אבטחת תקשורת באינטרנט באמצעות חילוף מפתחות דיפי-הלמן ואלגוריתם הצפנה RC4

*על ידי רון ליפל*

*עבודת גמר במחשבים, כיתה י"א - י"ב*

*מועד הגשה: ינואר 2015*

*מנחה: עומר בוהם*

*תוכן עניינים*

* מבוא 3
* הצפנה באינטרנט 4
* הבעיה המתמטית 5
* מימוש בפועל 7
  + RC4 8
  + המפתח של RC4 8
* שרת צ'אט מרובה לקוחות 11
  + צד השרת 11
  + צד הלקוח 12
* מנקודת מבטו של מצותת 13
* סיכום 15
* בבליוגרפיה 16
* קוד 17
  + DiffieHellman.cs 17
  + RC4Stream.cs 18
  + dh\_server (Server Program) 20
    - ServerMain.cs 20
    - Client.cs 24
    - SecureClient.cs 27
    - DiffieHellmanParams.cs 29
  + dh\_client (Client Program) 31
    - ClientMain.cs 31
    - Connection.cs 35
    - SecureConnection.cs 38

*מבוא*

אנו חיים בעידן האינטרנט, עידן בו אנחנו מסוגלים לתקשר באופן ששואף למיידי עם אדם הנמצא אלפי קילומטרים מאיתנו, רק על ידי כמה לחיצות מקלדת. ביכולתנו לרכוש מוצרים הנמצאים בצד השני של העולם באמצעות לחיצת כפתור או לגשת למידע שעד לפני 30 שנה, הצטרכנו לגשת פיזית לספריה ולמצוא בשבילו ספר מתאים. האינטרנט מעשיר את חיינו באין-סוף תחומים, אך האם יש לו גם מגרעות?

השפעה אחת של האינטרנט, לגביה לא מוטל כל ספק- היא היעדר הפרטיות. כשאנו משתמשים באינטרנט, המידע הקרוב אלינו ביותר כבר לא נמצא תחת שליטתנו. מידע אישי, פרטי כרטיס אשראי, ומידע אחר הרגיש לנו עובר באופן דיגיטלי במבואות האינטרנט, בין נקודת קצה אחת, לנקודת קצה שניה. אך הדרך אותה עושה המידע שאנו מעבירים באינטרנט, היא מפותלת ועוברת במספר תחנות ביניים, שהן לאו דוקא סימפטיות במיוחד, ולחלקן אף כוונות זדוניות.

עם התרחבות האינטרנט והתווספות שימושים שונים הנוגעים במידע רגיש, צמח במקביל תחום האבטחה אשר כל תכליתו היא להגן על המידע שלנו ולוודא שלא נעשה לנו עוול. כאשר אנו כיום רוכשים מוצר מ- ebay או amazon פרטי כרטיס האשראי שלנו עוברים לשרתים של אותם אתרים באופן מאובטח כזה, שצדדים בלתי רלוונטים לתקשורת לא יוכלו לעשות איתם מאומה. אותה אבטחה חלה על מספר גדול של התקשרויות אינטרנטיות, מצ'אטים מקוונים, לאימייל ולשיתוף קבצים.

מספר רב של פריצות דרך ורעיונות הובילו ליכולת האבטחה שקיימת כיום. פותחו שיטות הצפנה חדשות, ורעיונות שונים, אשר הזניקו את התחום קדימה וכיום מיושמים באין-ספור אפליקציות. פריצת דרך אחת, אותה אני הולך לסקור בעבודה זו, פותחה בשנות ה-70, ונחשבת לבין ההתפתחויות החשובות בתחום.

*הצפנה באינטרנט*

כאשר יש לנו חיבור תקשורתי בין שתי נקודות באינטרנט (בפרוטוקול IP), אנו שולחים פאקטות (packets) הלוך ושוב המכילות את המידע שברצוננו להעביר. ללא כל סוג של הצפנה כזו או אחרת, המידע שאנו מעבירים חשוף לגמרי ויהיה ניתן לקריאה חופשית על ידי כל גורם שביכולתו לעקוב אחר תעבורה בערוץ התקשורת (ע"י לדוגמא, packet sniffing), או כל גורם שהמידע הגיע אליו בשוגג, או עבר דרכו.

עקב כך, התפתח הצורך להצפין תעבורה מסוג זה כדי שרק הצדדים הרלוונטיים בתקשורת אשר מקבלים את המידע, יוכלו לפרש אותו. בשביל כך, נעשה שימוש באלגוריתם הצפנה מוסכם.

אלגוריתם ההצפנה הקלאסי מקבל פרמטר מסוים הנקרא מפתח, ולפיו מצפין או מפענח את המידע הנתון. כאשר המידע מוצפן על ידי מפתח, ניתן לפענח את המידע המוצפן רק באמצעות אותו המפתח- שיטה זאת נקראת הצפנה סימטרית, הצפנה בה נעשה שימוש במפתח יחיד לשם הצפנה ולשם פיענוח.

מכך נובע שלמפתח חשיבות בטיחותית גבוהה ביותר, ואסור שאף צד אחר לא ידע אותו חוץ מאותם צדדים רלוונטיים המשתתפים בתקשורת. וכאן נוצרת הבעיה - כאשר יש לנו שני צדדים שלא נפגשו מעולם, כיצד הם משתפים את המפתח הסודי בינהם באופן שצדדים המצותתים להם לא יוכלו לקבל אותו?

לבעיה זאת פותחו מספר פתרונות. הפתרון האפקטיבי הנפוץ כיום, שמהווה אלטרנטיבה להצפנה סימטרית, זה הצפנה א-סימטרית (asymmetric/public-key cryptography).

בהצפנה אסימטרית, יש מפתחות שונים להצפנה. כל צד מפתח זוג מפתחות כאלו, ושולח לצד השני רק את המפתח ה"פומבי", שהוא מפתח ההצפנה. לאחר ההחלפה הצדדים מצפינים את המידע עם מפתחות ההצפנה שניתנו להם מהצד השני, שולחים את המידע, ואז המידע מפוענח עם מפתח הפיענוח התואם שנמצא אצל הנמען. כך למעשה, כשמשתמשים בשיטה זו, צד שלישי המצותת לתקשורת ישיג את מפתח ההצפנה, אך לא יהיה מסוגל לפעח שום מידע מוצפן.

השיטה המעשית הראשונה לשיתוף מפתח סודי, שהומצאה עוד לפני ההצפנה האסימטרית, היא שיטת הDiffie-Hellman, שפותחה ע"י וויטפילד דיפי ומרטין הלמן ב1976, ונקראת אחר שמם. למען האמת, השיטה נמצאה כמה שנים לפני כן על ידי המוסד הבריטי, אך נשמרה מסווגת.

השיטה הזו פתרה את בעיית העברת עותק של מפתח הצפנה על גבי ערוץ בלתי מאובטח, בין משתתפים שלא נפגשו מעולם. לפתרון דיפי-הלמן יש מאפיינים של חילוף אסימטרי (פרמטרים ציבוריים ופרמטרים סודיים, שמשתלבים למען יצירת תוצר סודי שישמש כמפתח), אך למעשה מאפשר הצפנה סימטרית באמצעות מפתח יחיד, הנשמר ע"י שני הצדדים.

*הבעיה המתמטית*

פתרונות אבטחה רבים מתבססים על פונקציות חד כיווניות- כלומר פעולות מתמטיות שקל לבצע בכיוון אחד, אך קשה מאוד לבצע אותן בכיוון ההפוך. בעיית דיפי-הלמן היא אחת מהן. אך לפני שניגש לבעיית דיפי הלמן, נציג את הבעיה עליה מתבססים דיפי והלמן- בעיית **הלוגריתם הדיסקרטי**. הבעיה היא:

*בהינתן g ו- ,gxמה הוא הערך של x?*

כדי לפתור את הבעיה, נכון להיום לא הומצאה שיטה אפקטיבית יותר מהצבות חוזרות של x בפונקציה עד שינתן לנו gx.   
כאשר מדובר במספרים קטנים, התהליך ניתן לביצוע במהירות סבירה יחסית. אך ככל שנגדיל את המספרים, ונבחר אותם ע"י מספר חוקים שאפרט עליהם, הבעיה נהיית קשה מאוד לפתירה אפילו על ידי המחשבים החזקים ביותר.  
כל המתמטיקה סביב הבעיה נעשית בחשבון מודולרי, אותו נכיר כעת:

חשבון מודולרי הוא חשבון בו אנו מחליפים מספרים הגדולים מ-n בשארית החלוקה שלהם עם n. דוגמא טובה לחשבון מודולרי היא בשעון, בו n=24, כלומר מודולו 24. נניח השעה היא 7:00 בבוקר, ואנחנו רוצים לחשב את השעה לאחר 4 שעות. נעשה זאת באמצעות חשבון מודולרי:

7 + 4 mod 24 = 11

כאן התוצאה קטנה מ-24, ולכן לא נעשה כל שינוי. אך אם נרצה לחשב את השעה 5 שעות לאחר 22:00:

22 + 5 mod 24 = 3

התוצאה שווה ל-3 מכיוון ש-27 גדול ממספר המודולו שלנו (שעה כזו לא קיימת בשעון), אז אנו מחלקים אותו ב- 24 ומשתמשים בשארית, שהיא 3. דוגמאות לחישובים במודולו נוספים:

5 + 6 mod 7 = 4 (11 / 7)  
20 + 50 mod 30 = 10 (70 / 30)

עכשיו, כאשר אנו מכירים חשבון מודולרי, ננסה לפתור את בעיה הלוגריתם הדיסקרטי- נבחר מודולו p, בסיס g, ומעריך x, ו נציב ב- gx mod p:

p = 23, g = 5, x = 6  
56 mod 23 = 15625 % 23 = **8**

כעת, נניח שאנו יודעים את כל הנתונים חוץ מ-x. במצב כזה, כדי למצוא את ה-x עלינו לנסות להציב מספר אחר מספר בפונקציה עד שנגיע לתוצאה 8.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | 1 | 2 | 3 | 4 | 5 | 6 |
| 5x | 5 | 25 | 125 | 625 | 3125 | 15625 |
| 5x mod 23 | 5 | 2 | 10 | 4 | 20 | **8** |

מהטבלה נובע כי 6 הוא ה-x שחיפשנו.

שיטה זו לפיתרון הלוגריתם הדיסקרטי אינה אפקטיבית כלל בהינתן מודולו ו-x גדולים. כל זאת כאשר אנחנו בוחרים את p, ו-g תחת החוקים הבאים:  
 - על p (המודולו שלנו) להיות מספר ראשוני.  
 - על g להיות שורש פרימיטיבי (גנרטור) של מודולו p.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | 1 | 2 | 3 | 4 | 5 | 6 |
| 3x | 3 | 9 | 27 | 81 | 243 | 729 |
| 3x mod 7 | 3 | 2 | 6 | 4 | 5 | 1 |

שורש פרימיטיבי של מספר ראשוני p הוא מספר שכל החזקות שלו מ-1 ל- (p-1) במודולו p מניבות תוצאות שונות. לדוגמא, 3 הוא שורש פרימיטיבי של 7 מכיוון שאם נעלה את 3 בחזקת 1,2,3,4,5,6 מודולו 7- נקבל תוצאות שונות.

בעצם כאשר אנו משתמשים ב-p ראשוני וב-g שורש פרימיטבי, יווצר לנו מערך של מספרים מעורבבים, בין 1 ל- (p-1), שאידיאלי לשימוש בהצפנה.

נמשיך להגדרה הבסיסית של בעית דיפי הלמן, הנשענת על בעית הלוגריתם הדיסקרטי, שהיא:

*בהינתן מספר בסיס g, והערך של ga ו- gb (המספרים a,b שלמים): מה הוא הערך של gab?*

אפשר להבין שפתירת הבעיה כשאנחנו יודעים רק את הערכים של g, *ga ו- gb*, היא מאוד קשה ללא ידיעת הערכים של a ו- b.  
בפרק הבא נראה כיצד אנחנו זוקפים את הקושי הזה לזכותנו, כאשר ננצל אותו למען יצירת מפתח הצפנה סודי משותף בין שני צדדים שלא נפגשו מעולם.

*מימוש בפועל*

נתאר כעת את תהליך חילוף המפתחות של דיפי-הלמן בין שני צדדים. נניח שבצד א' יושב בוב, ובצד ב' יושבת אליס:

1. בוב ואליס מסכימים על הפרמטרים g ו-p. נניח שבוב, אשר יזם את התקשורת, החליט על g=3, p=223, ושלח את הפרמטרים לאליס על גבי הערוץ הבלתי מאובטח.

g = 3, p = 223

1. אליס מחליטה על מספר סודי a = 9. לאחר מכן מחשבת *A* = *g****a*** mod *p*:

A = 39 mod 233 = 59

1. בוב מחליט על מספר סודי b = 17. לאחר מכן מחשב *B* = *g****b*** mod *p*:

B = 317 mod 233 = 194

1. אליס שולחת לבוב את A, ובוב שולח לאליס את B.

בוב

אליס

59

194

1. אליס שקיבלה את B מבוב, מחשבת את *s* = *Ba* mod *p*:

s = 1949 mod 233 = **111**

1. בוב שקיבל את A מאליס, מחשב את *s* = *Ab* mod *p*:

s = 599 mod 233 = **111**

1. בוב ואליס כעת חולקים מספר סודי משותף: המספר 111. במפתח זה נוכל להשתמש להצפנת התקשורת מאילך.

בוב ואליס הגיעו לאותו מספר, מכיוון ש - (*ga* mod *p*)*b*mod *p* שווה ערך ל- (*gb* mod *p*)*a* mod *p*. שימו לב ש- p, g, *ga*, *gb* נשלחים בתקשורת באופן פומבי, וכל צד שלישי המצותת יכול לשים את ידו על הנתונים הללו. אבל האם ניתן לחשב את *gab* על סמך הנתונים הללו בלבד? לא בלי לגלות את a ו-b, בגלל בדיוק אותה בעיה מתמטית שהוסברה בפרק הקודם.

חילוף המפתחות שהרגע ביצענו הוא החלק הראשון בתהליך של אבטחת התקשורת בין שני הצדדים.  
החלק השני יהיה לבחור אלגוריתם הצפנה שמתאים לצרכים שלנו, ולדאוג להפוך את המפתח שלנו לאחד שמותאם לקריטריונים של האלגוריתם. השלב השלישי הוא להצפין את המידע היוצא ולפענח את המידע הנכנס לתוכנה ומהתוכנה שלנו, על ידי אותו אלגוריתם.

**RC4**

אלגוריתם ההצפנה הסימטרי שבחרתי להשתמש בו בפרויקט זה הוא RC4.  
**RC4**, הוא “stream cipher” (צופן "זרם"), הנמצא בשימוש רב מאוד בעולם התוכנה. הוא פותח על ידי רון ריבסט ב-1987, והוא מאופיין בתור קל למימוש, מהיר, ויחסית מאוד פשוט. הוא ממומש באפליקציות רבות, ביניהם הXBOX של Microsoft (בהגנות נגד העתקת תוכנה), SSL (פרוטוקול תקשורת מאובטחת), WEP (אבטחת רשתות אלחוטיות).

במהלך השנים, קריפטו-אנליזה של האלגוריתם RC4 מצאה כמה חולשות אבטחה, וכיום הצופן לא עומד בסטנדרטים המחמירים של האבטחה בימינו. למרות כך, הוא מתאים לצרכי הפרויקט, כיון שאני אוכל לממש אותו בעצמי במקום להשתמש במימושים של אלגוריתמים על ידי צד שלישי. בנוסף, פשטותו מאפשרת לי להסבירו בלי הסטת הזרקור מאלגוריתם Diffie-Hellman ושאר המרכיבים בתוכנה, עליהם נושא הפרויקט.

צופן "זרם" הוא שם כולל למשפחה של צפנים שפועלים בדרך של שינוי בית אחר בית ע"י שילובם עם הבית המתאים ב"זרם" המפתח (keystream).

מה שה RC4עושה, בפשטות יתרה, זה להפוך מפתח בגודל בין 5 ל-256 בתים (אותו אנחנו מספקים), לזרם בתים פסודו-אקראי, ה-keystream.  
כאשר אנחנו שולחים להצפין בית על ידי האלגוריתם, פונקצית ההצפנה מחשבת את הבית הבא בזרם המפתח, ומפעילה את פעולת ה הXOR\* (פעולה בינארית דו-כיוונית) על בית הקלט עם בית המפתח. כשאנו מצפינים רצף בתים (ההודעה שלנו), התוצאה היא שילוב של הקלט שלנו עם זרם המפתח, שלא ניתן לשחזר בלי המפתח ששומש בהצפנה.  
ממבט אבסטרקטיבי, מאתחלים את ה-keystream על ידי פונקציה: stream = RC4Stream(key), כאשר key זה המפתח שלנו, ולאחר מכן פונקצית ההצפנה/פיענוח תראה כך- *ciphertext = stream.RC4(text)*, כאשר text זהו המידע שברצוננו להצפין וה- ciphertext זהו פלט ההצפנה.  
מכיוון שמדובר בפונקציה הצפנה סימטרית, הפיענוח נעשה באותה צורה: *text = stream.RC4(ciphertext).*

לאחר שימוש בפונקצית ההצפנה/פיענוח, ה-keystream "יתקדם" הלאה והטקסט הבא שנצפין/נפענח לא יתן לנו את אותו הפלט.

XOR (exclusive-or): פעולה הלוקחת שני ביטים- אם הביטים שווים, התוצאה היא 0. אם הביטים שונים, התוצאה היא 1. טבלת חישוב:

|  |  |  |
| --- | --- | --- |
| XOR | b | a |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

מכאן נובעות כמה תכונות:

- XOR היא פעולה דו-כיוונית: *(a XOR b) XOR a = b*, זה הופך את הפעולה לאידיאלית בשימוש בצופן סימטרי- בו נצפין ונפענח באמצעות אותו המפתח.  
- XOR של מספר עם עצמו, שווה ל- 0.  
- XOR היא פעולה חילופית: *a XOR b XOR c = c XOR a XOR b*, סדר האופרנדים לא משנה, ולסוגריים אין משמעות.  
- XOR הינה פעולה שבה הסתברות התוצאות היא חצי-חצי. תכונה זו הופכת אותה לאידיאלית בקריפטוגרפיה.

**המפתח של RC4**

כאשר משתמשים בצופן זרם כגון RC4, אנחנו נחשפים לחולשה הנקראת "Reused key attack"- חולשה שצד שלישי יכול לנצל כאשר אנחנו משתמשים באותו key להצפנה של יותר מהודעה אחת.

נניח שאנחנו משתמשים באותו keystream להפצנה של יותר מהודעה אחת בתקשורת. אנו בעצם מבצעים את ההצפנה של הודעה א' והודעה ב' באופן הבא:

RC4(messageA) = messageA XOR keystream  
RC4(messageB) = messageB XOR keystream

כאשר הXOR מתבצע בית-אחר-בית.  
כעת נניח שמישהו מצותת לנו לתקשורת, והצליח ליירט את שתי ההודעות הללו. עכשיו הוא יכול לחשב:

RC4(messageA, key) XOR RC4(messageB, key) =   
(messageA XOR keystream) XOR (messageB XOR keystream)

ציינו קודם שפעולת ה- XOR היא פעולה חילופית, כלומר הביטוי למעלה שווה ערך ל:

messageA XOR messageB XOR keystream XOR keystream

מכיוון שXOR של מספר עם עצמו שווה ל0, הביטוי שווה ערך ל:

messageA XOR messageB

כלומר, מXOR של שתי ההודעות המוצפנות השגנו את ערך הXOR של שתי ההודעות המקוריות. אנחנו אמנם לא יודעים את התוכן של ההודעות, אבל אם אנחנו יודעים שההודעות מכילות מילים למשל באנגלית, אפשר להתקרב לערך המקורי של ההודעות על ידי השוואה של חלקים שונים של הצופן עם תווים או מילים נפוצות בשפה האנגלית (מתקפה שידועה בשם Correlation attack) אם נשיג יותר הודעות, זה אף יקל על התהליך ויהיה ניתן לפצח את הצופן בזמן קצר.

לכן, כאשר אנחנו משתמשים בצופן-זרם, **אסור לנו להשתמש במפתח יותר מפעם אחת**.

הבעיה הזו רלוונטית לנו מכיוון שכאשר אנחנו רוצים להשתמש ב RC4עבור ערוץ תקשורת דו כיווני (כלומר, שני הצדדים גם שולחים וגם מקבלים מידע בערוץ), אם נשתמש באותו זרם ההצפנה: נצטרך להתמודד עם מצב בו צד א' מצפין הודעה (וזרם המפתח מתקדם הלאה) אל צד ב', אך צד ב' לא מסונכרן לגבי מצב הזרם ומצפין הודעה לצד א' עם אותו זרם המפתח.

כאשר נממש את ה-RC4 בפרויקט, נוודא כי לא קורה מצב בו ה-keystream חוזר על עצמו חלילה.

למעשה, נפצל את המפתח שיצרנו בתהליך חילוף המפתחות לשניים, וניצור שני ערוצים (RC4Stream) של הצפנה/פיענוח, המשתמשים במפתחות שונים. ערוץ RC4 אחד יהיה עבור תקשורת מצד א' לצד ב', והערוץ השני יהיה מיועד לכיוון ההפוך, מצד ב' לצד א'. כך, אנחנו מאפשרים יותר שליטה על מצב הזרמים וביכולתנו לוודא שה- keystreams פועלים באופן של אל-חזור- מכיוון שברגע שנצפין הודעה באמצעות אחד מה- RC4Streams, השלב של הזרם יתקדם הלאה ואין RC4Stream דומה לו שעלול להצפין משהו באותו שלב של הזרם.

כעת, אנחנו צריכים להפוך את המפתח דיפי-הלמן לזוג מפתחות RC4. חוץ מהצורך שלנו להוציא מפתחות שונים בהחלט כדי ליצור שני זרמי RC4 שונים, אנחנו צריכים מפתחות בגודל המתאים לאלגוריתם (רצוי בין 20 ל-40 בתים). לשם כך נעשה שימוש בפונקצית גיבוב.

* פונקצית גיבוב (hash function): פונקציה שלוקחת קלט בעל אורך בלתי מוגדר, וממירה אותו לפלט באורך קבוע. פונקצית גיבוב טובה היא פונקציה שכל שינוי קטן בקלט המקורי שלה יהווה שינוי משמעותי בפלט. לא ניתן לשחזר פלט של פונקצית גיבוב חזרה לקלט המקורי.

תחילה, ניקח את מפתח הדיפי-הלמן, ונחבר אליו מחרוזת- כל מפתח מחרוזת אחרת (למען הנוחות, נחבר מחרוזת המתארת את כיוון הזרם שאותו המפתח ישמש). בגלל חיבור מחרוזות שונות למפתח, פונקצית הגיבוב תבצע שינוי מספיק משמעותי כדי ליצור לנו מפתחות סופיים שונים לגמרי אחד מהשני.

נעשה שימוש בפונקצית גיבוב הנקראת SHA-1. זוהי פונקצית גיבוב נפוצה שפותחה על ידי רשות ה-NSA האמריקאית ב-1995, והיא תואמת את הצורך שלנו מבחינת התאמת אורך הפלט שלה לדרישות אלגוריתם ה-RC4. בסופו של דבר, זוג המפתחות שלנו יראה כך:

*RC4\_Key1 = SHA1(DH\_key + “server-to-client”) = 0x...  
RC4\_Key2 = SHA1(DH\_key + “client-to-server”) = 0x...*

הפונקציה תניב לנו שני מפתחות סופיים באורך 160 ביט (20 בתים(, בהם נשתמש להצפנה/פענוח המידע, בהתאם לזרם המתאים.

*שרת צ'אט מרובה לקוחות*

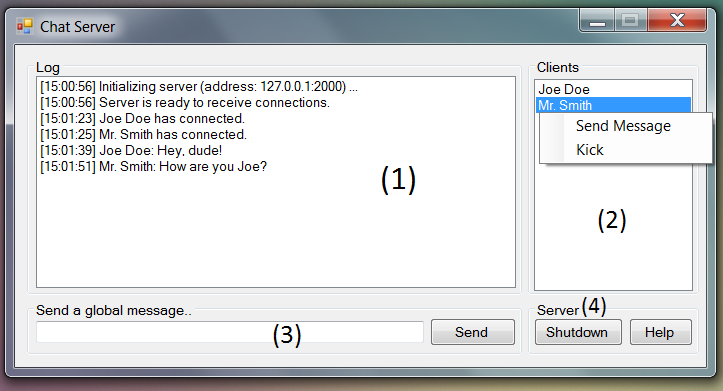
בפרויקט זה בחרתי ליצור תוכנת צ'אט מרובה משתתפים, שהדגש בה היא אבטחת התקשורת ביניהם.

את קונספט אבטחת ערוץ התקשורת בין שני צדדים אני מיישם במודל שרת-לקוח.

**צד השרת:**

השרת הוא תוכנה מרכזית שמקבלת התקשרויות מלקוחות- השרת מהווה את בסיס תקשורת הצ'אט בכך שהוא אחראי על ניהול הלקוחות, העברת מסרים מלקוח אחד ללקוח שני ומהווה מרכז שליטה בעל פיצ'רים אדמינסטרטביים כגון גירוש לקוחות, שליחת הודעות-מנהל לחברי הצ'אט, וכו'.  
באחראיות צד השרת:

* אחראי על בחירת הפרמטרים הפומביים (g, p) של Diffie-Hellman.
* יוזם את חילוף המפתחות: השרת שולח ללקוח את הפרמטרים של Diffie Hellman ואת המפתח הפומבי שלו, ולפי פרמטרים אלו מבצע חילוף המפתחות.
* בעל רשימה של כל הלקוחות המחוברים, כאשר לכל לקוח:
  + ערוץ תקשורת TCP שדרכו הוא שולח ומקבל הודעות.
  + מפתח דיפי הלמן שונה, תוצר מחילוף מפתחות שנעשה עם פרמטרים הייחודים לאותו לקוח.
  + מפתח הצפנה ומפתח פיענוח RC4 היחודיים לערוץ התקשורת של השרת ואותו לקוח (נגזרת של מפתח דיפי-הלמן ייחודי).
* שורת פקודות, בה מנהל השרת יוכל להריץ פקודות שמממשות פונקציות אדמין בסיסיות, כגון שליחת הודעה לכל הלקוחות והעפת משתמש מסוים.
* "לוג" (לוח הודעות) ויזואלי שמתעד את כל ההודעות והערות שעוברות דרך השרת.

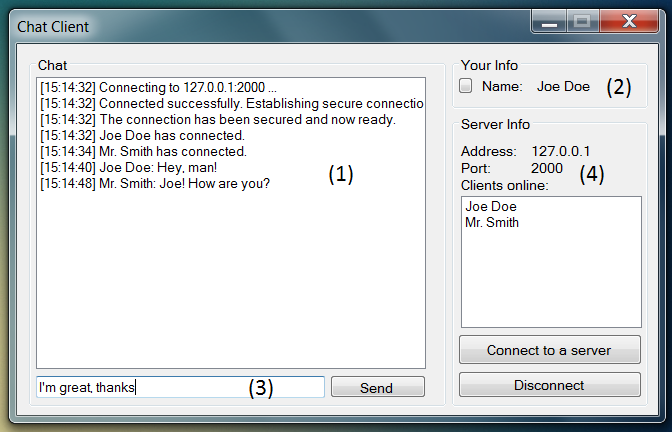


1. לוג הודעות: בחלקה זו של התוכנה יכתבו תיעודים של כל ההודעות הנשלחות על ידי הלקוחות, אירועים (התחברויות, התנתקויות, וכו') שמתרחשים בשרת.
2. רשימה של כל הלקוחות המחוברים לשרת. לחיצה ימנית על אחד הלקוחות יפתח תפריט של אפשרויות לגבי אותו לקוח. ניתן לשלוח לאותו לקוח הודעה, או להוציא אותו מהשרת.
3. בשורת קלט זו ניתן לכתוב הודעה ולשלוח אותה לכל הלקוחות המחוברים בשרת.
4. אפשרויות של סגירת השרת או הסבר קצר על התוכנה.

**צד הלקוח:**

הלקוח הוא משתמש בצא'ט, הבוחר לאיזה שרת להתחבר באמצעות כתובת IP ופורט. כאשר המשתמש בוחר להתחבר לשרת, התוכנה יוזמת התחברות.

* הצד הפסיבי בחילוף המפתחות. לאחר יצירת התקשורת, תוכנת הלקוח מחכה להודעה המודיעה לה על הפרמטרים שעליה להשתמש בהם (g, p). לאחר קבלת הפרמטרים מהשרת, הלקוח יוצר מימוש של דיפי-הלמן לפי פרמטרים אלו. רק לאחר שהלקוח שולח לשרת את המפתח הפומבי שלו, ניתן להתחיל את התקשורת המוצפנת בין שני הצדדים.
* בעל ערוץ תקשורת אחד, והוא עם השרת. דרך השרת הלקוח מעביר הודעות לשאר הלקוחות.
* תוכנת הלקוח בעלת ממשק נוח למשתמש, שמספקת:
  + שורת קלט בה המשתמש כותב את הודעותיו לשרת.
  + פרטים כלליים לגבי השרת אליו הלקוח מחובר, כגון כתובת ורשימה של כל הלקוחות האחרים המחוברים לשרת, שמתעדכנת בהתאם.
  + אפשרות לשנות את שם הלקוח שמייצג אותו בשרת.
  + אפשרות להתחבר לשרת לפי כתובת, ולסגור את התקשורת על ידי התנתקות.



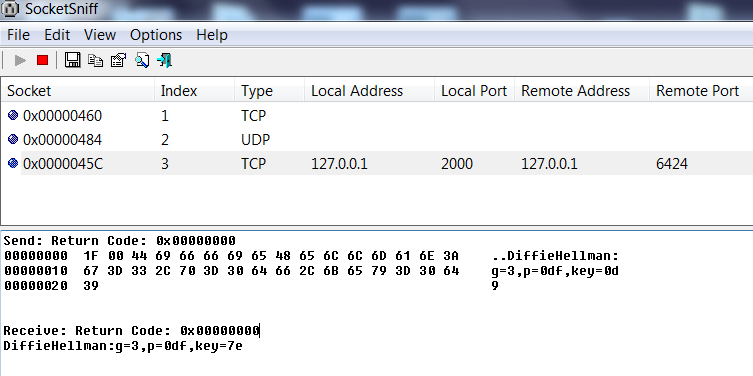
1. צא'ט: כאן המשתמש מקבל את כל ההודעות הנשלחות אליו מהשרת ומשאר הלקוחות.
2. שם המשתמש: השם איתו מיוצג הלקוח בצד השרת ומול שאר הלקוחות. ניתן לשנות את השם (כאשר במצב התוכנה במצב ניתוק משרת) על ידי לחיצה על הכפתור ליד.
3. שורת הקלט: בשורת הקלט המשתמש כותב את הודעותיו, ושולח אותם לשרת לשאר הלקוחות.
4. פרטי השרת ואפשרויות: כאן כתובת השרת הנוכחי ורשימה של הלקוחות המחוברים לשרת. נמצאת כאן האפשרות להתחבר לשרת על פי כתובת, ולהתנתק מהשרת הנוכחי.

*מנקודת מבטו של מצותת*

מכיוון שהרעיון המרכזי של הפרויקט ניכר מאחורי הקלעים, בתקשורת, אמחיש את התוצר הסופי של הפרויקט על ידי דימוי צד שלישי המצותת לתקשורת.

לצורך ההדגמה, אשתמש בקומבינצית g וp חלשה (p קטן מ-4 בתים(, כדי שיהיה ניתן לראות את חילוף המפתחות בבירור.

השתמשתי בתוכנה הנקראת SocketSniff על ידי ניר סופר כדי לצותת לתקשורת, והתחלתי את תוכנת השרת ותוכנת הלקוח. הנה התוצאות:



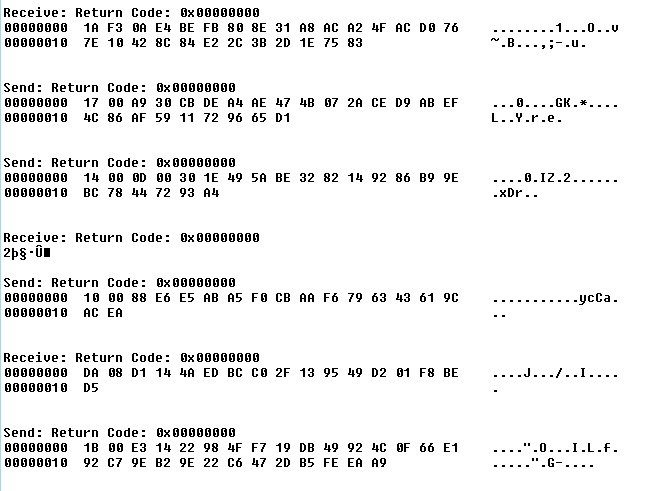
כאן ניתן לראות את שתי ההודעות הראשונות בתקשורת, שהן חילוף המפתחות. מימשתי בפרויקט כך שהודעה שקשורה לחילוף המפתחות תתחיל ב-DiffieHellman ונקודתיים. לאחר מכן יצוינו הפרמטרים, g, p, ו-key (המפתח הפומבי(.

ניתן לראות ב-Send, כי השרת שולח ללקוח את פרטי הדיפי-הלמן 0x3 (g), 0x0df (p), ו-0x0d9 (key).  
אלו אותם ערכים שהשתמשתי להדגמה של דיפי-הלמן בתחילת המסמך. השרת, באמצעות המפתח הפרטי שלו (ערך סודי כלשהו) חישב את 0x0d9.

כעת, הלקוח מקבל את פרטי הדיפי-הלמן מהשרת, ומאתחל אצלו תהליך Diffie-Hellman בהתאם. הוא מאמץ את הערכים g ו-p אשר ניתנו לו מהשרת, ממציא מפתח פרטי רנדומלי (ערך סודי שלא ידוע לנו) ומחשב את המפתח הפומבי בהתאם. לאחר מכן, הוא שולח בחזרה לשרת את הפרטים, כפי שניתן לראות ב-Receive:

0x3 (g), 0x0df (p), ו-0x7e (key).

בשלב הבא, הנסתר מעינינו, השרת והלקוח מחשבים את המפתח המשותף, על ידי המפתח הסודי שברשותם והמפתח הפומבי שקיבלו מהצד השני. לאחר חישוב המפתח המשותף, כל צד יוצר RC4Stream ומתחיל להצפין את המידע. מכאן הלאה, זה בערך המידע שנראה:



המידע הזה נראה כמו ג'יבריש, וזאת מכיוון שהוא מוצפן. למעשה, רק שתי ההודעות הראשונות לא מוצפנות וניתן לקרוא אותם. כל הודעה לאחר חילוף המפתחות מוצפנת ב-RC4, באופן שהמצותת לא יוכל לפענח אותה.

*סיכום*

בסופו של דבר, הפרויקט מהווה מן מימוש דל של SSL (Secure-Socket Layer, בו יש אפשרות להשתמש בקומבינציה של Diffie Hellman & RC4), רק בלי אותנטיקציה.

האם המערכת הזו חסינה לכל מתקפה? בהחלט לא.

אם נשתמש בפרמטרים חזקים וגדולים מספיק (בפרויקט אני משתמש ב-p בגודל של 2048 ביט (מספר בעל יותר מ-2000 ספרות) שזה נחשב מספיק חזק בשביל מימושים מסחריים), רמת ההצפנה שלנו בהחלט תהיה טובה מספיק כדי להתמודד עם כל איום שקיים בימינו. למען המחשה, כדי לפתור את בעיית דיפי-הלמן ולפצח את ההצפנה שלנו (כשאנו משתמשים ב-p בגודל הנוכחי), למחשב רגיל יקח יותר מאלף שנים של חישובים כדי לעשות זאת.

אך יש כמה מתקפות שבפרויקט זה לא ביצעתי הגנות מפניהן, כמו Man-in-the-middle- אם נניח שבין צד א' לצד ב' יושב צד ג' שדרכו עוברות כל ההודעות בתקשורת, הוא מסוגל "לחקות" את הצד השני ובעצם לנהל עם הצדדים תקשורת כאילו הוא המשתמש האמיתי: אם יבצע חילוף מפתחות מול השרת כאילו הוא הלקוח, ועם הלקוח כאילו הוא השרת, ולאחר מכן יעביר את התקשורת בין השרת ללקוח כאילו אין אף אחד באמצע, הוא מקנה לעצמו את היכולת לפענח את המידע העובר בין שני הצדדים (ואף לשנות אותו כרצונו).

ניתן למנוע מתקפות מסוג man-in-the-middle באמצעות אותנטיקציה (authentication). אותנטיקציה היא תהליך שבו צד א' מאשר שצד ב' הוא מי שהוא טוען שהוא, באצמעות בדיקה אצל צד ג' הנחשב אמין (רשות סרטיפיקציות כלשהי).

את המטרה הבסיסית של של הפרויקט, שהיא ליצור תקשורת מוצפנת, הגשמתי במלואה. קיימות הרבה אלטרנטיבות שיכלו להגשים את המטרה הזו באופן לא פחות טוב מדיפי-הלמן ו-RC4- כגון RSA, הצפנה א-סימטרית שפותחה ב-1977 ונחשבת לפורצת דרך בתולדות ההצפנה המודרנית.

בחרתי באלגוריתם דיפי-הלמן מכיוון שהוא הקלאסי מהאלגוריתמים. הוא הדוגמא הממשית הראשונה ביותר להצפנה פומבית, והביא להתפתחויות רבות בעולם הקריפטוגפיה. כאשר קראתי עליו לראשונה, הפשטות (היחסית) של האלגוריתם הרשימה אותי מאוד, והחלטתי שאני רוצה לממש אותו כחלק מפרויקט גדול יותר, של אבטחה באינטרנט.

המחקר והכתיבה של הפרויקט הקנו לי המון ידע בנושאים שעניינו אותי מאוד: הצפנות, פרוטוקולים של תקשורת, אבטחת מידע, ומידע רב לגבי חולשות, מתקפות שונות, ובעיות שונות שהאינטרנט מתמודד איתן כיום.

*בבליוגרפיה*

* Diffie, W.; Hellman, M.E., "New directions in cryptography," Information Theory, IEEE Transactions on , vol.22, no.6, pp.644,654, Nov 1976  
  doi: 10.1109/TIT.1976.1055638  
  URL: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1055638&isnumber=22693>
* The RC4 Stream Encryption Algorithm  
  2005 William Stallings  
  URL: <http://cse.spsu.edu/afaruque/it6833/rc4.pdf>
* Paar, C.; Pelzl, J., “Understanding Cryptography”, A Textboox for Students and Practitioners  
  2010, XVIII, p. 29-54 (Chapter 2 – Stream Ciphers), Springer  
  ISBN: 978-3-642-04100-6
* T. Kivinen; M. Kojo, “More Modular Exponential Diffie-Hellman groups for Internet Key Exchange”  
  May 2003, RFC 3526  
  URL: https://www.ietf.org/rfc/rfc3526.txt

*קוד*

*\*לא כולל קבצי Designer, Auto-Generated של Visual C#, ו-Forms משניים.  
ניתן להוריד את קבצי הפרויקט בשלמותם דרך Dropbox, בלינק:*

https://www.dropbox.com/sh/bqpa4782pqw524b/AADry4KPTUVU-3iZHBMA58W3a?dl=*0*

DiffieHellman.cs

using System;

using System.Numerics;

using System.Security.Cryptography;

public class DiffieHellman

{

/\*\*\* Config \*\*\*/

const int SECRET\_KEY\_SIZE = 16; // Minimum byte length of secret key

/\*\*\* Diffie Hellman variables \*\*\*/

int g; // Generator number

BigInteger p; // Prime modulo number

BigInteger secretKey;

public BigInteger publicKey { get; private set; }

/\* Default class constructor: Use predefined values \*/

public DiffieHellman(BigInteger prime, int generator)

{

this.p = prime;

this.g = generator;

GenerateKeys();

}

/\* Generate the secret and public keys \*/

void GenerateKeys()

{

this.secretKey = GenerateRandomKey(SECRET\_KEY\_SIZE);

this.publicKey = CalculatePublicKey();

}

/\* Calculate and return the public distributable key \*/

BigInteger CalculatePublicKey()

{

return BigInteger.ModPow(this.g, this.secretKey, this.p);

}

/\* Generate a cryptographicly strong random integer \*/

BigInteger GenerateRandomKey(int sizebytes)

{

byte[] bytes = new byte[sizebytes + 1];

// Use RNGCryptoServiceProvider class to generate a random sequence of bytes:

new RNGCryptoServiceProvider().GetBytes(bytes);

bytes[sizebytes] = 0x0;

return new BigInteger(bytes);

}

/\* Calculate the shared Diffie-Hellman secret using the other party's public key \*/

public BigInteger CalculateSharedSecret(BigInteger otherKey)

{

return BigInteger.ModPow(otherKey, this.secretKey, this.p);

}

/\* Pack g, p, and publicKey so we can send it to the other DH party \*/

public string ToExchangeString()

{

return string.Format("DiffieHellman:g={0:x},p={1:x},key={2:x}", this.g, this.p, this.publicKey);

}

}

RC4Stream.cs

class RC4Stream

{

/\* RC4 state table \*/

byte[] table = new byte[256];

/\* RC4 Stream constructor: Key-scheduling algorithm (KSA) \*/

public RC4Stream(byte[] key)

{

// Initialize the permutation:

for (int i = 0; i < 256; i++)

table[i] = (byte)i;

// Scramble the permutation according to the key:

int j = 0;

for (int i = 0; i < 256; i++)

{

j = (j + table[i] + key[i % key.Length]) % 256;

// Swap values of table[i] and table[j]

byte t = table[i];

table[i] = table[j];

table[j] = t;

}

}

/\* RC4 Cipher function: Encrypt or decrypt input (PRGA) \*/

public byte[] RC4(byte[] input)

{

byte[] output = new byte[input.Length];

int counter = 0;

int i = 0;

int j = 0;

while (counter < input.Length)

{

i = (i + 1) % 256;

j = (j + table[i]) % 256;

// Swap values of table[i] and table[j]

byte t = table[i];

table[i] = table[j];

table[j] = t;

output[counter] = (byte)(input[counter] ^ table[(table[i] + table[j]) % 256]);

counter++;

}

return output;

}

}

dh\_server (Server Program)

ServerMain.cs

using System;

using System.Collections.Generic;

using System.Windows.Forms;

using System.Net;

using System.Net.Sockets;

namespace dh\_server

{

/\* ServerMain class: Main class of the server program

\* Contains the main functionality of the server, accepting clients, forwarding messages,

\* and executing commands of the admin.

\*/

public partial class ServerMain : Form

{

/\* "Safe" delegates to add listBox items from multiple threads \*/

delegate void safelyEdit(string text); // To edit controls from multiple threads

const string SERVER\_IP = "127.0.0.1"; // \*\* Run server on local host \*\*

const int SERVER\_PORT = 2000;

TcpListener tcpListener;

List<Client> clients = new List<Client>();

/\* ServerMain(): Main of the program \*/

public ServerMain()

{

InitializeComponent();

}

/\* ServerMain\_Load(): On form load, initialize application \*/

void ServerMain\_Load(object sender, EventArgs e)

{

WriteLog(string.Format("Initializing server (address: {0}:{1}) ...", SERVER\_IP, SERVER\_PORT));

// Initialize Diffie Hellman parameters:

DiffieHellmanParams.Initialize();

// Set up TcpListener..

tcpListener = new TcpListener(IPAddress.Parse(SERVER\_IP), SERVER\_PORT);

tcpListener.Start();

// Set AcceptCallback to be called when there's an incoming connection

tcpListener.BeginAcceptSocket(new AsyncCallback(AcceptCallback), "state");

WriteLog("Server is ready to receive connections.");

AppDomain.CurrentDomain.ProcessExit += new EventHandler(OnProcessExit);

clientsBox.ContextMenuStrip = clientMenuStrip;

}

/\* AcceptCallback(): Callback for incoming clients \*/

void AcceptCallback(IAsyncResult ar)

{

Socket socket = tcpListener.EndAcceptSocket(ar);

tcpListener.BeginAcceptSocket(new AsyncCallback(AcceptCallback), "state");

// Create a new client object:

Client client = new SecureClient(socket, this);

// Now wait for exchange to finish.. (client to send 'OKAY')

}

/\* Broadcast(): Send a message to all connected clients \*/

void Broadcast(string message)

{

foreach (Client client in clients)

{

client.Send(message);

}

}

/\* OnClientMessage(): Called when a client sent the server a message \*/

public void OnClientMessage(Client client, string message)

{

if (message.StartsWith("Hey! My name is... :") && clients.IndexOf(client) == -1) // Message that indicates key exchange finish

{

// Key exchange complete and connection with client established

clients.Add(client);

// Set client name:

int colon = message.IndexOf(':');

string newName = message.Substring(colon + 1);

if (newName.Length >= 3)

client.name = newName;

// Announce new client:

WriteLog(string.Format("{0} has connected.", client.name));

Broadcast(client.name + " has connected.");

AddClientItem(client.name);

// Send the client the list of online chatters:

SendClientList(client);

return;

}

// Sent it to all clients:

string broadcast = string.Format("{0}: {1}", client.name, message);

Broadcast(broadcast);

// Diaplay it in the log

if (broadcast.Length > 40)

broadcast = broadcast.Substring(0, 40) + "...";

WriteLog(broadcast);

}

/\* OnClientMessage(): Called when a client disconnected \*/

public void OnClientDisconnect(Client client)

{

clients.Remove(client); // Remove the client from the clients list

WriteLog(string.Format("{0} has disconnected.", client.name));

Broadcast(client.name + " has disconnected.");

RemoveClientItem(client.name);

}

/\* SendClientList(): Send the list of connected clients to a client \*/

void SendClientList(Client toClient)

{

string listMessage = "CLIENTLIST" + Convert.ToChar(0); // Append CLIENTLIST a 0x0 byte so users can't exploit this

foreach (Client client in clients)

{

listMessage += client.name + ",";

}

toClient.Send(listMessage);

}

/\* When a client in the clients list is clicked \*/

private void clientsBox\_MouseDown(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Right)

{

//select the item under the mouse pointer

clientsBox.SelectedIndex = clientsBox.IndexFromPoint(e.Location);

if (clientsBox.SelectedIndex != -1)

{

clientMenuStrip.Show();

}

}

}

/\* Excecute admin command from server command line ('send' button) \*/

void cmdSend\_Click(object sender, EventArgs e)

{

string msg = cmdLine.Text;

if (msg.Length > 0)

{

cmdLine.Text = "";

Broadcast("Admin: " + msg);

WriteLog("You have sent a global message.");

}

}

/\* OnProcessExit(): Called when the application exists \*/

void OnProcessExit(object sender, EventArgs e)

{

tcpListener.Server.Close();

tcpListener.Stop();

}

/\* WriteLog(): Write a line to the log listBox \*/

void WriteLog(string message)

{

// Handle calls from secondary threads:

if (logBox.InvokeRequired)

{

this.Invoke(new safelyEdit(WriteLog), new object[] { message });

}

else

{

string time = DateTime.Now.TimeOfDay.ToString(@"hh\:mm\:ss");

logBox.Items.Add(string.Format("[{0}] {1}", time, message));

}

}

/\* AddClientItem(): Add a name to the clients listBox \*/

void AddClientItem(string name)

{

// Handle calls from secondary threads:

if (clientsBox.InvokeRequired)

this.Invoke(new safelyEdit(AddClientItem), new object[] { name });

else clientsBox.Items.Add(name);

}

/\* RemoveClientItem(): Add a name to the clients listBox \*/

void RemoveClientItem(string name)

{

if (clientsBox.InvokeRequired)

this.Invoke(new safelyEdit(RemoveClientItem), new object[] { name });

else clientsBox.Items.Remove(name);

}

/\* clientMenuStrip\_ItemClicked(): When an option in the client menu is pressed \*/

private void clientMenuStrip\_ItemClicked(object sender, ToolStripItemClickedEventArgs e)

{

string clientName = clientsBox.Items[clientsBox.SelectedIndex].ToString();

// Find client object based on name:

Client client = null;

foreach (Client c in clients)

{

if (c.name == clientName)

{

client = c;

break;

}

}

if (client == null)

return;

ToolStripItem item = e.ClickedItem;

if (item.Text == "Send Message") // Send a message to selected client

{

ClientSendDialog dialogForm = new ClientSendDialog(clientName);

dialogForm.ShowDialog(this);

string msg = dialogForm.GetMessageText();

dialogForm.Dispose();

if (!string.IsNullOrEmpty(msg))

{

client.Send(msg);

WriteLog("You have sent " + clientName + " a message.");

}

}

else if (item.Text == "Kick") // Kick selected client

{

WriteLog(clientName + " has been kicked from the server.");

client.End();

}

}

/\* shutdown\_Click(): When shutdown button is pressed \*/

private void shutdown\_Click(object sender, EventArgs e)

{

Application.Exit();

}

/\* help\_Click(): When help button is pressed \*/

private void help\_Click(object sender, EventArgs e)

{

string text = "Secure Client-Server by Ron Lipel\n\n" +

"This server manages secure connections with clients and forwards each other's messages.\n" +

"The traffic between the server and the clients is encrypted with RC4 & Diffie Hellman.\n" +

"Right click a connected client in the clients list in order to send a specific message, or kick selected client.\n" +

"You can also send a global admin message using the text input.";

MessageBox.Show(text, "Help");

}

}

}

Client.cs

using System;

using System.Text;

using System.Net;

using System.Net.Sockets;

namespace dh\_server

{

/\* Client class:

\* This class represents a single client that is connected to the server.

\* This class contains all the network functionality regarding to the client: Sending & receiving messages, etc.

\* \*/

public class Client

{

/\* A link to the main form- so we could call general methods from here \*/

ServerMain main = null;

/\* Client fields \*/

Socket clientSocket;

public string name;

/\* Client(): Client constructor \*/

public Client(Socket sock, ServerMain form)

{

main = form;

clientSocket = sock;

// Set default client name to its host name:

IPEndPoint remoteIpEndPoint = clientSocket.RemoteEndPoint as IPEndPoint;

IPAddress clientIP = remoteIpEndPoint.Address;

name = Dns.GetHostEntry(clientIP).HostName;

StartReceive();

}

/\* End(): Dispose the client and kill the connection \*/

public void End()

{

if (clientSocket != null)

{

clientSocket.Shutdown(SocketShutdown.Both);

clientSocket.Close();

clientSocket.Dispose();

clientSocket = null;

main.OnClientDisconnect(this);

}

}

/\* Send(string): Transform the string into bytes and call Send(byte[]) \*/

public virtual int Send(string message)

{

return Send(Encoding.ASCII.GetBytes(message));

}

/\* Send(byte[]): Send the data to the client \*/

public virtual int Send(byte[] data)

{

// Defines the length of the data with a 2-byte header attached to the beginning of the packet:

int datalen = data.Length;

byte[] lengthHeader = new byte[2];

lengthHeader[0] = (byte)(datalen & 0xFF);

lengthHeader[1] = (byte)((datalen >> 8) & 0xFF);

// Header ready, attach it to the data:

byte[] result = new byte[2 + datalen];

lengthHeader.CopyTo(result, 0);

data.CopyTo(result, 2);

// Final packet ready. Send

clientSocket.BeginSend(result, 0, result.Length, 0,

new AsyncCallback(SendCallback), "state");

return data.Length;

}

/\* HandleMessage(): Handle method for incoming messages; forward it to the main program \*/

protected virtual void HandleMessage(byte[] messageData)

{

string message = Encoding.ASCII.GetString(messageData);

main.OnClientMessage(this, message);

}

/\* StartReceive(): Wait for incoming data from socket \*/

void StartReceive()

{

byte[] headerBuffer = new byte[2];

clientSocket.BeginReceive(headerBuffer, 0, 2, SocketFlags.None,

new AsyncCallback(OnReceiveHeader), headerBuffer);

}

/\* OnReceiveHeader(): Received a message length header \*/

void OnReceiveHeader(IAsyncResult result)

{

byte[] header = (byte[])result.AsyncState;

int receivedBytes = 0;

try

{

receivedBytes = clientSocket.EndReceive(result);

}

catch { }

if (receivedBytes == 0)

{

// Connection error

End();

}

else if (receivedBytes == 2)

{

// Transform the two length header bytes into an integer, messageSize

int messageSize = (header[1] << 8) | (header[0] & 0xFF); // Little Endian

if (messageSize > 0)

{

byte[] messageBuffer = new byte[messageSize];

clientSocket.BeginReceive(messageBuffer, 0, messageSize, SocketFlags.None,

new AsyncCallback(OnReceiveMessage), messageBuffer);

}

}

}

/\* OnReceiveMessage(): Received a message after length header \*/

void OnReceiveMessage(IAsyncResult result)

{

byte[] message = (byte[])result.AsyncState;

clientSocket.EndReceive(result);

HandleMessage(message);

StartReceive();

}

/\* Socket asynchronous send callback \*/

void SendCallback(IAsyncResult ar)

{

try

{

clientSocket.EndSend(ar);

}

catch

{

// Connection error

End();

}

}

}

}

SecureClient.cs

using System.Text;

using System.Net.Sockets;

using System.Text.RegularExpressions;

using System.Numerics;

using System.Security.Cryptography;

namespace dh\_server

{

/\* SecureClient class:

\* This class inherits from Client class and adds the encryption and decryption functionality.

\* This class is for the server side of the connection- it decides on the DH parameters and sends it to the client.

\* \*/

public class SecureClient : Client

{

/\* SHA-1 abstract object for SHA-1 hash \*/

static SHA1 sha = new SHA1CryptoServiceProvider();

/\* Diffie Hellman object and DH shared secret: \*/

DiffieHellman DHObject;

BigInteger sharedSecret = BigInteger.Zero;

/\* RC4 streams \*/

RC4Stream rc4InStream; // RC4 from client to server

RC4Stream rc4OutStream; // RC4 from server to client

/\* Security status: True after key exchange completion and RC4 initialization (ready for communication) \*/

bool clientReady = false;

/\* SecureClient(): Constructor \*/

public SecureClient(Socket sock, ServerMain form) : base(sock, form)

{

/\* Create an instance of Diffie Hellman \*/

DHObject = new DiffieHellman(DiffieHellmanParams.RFC\_P, DiffieHellmanParams.RFC\_G);

/\* Tranform the DH object to a transferable string and send it to the client \*/

string exchangeString = DHObject.ToExchangeString();

base.Send(exchangeString);

}

/\* KeyExchangeComplete(): This is called upon key exchange completion \*/

void KeyExchangeComplete()

{

if (sharedSecret != BigInteger.Zero)

{

// We have a shared Diffie-Hellman key. Now create RC4 streams for in & out traffic:

byte[] RC4InKey = CreateRC4Key(sharedSecret, "client-to-server"); // Indicating in stream direction

rc4InStream = new RC4Stream(RC4InKey);

byte[] RC4OutKey = CreateRC4Key(sharedSecret, "server-to-client"); // Indicating out stream direction

rc4OutStream = new RC4Stream(RC4OutKey);

// Set security status to ready and allow further communication

clientReady = true;

}

}

/\* CreateRC4Key(): Transform the shared DH key combined with a string to an RC4 key \*/

byte[] CreateRC4Key(BigInteger masterKey, string otherInfo)

{

byte[] masterKeyBytes = masterKey.ToByteArray();

// Concatenate parameters:

byte[] concatenation = new byte[masterKeyBytes.Length + otherInfo.Length];

masterKeyBytes.CopyTo(concatenation, 0);

byte[] otherInfoBytes = Encoding.ASCII.GetBytes(otherInfo);

otherInfoBytes.CopyTo(concatenation, masterKeyBytes.Length);

// Hash the result:

return sha.ComputeHash(concatenation);

}

/\*\*\*\* Overridden Client class methods: \*\*\*\*/

/\* These are overriden send & receive methods that add decryption and encryption of the traffic \*/

/\* Send(byte[]): Encrypt the data, then use the base's send function to send it \*/

public override int Send(byte[] data)

{

if (clientReady)

{

byte[] encrypted = rc4OutStream.RC4(data);

return base.Send(encrypted);

}

else return base.Send(data);

}

/\* HandleMessage(): Handle an incoming message \*/

protected override void HandleMessage(byte[] messageData)

{

// Are we awaiting an exchange message from the client?

if (!clientReady)

{

string exchangeString = Encoding.ASCII.GetString(messageData);

if (exchangeString.StartsWith("DiffieHellman"))

{

// Use Regular Expressions to extract key:

Match match = Regex.Match(exchangeString, "key=(.\*)$");

if (match.Success)

{

string keyString = match.Groups[1].Value;

BigInteger key = BigInteger.Parse(keyString, System.Globalization.NumberStyles.HexNumber);

sharedSecret = DHObject.CalculateSharedSecret(key);

KeyExchangeComplete();

}

}

}

else

{

// Handling an encrypted message

// Decrypt the message

byte[] decrypted = rc4InStream.RC4(messageData);

// Forward it to Client class

base.HandleMessage(decrypted);

}

}

}

}

DiffieHellmanParams.cs

using System;

using System.Numerics;

public class DiffieHellmanParams

{

/\* \*\*\*\*\*\*\* Diffie Hellman Parameters \*\*\*\*\*\*\*\*

\* This is a 2048-bit (256 bytes) prime number p and its generator, g.

\* These are taken from RFC 3526 (https://www.ietf.org/rfc/rfc3526.txt)

\* and are considered strong enough for today's security standards.

\* \*/

public static BigInteger RFC\_P;

public static int RFC\_G;

public static void Initialize()

{

RFC\_P = BigInteger.Parse

System.Globalization.NumberStyles.HexNumber);

RFC\_G = 2;

/\*RFC\_P = 223; // For demonstration purposes

RFC\_G = 3; \*/

}

}

dh\_client (Client Program)

ClientMain.cs

using System;

using System.Windows.Forms;

namespace dh\_client

{

public partial class ClientMain : Form

{

delegate void safelyEdit(string edit); // To edit controls from multiple threads

Connection connection = null;

string userName;

/\* ClientMain(): Main function to the program \*/

public ClientMain()

{

InitializeComponent();

// Set default username to hostname:

userName = System.Net.Dns.GetHostName();

nameLabel.Text = userName;

// Add OnProcessExit callback for when the app exits

AppDomain.CurrentDomain.ProcessExit += new EventHandler(OnProcessExit);

}

/\* Connect(): Connect to a server \*/

void Connect(string ipaddress, int port)

{

if (connection != null)

return; // There's an ongoing connection

WriteChat(string.Format("Connecting to {0}:{1} ...", ipaddress, port));

serverIpLabel.Text = "Connecting..";

serverPortLabel.Text = "...";

connection = new SecureConnection(ipaddress, port, this);

if (connection.Connect())

{

serverIpLabel.Text = ipaddress;

serverPortLabel.Text = port.ToString();

WriteChat("Connected successfully. Establishing secure connection..");

}

else

{

connection = null;

WriteChat("Failed to connect.");

serverIpLabel.Text = "-";

serverPortLabel.Text = "-";

}

}

/\* OnServerMessage(): Called upon message from server \*/

public void OnServerMessage(string message)

{

if (message.StartsWith(("CLIENTLIST" + Convert.ToChar(0)))) // A clients list message from server

{

ClientListClear();

string list = message.Substring(message.IndexOf(Convert.ToChar(0)) + 1);

foreach (string name in list.Split(','))

{

if(name.Length >= 3)

AddClientItem(name);

}

return;

}

else if (message.EndsWith(" has disconnected.") && message.IndexOf(':') == -1) // A client has disconnected; update the clients list

{

int i = message.IndexOf(" has disconnected.");

string name = message.Substring(0, i);

RemoveClientItem(name);

}

else if (message.EndsWith(" has connected.") && message.IndexOf(':') == -1) // A client has connected; update the clients list

{

int i = message.IndexOf(" has connected.");

string name = message.Substring(0, i);

AddClientItem(name);

}

// Regular message, display in chat

WriteChat(message);

}

/\* OnDisconnect(): Called when the connection ended \*/

public void OnDisconnect()

{

connection = null;

WriteChat("Disconnected from the server.");

SetIPLabel("-");

SetPortLabel("-");

ClientListClear();

WriteChat("---");

}

/\* OnSecureReady(): Called upon key exchange completion \*/

public void OnSecureReady()

{

// Send the key exchange completion message, along with client name:

string introductionMessage = string.Format("Hey! My name is... :{0}", userName);

connection.Send(introductionMessage);

WriteChat("The connection has been secured and now ready.");

}

/\* OnProcessExit(): Called when the app exits \*/

void OnProcessExit(object sender, EventArgs e)

{

if (connection != null)

connection.End();

}

#region UI sets

/\* WriteChat(): Write a line to the chat \*/

void WriteChat(string message)

{

// Handle calls from secondary threads:

if (chatBox.InvokeRequired)

this.Invoke(new safelyEdit(WriteChat), new object[] { message });

else

{

string time = DateTime.Now.TimeOfDay.ToString(@"hh\:mm\:ss");

chatBox.Items.Add(string.Format("[{0}] {1}", time, message));

}

}

/\* AddClientItem(): Add a name to the clients listBox \*/

void AddClientItem(string name)

{

// Handle calls from secondary threads:

if (clientsList.InvokeRequired)

this.Invoke(new safelyEdit(AddClientItem), new object[] { name });

else clientsList.Items.Add(name);

}

/\* RemoveClientItem(): Add a name to the clients listBox \*/

void RemoveClientItem(string name)

{

// Handle calls from secondary threads:

if (clientsList.InvokeRequired)

this.Invoke(new safelyEdit(RemoveClientItem), new object[] { name });

else clientsList.Items.Remove(name);

}

/\* ClientListClear(): Clear the clientsList listbox \*/

void ClientListClear(string unused="")

{

// Handle calls from secondary threads:

if (clientsList.InvokeRequired)

this.Invoke(new safelyEdit(ClientListClear), new object[] { unused });

else clientsList.Items.Clear();

}

/\* SetIPLabel(): Set the value of the IP label \*/

void SetIPLabel(string value)

{

if (serverIpLabel.InvokeRequired)

this.Invoke(new safelyEdit(SetIPLabel), new object[] { value });

else serverIpLabel.Text = value;

}

/\* SetPortLabel(): Set the value of the port label \*/

void SetPortLabel(string value)

{

if (serverPortLabel.InvokeRequired)

this.Invoke(new safelyEdit(SetPortLabel), new object[] { value });

else serverPortLabel.Text = value;

}

#endregion

#region UI button clicks

/\* connectButton\_Click(): Called when the connect button is pressed \*/

private void connectButton\_Click(object sender, EventArgs e)

{

if (connection != null)

{

MessageBox.Show("You should first disconnect from the current server.", "Connect to a server");

return;

}

ConnectDialog dialogForm = new ConnectDialog();

string ip = null;

int port = 0;

dialogForm.ShowDialog(this);

ip = dialogForm.GetServerIP();

port = dialogForm.GetServerPort();

dialogForm.Dispose();

if (ip != null && port > 0 && port <= 65535)

{

// if already connected to a server, Disconnect();

Connect(ip, port);

}

}

/\* connectButton\_Click(): Called when the disconnnect button is pressed \*/

private void disconnectButton\_Click(object sender, EventArgs e)

{

if (connection != null)

connection.End();

}

/\* userSend\_Click(): Called when the send button is pressed \*/

private void userSend\_Click(object sender, EventArgs e)

{

string input = userInput.Text;

if (input.Length > 0)

{

if (connection != null)

{

connection.Send(input);

}

userInput.Text = "";

}

}

/\* nameEdit\_Click(): Called when the edit name button is pressed \*/

private void nameEdit\_Click(object sender, EventArgs e)

{

if (connection != null)

{

MessageBox.Show("You can't change the name while you're connected to a server.", "Change name");

}

else

{

NameDialog dialogForm = new NameDialog(userName);

dialogForm.ShowDialog(this);

string newName = dialogForm.GetNewName();

if (newName != userName && newName.Length > 3 && newName.IndexOf(':') == -1)

{

userName = newName;

nameLabel.Text = newName;

}

}

}

#endregion

}

}

Connection.cs

using System;

using System.Text;

using System.Net;

using System.Net.Sockets;

namespace dh\_client

{

/\* Connection class:

\* This class abstracts all connection-related functionality.

\* Connection represents the connection to the server, and handles sending and receiving from the server.

\* \*/

public class Connection

{

/\* A link to the main form- so we could call general methods from here \*/

protected ClientMain main = null;

/\* Client fields \*/

Socket clientSocket;

string server\_ip;

int server\_port;

/\* Connection(): Connection constructor \*/

public Connection(string ipaddress, int port, ClientMain form)

{

main = form;

clientSocket = new Socket(SocketType.Stream, ProtocolType.Tcp);

server\_ip = ipaddress;

server\_port = port;

}

/\* Connect(): Attempt to connect to server \*/

public bool Connect()

{

try

{

clientSocket.Connect(server\_ip, server\_port);

}

catch

{

return false;

}

StartReceive();

return true;

}

/\* End(): Disconnect and dispose \*/

public void End()

{

if (clientSocket != null)

{

clientSocket.Shutdown(SocketShutdown.Both);

clientSocket.Close();

clientSocket.Dispose();

clientSocket = null;

main.OnDisconnect();

}

}

/\* Send(string): Transform the string into bytes and call Send(byte[]) \*/

public int Send(string message)

{

return Send(Encoding.ASCII.GetBytes(message));

}

/\* Send(byte[]): Send the data to the client \*/

public virtual int Send(byte[] data)

{

// Defines the length of the data with a 2-byte header attached to the beginning of the packet:

int datalen = data.Length;

byte[] lengthHeader = new byte[2];

lengthHeader[0] = (byte)(datalen & 0xFF);

lengthHeader[1] = (byte)((datalen >> 8) & 0xFF);

// Header ready, attach it to the data:

byte[] result = new byte[2 + datalen];

lengthHeader.CopyTo(result, 0);

data.CopyTo(result, 2);

// Final packet ready. Send

clientSocket.BeginSend(result, 0, result.Length, 0,

new AsyncCallback(SendCallback), "state");

return data.Length;

}

/\* HandleMessage(): Handle method for incoming messages; forward it to the main program \*/

protected virtual void HandleMessage(byte[] messageData)

{

string message = Encoding.ASCII.GetString(messageData);

main.OnServerMessage(message);

}

/\* StartReceive(): Wait for incoming data from socket \*/

void StartReceive()

{

byte[] headerBuffer = new byte[2];

clientSocket.BeginReceive(headerBuffer, 0, 2, SocketFlags.None,

new AsyncCallback(OnReceiveHeader), headerBuffer);

}

/\* OnReceiveHeader(): Received a message length header \*/

void OnReceiveHeader(IAsyncResult result)

{

byte[] header = (byte[])result.AsyncState;

int receivedBytes = 0;

try

{

receivedBytes = clientSocket.EndReceive(result);

}

catch { }

if (receivedBytes == 0)

{

// Connection error

End();

}

else if (receivedBytes == 2)

{

// Transform the two length header bytes into an integer, messageSize

int messageSize = (header[1] << 8) | (header[0] & 0xFF); // Little Endian

if (messageSize > 0)

{

byte[] messageBuffer = new byte[messageSize];

clientSocket.BeginReceive(messageBuffer, 0, messageSize, SocketFlags.None,

new AsyncCallback(OnReceiveMessage), messageBuffer);

}

}

}

/\* OnReceiveMessage(): Received a message after length header \*/

void OnReceiveMessage(IAsyncResult result)

{

byte[] message = (byte[])result.AsyncState;

clientSocket.EndReceive(result);

HandleMessage(message);

StartReceive();

}

/\* Socket asynchronous send callback \*/

void SendCallback(IAsyncResult ar)

{

try

{

clientSocket.EndSend(ar);

}

catch

{

// Connection error

End();

}

}

}

}

SecureConnection.cs

using System;

using System.Text;

using System.Text.RegularExpressions;

using System.Numerics;

using System.Security.Cryptography;

namespace dh\_client

{

/\* SecureConnection class:

\* This class inherits from Connection class and adds the encryption and decryption functionality.

\* This class is for the client side of the connection, as it receives Diffie-Hellman parameters from the server and exchanges the key accordingly.

\* \*/

public class SecureConnection : Connection

{

/\* SHA-1 abstract object for SHA-1 hash \*/

static SHA1 sha = new SHA1CryptoServiceProvider();

/\* Diffie Hellman object and DH shared secret: \*/

DiffieHellman DHObject;

BigInteger sharedSecret = BigInteger.Zero;

/\* RC4 streams \*/

RC4Stream rc4InStream; // RC4 from server to client

RC4Stream rc4OutStream; // RC4 from client to server

/\* Security status: True after key exchange completion and RC4 initialization (ready for communication) \*/

bool clientReady = false;

/\* SecureConnection(): Constructor \*/

public SecureConnection(string ipaddress, int port, ClientMain form) : base(ipaddress, port, form)

{

/\* because it is clientside, we first need to get the DH parameters from the server in order to initialize \*/

}

/\* GotDHParameters(): Called once received the DH info from server - all initialization in here \*/

void GotDHParameters(int g, BigInteger p, BigInteger otherKey)

{

// Create Diffie Hellman instance using p, g:

DHObject = new DiffieHellman(p, g);

// Send the server the Diffie Hellman client key:

string exchangeString = DHObject.ToExchangeString();

base.Send(exchangeString);

// Calculate sharedSecret:

sharedSecret = DHObject.CalculateSharedSecret(otherKey);

// End key exchange:

KeyExchangeComplete();

}

/\* KeyExchangeComplete(): This is called upon key exchange completion \*/

void KeyExchangeComplete()

{

if (sharedSecret != BigInteger.Zero)

{

// We have a shared Diffie-Hellman key. Now create RC4 streams for in & out traffic:

byte[] RC4InKey = CreateRC4Key(sharedSecret, "server-to-client"); // Indicating in stream direction

rc4InStream = new RC4Stream(RC4InKey);

byte[] RC4OutKey = CreateRC4Key(sharedSecret, "client-to-server"); // Indicating out stream direction

rc4OutStream = new RC4Stream(RC4OutKey);

// Set security status to ready and allow further communication

clientReady = true;

main.OnSecureReady();

}

}

/\* CreateRC4Key(): Transform the shared DH key combined with a string to an RC4 key \*/

byte[] CreateRC4Key(BigInteger masterKey, string otherInfo)

{

byte[] masterKeyBytes = masterKey.ToByteArray();

// Concatenate parameters:

byte[] concatenation = new byte[masterKeyBytes.Length + otherInfo.Length];

masterKeyBytes.CopyTo(concatenation, 0);

byte[] otherInfoBytes = Encoding.ASCII.GetBytes(otherInfo);

otherInfoBytes.CopyTo(concatenation, masterKeyBytes.Length);

// Hash the result:

return sha.ComputeHash(concatenation);

}

/\*\*\*\* Overridden Client class methods: \*\*\*\*/

/\* These are overriden send & receive methods that add decryption and encryption of the traffic \*/

/\* Send(byte[]): Encrypt the data, then use the base's send function to send it \*/

public override int Send(byte[] data)

{

if (clientReady)

{

byte[] encrypted = rc4OutStream.RC4(data);

return base.Send(encrypted);

}

else return base.Send(data);

}

/\* HandleMessage(): Handle an incoming message \*/

protected override void HandleMessage(byte[] messageData)

{

// Are we awaiting an exchange message from the server?

if (!clientReady)

{

string exchangeString = Encoding.ASCII.GetString(messageData);

if (exchangeString.StartsWith("DiffieHellman"))

{

// This is the Diffie Hellman message from the server- parse it.

// Use Regular Expressions to extract g, p, and key:

Match match = Regex.Match(exchangeString, "g=(.\*),p=(.\*),key=(.\*)$");

if (match.Success)

{

string gString = match.Groups[1].Value;

string pString = match.Groups[2].Value;

string keyString = match.Groups[3].Value;

int g = Convert.ToInt32(gString, 16);

BigInteger p = BigInteger.Parse(pString, System.Globalization.NumberStyles.HexNumber);

BigInteger key = BigInteger.Parse(keyString, System.Globalization.NumberStyles.HexNumber);

GotDHParameters(g, p, key);

}

}

}

else

{

// Handling an encrypted message

// Decrypt the message

byte[] decrypted = rc4InStream.RC4(messageData);

// Forward it to Connection class

base.HandleMessage(decrypted);

}

}

}

}