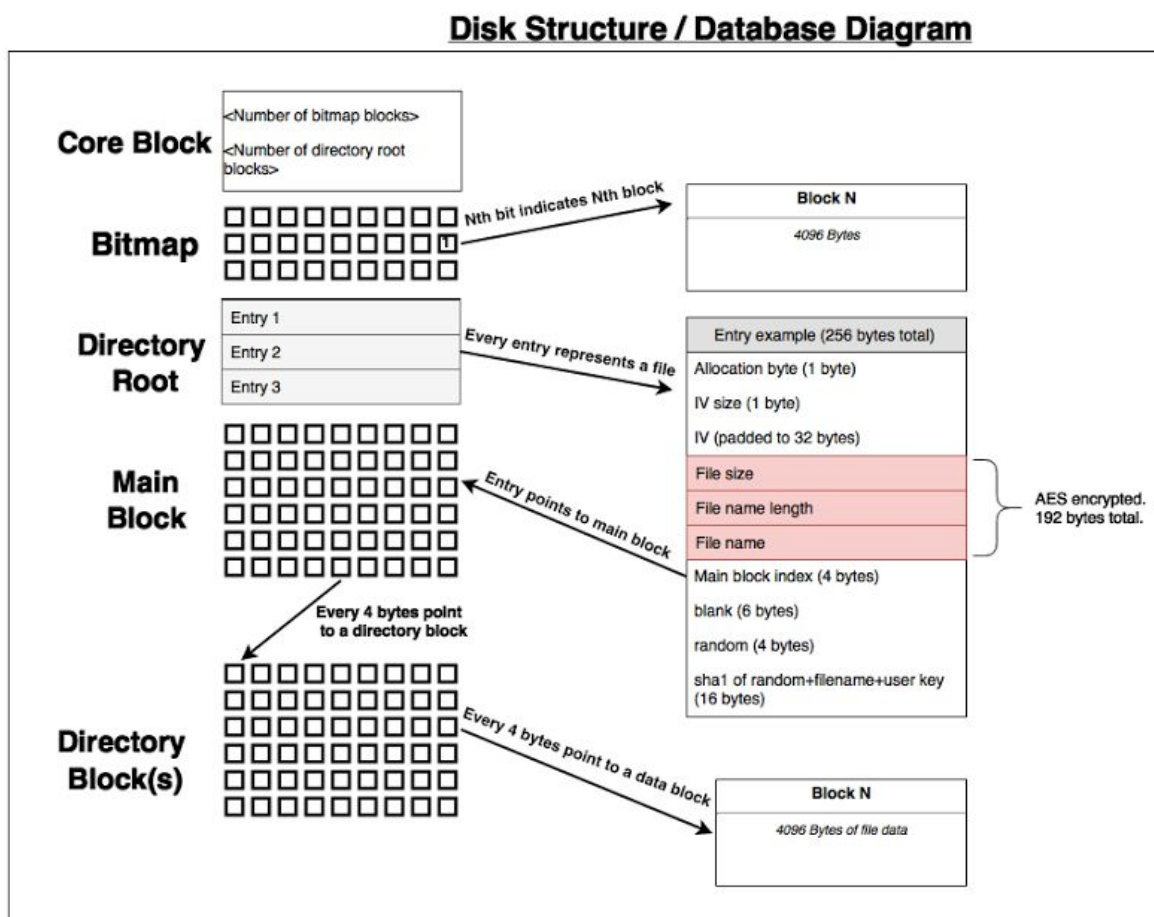
- [תקשורת אסינכרונית](#) – סוג תקשורת בין שני גורמים המאפשר לאחד מהם או לשניהם להריץ תהליכים נפרדים לצד תהליך התקשורת, לפני שהתהליך המקורי הסתיים.
- [POSIX](#) – אוסף תקנים בסיסיים שנאגד במטרה לשמור על תאימות בין מערכות הפעלה, בעיקר בין מערכות מבוססות UNIX.
- [daemon](#) – תכנית מחשב שרצה כתהליך ברקע ואינה נמצאת בשליטה של אינטרקציה עם משתמש. בפרויקט זה קיימת תמיכה של הרצת תכניות השרתים בתור תכניות daemon.

שפות תכנות ומושגי הנדסת תוכנה

- תכנות מונחה עצמים – שיטת תכנות המתבססת על חלוקת התכנית לאובייקטים בעלי תכונות ופונקציות נפרדות. בשיטת תכנות זו, למשל, יהיה קיים אובייקט נפרד לתיאור כל רכיב תקשורת וכל שירות שהרכיב מספק.
- תכנות מונחה אירועים – סגנון תכנות המבוסס על הרכבת תכנית ממספר גורמים הממתינים לקבלת אות לצורך "התעוררות" וביצוע פונקציות. בפרויקט זה האובייקטים הממתינים הם חיבורי הרשת בין הגורמים השונים, למשל חיבור של שירות הורדת קובץ שממתין לאירועי קריאה של בלוקים יחידים מהכספת על מנת להתעורר ולעבד אותם.
- Context-Based Programming – תכנות שמבוסס על הרכבת מבנה נתונים מרכזי בתכנת, Context, והעברה שלו בין מחלקות וגורמים שונים במהלך הריצה. פרויקט זה מבוסס על סוג זה של תכנות באמצעות יצירה של מבני נתונים מסוג מילון dictionary.
- [קובץ log](#) – קובץ שמטרתו לתעד את האירועים ואת השגיאות שמתרחשים במהלך ריצה של תכנית. פרויקט זה תומך בהגדרת מסמך log, אליו נכתבים פרטים של חיבורי רשת שנפתחים ונסגרים במהלך הריצה, כמו גם תיעוד של שגיאות אפשריות.
- [Python](#) – שפת תכנות נפוצה ששמה דגש על קריאות קוד והרכבת תכניות ומבני נתונים מסובכים בדרך קצרה ופשוטה. בשפה זו נכתב רוב הפרויקט.
- [HTML](#) – שפה מבוססת תגיות המאפשרת עיצוב של דפי אינטרנט שתצוגתם נתמכת בדפדפן.
- [CSS](#) – פורמט לעיצוב דפי אינטרנט.

מושגי הצפנה

- [הצפנה](#) – תהליך עיבוד מידע שמטרתו לאבטח את המידע ולהפוך אותו לנגיש לגורמים מסוימים בלבד.
- [הצפנה בצופן סימטרי](#) – סוג אלגוריתם הצפנה שעושה שימוש במפתח זהה לצורך הצפנת מידע ולצורך פענוח בצורה שבה רק מחזיק המפתח מסוגל לגשת למידע המקורי.
- [צופן בלוקים](#) – סוג הצפנה שעובדת על מחרוזת באורך קבוע הנקראת בלוק ומבצעת עליה טרנספורמציה קבוע. פענוח הבלוק המוצפן מתבצע באמצעות אופן דומה, כאשר האלגוריתם הפענוח מקבל את הבלוק המוצפן ואת מפתח ההצפנה המקורי.
- [הצפנת AES](#) – הצפנת בלוקים סימטרית נפוצה שנחשבת לבעלת יכולת אבטחה גבוהה. בהצפנה זו נעשה שימוש בפרויקט, והיא מצפינה בלוקים באורך קבוע באמצעות מפתח ובאמצעות וקטור אתחול (IV) שנתון מראש. בכל מנגנון הצפנה בפרויקט זה משתמש ב-AES המפתח והוקטור נגזרים באופן ייחודי, והאופנים השונים מוסברים בפרק על הצפנות במסמך זה.
- [Hash Function](#) – פונקציית גיבוב, פונקציה שממירה פלט חופשי לקלט באורך קבוע. בפרויקט זה נעשה שימוש בפונקציית גיבוב מסוג sha1.



לצורך ארגון יעיל ופשטני של הדיסק אליו מועלה התוכן על ידי המשתמש, בשרתי הבלוקים מומשה מערכת הקבצים הבאה, המורכבת ממספר חלקים לכל דיסק ומפורטת גם בדיאגרמה שלעיל:

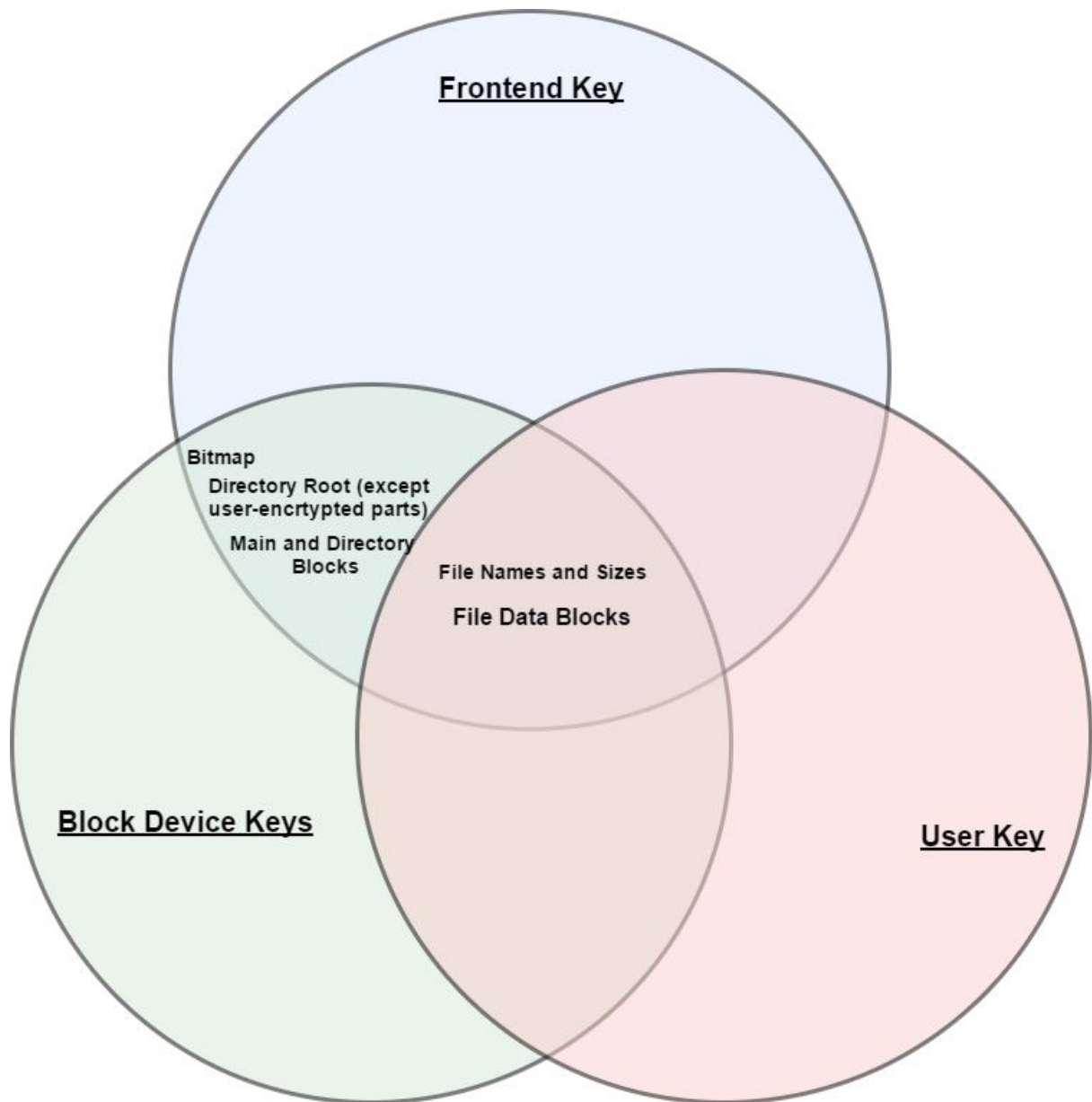
- **Core Block** - בתחילת כל דיסק נמצא בלוק אחד המכיל בתוכו מידע על הדיסק. בעת אתחול המערכת על ידי האדמין, שירות האתחול כותב לבלוק זה את כמות בלוקי ה-**Bitmap** וה-**Directory Root** שמאותחלים בדיסק.
- **Bitmap** – "מפת הביטים" היא מספר מוגדר מראש של בלוקים (של 4096 בטים), בהם כל ביט באינדקס כלשהו מתייחס למצב הבלוק בדיסק שמקומו זהה. לדוגמה, כאשר המערכת תרצה לכתוב לבלוק ה-1000 בדיסק, היא תבדוק לפני כן את מצבו של הביט ה-1000 במפת הביטים. אם הביט כבוי (0) אז הבלוק המתאים בדיסק ריק, וניתן לכתוב אליו, ואם הביט דולק (1), אז הבלוק תפוס.
- **Directory Root** – "שורש הדיסק" מורכב ממספר מוגדר מראש של בלוקים, ובחלק זה נשמר מידע כללי על כל קובץ ששמור במערכת, בצורה של Entry בגודל 256 בטים עבור כל קובץ. Entry של קובץ מכיל מידע כגון שם הקובץ, גודל הקובץ והרשאות גישה לקובץ, ומידע זה יפורט בהמשך במלואו. כל Entry של קובץ מכיל הפניה לבלוק הראשי של הקובץ, **Main Block**.
- **Main Block** – הבלוק הראשי של הקובץ מכיל הפניות לבלוקי המידע של הקובץ. כל הפניה היא אינדקס בגודל 4 בטים, ולאחר ההפניה האחרונה מרופד הבלוק הראשי עד סופו בהפניות אקראיות

- על מנת להסתיר את גודל הקובץ האמיתי. לכל קובץ קיים בלוק ראשי יחיד, בהבדל מבלוקי המידע שמשרתים מטרה דומה אך אינם יחידים לכל קובץ.
- Directory Block – כל בלוק מידע של קובץ מכיל הפניות לבלוקים בדיסק שבהם כתוב תוכן הקובץ. כל הפניה היא בגודל 4 בתים ובלוק המידע מרופד בסופו בהפניות אקראיות בדומה לאופן ריפוד הבלוק הראשי.
- Data Block – בלוק תוכן. בלוקים אלה מרכיבים עבור כל קובץ את התוכן עצמו. כל בלוק תוכן מכיל 4080 בתים של תוכן מהקובץ, בצירוף 16 בתים של initial vector שמשמש להצפנת הבלוק, ועל כך יוסבר בהמשך באופן מפורט.

מבנה File Entry

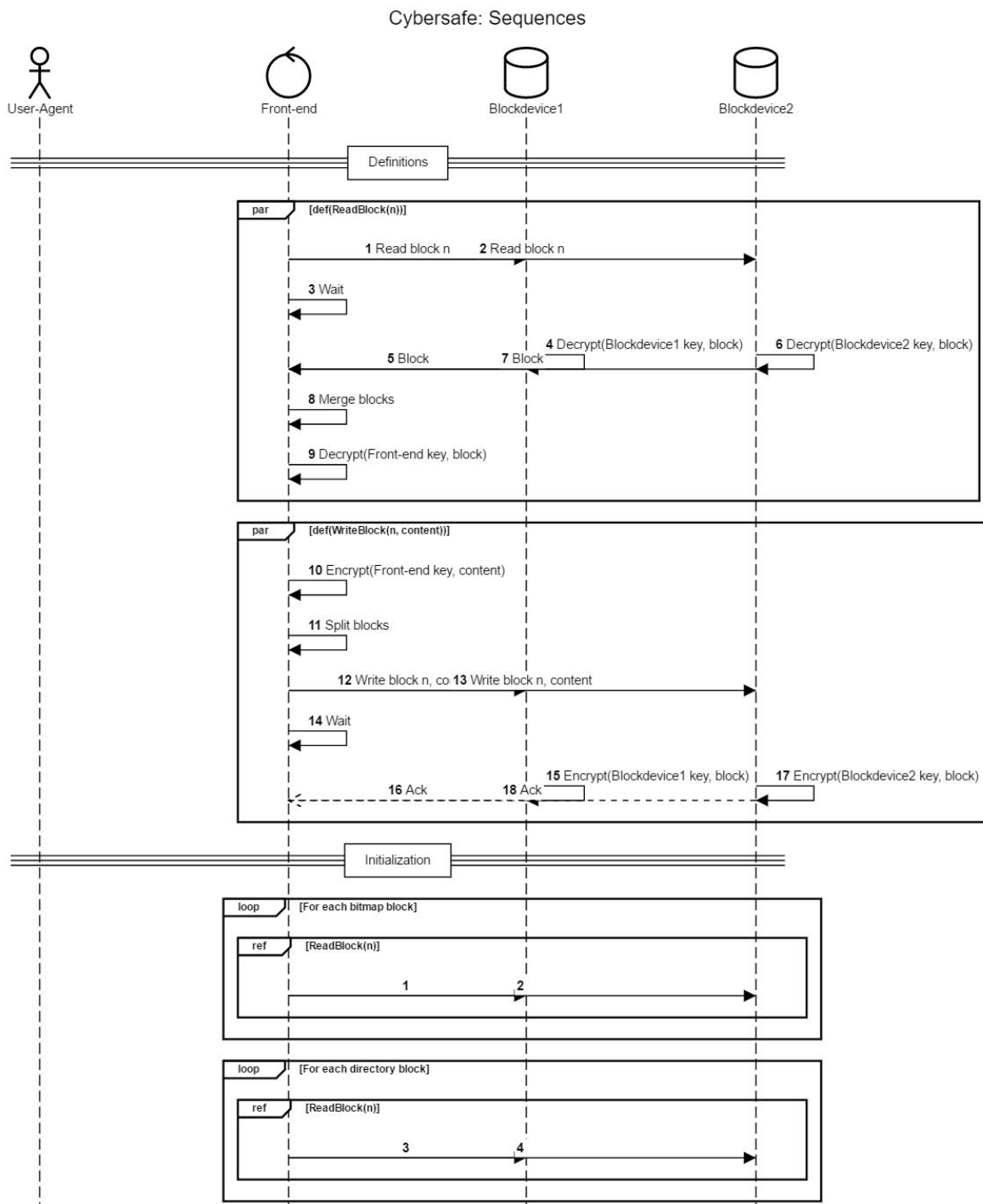
- כאמור לעיל, ה-Directory Root, שורש הדיסק, מורכב מחלקים בגודל 256 בתים, וכל אחד מהחלקים האלה נקרא Entry ומתייחס לקובץ ששמו במערכת. לשם אבטחה מיטבית של המערכת ושל הקבצים, מבנה כל Entry הוא מורכב, ולכן יפורט באופן מילולי כאן ובאופן מומחש בדיאגרמה מצורפת לעיל:
- Allocation Byte – בית אחד בתחילת כל Entry מקבל את הערך 0 או 1. אם הערך הוא 0, סימן שה-Entry ריק, ואם הערך 1 אז ה-Entry מלא. הדבר מאפשר מחיקה יעילה של Entry כשעולה הצורך.
- IV Size – בית אחד, גודל שדה ה-iv ב-Entry (יוסבר בהמשך).
- IV – שדה מרופד עד 32 בתים המכיל את הווקטור ההתחלתי של ההצפנה. בכל Entry קיים שדה עם מידע רגיש, אותו יש להצפין בהצפנת AES, אליה יש לספק מפתח משתמש וווקטור התחלתי. בעוד מפתח המשתמש מסופק בכל קריאה למערכת, הווקטור ההתחלתי מוגרל באופן אקראי ונשמר בשדה הנוכחי לשימוש עתידי.
- Encrypted Field – זהו שדה מוצפן בגודל של 192 בתים לאחר ההצפנה המכיל את המידע הרגיש של הקובץ. שדה זה מכיל את גודל הקובץ, גודל שם הקובץ ואת שם הקובץ. על מנת להגן על מידע זה מפני איומים חיצוניים אפשריים, שדה זה מוצפן בהצפנת AES עם מפתח שמסופק על ידי המשתמש ובאמצעות ה-IV שנשמר בשדה הקודם.
- Main Block Index – הפניה בגודל 4 בתים לבלוק הראשי של הקובץ.
- Blank – שישה בתים שנותרו ריקים לצורך שימוש עתידי אפשרי ולצורך הפיכת גודל ה-Entry לגודל שמחלק את 4096 (גודל בלוק) ללא שארית.
- Random – שדה אקראי בגודל 4 בתים.
- Sha – שדה זה אורכו 16 בתים, והוא מכיל hash מוג sha1 של שם הקובץ, מפתח המשתמש והערך האקראי בשדה הקודם. בהינתן שם קובץ ומפתח משתמש, המערכת מבצעת עליהם hash ומשווה לערך זה על מנת לקבוע באופן מהיר האם ה-Entry הנוכחי הוא זה שמתאים לשם הקובץ.

להלן דיאגרמה הממחישה את מעגלי ההגנה של הכספת. כל מעגל מסמל אחד משלושת מפתחות ההצפנה: מפתח ה-Frontend, מפתחות ה-Block Device ומפתח המשתמש. באזורי החיתוך של המעגלים נמצאים חלקי הדיסק אותן ניתן לפענח בהינתן שילוב של המפתחות המתאימים. כפי שניתן לראות, אל המידע הרגיש והחשוב ביותר במערכת ניתן לגשת אך ורק על ידי שילוב כל המפתחות. במקרה שבו גורם חיצוני עוין מצליח לפרוץ מעגל ההגנה, הדבר עדיין לא מספיק על מנת לפענח כל סוג של מידע מהכספת:



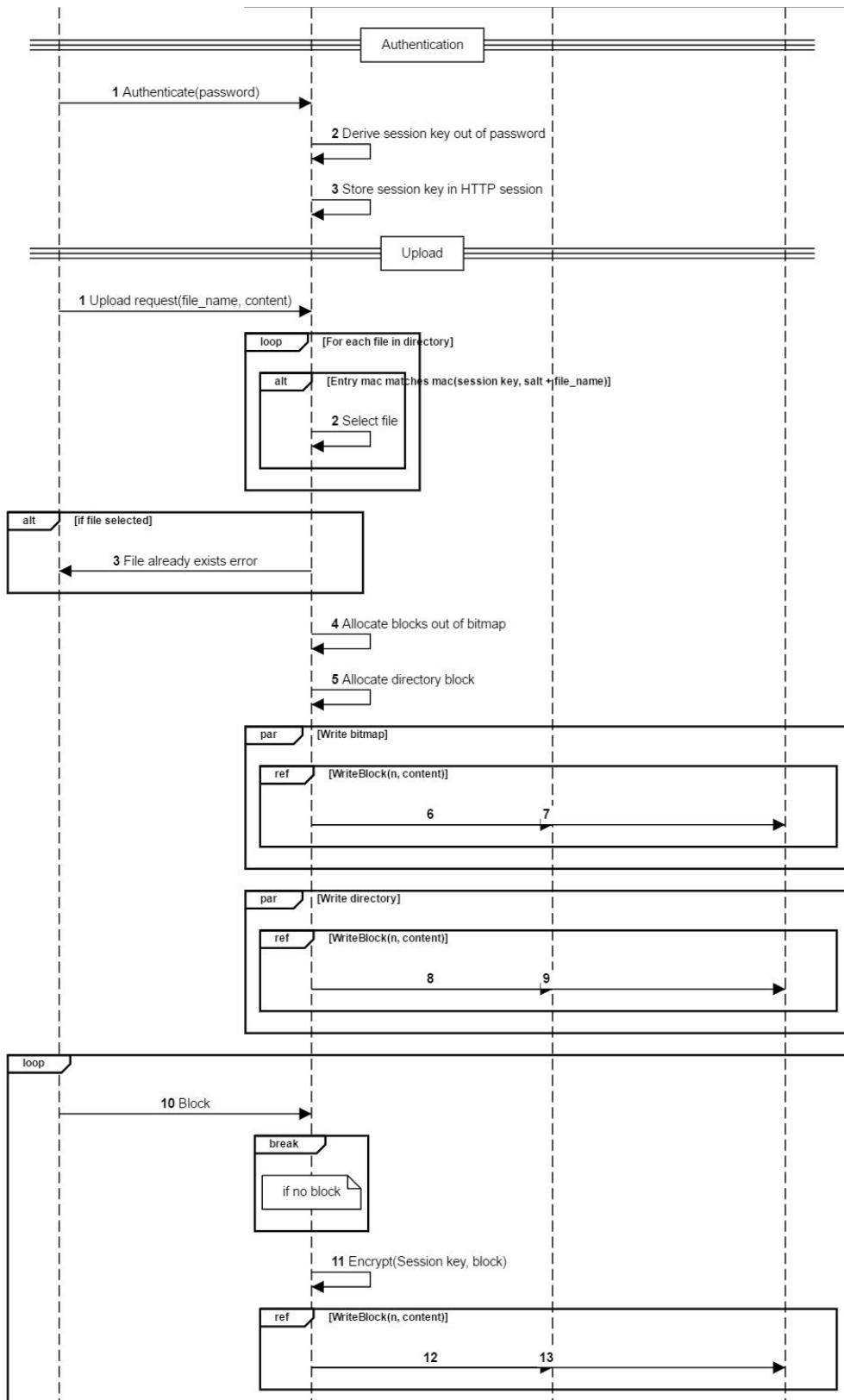
אופן פעולת השירותים

על מנת להמחיש את תפקוד מבנה זה, להלן תיאור פעולתם של חמשת השירותים המרכזיים בפרויקט, כפי שהם מומחשים בדיאגרמות שלהלן:



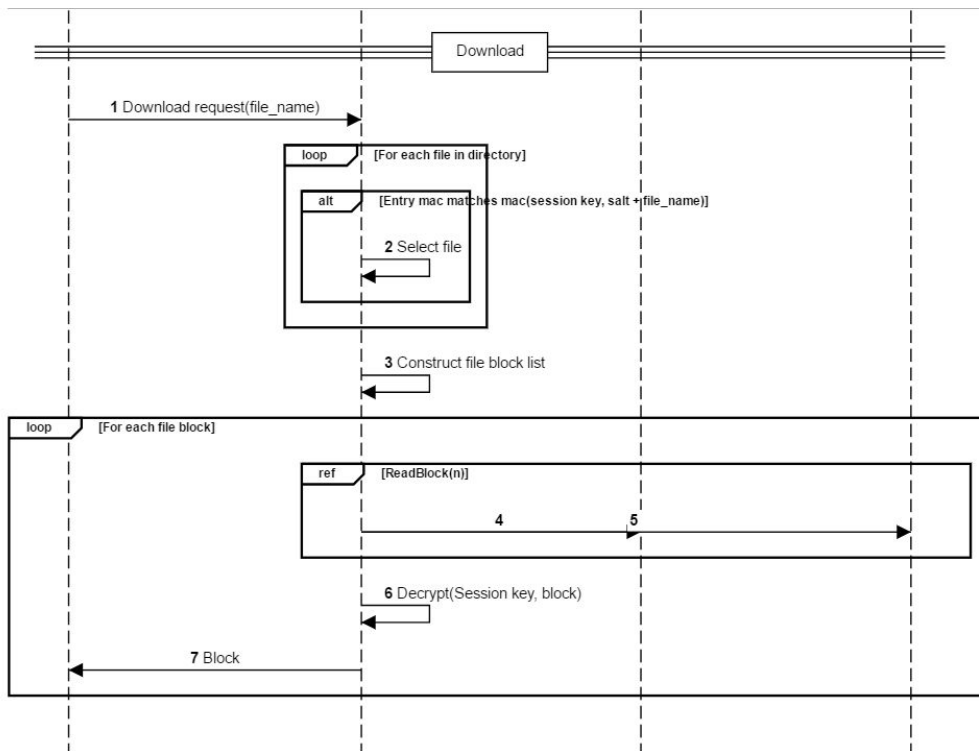
● תהליך אתחול המערכת:

- שירות אתחול המערכת ניתן לקריאה על ידי משתמש האדמין בלבד, ולו סיסמה מיוחדת.
- תהליך האתחול נועד לאתחל את הדיסקים של שרתי הבלוקים ולהפוך אותם מקבצים ריקים למערכות מוכנות לכתיבה ולקריאה.
- שרת ה-Frontend מקבל מהאדמין את מספרי הבלוקים שברצונו להקציב בדיסק עבור Bitmap ועבור Directory Root. ערכים אלה יכולים לנוע מ-1 ועד 255 בלוקים לכל חלק.
- השרת כותב לשרתי הבלוקים את הבלוק הראשון במערכת, ובלוק זה מכיל את הערכים שקבע האדמין. בלוק זה ייקרא על ידי שרת ה-Frontend בכל פעם שירצה לקרוא את ה-Bitmap ואת ה-Directory Root.
- השרת לאחר מכן כותב לשרתי הבלוקים את כל הבלוקים של ה-Bitmap, כאשר כולם פרט לראשון נכתבים ריקים, ואילו בבלוק הראשון השרת מדליק את הביטים שמצביעים על הבלוקים שהוא מאתחל, על מנת לסמן כתפוסים בפני כתיבה.
- השרת כותב לשרתי הבלוקים את כל הבלוקים של ה-Directory Root. על מנת להגביר את האבטחה על המערכת, השרת אינו כותב Root ריק, אלא מגריל את הערכים של כל Entry ולאחר מכן מסמן את ה-Entry כפנוי לכתיבה באמצעות כיבוי הבית הראשון שלו.
- השרת מחזיר ללקוח (לאדמין) הודעת הצלחה או הודעת שגיאה.



• תהליך העלאת קובץ לשרת:

- שרת ה-Frontend מקבל מהלקוח את שם הקובץ החדש, את תוכן הקובץ וסיממת הזדהות של הלקוח.
- השרת קורא משרתי הבלוקים את ה-Bitmap ואת ה-Directory Root.
- השרת מוצא מקום פנוי ב-Directory Root ויוצר בו Entry עבור הקובץ החדש. באמצעות ה-Bitmap השרת מוצא מקום ליצור בו את הבלוק הראשי של הקובץ ומדליק את הביט המתאים.
- עד סוף קריאת תוכן הקובץ מהלקוח, השרת יוצר Directory Block, יוצר עבורו הפנייה ב-Main Block, ועבור כל בלוק תוכן מקצה לו מקום באמצעות ה-Bitmap, כותב אותו למקום זה ויוצר אליו הפנייה ב-Directory Block.
- אם Directory Block מתמלא, השרת כותב אותו לדיסק ויוצר Directory Block חדש עם הפנייה אליו מה-Main Block.
- השרת מחזיר ללקוח הודעה על הצלחת הכתיבה, אלא אם התרחשה שגיאה, כגון חוסר במקום במערכת, ובמקרה זה מוחזרת הודעת שגיאה.



● תהליך הורדת קובץ מהשרת:

- שרת ה-Frontend מקבל מהלקוח את שם הקובץ הרצוי להורדה ואת סיסמת ההזדהות של הלקוח.
- השרת קורא משרתי הבלוקים את ה-Directory Root, מוצא בו את ה-Entry של הקובץ המבוקש (אם קיים) ומחלץ ממנו את האינדקס של ה-Main Block של אותו קובץ ואת גודל הקובץ. אם סיסמת ההזדהות של הלקוח לא תואמת את הנתונים ב-Entry, השרת מעלה הודעת שגיאה.
- השרת קורא את ה-Main Block ושומר אותו בזיכרון.
- עד שכל גודל הקובץ נשלח את הלקוח, השרת קורא בכל פעם את ההפניה הבאה של ה-Main Block וקורא את ה-Directory Block המתאים.
- השרת קורא את ההפניות מה-Directory Block עד סוף הבלוק או עד שנשלח כל תוכן הקובץ ללקוח, ועבור כל הפנייה קורא מהשרת את בלוק התוכן המתאים, מפענח את ההצפנה שלו ושולח ללקוח.
- במקרה של שגיאה, כגון שם קובץ שלא קיים במערכת, השרת מחזיר ללקוח הודעת שגיאה.

● תהליך הצגה של רשימת הקבצים במערכת:

- שרת ה-Frontend מקבל מהלקוח את סיסמת ההזדהות שלו, שתשמש בהמשך לזיהוי הקבצים שבבעלותו.
- השרת קורא משרתי הבלוקים את ה-Directory Root.
- השרת עובר על ה-Directory Root, ועבור כל Entry מחלץ את שם הקובץ, מבצע עליו hash עם מפתח הלקוח ועם המספר האקראי שב-Entry (התהליך פורט לעיל). אם התוצאה זהה ל-hash השמור ב-Entry, סימן שזהו קובץ בבעלות הלקוח שקרא לשירות, והשרת מצרף אותו לרשימת הקבצים שתוצג בפני הלקוח. אם התוצאה שונה, השרת עובר לקובץ הבא.
- השרת שולח את רשימת הקבצים בפורמט HTML נוח לקריאה ללקוח, או הודעת שגיאה אם התרחשה שגיאה בתהליך.

● תהליך מחיקה של קובץ מהשרת:

- שרת ה-Frontend מקבל מהלקוח שם קובץ למחיקה ואת סיסמת ההזדהות של הלקוח.
- השרת קורא משרתי הבלוקים את ה-Bitmap ואת ה-Directory Root.
- בתהליך זה לזה שבשירות הורדת הקבצים, השרת מאתר ב-Directory Root את ה-Entry המתאים לקובץ שברצונו של הלקוח למחוק ומוודא את בעלותו של הלקוח על הקובץ. השרת מכבה את הבית הראשון ב-Entry כדי לסמן אותו בתור מחוק.
- השרת קורא את ה-Main Block של הקובץ ומכבה את הביט המתאים ב-Bitmap על מנת למחוק את הבלוק מהמערכת. מה-Main Block נקראים כל ה-Directory Blocks, ועבור כל אחד מהם השרת מוחק את הבלוק מהמערכת, וכן את כל בלוקי התוכן שהבלוק מפנה אליהם.
- השרת כותב לשרתי הבלוקים את כל הבלוקים שעודכנו (ה-Bitmap וה-Directory Root) ומחזיר למשתמש הודעת הצלחה או שגיאה בהתאם.

פרוטוקול תקשורת

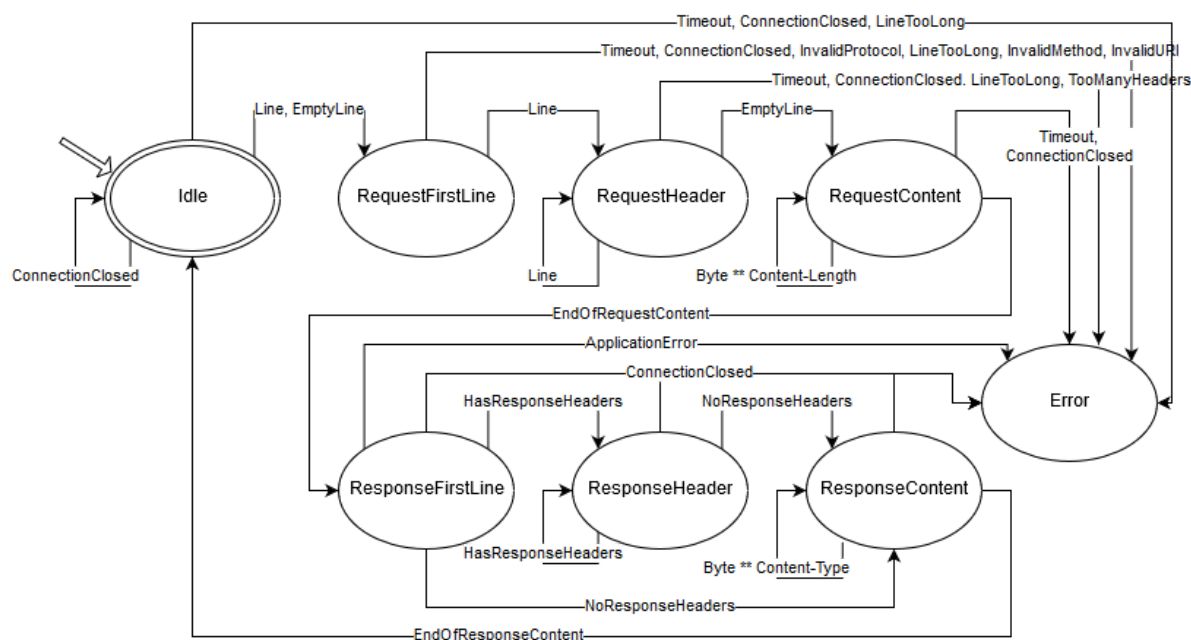
התקשורת בין הלקוח לבין שרת ה-Frontend ובין שרת ה-Frontend לבין שרתי הבלוקים נעשית באמצעות פרוטוקול HTTP (קובץ rfc של הפרוטוקול: <https://tools.ietf.org/html/rfc2616>) בקשות של הלקוח למערכת נשלחות באמצעות הדפדפן בתור בקשות GET (או בקשת POST עבור העלאת קובץ), ובקשות הקריאה והכתיבה של שרת ה-Frontend לשרתי הבלוקים נעשית באמצעות בקשות GET.

לצורך מימוש עבודה יעילה וכללית עם פרוטוקול HTTP, במחלקות השרתים נמצאות מכונות מצבים לעבודה עם השלבים הבאים של בקשת HTTP:

- שלב קבלת שורת הסטטוס, ממנה השרת מקבל את שם השירות שברצונו של הלקוח להשתמש.
- שלב קבלת ה-Headers, שורות ה-HTTP שמספקות מידע בנוגע לבקשה, כגון אורך התוכן וסוג התוכן.
- שלב קבלת תוכן הבקשה (במקרה שבו המשתמש מעלה קובץ, תוכן הבקשה מכיל את תוכן הקובץ).
- שלב שליחת שורת הסטטוס, בה השרת מודיע ללקוח על הצלחה או על כישלון של הפעולה.
- שלב שליחת ה-Headers, בהם נכתב מידע בנוגע לתשובת השרת, כגון אורך התשובה וסוג המידע הנשלח.
- שלב שליחת תוכן התגובה (אם המשתמש מוריד קובץ מהשרת, תוכן התגובה הוא תוכן הקובץ. אם המשתמש ביקש מהשרת רשימת קבצים או את תפריט הממשק הראשי, תוכן הבקשה הוא דף HTML שמכיל את הדף הרצוי. בכל שירות אחר, תוכן הבקשה הוא דף HTML של הודעת הצלחה או כישלון של השירות).

בשלב קריאת שורת הסטטוס, השרת מוציא ממנה את שם השירות שלו הלקוח קורא. לאחר מכן השרת יוצר אובייקט ממחלקה מתאימה לשירות, כאשר לכל שירות מחלקה נפרדת. מחלקות השירותים השונים מכילות פונקציות שמתאימות למצבים השונים של HTTP, ועבור כל מצב HTTP השרת קורא לפונקציה המתאימה מתוך השירות.

להלן דיאגרמה המתארת את המצבים השונים של פרוטוקול HTTP ואת המעברים ממצב למצב, התלויים בקבלת התוכן ובשליחת התוכן ללקוח. מכל מצב, אם מתרחשת שגיאה משמעותית, השרת עובר למצב שגיאה, בו ישלח ללקוח הודעת שגיאה ולא ימשיך לטפל בבקשה.



העבודה עם פרוטוקול HTTP נעשית באופן אסינכרוני. לצורך כך מומשה בפרויקט מחלקה של שרת קריאה/כתיבה אסינכרוני, אשר עובד באמצעות שיטת polling. לפי שיטה זו, השרת מחזיק מערך של חיבורי socket של הגורמים השונים, ופעם בפרק זמן קבוע בודק האם אחד מהם פנוי לקריאה או לכתיבה, והאם התרחשה בו שגיאה. עבור כל אחד מהמקרים השונים, ה-poller מעיר את החיבור המתאים וקורא לפונקציה שלו שתטפל בסוג הקריאה שהתקבל. בין כל שני מעברים של אובייקט ה-poller על מערך החיבורים, האובייקט ישן פרק זמן השווה לפרמטר ה-timeout שסופק בתחילת הרצת התכנית.

הפרויקט תומך בשתי שיטות Polling, האחת היא select והשנייה היא poll. שתי השיטות ממומשות באמצעות מחלקות קיימות ב-Python ובאמצעות מחלקות שנכתבו במסגרת הפרויקט על מנת להתאים את המימוש של שתי השיטות לקלט ופלט זהים.

פרמטרים וסוגי בקשות

עבור כל שירות במערכת שמקיים תקשורת בין חלקיה השונים מועברים פרמטרים שונים בשורת הסטטוס של בקשת ה-HTTP ותוכן בקשה שונה, ולהלן פירוט על הקלט והפלט של כל שירות:

כל השירותים, מלבד שירות העלאת קובץ, ייקראו באמצעות מתודת GET של HTTP בצורה הבאה:

GET /service_name?parameters HTTP/1.1

או על ידי הקלדה של הטקסט הבא בשורת הכתובת:

frontend_address:frontend_port/service_name?parameters

כאשר במקום "service_name" יופיע שם השירות ובמקום parameters יופיעו פרמטרים במידת הצורך.

במקום "frontend_address" ו"frontend port" יופיעו כתובת ה-IP והפורט של שרת ה-Frontend בהתאמה.

עבור שירותי העלאת קבצים, הורדת קבצים, רשימת קבצים, מחיקת קבצים ואתחול המערכת יש להעביר הזדהות משתמש בסיסמה דרך אחד הפרמטרים בשם "password" או דרך Cookie Header.

- שירות רשימת קבצים:
שם השירות שיש להעביר בבקשה הוא "list".
במקרה של הצלחה, התוכן שיוחזר יכיל דף HTML עם טבלת קבצים מעוצבת.
- שירות הורדת קובץ:
שם השירות שיש להעביר בבקשה הוא "download".
יש לספק פרמטר מסוג מחרוזת בשם "filename" המציין את שם הקובץ שברצון הלקוח להוריד.
במקרה של הצלחה, התוכן שיוחזר יכיל את תוכן הקובץ בתור attachment.
- שירות מחיקת קובץ:
שם השירות שיש להעביר בבקשה הוא "delete".
יש לספק פרמטר מסוג מחרוזת בשם "filename" המציין את שם הקובץ שברצון הלקוח להוריד.
התוכן שיוחזר הוא דף HTML המציין האם הפעולה נכשלה או הצליחה.
- שירות העלאת קובץ:
שם השירות שיש להעביר בבקשה הוא "fileupload".
בהבדל מהשירותים האחרים, שירות זה אינו נקרא באמצעות מתודת GET, אלא באמצעות POST.
תוכן הבקשה יכיל את תוכן הקובץ שברצון הלקוח להעלות, בפורמט בקשה של multipart form: <https://www.ietf.org/rfc/rfc2388.txt>
התוכן שיוחזר הוא דף HTML המציין האם הפעולה נכשלה או הצליחה.
- שירות אתחול המערכת:
שם השירות שיש להעביר בבקשה הוא "init".
יש להעביר פרמטרים מסוג integer בשם "bitmaps" ו"dir_roots" המציינים בהתאמה את כמות בלוקי ה-Bitmap ובלוקי ה-Directory Root שברצון האדמין לאתחל. על פרמטרים אלה להיות בטווח (0, 255).
הסיסמה שתועבר בשירות זה חייבת להיות תואמת את סיסמת האדמין השמורה במערכת.
התוכן שיוחזר יכיל דף HTML המציין האם הפעולה נכשלה או הצליחה.

בנוסף תומכת המערכת בשירות הורדת קבצים מתוך תיקיית הקבצים המוגדרת מראש במחשב ה-Frontend (בהבדל משירות הורדת הקבצים מכספת). שירות זה אינו שירות חובה במערכת, והוא משמש את הפרויקט במסירה של דפי HTML וקבצי CSS לדפדפן עבור ממשק משתמש נוח. הפרמטר שמועבר לשירות זה הוא שם הקובץ שברצון המשתמש להוריד.

כמו כן, בעבודת שרת ה-Frontend אל מול שרתי הבלוקים, הוא משתמש בקריאות לשני שירותים שמספקים שרתי הבלוקים (קריאה וכתובה). השירותים נקראים גם הם באמצעות מתודת GET, ובכל קריאה יש לספק Authentication header שמכיל הזדהות מסוג Basic, שם משתמש וסיסמה שמזהים את ה-Frontend בפני שרתי הבלוקים.

- שירות קריאה של בלוק:

שם השירות שיש להעביר הוא "read".

יש לספק פרמטר מסוג integer המציין מספר בלוק חוקי שברצון ה-Frontend לקבל. התוכן המוחזר יהיה תוכן הבלוק, או הודעה במקרה של שגיאה.

- שירות כתיבה של בלוק:

שם השירות שיש להעביר הוא "write".

יש לספק פרמטר מסוג integer המציין את מספר הבלוק שאליו ה-Frontend רוצה לכתוב. תוכן הבקשה יהיה תוכן הבלוק שיש לכתוב. התוכן המוחזר יהיה סטטוס הצלחה או הודעה במקרה של שגיאה.

בעיות ידועות

- הפרויקט תומך בסביבת Unix בלבד. ניתן להוסיף תמיכה במערכת ההפעלה Windows על ידי הגדרה חלופית לכל הספריות שאינן נתמכות ב-Unix ובהן נעשה שימוש בפרויקט.
 - גורם חיצוני עדיין שניגש לקבצי ה-Config של השרתים מסוגל לחלץ משם בקלות את סיסמת האדמין. פתרון אפשרי לבעיה זו הוא מימוש הצפנה על הסיסמה, או לחלופין מחיקת הסיסמה, באופן ידני או דרך התוכנית, מקובץ הקונפיגורציה לאחר אתחול המערכת, שכן אין שימוש לסיסמה זו לאחר מכן.
 - הדגש בפרויקט זה הוא על אבטחת המידע על הדיסקים, ולמרות השכבות ההגנה המרובות שקיימות, קיימת פרצה זעירה שיכולה להקל על פריצתן. כשהדיסק מאותחל, רוב ה-Bitmap ריק, משום שטרם נכתב דבר לדיסק. גורם חיצוני עדיין שיוודע זאת ובעל גישה לשרתי הבלוקים יכול לחלץ את ה-Bitmap, ועל ידי ידיעת התוכן שלו לפרוץ את הצפנת ה-AES של שרתי הבלוקים ושל ה-Frontend.
- ניתן להוסיף בעתיד אפשרות שתתגבר על פרצה זו, והיא להגריל מספר אקראי בתחילת העבודה, ועבור כל בית ב-Bitmap, להגדיל את אותו בית בערך השווה למכפלת המספר האקראי במיקום הבית (ההגדלה נעשית בחשבון מודולרי על מנת למנוע חריגה מערך הבית המקסימלי). כשהבלוק נקרא, נעשה תהליך הפוך, בו מפחיתים מערך כל בית את הערך הדרוש, וכך נוצר מצב שבו אין תוכן ידוע שמצוי בדיסק, ולא ניתן לפרוץ בקלות את ההצפנות.

התקנה ותפעול

התקנה

יש להתקין בנפרד כל שרת במערכת. ניתן להתקין את שרת ה-Frontend ואת שרתי הבלוקים על מחשב יחיד ב-ports שונים, אך ניתן גם לפצל אותם למחשבים שונים. עבור כל שרת, יש להוריד למחשב את תיקיית הפרויקט ולשנות את קובץ הקונפיגורציה. להלן פירוט הפרמטרים בקבצי הקונפיגורציה השונים: הפרמטרים של שרת ה-Frontend יוצרו בקובץ הקונפיגורציה שבתקיית ה-Frontend. בקובץ חייבת להיות Section עם פרמטרים על שרת ה-Frontend בשם "frontend", וכן Section נוסף עבור כל שרת בלוקים שמחובר למערכת, כאשר שם ה-Section אינו משנה.

Section Name	Key Name	Value Type	Default Value	Explanation
frontend	admin.password	string	admin	הסיסמה שתשמש את האדמין להזדהות במערכת.
frontend	bind.address	string	0.0.0.0	כתובת ה-IP שאליה מבצע השרת bind.
frontend	bind.port	int	8888	פורט ההאזנה של השרת.
frontend	key	string		מפתח שימש את השרת בשכבת הצפנת AES שלו.
frontend	ivkey	string		מחרוזת שבאמצעותה ייגזרו ערכי ה-iv עבור הצפנת AES של השרת.
block device> <name	address	string	127.0.0.1	כתובת ה-IP של שרת בלוקים זה.
block device> <name	port	int		פורט ההאזנה של שרת בלוקים זה.
block device> <name	username	string		שם המשתמש באמצעותו יש להזדהות בפני שרת בלוקים זה.
block device> <name	password	string		הסיסמה באמצעותה יש להזדהות בפני שרת בלוקים זה.
block device> <name	bd_identifier	int	1	שדה שמציין את היותו של Section זה הגדרה של שרת בלוקים.

הפרמטרים של כל שרת בלוקים ייקבעו בקובץ הקונפיגורציה בתיקייה של כל שרת בלוקים נפרד ב־Section בשם "blockdevice" באופן הבא:

Section Name	Key Name	Value Type	Default	Explanation
blockdevice	bind.address	string	0.0.0.0	כתובת ה-IP שאליה מבצע השרת bind.
blockdevice	bind.port	int		פורט ההאזנה של השרת.
blockdevice	file.name	string	disk	שם הקובץ שישמש בתור הדיסק הפיזי לקריאה ולכתיבה. במקרה שבו יש מספר שרתי בלוקים שרצים על אותו מחשב, יש לוודא ששם הקובץ שונה בין כולם על מנת למנוע התנגשויות ומחיקות מידע.
blockdevice	file.size	int	1024000	גודל הקובץ שישמש בתור הדיסק הפיזי.
blockdevice	key	string		מפתח שישמש את השרת בשכבת הצפנת AES שלו.
blockdevice	ivkey	string		מחרוזת שבאמצעותה ייגזרו ערכי ה־iv עבור הצפנת AES של השרת.
blockdevice	username_hash	encrypted_string		הצפנה של שם המשתמש באמצעותו יזדהה ה־Frontend בפני שרת הבלוקים.
blockdevice	password_hash	encrypted_string		הצפנה של הסיסמה באמצעותה יזדהה ה־Frontend בפני שרת הבלוקים.
blockdevice	salt	base64 string		גורם שישמש בהצפנת שם המשתמש והסיסמה שלעיל.

- salt – ערך שרירותי כלשהו שמטרתו תוסבר בסעיפים הבאים.
 - username_hash – ערך hash מסוג sha1 בבסיס 64 על ערך שם המשתמש שבאמצעותו שרת ה־Frontend יזדהה בגישה לשרת הבלוקים ועל ערך ה־salt. לצורך קביעת ערך זה באופן פשוט קיים בתיקיית block_device קובץ בשם userpass_encryption_tool, אותו ניתן להריץ ולספק לו את שם המשתמש, הסיסמה וה־salt, לקבלת ערך שדה זה בתור פלט.
 - password_hash – ערך דומה לערך username_hash, אך עם סיסמת ההזדהות במקום שם המשתמש. גם ערך זה מתקבל בהרצת הקובץ userpass_encryption_tool.
- הערכים שמוזנים לתוך קובץ userpass_encryption_tool בתור username ובתור password הם גם אלה שצריכים להופיע בקובץ ה־Config של ה־Frontend בתור שם משתמש וסיסמה עבור שרת בלוקים זה (בשדות "username" ו"password").

הרצה

יש להריץ כל שרת מהמחשב עליו הותקן באמצעות cygwin או כל תוכנה שמדמה סביבת מערכת Unix. עבור שרת Frontend יש להיכנס לתיקיית הפרויקט ולהריץ את הפקודה הבאה: `Python -m frontend`. עבור שרת בלוקים יש להיכנס לתיקיית הפרויקט ולהריץ את הפקודה הבאה: `Python -m block_device`.

עבור כל אחד מהשרתים ניתן לספק בעת ההרצה פרמטרים. סיפוק הפרמטר `-h` ידפיס בקונסולה הסבר מפורט על כל הפרמטרים המותרים ועל מטרותיהם, ולהלן הסבר נוסף עליהם:

Name	Type	Default Value	Explanation
max-connections	int	10	מספר החיבורים בהם השרת תומך בעת ובעונה אחת.
log-file	string		שם הקובץ אליו ייכתבו הדפסות debug במהלך ריצת התכנית. אם סופק פרמטר והקובץ לא קיים, המערכת תיצור בעצמה קובץ בעל שם זה. אם לא סופק פרמטר, הדפסות debug ייכתבו ישירות לקונסולת ההרצה.
base	string	./files	ה-path לתיקייה בה נמצאים קבצי עזר ודפי HTML הנשלחים לדפדפן.
timeout	int	1	הזמן שהשרת האסינכרוני יחכה בין כל פעולת poll (הוסבר בפרק המימוש)
foreground	boolean	false	אם סופק הפרמטר, השרת ירוץ בתור תהליך רגע בצורת daemon.
event-method	string	מפורט אחרי הטבלה	פרמטר הקובע את שיטת ה-polling שבאמצעותה עובד השרת האסינכרוני (הוסבר בפרק המימוש)
config	string	מפורט אחרי הטבלה	ה-path לקובץ הקונפיגורציה ממנו יקרא השרת את הקבועים שלו בתחילת ההרצה.

- `event-method`: ערך ברירת המחדל הוא "poll". אם תמומש בעתיד תמיכה במערכת Windows ברירת המחדל בסביבה זו תהיה "select".
- `config`: ערכי ברירת המחדל הם "frontend/config.ini" עבור שרת Frontend ו"block_device/config.ini" עבור שרת בלוקים.

דוגמה לקבצי קונפיגורציה תקינים עבור תצורה של 3 שרתי בלוקים שכולם מורצים על אותו המחשב (כתובת IP זהה):

עבור שרת ה-Frontend:

```
[frontend]
admin.password = admin
bind.address = 0.0.0.0
bind.port = 8888
key = frontend-encryption-key
ivkey = random-frontend-key-string
```

```
[blockdevice.1]
address = 127.0.0.1
port = 8081
username = username1
password = password1
bd_identifier = 1
```

```
[blockdevice.2]
address = 127.0.0.1
port = 8082
username = username1
password = password1
bd_identifier = 1
```

```
[blockdevice.3]
address = 127.0.0.1
port = 8083
username = username1
password = password1
bd_identifier = 1
```

עבור שרת בלוקים ראשון:

```
[blockdevice]
bind.address = 0.0.0.0
bind.port = 8081
file.name= disk
file.size = 1024000
key = some-encryption-key
ivkey = random-key-string
username_hash = lSdrokXcyx/tuSsjO8m1ts+S0mY=
password_hash = hnYtp6yd/jXgfd2t8KcqWtVuAnE=
salt = c2FsdDE=
```

עבור שרת בלוקים שני:

```
[blockdevice]
bind.address = 0.0.0.0
bind.port = 8082
file.name= disk
file.size = 1024000
key = some-encryption-key
ivkey = random-key-string
username_hash = lSdrokXcyx/tuSsjO8m1ts+S0mY=
password_hash = hnYtp6yd/jXgfd2t8KcqWtVuAnE=
salt = c2FsdDE=
```


עבור שרת בלוקים שלישי:

```
[blockdevice]
bind.address = 0.0.0.0
bind.port = 8083
file.name= disk
file.size = 1024000
key = some-encryption-key
ivkey = random-key-string
username_hash = lSdrokXcyx/tuSsjO8mlts+S0mY=
password_hash = hnYtp6yd/jXgfd2t8KcqWtVuAnE=
salt = c2FsdDE=
```


הפקודות שיש לכתוב בשורת ההרצה לאחר ניווט לתיקיית הפרויקט:

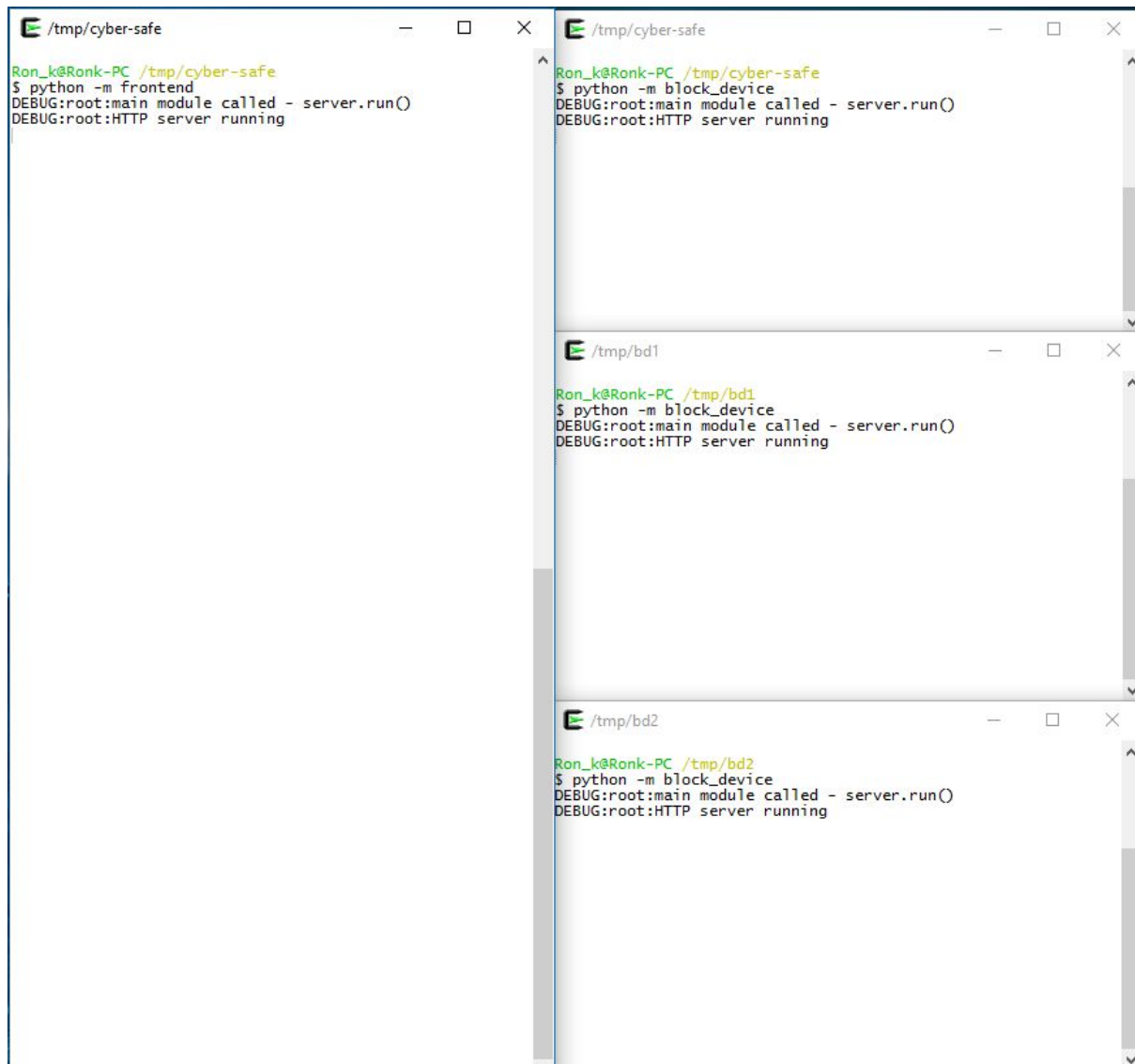
עבור Frontend:

```
python -m frontend
```

עבור שרת בלוקים:

```
python -m block_device
```

להלן תמונה הממחישה הפעלה תקינה של שרת Frontend ושלושה שרתי בלוקים בסביבת cygwin מתוך תיקיית הפרויקט:



גישה של לקוח למערכת

לאחר הרצת כל השרתים הדרושים, לקוח של המערכת מסוגל לגשת אליה מכל דפדפן Chrome שנמצא על מחשב על ידי הזנה של כתובת השרת ופורט השרת בשורת הכתובת בצורה הבאה:

Frontend_address:Frontendport

הדבר יפנה את הלקוח לתפריט המערכת הראשי, שהוא פשוט וקל לשימוש. בתפריט הראשי קיים שדה להזנת סיסמה, ושני כפתורים: אחד לכניסה רגילה למערכת ואחד לכניסת משתמש אדמין.

ממשק המשתמש הרגיל מכיל את רשימת הקבצים, ומתחתיה כפתורים המאפשרים להוריד ולמחוק את הקובץ שנבחר מהרשימה, כמו גם לבחור קובץ מהמחשב של הלקוח ולהעלות אותו לשרת. ממשק האדמין מכיל כפתור לאתחול המערכת ושני שדות להזנת מספרי הבלוקים של Bitmap ושל Directory Root, אתם מבקש האדמין לאתחל את המערכת. לאחר כל קריאה לשירות, בפני המשתמש תוצג הודעת שגיאה או הודעת הצלחה עם כפתור הפניה לתפריט הראשי.

להלן מספר תמונות הממחישות את ממשק המשתמש:

תפריט ראשי

Cyber-Safe

User Code

Your Password

Enter

Enter as admin


רשימת קבצים לאחר הזנת סיסמת משתמש. ניתן לראות 4 קבצים שהועלו למערכת ואת הגדלים שלהם.

File List	
File Name	File Size (Bytes)
• devito.png	11871
• kot_blini_in_space.jpg	241133
• seagul yawn.jpg	12374
• BIG NEWS.jpg	18236

Download

Delete

Submit file



הודעת הצלחה לאחר מחיקת קובץ

File deleted successfully

Back to menu

הודעת שגיאה לאחר קריאה לשירות הורדת הקבצים ללא סיפוק שם קובץ

File name missing

Back to menu

ממשק האדמין לאחר סיפוק סיסמת אדמין מיוחדת שנקבעה מראש

Admin Panel

Number of Bitmap Blocks

Number of Directory Root Blocks

Initialize disk

תכניות עתיד

- אבטחת מידע נע: פרויקט זה מגן בצורה טובה על המידע הנח בשרתים, אך המידע שעובר בתקשורת בין הגורמים השונים חשוף לחלוטין להאזנות ולאימים דומים. בעתיד ניתן יהיה להגן על התקשורת בין כל הנקודות באמצעות פרוטוקול TLS, פרוטוקול המאפשר שיחות רשת מאובטחות בשיטות קריפטוגרפיות למניעת ציתות וזיוף של המידע. בין כל שני צדדים ניתן יהיה לפתוח ערוץ תקשורת פרטי תוך אילוץ הזדהות חזקה באמצעות מפתח ותעודות הרשאה דיגיטליות.
 - בעתיד ניתן יהיה לוותר על שימוש בדפדפן בתור ממשק משתמש ולממש ממשק משתמש נפרד, בו ניתן יהיה להצפין את המידע שנשלח לשרת, בניגוד מבבקה הנוצרת ונשלחת על ידי הדפדפן ללא שליטת הלקוח.
- תמיכה בשינויים למערכת הכספות: בעתיד ניתן יהיה להוסיף תמיכה בפעולות של הוספת כספת חדשה למערכת (שרת בלוקים נוסף) או הורדה של כספת מהמערכת. כשברצון האדמין לשנות את מערכת הכספות, המידע שבמערכת הקבצים ייקרא מהמערכת הישן ויכתב במקביל למערכת החדש על מנת לשמור על כל הקבצים שכבר הועלו קודם לכן.
- תמיכה במספר מערכי כספות: בעתיד ניתן יהיה להוסיף אפשרות לתמיכה במספר מערכות קבצים במקביל דרך שרת Frontend יחיד. כלומר בהינתן שני מערכים של שרתי בלוקים, ניתן יהיה לנהל בכל אחד מהם מערכת קבצים נפרדת בעת ובעונה אחת.

פרק אישי

פרויקט זה הוא ההתנסות הראשונה שלי בכתיבת מערכת רחבת היקף במקצוע מדעי המחשב, וקשה להפריז בכמות הנושאים שלא הכרתי קודם לכן ושלמדתי במהלך שנה זו. העבודה על הפרויקט הקנתה לי אין-ספור מיומנויות מעולם הנדסת התכנה, מיומנויות שאני משוכנע שיעזרו לי במידה ניכרת בהמשך דרכי בעולם המחשבים.

מיומנויות וכישורים אלה – כגון: היכולת לתכנן מערכת רחבת היקף, היכולת לכתוב קוד מאורגן ומסודר לפי מחלקות, היכולת להתמודד עם תקלות בהרצת הקוד – כל אלה אמנם מרגישים עבורי טבעיים יותר כעת, אך במבט לאחור אני נזכר באתגרים הרבים שהיה עליי להתמודד עמם על מנת להגיע לשלב המוצלח שהפרויקט עומד בו בהווה:

במהלך הפרויקט נאלצתי לבלות לעתים זמן ממושך בחיפוש אחר שגיאות בתכנית, דבר שהיה בגדר המכשול המתסכל יותר של תהליך העבודה, משום שהוא מנע ממני מלהשקיע את אותו הזמן בהתקדמות בהנדסת השלבים הבאים של המערכת.

עם זאת, בכל פעם נוכחתי לדעת שכל בעיה – סופה להיפתר, ולאחר שהתמודדתי עם הדברים, לעתים בהתייעצות עם מנחי הפרויקט שלי, זכיתי לחוות סיפוק רב כאשר חזיתי בהתקדמות ממשית של כל המערכת שבנתי.

אין ספק שהעבודה על הפרויקט נטעה בי עניין רב בתחום הנדסת התכנה ואבטחת המידע, וסיפקה לי שאלות רבות במגוון נושאים, עליהן אני מקווה לענות בהמשך הדרך שלי במסלול זה.

קוד פרויקט

להלן קישור ל-Release העדכני של הפרויקט, אשר מכיל בנוסף קובץ rar של התיעוד.
<https://github.com/TheClownFromDowntown/Cyber-Safe/releases/tag/1.0.4>

Sequence Diagram Source - 'נספח א'

להלן הקוד בו נעשה שימוש על מנת ליצור את דיאגרמת הרצף באתר

[:https://www.websequencediagrams.com](https://www.websequencediagrams.com)

```
title Cybersafe: Sequences

actor User-Agent
control Front-end
database Blockdevice1
database Blockdevice2

==Definitions==

autonumber 1
par def(ReadBlock(n))
    parallel
        Front-end->>Blockdevice1: Read block n
        Front-end->>Blockdevice2: Read block n
    parallel off
        Front-end->>Front-end: Wait
    parallel
        Blockdevice1->>Blockdevice1: Decrypt(Blockdevice1 key, block)
        Blockdevice1->>Front-end: Block
        Blockdevice2->>Blockdevice2: Decrypt(Blockdevice2 key, block)
        Blockdevice2->>Front-end: Block
    parallel off
        Front-end->>Front-end: Merge blocks
        Front-end->>Front-end: Decrypt(Front-end key, block)
    end
end

par def(WriteBlock(n, content))
    Front-end->>Front-end: Encrypt(Front-end key, content)
    Front-end->>Front-end: Split blocks
    parallel
        Front-end->>Blockdevice1: Write block n, content
        Front-end->>Blockdevice2: Write block n, content
    parallel off
        Front-end->>Front-end: Wait
    parallel
        Blockdevice1->>Blockdevice1: Encrypt(Blockdevice1 key, block)
        Blockdevice1-->>Front-end: Ack
        Blockdevice2->>Blockdevice2: Encrypt(Blockdevice2 key, block)
        Blockdevice2-->>Front-end: Ack
    parallel off
end

==Initialization==

autonumber 1
loop For each bitmap block
    ref ReadBlock(n)
    parallel
        Front-end->>Blockdevice1:
        Front-end->>Blockdevice2:
    parallel off
    end
end
loop For each directory block
```



```

        ref ReadBlock(n)
        parallel
            Front-end->Blockdevice1:
            Front-end->Blockdevice2:
        parallel off
    end
end

==Authentication==

autonumber 1
User-Agent->Front-end: Authenticate(password)
Front-end->Front-end: Derive session key out of password
Front-end->Front-end: Store session key in HTTP session

==Upload==

autonumber 1
User-Agent->Front-end: Upload request(file_name, content)
loop For each file in directory
    alt Entry mac matches mac(session key, salt + file_name)
        Front-end->Front-end: Select file
    end
end

alt if file selected
    Front-end->User-Agent: File already exists error
end

Front-end->Front-end: Allocate blocks out of bitmap
Front-end->Front-end: Allocate directory block

par Write bitmap
    ref WriteBlock(n, content)
    parallel
        Front-end->Blockdevice1:
        Front-end->Blockdevice2:
    parallel off
end
end

par Write directory
    ref WriteBlock(n, content)
    parallel
        Front-end->Blockdevice1:
        Front-end->Blockdevice2:
    parallel off
end
end

loop
    User-Agent->Front-end: Block
    break
    note over Front-end: if no block
    end
    Front-end->Front-end: Encrypt(Session key, block)
    ref WriteBlock(n, content)
    parallel
        Front-end->Blockdevice1:
        Front-end->Blockdevice2:
    parallel off
end

```



```

end

==Download==

autonumber 1
User-Agent->Front-end: Download request(file_name)
loop For each file in directory
    alt Entry mac matches mac(session key, salt + file_name)
        Front-end->Front-end: Select file
    end
end

end

Front-end->Front-end: Construct file block list

loop For each file block
    ref ReadBlock(n)
    parallel
        Front-end->Blockdevice1:
        Front-end->Blockdevice2:
    parallel off
end
Front-end->Front-end: Decrypt(Session key, block)
Front-end->User-Agent: Block
end

```

