סוכם על ידי אבי שוע –
shuaavi@gmail.com
http://www.cs.huji.ac.il/~shuaavi
אני מקווה שהסיכומים יעזרו לכם ולעוד רבים – אבל אני חייב להעיר – ייתכן, ואפילו סביר, שיש בהם טעויות. **אני (ואף אחד אחר) לא לוקח אחריות** לאף ציון / נזק שיגרם בגללם.

אבי שוע

<u>מבוא להצפנה ולאבטחת מידע</u>

כדאי להכיר אלגוריתמים ותקשורת.

אוריינטציית הקורס תהיה די מעשית.

נסקור בו כלים וטכנולוגיות בנושאי הצפנה ואבטחת מידע.

הקורס אינו מתמטי, ולא יפתור את כל בעיות האבטחה.

הערה: בד"כ בקורס לא נעשה הפסקה, אלא נעשה שעה וחצי רצוף.

<u>תרגילים ומשימות</u>

אין מתרגל, ואין בודקים. עם זאת, יהיו 6 תרגילים, אבל לא ברור מה יהיה המנגנון של בדיקת התרגילים.

התרגילים יהיו תאורטיים.

המבחן בשנים האחרונות היה בעל 12 שאלות, 3 שעות, בלי בחירה. כל שאלה דנה בנושא מסוים, ובעלת כמה תשובות, שחלקן נכונות. הוא נותן case analysis, לא בודק כמה אתה זוכר טוב את הקורס. המבחן קשה גם לחיבור וגם לענות. המבחן לא קל.

רוב חומר הקריאה נמצא בספריה, אפילו כמה עותקים.

אתר הקורס ינוהל על ידי דינה (http://www.notes-heaven.com) שהתנדבה לעניין (**באופן מפליא**).

הקורס יתחלק לשלוש חלקים:

1. מבוא (שבוע).
2. אלגוריתמי הצפנה, חתימות, וכיו"ב (טיפה יותר מתמטי)
3. פרוטוקולים ומערכות (טיפה יותר סיסטמי).

<u>מבוא לאבטחת מידע</u>

- מורכבות העולם –
  - מחשב אישי
  - טלפון
  - מכשיר סלולרי
  - כספומט
  - GPS
  - שרתי מידע
  - טלויזיה
  - אינטרנט
  - מכוניות

כולם עובדים אלקטרונית, ובצורה ממוחשבת כלשהי. כולם נהפכים יותר ויותר נגישים. בכל תחום יש חברות שונות, עם מהנדסים שונים, והתמחות בפני עצמה. מדובר במערכות לא קוהרנטיות, ולמרות זאת

– מנסים שהכל ידבר עם הכל. יש לכן מאות ממשקים בין המערכות (אם יש n מערכות, יש כ $\sim n^2$ ממשקים) – ויש יותר פתח לבעיות, ועם זה – תקלות אבטחה. הסיכוי שהמערכות עם הסתבכותם יהיו נקיות מתקלות קטן עם הפיכת המערכות למורכבות יותר קטן באופן מהותי.
נשים לב, שבעולם אידילי, לא בהכרח היינו צריכים להתמודד עם אבטחת מידע – אם לא היו גנבים. אבל אנחנו לא חיים בעולם אידילי. אבטחת מידע זה השקעת משאבים כדי להקטין סיכוי לנזק.

'אין סיכוי, שאף מערכת מהסוג שאנו מדברים עליה, תהיה חפה מתקלות'. השאלה היא איך אפשר להתמודד עם זה.

<u>הגדרה</u>: אבטחה –

- <u>למנוע</u> מדברים רעים לקרות.
- הקטנת הנזקים מדברים רעים.
- ספיגת עלויות סבירות. (לא הגיוני להשקיע בביטוח רכב יותר משוויו).

<u>אתגרים / בעיות אבטחה</u>

- אובדן מידע
- גניבת מידע
- השחתת מידע (שתילת מידע מפוברק, שנדמה לי שהוא אותנטי)
- מניעת גישה (Denial Of Service)
- הפצת מידע לא חוקית
- התחזות
- הונאה

<u>מונחים</u>

- Eavesdropping – האזנה (לכן קוראים למאזין בד"כ Eve). בד"כ בהקשרי תקשורת.
- snooping – 'הצצה' פנימה (לדוגמא, מישהו שמחטט לך בקבצים במחשב). בד"כ בהקשרי פריצה.
- Tampering – 'התעסקות' ושינוי מידע – כמו snooping אבל גם שינוי. בד"כ בהקשרי חומרה.
- spoofing – התחזות. להציג את עצמי כמישהו אחר.
- jamming – חסימה. מניעת גישה לשרת לדוגמא.
- Code injection – החדרת קוד עוין.

בקורס נדבר טיפה על Eavesdropping והתמודדות עמו ע"י **<u>הצפנה</u>**. נסביר טיפונת על התמודדות עם snooping באמצעות **<u>מנגנוני הרשאות גישה</u>**. לא נתעסק כמעט במנגנוני הגנות פיזיות. נדבר על

התמודדות עם spoofing באמצעות **זיהוי + אימות.** לא נדבר הרבה על התמודדות עם חסימה וJamming. לא נדבר הרבה על התמודדות עם קוד עוין.

נדבר בעיקר על כלים מתמטיים של הצפנה , אימות וזיהוי, ובחצי השני נדבר על מערכות שמתבססות על הכלים הנ"ל. נדבר על זה בעיקר כי יש לנו דרכי התמודדות עמו.

2 סוגים של בעיות

בעיות טבעיות

בעיות טבעיות הן בעיות שאין לנו שליטה עליהן – אין מישהו שמנסה לגרום להם. לדוגמא:

- נפילת מתח
- רעידת אדמה / שיטפון
- נפילת disk

המדדים הם סיכוי / נזק צפוי, והפתרון יהיה ע"י גיבוי מסוג כלשהו (כולל יתירות)

בעיות יזומות

בעיות שיש מישהו שגרם אליהם.

- התחזות
- פריצה
- גניבה
- שינוי מידע

גם כאן קיים 'סיכוי' – למרות שחישובו שונה, 'הערכת נזק', ו'מנגנוני הגנה'.

'סיכוי' – אם אין לאף אחד אינטרס (כלכלי, פוליטי,...) לפרוץ לאתר שלי, הסיכוי שהוא יפרץ קטן משמעותית. הסיכוי מושפע משיקולים שונים – לדוגמא, פוליטיים, או רמת הקושי לפרוץ למתחרים לדוגמא.

'הערכת נזק'- בהתאם לסוג המערכת (בטחונית, כלכלית, ...)

כאן, מנגנוני ההגנה בד"כ לא יהיו גיבוי.

מנגנון הגנה נוסף, שיכול לעזור בשני המקרים – **ביטוח**.

סוגי פורצים

1. חוקרים אקדמיים – מטרתם היא לחקור איך להגן / לפרוץ למערכות. יש בעולם אלפי חוקרים בנושא. בד"כ עובדים בצורה חוקית, **ומפרסמים את תוצאותיו**. בד"כ נשארים במחקר האקדמי.
2. יועצי אבטחה – עובדים בד"כ עבור חברות בצורה חוקית, אבל **לא יפרסמו בד"כ** את תוצאותיהם מחוץ לארגון.

3. האקרים עצמאיים – בודדים שבד"כ מוכנים להזדהות בצורה זו או אחרת, **ומפרסמים בד"כ כלים / דרכים** לפרוץ מערכות. CDC לדוגמא. מטרתם להציג לעולם כמה מערכות הן לא בטוחות. 'אם היית יודע שלGM יש מכונית מסוכנת, תפקידך לדווח על זה לעולם'. חלקם רואים את עצמם כפעילים חברתיים. יתכן שחלקם שייכים לקטגוריה 4. קיימים אתרים, כנסים, ועיתונות שלהם.
4. ארגוני פשע – קטגוריה זו ו5 הן לא חוקיות. מטרתם היא כסף/טרור/.... יש מעט, אבל הנזק שלהם בד"כ עצום בכל תקרית. כאן יש המון כסף, והמון תשתית.
5. חובבנים – כמו 4, אבל פורצים לשם אגו, הטרדה ועניין. 'בד"כ בנים בני 12 – 15 עם קוקו'. יש הרבה יותר כאלה מקטגוריה 4. בד"כ הנזקים שלהם די קטנים פר תקרית, אבל יש המון כאלה.

מי שמטריד אותנו זה קטגוריות 4 ו5, וגם פרסומים לגיטמיים שמגיעים למי שמטריד אותנו.

איך מתגוננים מפני חובבנים? יחסית די קל (להוריד Patchים). הם נתפסים גם די מהר, הבעיה שיש הרבה.

איך מתגוננים מפני ארגוני פשע? יותר קשה. אם מוצאים, מוצאים רק את דגי הרקק.

דוגמא:

מערכת הטלפוניה בארה"ב. יש $3 \cdot 10^8$ אנשים. כל אחד עושה כ30 שיחות ביום – כלומר, יש בערך 10^{10} שיחות ביום. נעריך שהשוק מגלגל כ 10^{12}\$ בשנה. נניח שהפורץ לוקח 0.1% מהעסק – מדובר ב 10^9\$. למה דווקא 0.1%? כי הם יכולים להרשות לעצמם הפסד כזה.

24.10.2007
הנדסת אבטחה

מסמך תכנון:
מכיל בד"כ:

- מדיניות - מה מותר ומה אסור?
  - פתוחה / סגורה
  - תשלום / חינם
  - גישה למידע של אחרים?
  - מידע חסוי / מידע גלוי
  -
- איומים אפשריים
  - האזנה, פריצה, גניבה, השחתה, אי נגישות
  - לדוגמא – אם המידע פתוח, אז האזנה לא כל כך מהווה איום בד"כ
- מטרות - ההפך מהאיומים בד"כ
  - סודיות
  - אמיתות מידע
  - זיהוי משתמשים
  - נגישות המערכת
- ישויות – מי הם הגופים הפעילים במערכת?
  - לדוגמא – בowl –,
  - גופים פעילים - מרצה ,סטודנט, אורח, עובד מנהלה, מנהל מערכת
- עצמים
  - גופים סבילים – לדוגמא קובץ
- הרשאות – פעולות מותרות
  - סטודנט יכול לקרוא קובץ
  - מרצה יכול לכתוב קובץ
  - ...
- תקלות – תמיד יהיו. איומים בלתי מכוסים
  -
- התאוששות
  - טיפול בתקלות שלא כיסינו. דרכים לחזור לפעילות.
  - לדוגמא – אם הבניין התמוטט, הביטוח ישלם.הפתרון הוא 'מחוץ למערכת'.

מטרתו ליצור תמונה רחבה לגבי 'מה יש לנו בידים' – ממה אנו מתמודדים, ממה לא, מה יכולותינו, וכיו"ב.

מטרות

- סודיות מידע – אף אחד לא רואה קובץ בלי הרשאה.
- אימות מידע – הקובץ שאני רואה בוודאות לא שונה בדרך.
- זיהוי ישות – לזהות מי הישות שאני עומד מולה.
- אימות ישות – לוודא שאתה מי שאתה **אומר** שאתה.
- אימות מקור מידע – לוודא מי באמת שלח את זה (לפעמים אפילו בלי לדעת מה המסר).
- אי התכחשות – שהשולח לא יכול לחזור בו מחתימתו .
- זמינות מידע
- זמינות מערכת
- פרטיות מידע – טביעת האצבע שלי לדוגמא לא סודית, אבל אני לא רוצה שכל אחד יוכל **להשתמש** בו. לדוגמא, איכון המקום שלי – לא סודי איפה אני נמצא, אבל אני לא רוצה שזה גם יפורסם באינטרנט.

- הכלת מידע – לדוגמא – רמת סיווג, שכל אחד יוכל לראות רק דברים בסיווג שלו או פחות. זה עובד גם על רשתות. מנגנון לשלוט על התפשטות המידע – שקובץ סודי ביותר לא יגיע לרשת לא בסיווג מספיק.
- אנונימיות – כמו לבוא לסופר עם כסף מזומן ומסכה על הפנים. לדוגמא, אנונימיות במשובי הוראה.

ישויות , עצמים והרשאות

	מערכת C	קובץ B	מחשב A	**עצמים/ישויות**
	הצצה	כתיבה / קריאה	V	מרצה
				סטודנט
	V	V	V	אדמין
	X	הצצה קריאה	X	אורח

תקלות והתאוששויות (מקת"גים – מקרים ותגובות)

תקלה	**התאוששות**
אבדן מידע	גיבוי
חוסר נגישות	שרותים אלטרנטיביים
נזק כללי	ביטוח

שיטות כלליות לאבטחה

מניעה: מניעת הפגיעה. הדבר הרצוי ביותר. לדוגמא, מנעול מונע כניסה של מי שאינו מורשה. Anti virus, הצפנה, השראות, שומר.

זיהוי: לא מונע פגיעה, אך מזהה את הפוגע. לדוגמא - מצלמה, אזעקה, ניטור, איתוראן, log files. מרתיה ולפעמים מקל על התאוששות / מניעת הפרצה בעתיד.

התאוששות: גיבוי, יתירות, ביטוח.

פתרון טוב יכיל מנגנונים מ3 הסוגים.

שכבות אבטחה:

1. מודעות משתמש
2. הגנה פיזית
3. הרשאות גישה
4. קריפטוגרפיה
5. ניטור
6. גיבוי
7. יתירות
8. הטעיה – לדוגמא honey pots – מלכודות שאמורות לתפוס פורצים.
9. התקפה -
10. הגנה משפטית

למה זה שכבות? לדוגמא - אין ערך לקריפטגורפיה, אם אין הרשאות גישה, הגנה ומודעות. וכך האלה. אפשר להסתכל על מה שאנו שומרים כנכס, המוגן בשכבות 'כמו בצל'. עדיין, הן לא מושלמות – תמיד תהיה דרך פנימה, אבל השכבות מקשות על הפורץ.

מודעות משתמש – בלעדיה, כלום לא יעזור. אם המשתמשים יתנו את הסיסמאות למי שלא מורשה, כל שאר המנגנונים יכשלו.

הגנה פיזית- אם המפתח הקריפטוגרפי נגיש פיזית לכל אורח, אין לו ערך.

עקרונות אבטחה

- least privilege – בצע את הפעולה עם הישות בעלת ההשראה הנמוכה ביותר שמסוגלת לבצע אותה. למה ? כמה שיש יותר הרשאות, יותר נזק יכול להגרם.

- trusted components – בכל מערכת יש רכיבים שחייבים לסמוך עליהם. יש להיות משוכנע שהם ראויים לכך. לדוגמא – הAdmin של רשת המחשבים. לדוגמא – מערכת ההפעלה שלי. לא תמיד זה יהיה מה שנראה לי כדבר הכי מסוכן – לדוגמא, היו מקלדות ששלחו את הסיסמאות שהקלידו עליהן.
- simple design – כדאי ללכת למערכות קטנות ומודולריות. קל לבנות , לתחזק, לאפיין ולחקור אותן.
- paranoia - תמיד ירדפו אחרינו. אם יש משהו שווה בארגון ,יהיה מי שינסה להשיג אותו. אנשי אבטחת המידע חייבים להיות בפרנויה – דברים משתנים, נפתחו חורי אבטחה, התגלו פרצות חדשות, ...
- no perfection – לעולם לא נגיע לשלמות.

<u>הגנות פיזיות:</u>

- חוטים – אפשר לצטט לחוט על ידי קירבה אליו וחישת הקרינה שלו. כדאי להגן עליו פיזית (יש כבלים מסוככים), או להצפין את המידע.
- רכיבים – אפשר לפרוץ ולהוציא מהם את המידע. לדוגמא – מעגל מודפס. אם המעגל מיישם אלגוריתם סודי, אפשר למצוא אותו לבסוף. יש רכיבים יותר בטוחים. יש רכיבים עם הגנה פיזית – שברגע שהם נפתחים המידע נאבד (מנגנון השמדה עצמית).
- מסכים – אפשר מהקרינה האלקטרומגנטית של המסך לזהות מה היה על המסך, ממרחק של כמה עשרות מטרים. לא רלוונטי בד"כ לLCD.
- דיסקים – בסופו של דבר, מכילים את האינפורמציה.

<u>ניהול סיכונים</u>

- הערכת נכסים
- זיהוי איומים / בעיות
- הערכת סיכויים לנזקים
- הערכת תוחלת נזקים
- סקירת מנגנוני אבטחה
- הערכה של הורדת נזק עם מגנונים מסוימים
- החלטה / בחירת מנגנונים.

29.10.2007

קריפטוגרפיה

קריפטו – סתר

גרפיה – כתב

מעניין לשים שבהרבה מובנים, הקריפטוגרפיה הקלאסית דומה להיום.

שימושים

1. סודיות מידע
2. אימות מידע
3. אימות מקור
4. אי התכחשות
5. תיאום מפתחות
6. הקמת שיחה מאובטחת
7. טראנסאקציות בטוחה

במה היא לא מתעסקת

1. אבדן מידע
2. השחתת מידע

סטגנוגרפיה – מתעסקת בהחבאת מידע – מתעסקת בהסתרת העובדה שהמידע הקיים. דוגמא מהעולם העתיק – כתבו את הטקסט על קרקפת של ראש של מישהו, ממתינים שהשיער יצמח, ואז שולחים אותו לדרכו. לא נעסוק הרבה בסטגנוגרפיה בקורס זה.

קריפטוגרפיה קלאסית

Substition ciphers

"ששך" ← "בבל"

א ← ת

ב ← ש

צופן קיסר – Caesar chiper: קח את הABC, ובחר מספר כלשהו (נניח 3). הפוך $A \to D$, $B \to E$, ...

כלומר - $E(M) = (M+3)_{\mod 26}$.

לדוגמא $E(\text{ATTACK FROM EAST}) = DWWD...$

כמובן ש $D(C) = (C-3)_{\mod 26}$

בעיות

1. **הסוד הוא האלגוריתם.** כבר בעולם העתיק הגיעו למסקנה שזה רע להשתמש באלגוריתם בתור סוד – כי מישהו יכול לספר אותו. בעולם המודרני מזמן לא חושבים על אופציה כזו – כי כולם מכירים את האלגוריתם. די ברור שהעובדה שהסוד הוא האלגוריתם גורמת את זה שחייו שאולים.

צופן קיסר מוכלל

$$E_k(M) = M + K \pmod{26}$$
$$D_k(C) = C - K \pmod{26}$$

כאן יש סוד – והוא K.

זה דומה לדלת – כל הדלתות מכילות את אותו אלגוריתם (צילינדר), אבל הסוד הוא המפתח. כיום תמיד האלגוריתם ידוע, והסוד הוא המפתח.

היתרון: את המכניזם אני בונה פעם אחת (את האלגוריתם). מפתחות אפשר להחליף מפעם לפעם – וזה יותר זול מלהחליף אלגוריתם.

התקפות:

1. מרחב חיפוש מאוד קטן – ב26 ניסיונות אני מוצא את המפתח. מנסים את כל האופציות עד שמוצאים משהו עם משמעות. נשים לב – אם הטקסט הוא באורך תו אחד, ההצפנה הזו היא בלתי שבירה. זה שביר כאשר יש טקסט ארוך – יש בד"כ רק הזזה אחת שתחזיר את זה למשהו בעל משמעות. במקרה הזה מגלים גם את המסר וגם את המפתח. להתקפה זו קוראים **חיפוש ממצה** (Exostive search). נרצה שחיפוש ממצה לא יהיה רלוונטי ליריב.

צופן כל התמורות

נבחר תמורה π_K. כאן מרחב החיפוש הוא $26!$. $E_K(M) = \pi_K(M)$. הסוד הוא התמורה - π_k.

חסרונות:

2. מבנה השפה – בכל שפה יש התפלגות מופע של תווים. התו 'e' נפוץ מאוד באנגלית לדוגמא, 't' מאוד נפוץ, 'z' לא נפוץ. אפשר למצוא התפלגות יותר מדויקת ככל שיודעים יותר על

מקור הטקסט. בהנחה שיודעים ששיטת ההצפנה היא פרמטוציה, אפשר לנסות על פי סטטיסטיקות לנחש את הפרמוטציות. כמובן שאפשר להשתמש בתדירויות של זוגות - ee, th, oo נמצאים יחסית הרבה.

זהו צופן **מונואלפבטי-** תמיד תו a יועתק לתו קבוע, בלי קשר למיקומו. מעניין לשים לב שגם הצפנות מודרניות הן מונואלפבתיות.

נקודה מעניינת נוספת היא **חוסר הפרקטיות של השיטה** – שכן, קשה לקודד פרמוטציה כמפתח ('הפרמוטציה החמישית' – כלומר, באורך $\log_2(26!)$). זה קשה במיוחד מאחר ויש פרמוטציות שלא נרצה להשתמש בהם, וצריך להסיר אותן מהמרחב. אפשר תאורטית לעשות טבלא מאינדקס לפרמוטציה, אבל היא תהיה טבלא ענקית.

Play fair

רושמים את הא"ב בטבלה של 5×5.

. הסוד הוא מילת קוד – לדוגמא – MONARCHY, שצריכה לקיים שאף אות לא חוזרת. לכן רושמים את המילה בטבלא , ולאחריה רושמים את כל הא"ב בו לא השתמשתי.

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

. לוקחים את הטקסט הלא מוצפן, לדוגמא – FUNNY. במידה ויש שתי אותיות זהות, אחת אחרי השניה, מוסיפים 'filler' – לדוגמא, Z, באמצע, כך שכרגע יש לנו FUNZNY. כעת, נצפין כל **זוג** אותיות – הזוג FU מתחלף עם הזוג שיוצר איתו מלבן – כלומר - EV. NZ מתחלפות עם RW. NY הוא 'מלבן מנוון' ומוחלף ב YG. הפתיחה היא זהה להצפנה.

יתרונות:

3. מכנית, מאוד פשוטה לישום.
4. יש הרבה מאוד מפתחות פשוטים.
5. מצפינים זוגות של אותיות – זה מטשטש תדירות מופעים.

נשים לב – זה לא מונואלפבטי, אלא פוליאלפבטי (פולי – זוגי). יש מי שיגיד שזה מונואלפבטי אם הא"ב שלך הוא $[A..Z]^2$ (זוגות). היתרון הוא שההיסטוגרמה של זוגות אותיות בא"ב היא הרבה יותר שטוחה. לפי ההגדרה הזו אנו עובדים בימינו.
נשים לב שזה פותח פתח לאלגוריתם דומה לשלשות, במקום לזוגות.

Hill Cipher

בוחרים m. בונים מטריצה של $m \times m$. לדוגמא, עבור $m = 3$, בונים:

$$(K) = \begin{pmatrix} k_{11} & k_{12} & k_{13} \\ k_{21} & k_{22} & k_{23} \\ k_{31} & k_{32} & k_{33} \end{pmatrix}$$

הצפנה : עבור מסר $M = p_1 p_2 \ldots$,

$$E_K(M) = (K)\begin{pmatrix} p_{3i+1} \\ p_{3i+2} \\ p_{3i+3} \end{pmatrix} = \begin{pmatrix} c_{3i+1} \\ c_{3i+2} \\ c_{3i+3} \end{pmatrix}$$

כאשר את החיבורים והכפל עושים $\bmod 26$ (גודל הא"ב).
כלומר – מצפינים m יות של אותיות כל פעם. זה לא מונואלפבטי בהגדרה הפשוטה, אבל כמו שאמרנו, אפשר להגדיר זאת כך.
צריך כמובן למצוא K שקיים לו K^{-1}.

$$D_K(C) = (K^{-1})(C)$$

השיטה טובה ויעילה – כאשר m גדול, אין אפשרות לשבור את השיטה באמצעות סטטיסטיקה.

Vigenere

ניצור טבלא עם כל ההזזות האפשרויות

	A	B	..	Z
A	A	B	..	Z
B	B	C	...	A
...				
Z				

ניקח מילת קוד $K = MONARCHY$. עבור טקסט $M = p_1 p_2 \ldots p_m$, הצפן את האות הראשונה עם M, את השניה O, ... וכך עד Y - ומשם חוזרים.

זה קוד פוליאלפבטי – כי לא כל האותיות מוצפנות באותה צורה. אפשר גם להתייחס לזה גם כמונואלפבטי של שמניות (במקרה של המפתח הזה).

זו שיטה מאוד נוחה לשימוש, גם בימינו.

מגבלות:

1. ככל שהמילה יותר ארוכה, איכות ההצפנה טובה יותר. מפתח אינסופי (או באורך המידע) היא בלתי שבירה.
2. איך שוברים? מנסים למצוא דברים שחוזרים (בטקסטים ארוכים). לפי זה מנסים להסיק את אורך מילת הקוד, ומשם ממשיכים.

נשים לב – יש כאן מנגנונים טובים. יש לבחון אותם לפי חוזקם וקושי המימוש.

31.10.2007

תרגיל ומצגות – באתר של דינה – http://www.notes-heaven.com

Transposition ciphers – צפני הזזה

לוקחים Buffer בן N תוים – הוא הטכסט הלא מוצפן. הטקסט המוצפן הוא הזזה שלו, לדוגמא, אם הטקסט המקורי הוא | T | H | E | D | O | G | A | N | D |, אז ההצפנה היא פרמוטציה שלו - | D | E | T | . | . | . | . | . | . |.

תכונות:

1. לא שובר סטטיסטיקה.
2. ככל שגודל הBUFFER גדול יותר, השיטה קשה יותר לשבירה (יש יותר פרמוטציות).
3. האלגוריתם ידוע, הסוד הוא הפרמוטציה שמשתמשיה בה, כלומר הוא המפתח היא התמורה, ומרחב המפתחות הוא $N!$ מפתחות אפשריים. אם $N=9$, אזי אפשר פשוט לנסות את הכל. אם $N=100$, אז אין סיכוי. נשים לב – בדרך כלל מתייחסים ל N כחלק מהאלגוריתם.

נשים לב שגם כאן צריך להעביר פרמוטציה – וזה לא קל – אינדקס, אבל לא מבין כל הפרמוטציות, כי יש כאלה שאנו לא רוצים להשתמש בהם.

Row column cipher

לוקחים טבלא, לדוגמא בגודל 5×5 -

A	T	T	A	C
K	F	R	O	M
E	A	S	T	A
T	D	A	W	N
A	N	D	T	H

.

בוחרים פרמוטציה, ומסדרים את העמודות ע"פ הפרמוטציה בשורות. המפתח הוא **סדר** קריאת העמודות (הפרמוטציה שלהן). חוזק ההצפנה נקבע על פי מספר העמודות. אפשר להצפין שוב עם מפתח אחר (את השורות), וזה יסבך את השבירה. נשים לב שאם בטבלא יש 25 תאים 5×5 אז כשאנחנו מצפינים שוב ושוב אנחנו כל הזמן עוברים בין פרמוטציות על 25 איברים – יש $25!$ כאלה, אז אפשר לעשות חיפוש ממצה ולשבור את זה. אבל אם קודם מנסים $5!$ תמורות ולכל אחת מהן $5!$ תמורות אז יש לנו $(5!)^2$ כאלה. אפשר לעשות זאת k פעמים, כל עוד $(5!)^k < 25!$ - מעבר לזה, לא נרוויח כלום.

Rotor Machines

מדובר במכונה עם 3 דיסקיות, (ציורים אצל דינה), רפלקטור, ועוד כל מיני דברים מוזרים.
כל דיסקית מממשת פרמוטציה. לכל אחת מהדיסקיות יש ציר סיבוב. כל פוזיציה של דיסקית מייצגת פרמוטציה – לכן כל דיסקית מייצגת 26 פרמוטציות.

מאחר ויש 3 דיסקיות, יכולות להיות עד $(26)^3$ פרמוטציות.

לאחר כל לחיצה, הגלגל הסתובב, וכך בעצם התחלפה הפרמוטציה.

מדובר בקוד פוליאלפבטי – שכן 'aa' לא יוצא את אותה אות כפולה (כי הפרמוטציה מתחלפת). המכונה היא קוד של Substition, אבל יש בתוכה גם אלמנטים של Transposition.

המפתח הוא המצב ההתחלתי של הדיסקיות. נשים לב שזה אומר שהמפתח הוא בגודל $(26)^3$ - לא כל כך הרבה. זה לא נחשב מספיק לצרכים צבאיים גם כשהשתמשו בו, במלחמת העולם השניה, ולכן הם אפשרו כל מיני קומבינות – החלפת גלילים, השתמשו מכונה עם 5 גלילים, וכיו"ב, ובסופו של דבר הגיעו ל $(10)^{20}$ אפשרויות.

איך שברו את זה? אין סיכוי לנסות את הכל. אם מנסים את כל האפשרויות בקצב של אפשרות לשניה, זה בערך 10^{13} שנים. הם עבדו באופן מקבילי – 60 מחשבים, שעובדים על בחירות הדיסקיות הרלוונטיות. זה מחלק ב60, לא משמעותי. בנוסף, הם ידעו שאם $a \to z$ אז $z \to a$, וזה עזר לשבור. בנוסף, הגרמנים בכל יום העבירו מפתח חדש כל יום בקו המאובטח. בנוסף, הם חזרו על המפתח הזה פעמיים. הדבר הזה מאוד עזר לבנות הברית – כי זה איפשר לפסול המון אופציות. בתחילת שנות ה40 הבריטים ידעו לשבור את השיטה המדוברת. הבעיה היתה , כמו תמיד, איך להשתמש באינפורמציה בלי לחשוף את זה שמפענחים את השדרים.

נגדיר E כמשפחה של פרמוטציות (לדוגמא – אוסף דיסקיות). k בוחר פרמוטציה אחת מתוכן (סידור הדיסקיות במקרה הזה).

נשים לב – הגרמנים היו בטוחים שאי אפשר לשבור את זה בשנים הבאות. נשאלת השאלה – האם יש שיטה שאי אפשר לשבור?

perfect crtypto

מדובר בשיטות שאי אפשר לשבור.

One Time Pad: בהינתן טקסט בינארי M באורך N ביטים, ומפתח K ביטים באורך N ביטים, נגדיר את תוצאת ההצפנה C כך - $C = M \oplus K$ ($\oplus = xor$).

כלומר - $E_K(M) = M \oplus K$.

פיענוח – $D_K(C) = C \oplus K$.

למה זה עובד? $A \oplus K \oplus K = A$.

הגבלות:

1. מפתח K באורך ההודעה M.
2. מפתח K אקראי בכל ביט שלו.
3. המפתח חד פעמי.
4. המפתח סודי.

תחת ההנחות הללו – לא ניתן לשבור את השיטה לעולם. למה? נדגים עבור $N = 1$. נניח שאני מעביר על הקו את הביט '1'. ההסתברות שההודעה המקורית היא 1 היא 50, וכך גם בנוגע ל0.

ההגדרה של קריפטגורפיה מושלמת - $\Pr(M) = \Pr(M \mid C)$ (ההסתברות של M היא שווה להסתברות של M כשרואים את C). כלומר – האזנה למידע לא נותנת לנו שום ידע (מעבר למה שידענו קודם, בנוגע להסתברות של מה שיעבור על הקו) על ההודעה. כלומר – ניסיון כל המפתחות יתן לנו את כל הטקסטים האפשריים.

אימות חד פעמי – One Time Mark

אימות – לקיחת הודעה M, העברתה דרך קופסה בשם 'Auth' וקבלת הודעה M עם $Y = f(M,K)$.

בצד השני – מעבירים לקופסת מאמת (Verifier)שמכירה את K את M,Y, והיא מוציאה את M, והודעה האם ההודעה 'תקינה' ולא שונתה על ידי אף אחד שאין לו את המפתח.

האלגוריתם שנציע הוא כזה –

P ראשוני גדול (לא סודי בהכרח). $k = (a,b)$ מפתח סודי חד פעמי, מוכר ל A,B (שני הצדדים). מתקיים $0 \le a,b < P$ ואקראיים. בנוסף, מתקיים - $0 < M < P$ (אחרת, נחלק את M. נשים לב שנצטרך מספר מפתחות אם מחלקים).

הפונקציה היא כדלקמן - $y = f(M,k) = (a \cdot M + b) \pmod p$

המוודא לוקח את M, מחשב את y באופן דומה, ובודק אם התוצאה זהה למה שהיא קיבל בתקשורת.

מה החשש באימות?

1. שמישהו ידע לשנות את M ואת y באופן שנחשוב שהיא אותנטית.

2. גילוי המפתח .

טענות:

1. לפני ראיית M, y, המתקיף יכול רק לנחש את k בהסתברות $\frac{1}{(p-1)^2}$.
2. אחרי שהמתקיף ראה זוג אחד M, y, הוא יכול לנחש את k בהסתברות $\frac{1}{(p-1)}$

הוכחה: עבור $b=1$ חשב a_1

$b=2$ חשב a_2.

... (זה אפשרי – חילוק בשדה).

$b=p-1$ חשב a_{p-1}.

<u>טענה:</u> אם מתקיף רואה (y_1, M_1) ו (y_2, M_2) (עם אותו k) אזי הוא יודע (a,b) בהסתברות 1 (בוודאות).

<u>הוכחה:</u> מדובר בשתי משוואות בשתי נעלמים בשדה.

מסקנה: חייבים להשתמש במפתח רק פעם אחת.

5.11.2007

היום נדבר קצת על קריפטוגרפיה מודרנית.
דיברנו על קריפטוגרפיה לא מותנית (מושלמת). לאחר מכן דיברנו על קריפטוגרפיה מותנית (חישובית).

קריפטוגרפיה חישובית מותנית בכושר החישוב של היריב (לדוגמא, צופן קיסר). זאת לעומת קריפטוגרפיה **לא מותנית** היא קריפטוגרפיה שלא מותנית ב:

- כושר חישוב של היריד
- כמות מחשבים
- כמות טקסט
- כמות זמן

לא משנה כמה מהנ"ל יהיה ליריב, הוא לא יוכל לשבור את ההצפנה.

קריפטוגרפיה מודרנית היא **מותנית** – היא **מניחה** שאין ליריב יכולת לשבור את השיטה.
למה לא משתמשים בד"כ בקריפטוגרפיה לא מותנית? היא לא פרקטית בד"כ. לדוגמא, One time pad דורש מפתח באורך המידע – מה שאין סיכוי להעביר.

ב5 שבועות הקרובים נדבר על

- קריפטוגרפיה סימטרית
  - הצפנה סימטרית
  - אימות סימטרי
- פונקציות HASH
- קריפטוגרפיה א סימטרית

<u>הצפנה סימטרית</u>

מונחים בהצפנה:

1. צדדים – A(lice),B(ob)
2. הודעה לא מוצפנת (M)
3. מפתח יחיד – k
4. קופסת הצפנה – ENC
5. הודעה מוצפנת - C
6. קופסת הפיענול – DEC

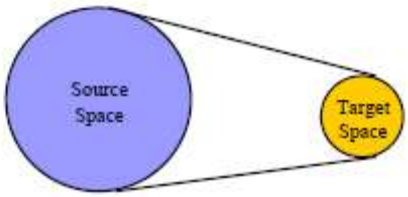
בד"כ בהצפנה סימטרית קופסת ההצפנה מאוד מאוד דומה לקופסת הפיענוח.
מונחים באימות:

1. צדדים – A(lice),B(ob)
2. הודעה לא מוצפנת (M)
3. מפתח יחיד – k
4. קופסת Auth
5. $y = f(M,k)$
6. קופסת אימות – Ver – שאומרת אם ההודעה אמיתית.

המודל הסימטרי מניח:

1. **תיאום מפתח.**

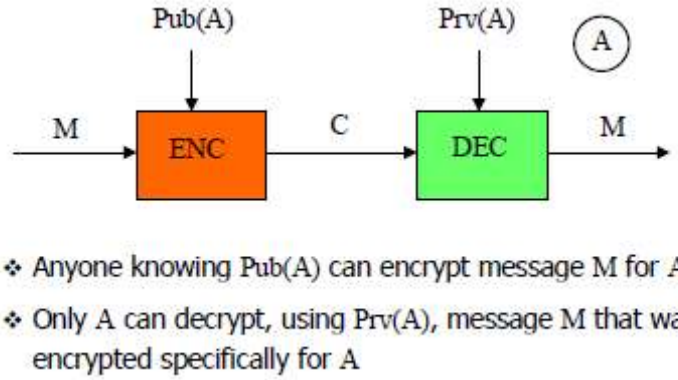
<u>פונקציות HASH</u>



ננסה למצוא **מאפיין קטן ואיכותי** לאובייקט דיגיטלי. לדוגמא – טביעת אצבע היא מאפיין קטן ואיכותי של אדם.

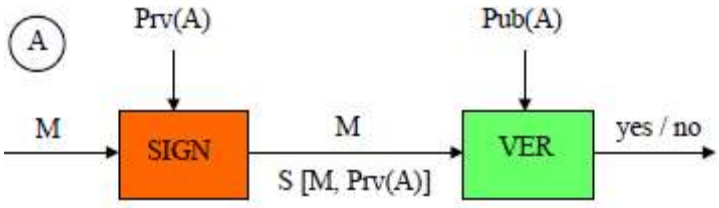
<u>קריפטוגרפיה א-סימטרית</u>

<u>הצפנה</u>



המשמעות היא שכל מי שרוצה שיצפינו **עבורו** דואג שיהיה מפתח ציבורי ופרטי. את המפתח הציבורי אני אפרסם (לכל העולם). להצפין עבורי אפשר עם המפתח הציבורי, אבל רק אני יכול לפתוח עם המפתח הפרטי.

<u>חתימה</u>



נשים לב – באימות סימטרי אי אפשר להשיג **אי התכחשות**, באסימטרי **אפשר**. **למה? כי במודל האסימטרי, רק לי יש את המפתח הפרטי.** באימות הסימטרי, גם המקבל (B) יכול לחתום.
(הערה להמשך – לפעמים אי אפשר לחתום על כל המידע, אז חותמים רק על Hash שלו).

<u>אלגוריתמים</u>

1. ידועים (כלליים)
2. פותחו על ידי חברה מסוימת
3. פותחו על ידי ממשלה / צבא
4. פותח על ידי יועץ
5. אני פיתחתי

עם איזה כדאי להשתמש? בד"כ לא כדאי באחד ש'אני פיתחתי'. עדיף להשתמש מהקבוצות 1 – 3. מה שחשוב זה שהאלגוריתם נבחן ע"י מאות חוקרים. למה חשוב ללכת על כזה? כי מדובר על קריפטוגרפיה לא מושלמת, והמדד היחיד לכמה הוא טוב הוא שלא הצליחו למצוא בו חולשה , למרות שניסו די הרבה.

<u>מה זה מפתח טוב?</u>

1. ארוך (כיום ± 100 ביט לסימטרי, ± 1000 לאסימטרי).

<u>דחיסה / הצפנה</u>

נשים לב – יש לדחוס לפני ההצפנה, ולא להפך. מידע מוצפן לא דחיס, מאחר והוא אמור להראות אקראי לאלגוריתם הדחיסה. בנוסף, קובץ דחוס מאבד חלק מהסטטיסטיקות שלו, ולכן מתקפות מסוימות על הצפנות לא יעבדו על מידע דחוס.

<u>סוגי התקפות</u>

<u>התקפות לא קריפטוגרפיות</u>

1. פריצה למחשב / לחדר / לדיסק
2. מציאת המפתח.
3. ניחוש המפתח.

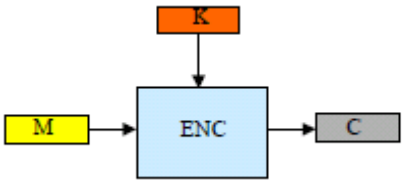
רוב המתקפות כיום הם לא קריפטוגרפיות.

<u>התקפות קריפטוגרפיות</u>

1. known ciphertext – המתקיף יודע רק את הטקסט המוצפן, ומנסה לגלות את הטקסט ו/או המפתח
2. known plaintext – המתקיף יודע את הטקסט, ומחפש את המפתח
   a. המתקיף יכול לצפות מה יעבור באיזורים מסוימים בהודעה.
3. chosen plaintext – המתקיף בוחר את הטקסט, ומחפש את המפתח.
   a. הצלחתי לגרום למישהו להצפין ולשלוח הודעה.

<u>Modern Ciphers</u>

<u>Block Ciphers</u>



מצפין בלוק לאחר בלוק. יש ריפוד בסוף הקובץ. דומה ל'substitution cipher'. המפתח **קבוע**. נשים לב – שני בלוקים זהים יוציאו שני מסרים מוצפנים זהים. אפשר להגיד שזה קוד החלפה מונואלפבטי (אבל הא"ב הוא 2 בחזקת גודל הבלוק).

נשים לב – אם M קטן, אפשר למפות את הפונקציה מאוד מאוד מהר (בהתקפה מסוג known plaintext). לכן עובדים בד"כ עם $64/128/256$ ביט.

מן הסתם C צריך להיות לפחות בגודל M. בד"כ $|C|=|M|$.

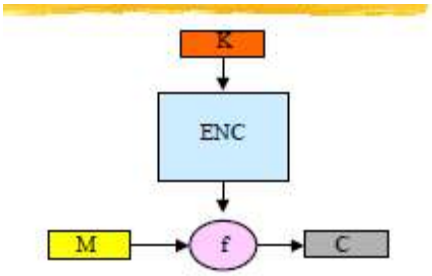
אם ההצפנה היא לדוגמא פרמוטציה, אז עבור 64 ביט, יש $2^{64}!$ תמורות כאלה. אם רוצים את כל התמורות אז $|k|=\log\left(2^{64}!\right)$. זה מספר לא סביר בעליל. לכן בד"כ בוחרים kים בסד"ג של $64-256$ ביטים. צריך לבחור את התמורות איתם כן נעבוד (מן הסתם – את הטובות).

נרצה כי:

1. שתי הודעות דומות יוצפנו שונה מאוד.
2. שני מפתחות דומים יצפינו שונה מאוד.

כלומר – נרצה פונקציות /תמורות אקראיות. באופן מעשי – עובדים עם תמורות פסאודו אקראיות. תמורות שנראות אקראיות במדדים. k צריך לבחור תמורה מתוך קבוצה E שהיא משפחה של תמורות פסאודו אקראיות.

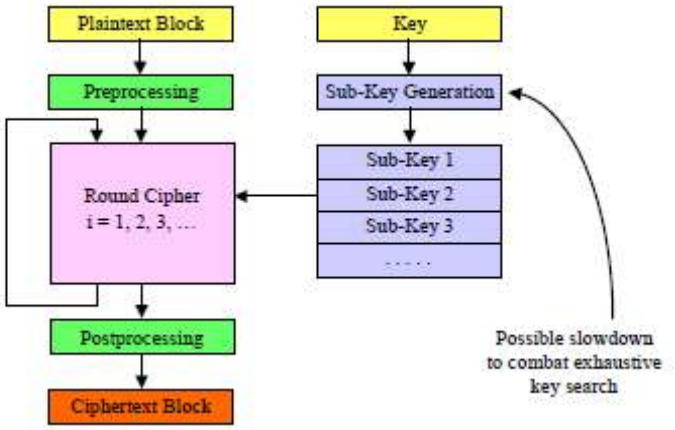
<u>Stream Ciphers</u>



מצפין ביט אחר ביט. בד"כ f תהיה xor, ההבדל יהיה שמנסים ליצור סימולציה של one time pad,
פשוט לא one time

7.11.07

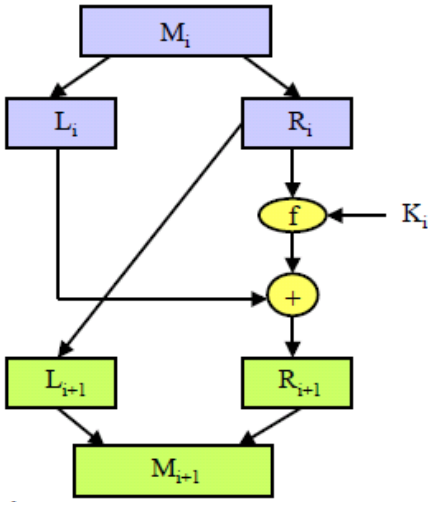
מבנה צופן בלוק



1. Pre Processing – בד"כ תפקידו ליעל את ההצפנה. לא תמיד קיים.
2. Post Processing - גם לא תמיד קיים.
3. Round Cipher – מבצע את **<u>תמורת הערבול</u>**. נשים לב – מפעילים על אותו בלוק את התמורות. אנחנו מגדילים את מרחב התמורות שלנו על ידי האיטרציות. כל Sub key ממש תמורה.

סכמת Feistel

$i = 1 \ldots r$



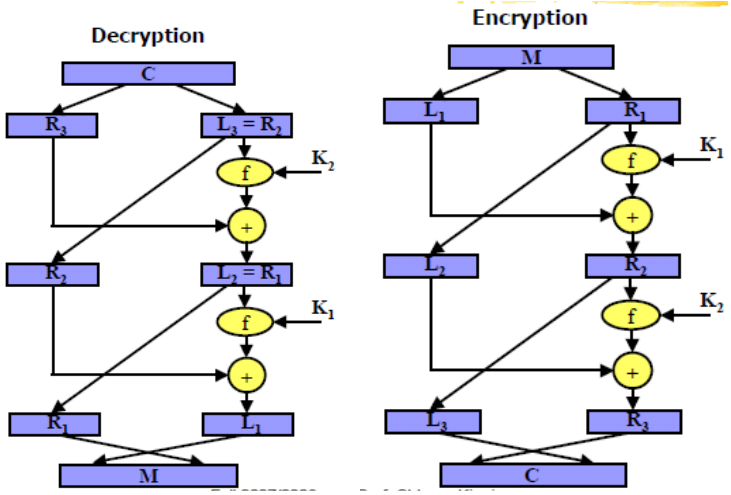
<u>(יש טעות בשקף – בסוף התהליך הופכים את הסדר של L_{i+1}, R_{i+1})</u>

1. חלק את החלק לחלק שמאלי וימני.
2. את החלק הימני העתק כמו שהוא לפלט של הסיבוב הנוכחי.
3. את החלק השמאלי הפוך להיות החלק הימני של התוצאה, לאחר XOR עם המפתח.

מה חסר בשביל אלגוריתם מלא?

1. r
2. k_i
3. $|M|$
4. f

דוגמא: $r=2$.



(הסבר למה הפתיחה פותחת את ההצפנה – אצל דינה).

איזה פונקציה תהיה טובה לתפקד כ f? אם נשים את פונקציית האפס, ההצפנה לא תצפין, אלא תחזיר את הזהות. **<u>אבל כל פונקציה תאפשר הצפנה ופענוח. אין צורך שיהיה קיים f^{-1}, או כל דרישה אחרת, כדי שאפשר יהיה לפענח. יש דרישות אם רוצים שההצפנה תהיה חזקה.</u>**

תוצאת הפונקציה f היא מתפקדת כInput לXOR – ואנחנו נרצה משהו שדומה לאקראי שם – לכן נרצה שהפלט שלה יהיה פסאודו רנדומי. הסודיות שלו באה מהמפתח.

<u>האלגוריתם DES</u>

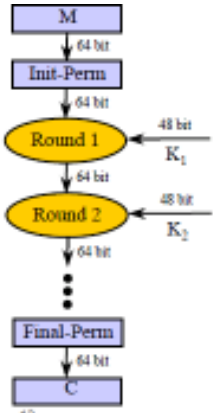
- DES = Data Encryption Standard
- הכוונה בהתחלת שנות ה70 שיהיה הסטנדרט בנושא ההצפנה.
- IBM הציעה את הפתרון שלה (LUCIFER) להיות הסטנדרט.
- התבצעה בחינה של האלגוריתם במשך 3 – 4 שנים.
- הNSA הכניס שינויים.
- ב75 פורסם, IBM ויתרה על פטנט שהיה לה על הסכמה המדוברת, כדי שזה יוכל להיות הסטנדרט.

הדרישה היתה שהאלגוריתם יהיה יעיל בחומרה, איטי בתוכנה (?!). הסיבה לכך היא שהצפנה נחשבה ככלי נשק, והמטרה היתה להגביל את השימוש בה בשוק האזרחי. זה דומה לכך שאסור לאדם פרטי לקנות נשק אוטומטי בארה"ב. אתה יכול להרוג, אבל לאט. באותו אופן – אתה יכול להצפין, אבל לאט.

<u>מבנה DES</u>

DES מממש את סכמת FEISTEL.

$|M|=64$. $r=16$. $|k|=64$ (מתוכו רק 56 ביט בשימוש – כל ביט שמיני אינו בשימוש – אפשר להשתמש בו כPARITY), מתוכו גוזרים 16 k_i ים. יש אלגוריתם דטרמניסטי שעושה את זה, לא נדבר עליו.



1. Init / Final permutation – מוגדרת לחלוטין בתקן. תפקידיה הן:
   a. איטיות בתוכנה.
   b. לערבב מעט את הקלט לפני ההצפנה (אבל זה לא נותן ביטחון בגלל שזה ידוע).

כעת נדבר על f. מדובר בפונקציה שמקבלת שני קלטים, אחד בן 32 ביט (הBLOCK) ואחד בן 48 ביט (k_i) בעלת 4 שלבים:

a. מתבצע expand לחלק מהביטים של הBLOCK – 16 מהביטים מועתקים פעם אחת, ו16 מהם מועתקים פעמיים. כך נהפך קטע בן 32 ביט לקטע בן 48 ביט. מדובר בטרנספוזיציה – מדובר בשכפול ובהזזה. נשים לב שגם שלב זה איטי יותר בתוכנה.
b. מתבצע XOR בין ה48 ביטים מהחלק הקודם עם המפתח (שהוא בן 48 ביט).
c. 48 הביטים נכנסים ל8 'קופסאות', s_i, $i..8$, כל קופסא ממשת $s_i : 2^6 \to 2^4$. סה"כ פלט 8 ה s_i הוא 32 ביט. s_1 היא טבלא . הביט הראשון והאחרון מהווים את השורה, שאר הביטים מהווים את העמודה, ואז מסתכלים בטבלה, וזה מגדיר את הפלט:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

נשים לב שמדובר בSubstition function.

d. מתבצעת פרמוטציה (מוגדרת כמובן).

אחד מהשינויים שהNSA הכניס זה בפונקציות s_i. במשך קרוב ל20 שנה היה חשש שהוא הכניס את השינוי כדי שיוכל לשבור את זה. לאחר 20 שנה, מאמר שפרסם עדי שמיר הראה שהקופסאות שבחר הNSA הן בן הטובות ביותר האפשריות – כלומר, ידע על התקפות שהתפרסמו רק 20 שנה אח"כ. רבים טוענים שזו סיבה להאמין שכבר בשנות ה70 יכל הNSA לבצע התקפת brute force על 56 ביט, ולא על 64 ביט.

ניזכר שדיברנו על כך שאנו רוצים שהודעות דומות (או הודעות זהות שמוצפנות עם מפתחות דומים) יוציאו הבדל שונות מהותית. כאן ה sים, בשילוב ה'expand' וה'permute' יגרמו לזה שביט אחד שמשתנה ישנה מהותית את הפלט.

מחקרים סטטיסטיים שנעשו מראים ש16 סיבובים מגיעים ליחס עלות תועלת אופטימלי בנוגע להההערה הקודמת (הודעות דומות -> הצפנה שונה מהותית). נעיר שגם היום האלגוריתם נחשב מצוין – החיסרון היחיד שלו הוא גודל מפתח קטן יחסית.

נשים לב - $2^{56} =\sim 10^{17}$. בהנחה שמחשב יודע לעשות 10^9 פעולות בשניה, ויש לך 10^4 מחשבים, ויש לך 10^4 שניות – אז סיימתי.

12.11.2007

בפעם הקודמת דיברנו על זה שהבעיה העיקרית של DES זה אורך מפתח קצר מדי. יש שתי אופציות עיקריות:

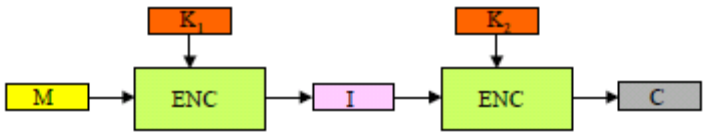
1. לחזק את DES
2. להחליף את DES

למה לחזק את DES ולא להחליף? כי הרבה משתמשים בו, ולא רוצים לזרוק. למה להחליף? כדי לא להשאר כל הזמן עם דבר ישן.

איך מחזקים את DES ?

נזכור שהבעיה העיקרית היא גודל המפתח. אפשר אולי להוסיף עוד סיבובים, ולהחליף את הSBoxים בין סיבובים. אבל זו אפשרות פנימית – של שינוי פנים האלגוריתם. נשים לב שאנחנו נחפש שיטות לחזק את DES בשיטות חיצוניות – בלי לגעת באלגוריתם עצמו – כלומר, להשתמש עדיין ב'ציפים' של DES שקיימים.

Double Des



הרעיון – להפעיל DES על DES.

מדובר תאורטית במפתח באורך $\left(2^{56}\right)^2 = 2^{116}$ - כמו מפתח באורך 112. אבל יש עם זה בעיות – והן נכונות לכמעט כל אלגוריתם שמופעל פעמיים.

התקפת Meet In The Middle (קיימת לא רק לDES, אבל נדבר על DES במקרה זה)

עולה בערך $O\left(2^{63}\right)$

הרעיון הוא כזה (נתבונן בשרטוט לעיל) – ננסה להצפין מכיוון קופסת ההצפנה השמאלית, ו'לפענח' דרך קופסת ההצפנה השמאלית – ולהיפגש באמצע.

מדובר בהתקפה של Known Plaintext (M,C ידועים):

נעבור על כל האופציות של K_1, ונסמן

$$E_{K_1^1}(M)=X_1$$
$$E_{K_1^2}(M)=X_2$$
...
$$E_{K_1^{2^{56}}}(M)=X_{2^{56}}$$

אותו דבר, ניקח את C,ונסה לעבור על כל המפתחות האפשריים K_2. נרשום:

$$Y_1=D_{K_2^1}(C)$$
$$Y_2=D_{K_2^2}(C)$$
...
$$Y_{2^{56}}=D_{K_2^{56}}(C)$$

כעת צריך לחפש i,j כך ש $X_i=Y_j$, ואז $\left(K_1^i,K_2^j\right)$ הוא מועמד. יתכנו כמובן כמה מועמדים (למצוא אותם עולה בערך 2^{60}).

<u>טענה:</u> יש בערך 2^{48} זוגות מועמדים.

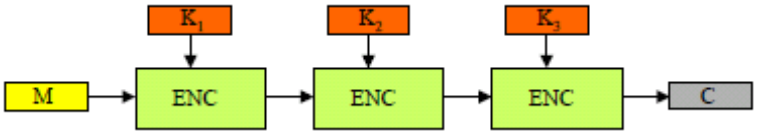
<u>הוכחה:</u> נניח שההתפלגות של הפלט של DES היא אחידה. מרחב הI (Intermediate) הוא 2^{56} (מתוך 2^{64}).

אותו דבר עושים על C – מקבלים 2^{56} מתוך 2^{64}. כלומר, בקירוב, יש 2^{48} זוגות (עבור כל Y הסיכוי שלו הוא $\frac{2^{56}}{2^{64}}$, ומאחר ויש 2^{56} כאלה, זה יוצא $2^{56}\cdot\frac{2^{56}}{2^{64}}=2^{48}$).

כעת, ננסה בלוקים נוספים של M,C. עבור הניסיון השני, נקבל בהסתברות טובה רק מפתח אחד (הוכחה –תרגיל לתלמיד החרוץ. עשוי להופיע במבחן המסכם.).

התקפה נוספת – יתכן ש $DES_{K_1}\left(DES_{K_2}(M)\right)=DES_{K_3}(M)$. אבל עבור DES זה לא מתקיים.

<u>Triple Des</u>

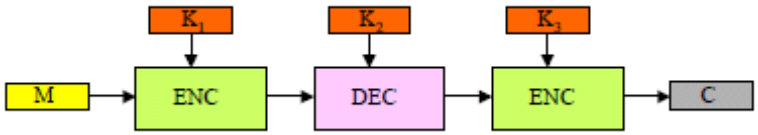


זה נקרא **EEE Mode** (נשים לב שיש 3 מפתחות). תאורטית – יש כאן מפתח של 168 ביט.

כמו Double Des – רק עם עוד אחד. נשים לב – הבטחון של זה כנגד Meet in the middle הוא בערך 2^{112}. כי אם נפגשים באמצע – חייבים בעצם לעבור 2 מצד אחד.

מה החיסרון של זה? צריך מפתח של 168 ביט בשביל חוזק של 112 ביט. אבל אפשר להשתמש בזה בתצורה של $K_3 = K_1$. זה חזק כמעט באותה מידה.

EDE MODE



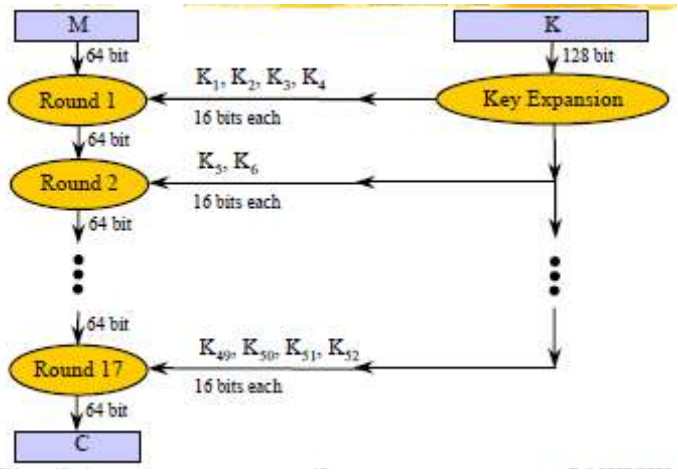
טעות בשרטוט – במקום לרשום K_3 צריך להיות כתוב K_1. אין K_3

נשים לב – אם $K_1 = K_2$ (כלומר, מפתח אחד), אזי הקופסא מתנהגת כמו DES רגיל. זה מאפשר באותו ציפ לעבוד כTriple DES וכDES רגיל. ואין באמת הבדל קריפטוגרפי בין DEC לENC.

נדבר על 2 אלגוריתמים נוספים:

IDEA – International Data Encryption Algorithm

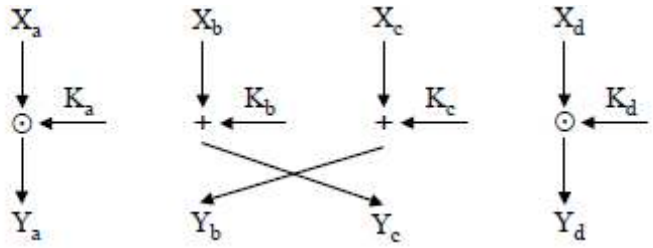
- משמש לדוגמא בPGP
- גודל בלוק – 64 ביט
- גודל מפתח – 128 ביט.
- 17 סיבובים (יש סיבובים אי זוגיים וזוגיים, דומה לפייסטל, אבל לא זהה).
- משתמש בפעולות הביטיות הבאות - XOR, $+_{\mod 2^{16}}$, $\cdot_{\mod 2^{16}+1}$
- תוכנן להיות מהיר מאוד בתוכנה.



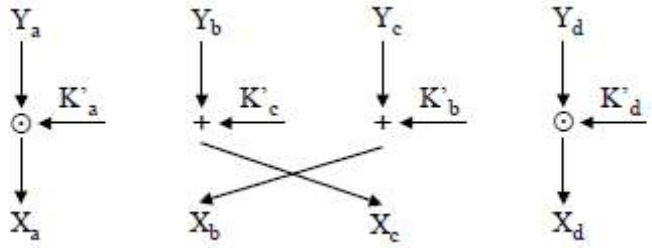
ההודעה מחולקת כ4 הודעות בנות 16 ביט – נחלק אותה ל x_a, x_b, x_c, x_d.

<u>סיבוב אי זוגי:</u>

המפתח הוא 128 ביט. נייצר ממנו 52 מפתחות קטנים בני 16 ביט. בסיבוב הראשון נשתמש במפתחות k_1, k_2, k_3, k_4 (נקרא להם k_a, k_b, k_c, k_d).

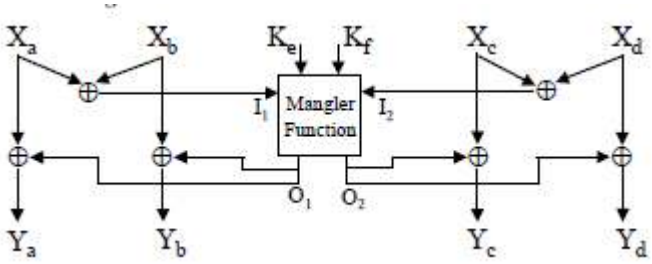


מה ההופכי לפעולה זו? לפעולת החיבור יש הופכי (חיסור). גם לפעולת הכפל יש הפכי. כלומר, כדי להפוך, כל מה שצריך לעשות זה לשים במקום k_a, k_b, k_c, k_d, את המספרים ההפכיים שלהם (חיבורית וכפלית בהתאם , לפי הצורך). לכן זה יראה כך:



<u>סיבובים זוגיים:</u>

<u>הצפנה:</u>



איך פותחים את זה? אותה קופסה מתנהגת גם כקופסת פתיחה. למה?

$$\left.\begin{aligned} y_a &= x_a \oplus o_1 \\ y_b &= x_b \oplus o_1 \end{aligned}\right\} \Rightarrow x_a \oplus x_b = y_a \oplus y_b$$

ואותו דבר גם בצד הימני.

<u>AES</u>

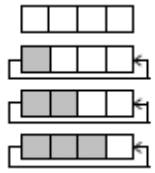
AES

- הרעיון היה ליצור סטנדרט חדש ומתקדם של DES. המוטיבציה באה מהאקדמיה.
- הצעות ובחינה- 97 – 2000
- האלגוריתם נבחר ב2001 (הצעה של שני בלגים)
- גודל בלוק – 128 ביט
- מפתח – בא בגדלים של 128 / 192 / 256 בתים
- מספר סיבובים – 10/12/14 בהתאמה לגדלי המפתחות לעיל.

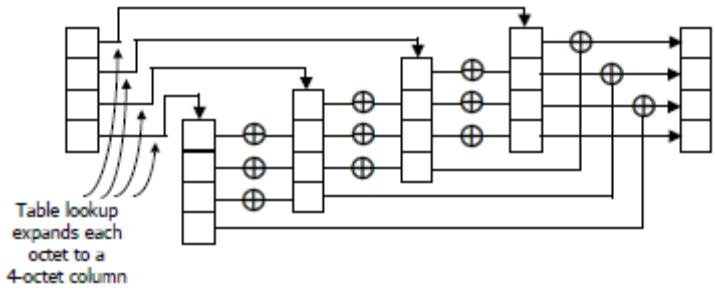
תיאור האלגוריתם – אין לי כוח להעתיק. יש אצל דינה / בשקפים / בוויקי. עברנו עליו די במהירות.

בקצרה:

1. XOR (ביטי) עם מפתח (128 ביט)
2. החלפה (S-Table) של כל Byte ע"פ טבלא.
3. Shift Row (בבתים)-



4. Mix Columns -



לצורך העניין, כל הפעולות הן פשוטות בכל דרך עיבוד מוכרת.

לכל הפעולות כאן יש הפכי , די פשוט בדרך כלל, וקל לממש אותן גם בצורה די קלה. האלגוריתם הזה מממש מתמטיקה די חזקה, שלא נפרט אותה כאן.

הצפנת טקסטים ארוכים

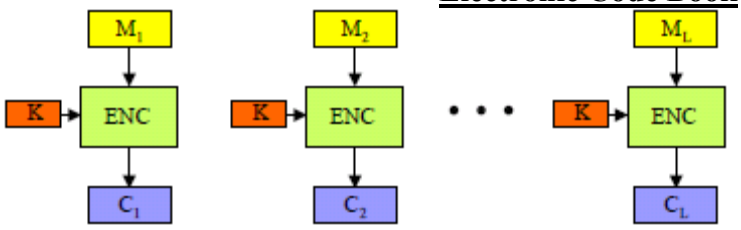
1. Block Ciphers
2. Stream Ciphers

Block Ciphers

השיטות:

1. ECB – Electronic Code Book
2. CTR – Counter Mode
3. CBC – Cipher block chaining

Electronic Code Book



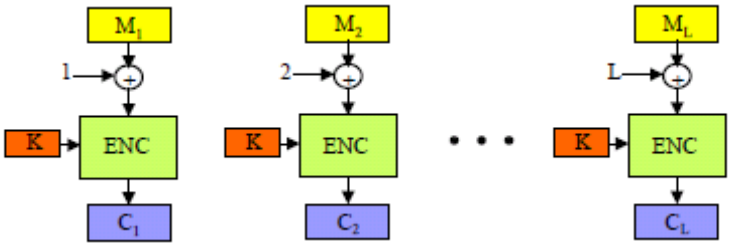
יתרונות:

1. פענוח/הצפנה מקבילית אפשרית
2. פענוח/הצפנה לבלוק כלשהו אפשרי.

חסרונות:

1. בלוקים זהים יוצפנו בצורה זהה. זה יכול לתת מידע למתקיף.
2. אין הגנה על סדר + תוכן הבלוקים (Man in the middle יכול להעיף בלוק ולשים במקומו בלוק אחר - אימות).

Counter Mode

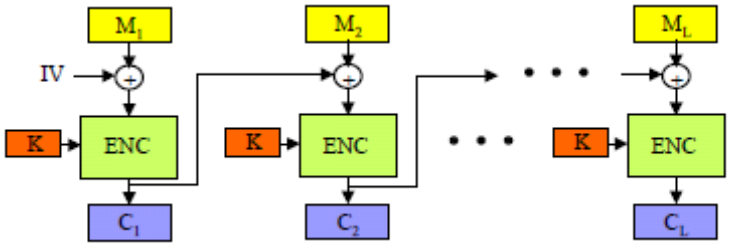


יתרונות

1. בלוקים זהים לא יוצפנו זהה.
2. הגנה מסוימת על סדר / תוכן
3. הצפנה / פענוח מקבילים אפשריים, אך נדרש סינכרון

(נשים לב – בד"כ לא עושים XOR עם המספר '1' – אלא בד"כ Hash(1) – וכך בעצם עושים Xor שמשפיע על הרבה ביטים,ולא רק הראשונים)

CBC – Cipher Block Chaining



יתרונות:

- פענוח יכול להיות מקבילי
- בלוקים זהים יוצפנו שונה
- **<u>סינכרון עצמי</u>** – אם בלוק אחד נפל, אי אפשר לפענח את האחד שאחריו. אבל מספיק שיגיע בלוק אחד תקין, כדי שאפשר יהיה לפענח את כל מה שבא אחריו.

<u>חסרונות:</u>

- איבוד מקביליות **בהצפנה** - חייבת להיות סדרתית

<u>מהו IV ? (הXOR של הבלוק הראשון)</u> – חכם לבחור אותו בצורה אקראית (IV = Initial Vector).
נשים לב, שחייבים לשלוח אותו בתקשורת, אז הוא לא עוזר באבטחה.
מה היתרון של IV אקראי? אם יומיים (שתי הודעות שונות, שלא קשורות אחת לשניה) שולחים את אותו קובץ, הוא יוצפן בצורה שונה לחלוטין. IV מאפשר לאקרא את ההודעות.

נשים לב – אם ביט אחד ישתנה באמצע, אז C_L (הבלוק המוצפן האחרון) ישתנה. לכן אפשר להשתמש בו כאימות.

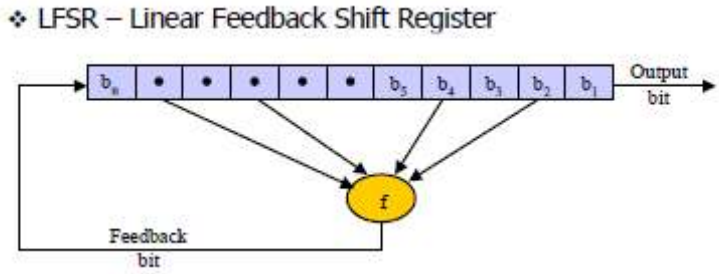
אבל איך מעבירים אותו? לשלוח את C_L פעמיים בסוף, כלומר לשלוח את $IV, C_1, .., C_L$ כתוכן, ואת C_L כאימות, לא יעבוד – אני יכול לדוגמא להתקיף זאת על ידי להעיף n בלוקים אחרונים, ולשים את האחרון כאימות.

מה עושים? שולחים כאימות את $C_L^{'}$ - שהוא הבלוק האחרון שמוצפן עם מפתח K'.

<u>Stream Ciphers</u>

ניזכר בOne time pad – פשוט עושים XOR ביט ביט. אנחנו ננסה ליצור קירוב לOne Time Pad.
ניצור ביטים עם מחזוריות מאוד מאוד גדולה, פסאודו רנדומית.
מייצר הביטים ישתמש במפתח כדי לייצר את הביטים.

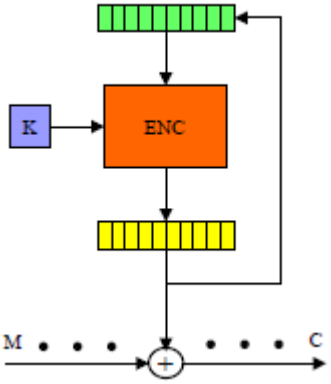
<u>LFSR</u>



מדובר בRegister, שלוקחת חלק מהביטים בRegister, ומוציאה ביט לתוך הRegister.
Linear – כי f לינארית.
Feedback – התוצאה נכנסת לRegister
Shift – מתבצע shift לRegister

בד"כ $f = xor$.
נשים לב - 0 חוזר ל0, לא משנה מה (אם $f = xor$). ברור שלא נרצה את 0 כמפתח (ערך ראשוני של הRegister). בנוסף, לא נרצה להגיע לעולם ל0. נרצה שהRegister יעבור על כל ה$2^n - 1$ אופציות שאינן 0.
התאוריה אומרת שאם מסתכלים על הRegister כסדרת מקדמים של פולינומים, והפולינום הוא פרימיטי מודול 2^n, נגיע לכל המצבים במעגל.
K = המצב ההתחלתי, שונה מ0.

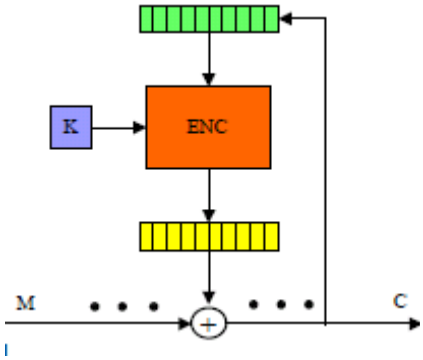
OFB - Output FeedBack



Enc יכולה להיות DES לדוגמא. הרעיון הוא להשתמש בDES כדי להצפין כל פעם בלוק, ואיתו לעשות XOR עם הOUTPUT.
מתחילים בBUFFER מסוים, מצפינים אותו עם מפתח. קיבלנו 64 ביט, ואיתם עושים XOR כל פעם ביט ביט. כרגע בעזרת FEEDBACK משנים את הInput להצפנה.

- מדובר באוטומט סופי.
  - אפשר לנתח את מספר המצבים שלו (ויתכן וזה יהיה במבחן).
- I = כמו IV קודם .
- אפשר להכין הרבה ביטים מראש (שיתפקדו כערך של הXOR).
- חיסרון – דרוש סינכרון חיצוני בין השולח והמקבל. אם נאבדו חלק מהביטים, אי אפשר לפתוח את המידע.

CFB = Cipher Feedback



(ירוק = I, צהוב = O). כאן הFeedback בא מהמידע המוצפן עצמו.
ניתוח:

- לא אוטומט סופי (לא דטרמניסטי) – כי הוא תלוי בM.
- אי אפשר להכין ביטים מראש (כי הוא תלוי בM)
- סינכרון עצמי.
- נשים לב – אין סיבה שI ההתחלתי יהיה סודי, כי בכל מקרה הוא יהפך לC בשלב כלשהו, שכן C נהפך להיות תוצאת ההצפנה.

כמה משוב נרצה? את כל הבלוק המוצפן פעם בגודל הבלוק (64 ביט לדוגמא), או על כל ביט. נשים לב – משוב קטן יותר אולי יותר בטוח, אבל יותר בזבזני (כי צריך להפעיל את ENC הרבה יותר פעמים).

נשים לב – גודל המשוב משפיע על זמן החזרה לסינכרון אם נאבד סינכרון.

19.11.2007

נדבר בעיקר על אימות של:

- ישות (עם מי אני מדבר)
- מקור (מאיפה ההודעה הגיעה)
- תוכן
- סדר (של חבילות בהודעה לדוגמא).
- זמן
- אי התכחשות

איך עושים את זה?

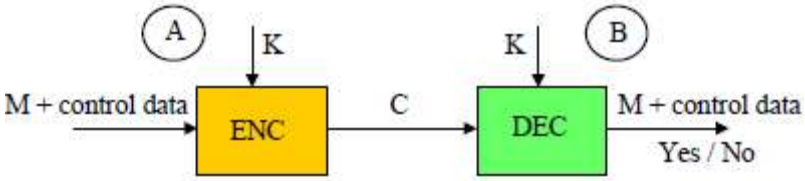
- אימות ע"י הצפנה
  - סימטרית
  - אסימטרית
- אימות על ידי MAC (Message Authentication code)
  - סימטרי (אין משמעות לMAC אסימטרי)
- אימות על ידי פונקציות HASH.

אימות ע"י הצפנה סימטרית

האם הצפנה בלבד נותנת אימות?

- לא של סדר
- בתלות באחוז ההודעות הנחשבות תקינות מתוך המרחב, קיים סיכוי שהודעה אקראית תפוענח.

מה עושים בשביל אימות מקור ותוכן? Control data.

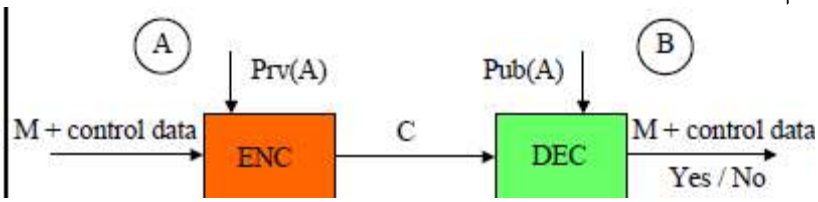


האימות מתבסס על כך שרק לA יש את המפתח K.

הצפנה אסימטרית

בהצפנה א סימטרית 'רגילה' אין שום אימות – לא מקור ולא תוכן, שכן כל אחד יכול להצפין עם המפתח הפרטי של המקבל B (השולח לא משתמש במפתח הפרטי שלו בהצפנה).

איך מאמתים עם הצפנה אסימטרי?



מצפינים את ההודעה שרוצים **לאמת** עם המפתח הפרטי של A – לתהליך זה קוראים חתימה(sign) .

המקבל יפתח זה עם המפתח הציבורי של השולח. זה נותן אימות מקור.

נשים לב שאנו מקבלים כך אימות של:

1. תוכן
2. מקור
3. **אי התכחשות** (רק A יכול לחתום על זה)

נשים לב שאנו לא מקבלים כאן הצפנה – כל אחד יכול לפענח את זה.

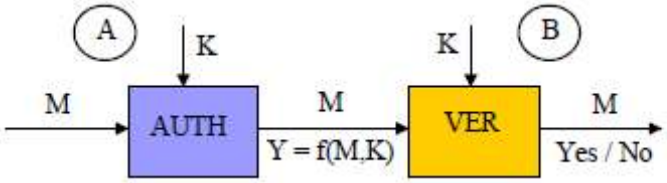
נחזור להצפנה סימטרית

מה הרעיון של הcontrol data שמוסיפים?

לגרום לכך שמרחב הMים (שכוללות את הControl data) האפשריים (בעלי המשמעות) דליל מאוד.
מה צריך לוודא? שפונקציית ההצפנה חזקה (אם אפשר לחלץ ממנה את המפתח, לא הרווחנו כלום).

צר שיהיה קל לזהות באופן אוטומטי הודעות תקינות ולא תקינות.

אימות על ידי MAC



הרעיון הוא כדלקמן – הAUTH מוציא את Y, שהוא הקוד המאמת. שולחים את M, y.
נשים לב שVER זהה לAUTH – VER מייצר את Y הצפוי , ובודק אם הוא שווה לY שהוא קיבל. זה ההבדל מול המודל הקודם – כאן שני הצדדים עושים אותו דבר, לעומת המודל הקודם, בו צד אחד הצפין והשני פענח. לכן אין כאן צורך בפונקציה הפיכה, להבדיל מהמודל הקודם.
ההבדל היחיד בין VER לAUTH זה שהוא משווה.

תכונות של MAC

חששות

- מתקיף רואה את (M, y) (כמובן שיותר מזוג אחד) ומחלץ את K.
- מתקיף יכול לייצר $(M, y) \neq (M', y')$ קביל.

ניזכר בOTM (One time mac). P ראשוני גדול, $K = (a, b)$, ו$f : y = (a \cdot M + b) \pmod p$. אנחנו נרצה לדבר על MTM – Many time mac.

תכונות רצויות

- נרצה פילוג Yים אחיד / אקראי.
- M דומים מועתקים לYים שונים משמעותית.
- כמובן שנרצה שY יהיה קטן משמעותית לM.

נשים לב שפונקציות הצפנה טובות ליצור MAC.

דוגמא (רעה):

$Y = E_K(M_1 \oplus M_2 \oplus M_3 .... \oplus M_n)$ (חלוקת ההודעה).

לא מתקיים התנאי השני – שינוי סדר של שני בלוקים יחזיר את אותו Y. גם חילוף מספר זוגי של ביטים יתן את אותו Y.

דוגמא (יותר טובה, אבל לא מושלמת):

$$C_L = CBC - E_K(M)$$
$$Y = (IV, C_L)$$

(כלומר – חלק את M לבלוקים, והצפן כל בלוק עם תוצאת הקודם. התוצאה היא הבלוק האחרון).
זו גם לא טכניקה מספיק טובה – היא אמנם נותנת פיזור אחיד ואקראי. הבעיה היא שהמתקיף יכול לזייף הודעה עם IV זהה או אחר וC_L. הסבר יותר מפורט בשקפים. לא בטוח שמה שהוא יכול לזייף אכן בעל משמעות, אבל זה עדיין לא מספק. בד"כ אפשר יהיה למצוא הודעה קצרה יותר עם אותו Y.

מסקנות:

- יש צורך להוסיף להודעה :

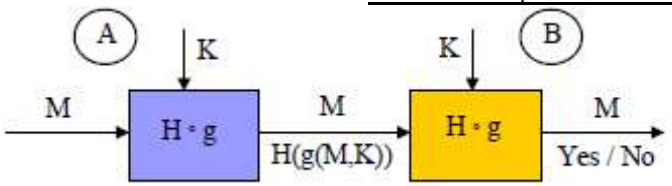
- זמן
- מספר סידורי
- אורך ההודעה

למעשה, ניקח את M, ונהפוך אותו ל $M \to (A, B, Time, Seq\#, length, rand, M)$, ואת זה נאמת. מה עושה איזה סוג זיוף כל דבר מיועד למנוע?

- A – מקור
- B – יעד
- Time – זמן
- Seq# - סדר
- Length – חיתוך
- rand - ?
- M – ההודעה עצמה

עם תוספת הגודל, CBC מתפקד כפוקנציית Y הנותנת MAC טוב.
נרצה כמובן שהMAC יהיה קצר ככל האפשר, ושנוכל להשתמש במפתח הרבה פעמים, ושהחישוב יהיה מהיר מאוד.

<u>אימות ע" פונקציות HASH</u>



נשים לב שזה מאוד דומה לMAC, אבל כאן לא משתמשים בפונקציית הצפנה, אלא בפונקציית HASH, ויש להן תכונות שונות.
גם כאן פונקציית הAUTH והVER זהות. ההבדל הגדול מול MAC זה שפונקציית HASH לא משתמשת במפתח – מחזירים פונקציה של הHASH והמפתח.

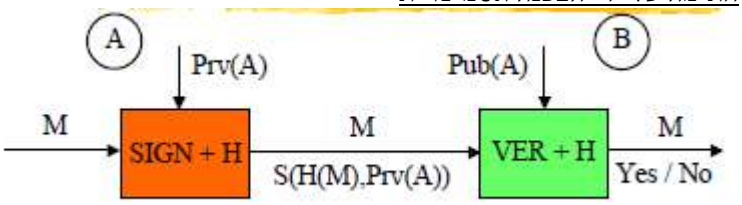
<u>במודל הזה משתמשים כיום בדרך כלל כh-MAC .</u>

סכמת הhMAC?
פרמטרים:

1. פונקציית h
2. מפתח k
3. הודעה M.

HMAC אומרת (בערך) לחשב $h(h(k||M)||k)$. [|| זה שרשור]. התכונה שהמפתח נמצא לפני וגם אחרי ההודעה מונע הרבה התקפות – אי אפשר לקצר או להאריך אותה.

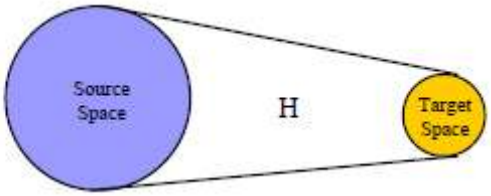
<u>חתימה על ידי הצפנה אסמיטרית</u>



מה ששולחים זה $Sign(h(M), PRV(A))$ (H היא פונקציית HASH). זו **<u>חתימה</u>** כמו **<u>שמשתמשים בה באמת בעולם האמיתי</u>**. אנחנו חותמים בד"כ על HASH(M), ולא על ההודעה עצמה – משיקולי זמן.
נשים לב שכאן (לעומת MAC) קודם כל מצמצמים את ההודעה, ורק אז המפתח נכנס לחגיגה – לעומת MAC, בו הוא משתתף בכל שלב.

21.11.2007

פונקציות HASH קריפטוגרפיות



המטרה היא לקבל מאפיין איכותי בגודל סופי (בד"כ באיזור ה100 ביט) של הודעה בגודל בלתי מוגבל.
נרצה:

1. מאפיין קצר
2. מאפיין איכותי
3. קשה ל'זיוף'

נשים לב – כמות גדולה (בפועל , אינסוף) של הודעות יתמפו לאותו ערך – אין ברירה אחרת.

תכונות :

אופרטיביות

1. תעבוד על כל קלט
2. מייצרת פלט באורך קבוע
3. קלה לחישוב

ביטחון:

1. One-Way – בהינתן y קשה למצוא x כך ש $h(x)=y$
2. weak collision resistence – **בהינתן** x_1, קשה למצוא $x_2 \neq x_1$ כך ש $h(x_1)=h(x_2)$
   (נשים לב שדרישה זו כוללת גם את הדרישה הקודמת)
3. strong collision resistence – קשה למצוא $x_2 \neq x_1$ כך ש $h(x_1)=h(x_2)$ (נשים לב שדרישה זו כוללת גם את הדרישה הקודמת, וזה עובד רקורסיבית). נשים לב שנסכים שיהיו בודדים כאלה, אבל לא יותר מזה.

שימושים לפונקציות HASH

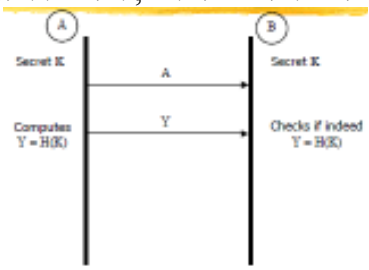
1. איפיון עצמים
2. פרוטוקולי אימות
3. פרוטוקולי להתחייבות
4. הצפנה
5. MAC
6. חתימה

איפיון עצמים

אם אני רוצה לוודא אם קובץ השתנה, אני יכול לשמור רק את הHASH שלו מאתמול, לחשב את הHASH שלו היום, ולראות אם זה אותו HASH.

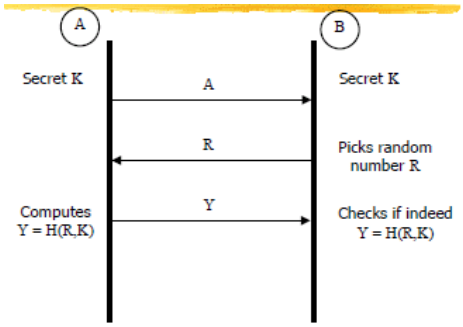
פרוטוקולי אימות

צד A וצד B מדברים, וצד A רוצה לוודא שהוא מדבר באמת עם צד B.



מעבירים פשוט HASH של המפתח הסודי המשותף.

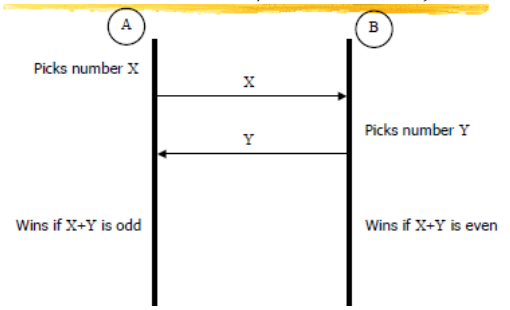
מה הבעיה של זה? שמאזין מכיר את $H(K)$, ויכול לשלוח אותו שוב. לכן, מה שעושים זה:



R כאן נותן חד פעמיות – Y ששולח A משתנה מפעם לפעם.

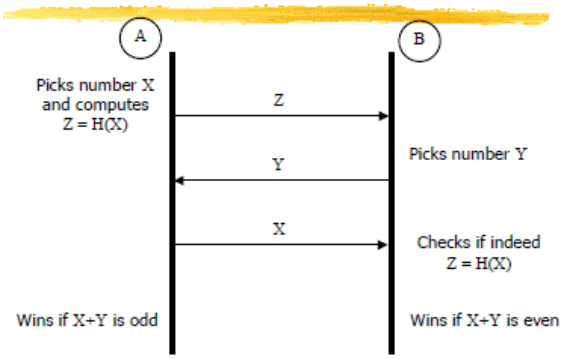
<u>פרוטוקול התחייבות</u>

נניח וA,B רוצים לשחק זוג / פרד. אופציה א'



הבעיה היא שאי אפשר לוודא בו זמניות במחשב. מי ששולח ראשון – חושף את הכוונה.

מה עושים?



כך B יכול לוודא שA לא רימה.

מה הבעיה? אם X,Y הם ממרחב מאוד קטן, אפשר לחשב את הHASH של כל המרחב, וכך לא הרווחנו כלום.

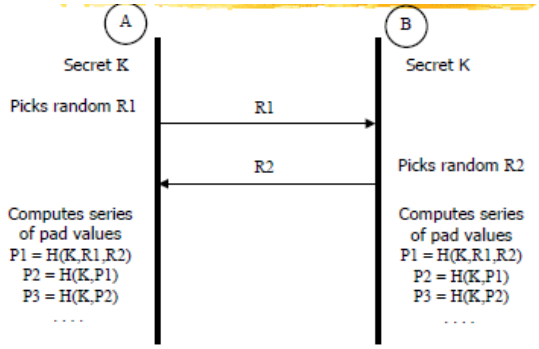
נשים לב שכדי שA לא יוכל לרמות, צריך פונקצית HASH שמקיימת Strong Coliision Resistance.

<u>הצפנה באמצעות פונקציות HASH</u>

למה לעשות הצפנה באמצעות HASH?

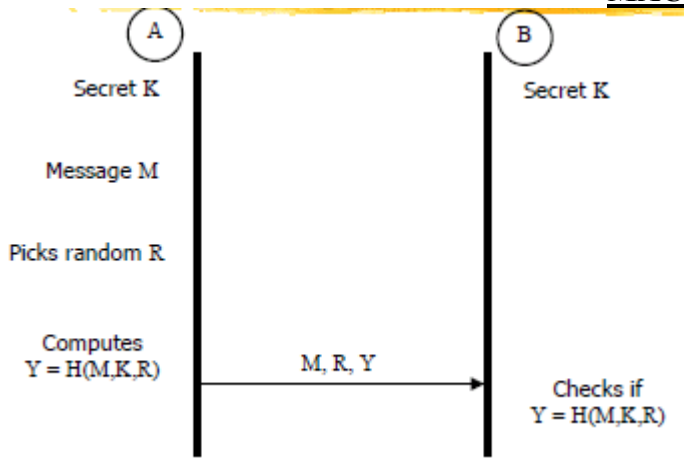
1. זה מגניב. למה לא?
2. הן לא מוגדרות ככלי בטחוני. כאשר היתה הגבלה על יצוא של הצפנה, אפשר היה להשתמש בזה.
3. הן מהירות מאוד בד"כ.

ההצפנה עצמה תהיה XOR. אנחנו נתעסק אך ורק עם יצר הPAD איתו עושים את הXOR.



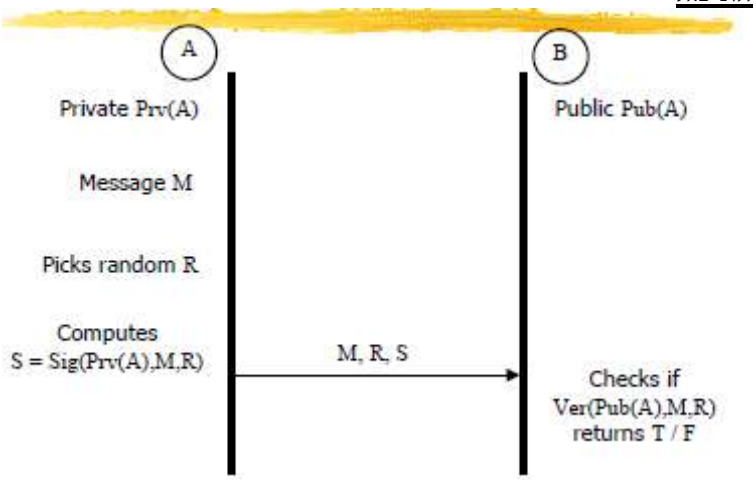
סדרת הPADים היא $P_1,...$.

MAC



למה הRANDOM ? שאותה הודעה פעמיים, לא יהיה לה אותו MAC.

חתימה



נעבור למימוש הפונקציות עצמן.

נשים לב שגודל הפלט חייב להיות גדול - אם זה קטן מדי, לדוגמא – 10 ביט, בטוח שאחרי $2^{10}+1$ נמצא זוג שנותנים את אותו HASH (מתקפת חיפוש ממצה). הגודל שעובדים איתו כיום הוא בד"כ 128 ביט.

פרדוקס יום היולדת:

אם יש 23 איש בכיתה– ההסתברות היא $\frac{1}{2}$ שיש שניים עם אותו יום הולדת.

העיקרון הוא כדלקמן - במרחב עם N תוצאות אפשריות, פילוג אקראי של נסיונות במרחב, מספיק לבחור $k=1.18\sqrt{N}$ נסיונות כדי להגיע להסתברות $1/2$ להתנגשות.

למה? נגדיר $Q(N,k)=$ ההסתברות שאין התנגשות לאחר k נסיונות.

$$Q(N,K)=1\cdot\frac{N-1}{N}\cdot\frac{N-2}{N}\cdot\ldots\cdot\frac{N-K+1}{N}=(1)\left(1-\frac{1}{N}\right)\left(1-\frac{2}{N}\right)\cdot\ldots\cdot\left(1-\frac{K+1}{N}\right)$$

$$\le e^{-1/N}\cdot e^{-2/N}\cdot\ldots\cdot e^{-k+1/N}=e^{-\frac{k(k+1)}{N}}$$

חשוב לשים לב שיש קשר בין k^2 לבין N.

<u>התקפת יום ההולדת</u>

ההתקפה אומרת שכדי לעמוד בתנאי Strong Coliision Resistance, לא מספיק שגודל הOUTPUT יהיה מחוץ לגודל החיפוש הממצה האפשרי, אלא צריך שחצי מגודל המפתח יהיה מחוץ לגודל.

מהי התקפת יום ההולדת?

נניח ויש מסמך שאני רוצה לחתום עליו – A. יש מסמך שרוצים שתהיה עליו חתימה (לא אני) – B.

אני חותם על HASH של A, כלומר מחשב את $sig(h(A))$, ושולח אותו יחד עם A. אבל מישהו אחר רוצה לשלוח את B, עם החתימה שלי.

מה המתקיף עושה? מייצר הרבה מאוד וריאציות של A,B , כך שיש להן אותה משמעות, אבל סינטקטית הם שונים (לדוגמא, רווחים, ...). זה לא כל כך קשה – לדוגמא, 64 מקומות של האם לשים רווח או לא נותן לנו 2^{64} אופציות.

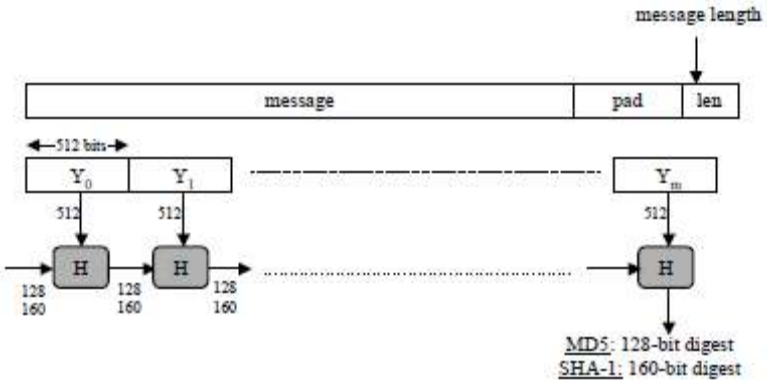
לאחר 2^{64} נסיונות, כנראה שנמצא שני מסמכים שיש להם אותו HASH. אז גורמים לחותם לחתום על הגרסה המתאימה של A, ויודעים שאפשר לשלוח במקומו את הגרסה המתאימה של B.

איך נמנעים מזה? מגדילים את המפתח. אופציה נוספת היא להוסיף לכל מסמך שאני חותם עליו 100 ביטים אקראיים , או משהו בסגנון, בזמן שאני חותם עליו. זה מונע ממישהו לגרום לי לחתום על מה שהוא רוצה בדיוק.

<u>איך בנויות פונקציות HASH ?</u>

פונקציות HASH שמשמשות בהן בימינו:

1. MD-5
2. SHA-1
3. SHA-256
4. …



המבנה הוא – בהינתן הודעה M, מחלקים אותה לבלוקים של 512 ביטים. מכניסים אותה לפונקציה H, שיודעת לקבל 512 ביטים ועוד 128 / 160 ביט (בהתאם לאלגוריתם). נשים לב לדימיון לCBC. הבלוק האחרון הוא הפלט.

נשים לב – בבלוק האחרון נהוג לרשום את גודל ההודעה. **זה הכרחי, כדי שאי אפשר יהיה להאריך את ההודעה.**

26.11.2007

נושאים מתורת המספרים + אלגברה

על סדר היום:

- GCD
- פונקציית אוילר
- חבורות
- שדות
- משפט קטן של פרמה
- בדיקת ראשוניות

GCD – Greatest common divisor

$\gcd(a,b)$ = מספר מירבי d שמחלק את a,b.

תכונה: נגדיר $LC(a,b)$ כקומבינציות הלינארית של a,b, כלומר $LC(a,b)=\{xa+yb \mid x,a\in\mathbb{Z}\}$.

מתקיים $\gcd(a,b)=\min_{>0}\{LC(a,b)\}$.

נשים לב – לכל a,b שלמים, קיימים x,y שלמים, כך ש $xa+yb=\gcd(a,b)$.

אלגוריתם אוקלידס לחישוב GCD

מתקיים $\gcd(a,b)=\gcd(b,a \bmod b)$.

לדוגמא - $\gcd(27,12)=\gcd(12,27 \bmod 12=3)=\gcd(3,0)=3$.

זה לוקח $O(\log(\min(a,b)))$.

על זה מתבסס אלגוריתם אוקלידס. אלגוריתם אוקלידס המוכלל מחזיר גם את x,y המקיימים $xa+yb=\gcd(a,b)$.

פונקציית אויילר

$\phi(N)$ היא פונקציית אוילר, והיא מוגדרת להיות מספר המספרים **הזרים** לN מ1 ועד N. זרים≡מחלק מרבי 1.

לדוגמא = $\phi(10)=4$. $\phi(11)=10$. נשים לב – לN ראשוני מתקיים $\phi(N)=N-1$. בכל מקרה, מתקיים $\phi(N)=N\prod_{\substack{P/N\\(\text{P that divides N})}}\left(1-\frac{1}{P}\right)$.

בנוסף, אם P,Q ראשוניים, מתקיים $\phi(P\cdot Q)=(P-1)(Q-1)$

חבורות

חבורה חיבורית

חבורה היא קבוצה ופעולה (נסמנה +), עם איבר הנחשב איבר היחידה. מקיימת:

1. סגור - $a,b\in S\Rightarrow a+b\in S$
2. איבר אפס - $\exists z\in S$ כך ש $\forall a\in S, a+z=z+a=a$
3. נגדי – לכל a יש איבר בחבורה שנסמנו $(-a)$ המקיים $a+(-a)=(-a)+a=0$
4. אסוציאטיביות - $(a+b)+c=a+(b+c)$.

נשים לב – מטריצה רגולריות עם פעולת הכפל מהווה חבורה. גם תמורות עם פעולת ההרכבה.
חבורה כפלית - $(S,\cdot)$.

1. סגור $a,b\in S\Rightarrow a\cdot b\in S$
2. יחידה – קיים $u\in S$ כך שלכל $a\in S$ מתקיים $a\cdot u=u\cdot a=a$
3. הפכי – לכל a קיים a^{-1} המקיים $a\cdot a^{-1}=a^{-1}\cdot a=u$
4. אסוציאטיביות $(a\cdot b)\cdot c=(a\cdot b)\cdot c$

שדה Z היא קבוצה S, עם פעולות $+,\cdot$, עם האיברים z,u, כך ש $S,+,z$ היא חבורה חיבורית, ו $S\setminus\{Z\},\cdot,u$ היא חבורה כפלית.
בנוסף, שדה Z מקיים את התכונת

1. חילופיות – $a\cdot b=b\cdot a, a+b=b+a$
2. דיסטריבוטיביות - $a\cdot(b+c)=a\cdot b+a\cdot c$

<u>חבורות מעל מודולו</u>

$\mathbb{Z}_N=(\{0,1,...,N-1\},+_{\bmod N})$.

נשים לב –זה חבורה חיבורית לכל N טבעי. היא בת N איברים.
חבורה כזו היא כפלית אם N הוא ראשוני.
נשים לב – אפשר להגדיר $Z_{10}^*=(\{1,3,7,9\},\cdot_{10})$ - אם יש בקבוצה רק את האיברים הזרים לN (10 במקרה זה).

<u>הגדרות:</u>
<u>חבורה חלופית / אובלית :</u>מקיימת $a\cdot b=b\cdot a$ (או חיבור כמובן). אנו נדבר רק על חבורות חילופיות.
<u>חבורה ציקלית:</u> יש איבר (לפחות אחד), g, שנקרא לו יוצר של החבורה – כלומר, $S=\{g^i\mid i\in\mathbb{N}\cup\{0\}\}$.
לדוגמא - Z_{10}^+ - היוצרים שלה הם 1, 3.

<u>משפט פרמה</u>
אם P ראשוני, אזי לכל a זר לP, מתקיים $a^{P-1}=1(\bmod p)$.
על כן, $a^{p-1}=a\cdot a^{p-2}=1\Rightarrow a^{p-2}=a^{-1}$.

<u>תכונות</u>
איבר a יקרא ריבוע אם קיים b המקיים $b^2=a(\bmod p)$. נשים לב – אם b ריבוע, אז $p-b$ גם ריבוע.

יהי g יוצר של Z_p^*, מתקיים $g^x=_{\bmod p}g^y$ אמם $x=y(\bmod(p-1))$.

<u>בדיקת ראשוניות</u>
נדבר על p בסדרי גודל של 2^{1000}. קשה למצוא ראשוניים כאלה. איך עושים זאת? האלגוריתם הבסיסי אומר לחפש עד גודל שורש p.
כיום משתמשים באלגוריתם ההסתברותי של מילר ורבין. האלגוריתם בודק ראשוניות של מספר.

<u>צפיפות הראשוניים</u>

נגדיר $\Pi(N)$ כמספר הראשוניים עד N. מתקיים $\Pi(N) =\sim \frac{N}{\ln N}$.

כלומר, בערך $\frac{1}{\ln N}$ מספרים ליד N יהיו ראשוניים.

על כן, אם אני רוצה למצוא מספר ראשוני בסדר גודל של 2^{1000}, אני יכול להגריל 1000 ביטים. מה הסיכוי שהמספר ראשוני? בערך $\frac{1}{1000}$. אפשר כמובן לנפות את הזוגיים בקלות, ולהגיע ל $\frac{1}{500}$.
ליד p אקראי בן 1000 ביט סביר להניח שיש ראשוני ב $\pm$ האי זוגיים לידו.

איך נמצא אז מספר אקראי? ניקח N אקראי אי זוגי.

<u>איך נבדוק אם N הוא כנראה ראשוני?</u>

1. בחר a אקראי המקיים $1 < a < N$.
2. מחשבים את $a^{N-1} \pmod N$.
3. אם התוצאה שונה מ1, מכריזים שN לא ראשוני (ע"פ משפט פרמה)
4. בודקים בחישובים של 2 אם ראינו $a^i \neq \pm 1$ אבל $a^{2i} = 1$ - אם כן, N לא ראשוני.

את פעולה זו נבצע t פעמים. ההסתברות ש N פריק ועובר מבחן 1 כראשוני $\geq \frac{1}{2}$. על כן, אם הוא עבר t בדיקות, אזי ההסתברות שהוא עבר את כולן היא $\frac{1}{2^t}$.

כמה זמן יקח לנו למצוא מספר ראשוני אקראי? $t \cdot 100$ בערך. עבור $t = 100$, זה בערך $100 \cdot 100$.
אבל צריך לבצע בערך 1000 כפלים כל פעם. מדובר בערך ב 10 מיליון פעולות כפל.
נשים לב שבד"כ נתפוס לא ראשוניים מאוד מהר – ולכן הפקטור של $t = 100$ אפשר לנוון, ובערך ב1000 כפלים כפול 100 – 200 סיבובים של N נקבל ראשוני. בערך – מיליון פעולות כדי למצוא מספר ראשוני של 1000 ביט.

28.11.2007

קריפטוגרפיה אסימטרית – Public key crypts

1976 – המודל התפרסם על ידי Diffie Hellman.

1976 – שיטת החלפת מפתח DH

1976 – merkle הוציא שיטת הצפנה שנשברה.

1977 – RSA.

1980 – אלגוריתם חתימה של Lamport.

1980 – El-gamal – הצפנה, חתימה.

מודל

לכל ישות (A) יש שני מפתחות:

- PRV(A) – פרטי רק לA.
- PUB(A) – ציבורי לכל העולם.

הצפנה: פונקציות D,E המקיימות:

$$C = E\left(M, Pub\left(A\right)\right)$$

$$M = D\left(C, \Pr v\left(A\right)\right)$$

כלומר – כל אחד יכול להצפין, רק A יכול לפתוח.

חתימה: פונקציות S,V - $\sigma = S\left(M, \Pr v\left(A\right)\right)$ רק A יכול לתחום.

כל אחד יכול לוודא - $V\left(M, \sigma, Pub\left(A\right)\right) = 0/1$.

דרישות

- V,S,D,E - פונקציות ידועות
- V,S,D,E - קלות לחישוב בהינתן מפתח מתאים
- בהעדר מפתח מתאים, **קשה מאוד** לחלץ את M או לזייף את $\left(M,\sigma\right)$.
- בהינתן Pub(A) יהיה קשה לחלץ את Prv(A). (גם אם ראינו הרבה הודעות).

מה אנחנו מחפשים? פונקציות 'חד כיווניות' – שקל מאוד לחשב אותם עם 'רמז', אבל בהעדר ה'רמז', מאוד מאוד קשה לחשב אותן. קוראים לזה 'one way trap door function'

שיטת ההצפנה של Merkel

מתבססת על בעיית knapsack. יש N עצמים $\left\{1,...,N\right\}$, לכל אחד מהם יש משקל חיובי $\left(k_i\right)$.

נגדיר $c = \sum x_i k_i$ כאשר x_i מציין אם עצם i נבחר. בהינתן c קשה מאוד לבחור

$X = \left(x_1.,...,x_N\right)$ המקיים את התנאי.

איך עושים מזה שיטת הצפנה?

בסיס: N עצמים $\left\{1,...,N\right\}$ עם משקולות k_i ידועים.

הודעה: $M = X_1,...,X_N$ ($X_i \in \left\{0,1\right\}$).

הצפנה (ניסיון 1): $C = \sum_{i=1}^{N} k_i x_i$.

מה הבעיה? זה לא מספיק בשביל לפענח (זה לא חח"ע).

נסיון 2 (ומוצלח)– נגדיר את $\vec{k}$ להיות super incresing, כלומר, מקיים $k_i > \sum_{j=1}^{i-1} k_j$. ההצפנה תמשיך להיות כזו - $C = \sum_{i=1}^{N} k_i x_i$

פענוח : סקור את $\vec{k}$ מסופו לתחילתו. ובחר k_i (עם i גדול ביותר) שעדיין לא גדול מC – הוא חייב להיבחר. ממשיכים איטרטיבית עם $C - k_i$.

נשים לב שעדיין אין לנו אלגוריתם הצפנה. מה עושים?

מגדירים $m > \sum_{i=1}^{N} k_i$. מצא w זר יחסית לm. (כלומר $\gcd(m,w)=1$). חשב את $w^{-1} \pmod m$.
הגדר $k^* = \left(w \cdot k_1 \pmod m, ..., w \cdot k_N \pmod m\right)$.

מפתח ציבורי - $\vec{k}^*$
מפתח פרטי - $\left(m, w, w^{-1}, \vec{k}\right)$.

<u>הצפנה:</u> בהינתן $M = x_1, ..., x_N$, הצפנה תהיה $C^* = \sum_{i=1}^{N} k_i^* x_i$.

<u>פענוח:</u> נגדיר $C = C^* \cdot w^{-1} \pmod m$. נשים לב כי $C = \sum_{i=1}^{N} k_i^* x_i w^{-1} \pmod m = \sum_{i=1}^{N} k_i x_i$.
מפה מחלצים את x_i ים כמו שראינו קודם – אפשרי שכן הוקטור $\vec{k}$ הוא super increasing. בעצם – מה שהמפתח הפרטי עשה זה למסך את המבנה הsuper increasing של הוקטור.

האלגוריתם נשבר כמה חודשים יותר מאוחר על ידי עדי שמיר – והשבירה היא שבעיית ה super increasing knapsack **מעורבלת** היא לא NP קשה .
נשים לב לשימוש בבעיות NP קשות.

<u>שיטת החתימה החד פעמית של Lamport</u>
A ימציא וקטור של זוגות באורך N. $\left(x_1, y_1\right), ..., \left(x_N, y_N\right)$. x_i, y_i - מספרים אקראיים גדולים.
<u>מפתח ציבורי:</u> $\left(H\left(x_1\right), H\left(y_1\right)\right), ..., \left(H\left(x_N\right), H\left(y_N\right)\right)$.

בהינתן $M = b_1 ... b_N$ (b_i-ביטים), החתימה של A תהיה פרסום של $z_1 ... z_N$ כאשר $z_i = \begin{cases} x_i, b_i = 0 \\ y_i, b_i = 1 \end{cases}$.
כדי לוודא ביט i - אם $b_i = 0$, בודקים האם $H\left(z_i\right) = H\left(x_i\right)$, אחרת בודקים אם $H\left(z_i\right) = H\left(y_i\right)$.

נשים לב – זה לא פרקטי לחלוטין. זה דורש שליחה של המון מידע, ודורש לבצע **המון** HASHים.
בלי קשר – מעניין לשים לב שזה מניח את החוזק של פונקציית הHASH.

מה עושים באמת כיום?
משתמשים בבעיות ה'קשות' הבאות:
1. לוג דיסקרטי – Diffie Helman, El Gamel, DSS, Elliptic curves.
2. פרוק מספר למרכיביו הראשוניים – RSA.

חשוב לשים לב – לא מדובר בבעיות NP קשות.

בעיית הלוג הדיסקרטי
בהינתן p ראשוני גדול, ובהנתן g יוצר של $\mathbb{Z}_p^*$ קל לחשב את $y = g^x \pmod p$. קשה לחשב את $x = D\log_{p,g} y$.
כאן בד"כ x יהיה המפתח הפרטי, y הציבורי.

בעיית הפירוק לגורמים
בהינתן p,q ראשוניים גדולים, קל לחשב את $N = p \cdot q$. בהינתן N פריק כנ"ל, קשה למצוא את p,q.
כאן p,q יהיו המפתח הפרטי, N הציבורי.

5.12.2007

היום נדבר על אלגוריתמים בהצפנה אסימטרית

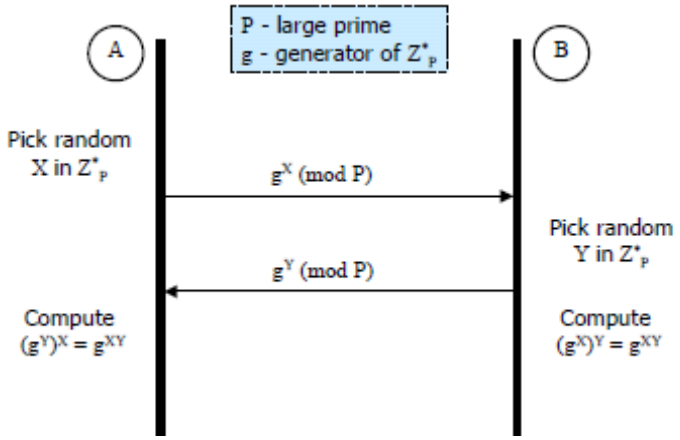
- D-H
  - החלפת מפתח
- El Gamal
  - הצפנה
  - חתימה
- DSS
  - חתימה
- RSA
  - הצפנה
  - חתימה
  - החלפת מפתח

קיצורים להיום – KE = Key Exchange. בדרך כלל מדובר במפתח סימטרי.

Diffie Hellman Key Exchange

הנחות :

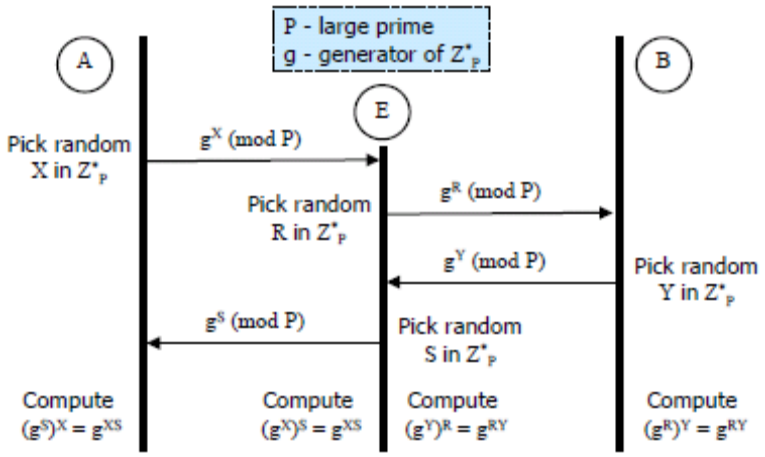
1. קיים P ראשוני גדול, ידוע לכולם.
2. קיים g יוצר של z_p^*, גם ידוע לכולם.



מה מכיר העולם? p, g, g^x, g^y . נסמן $k = g^{xy}$.

הנקודה כאן היא שלא ניתן לחשב את x, y מראיית הנתונים שעוברים. נשים לב, שאנחנו לא יכולים לחשב את k מראיית הנתונים – ניתן לחשב את g^{x+y}, לא את g^{xy}. ככה קיבלנו מפתח, שאפשר להשתמש בו בהצפנה סימטרית לדוגמה.

נשים לב –זה פגיע להתקפת **Man in the middle: - כלומר, אי אפשר לדעת עם מי אתה מדבר בוודאות.**

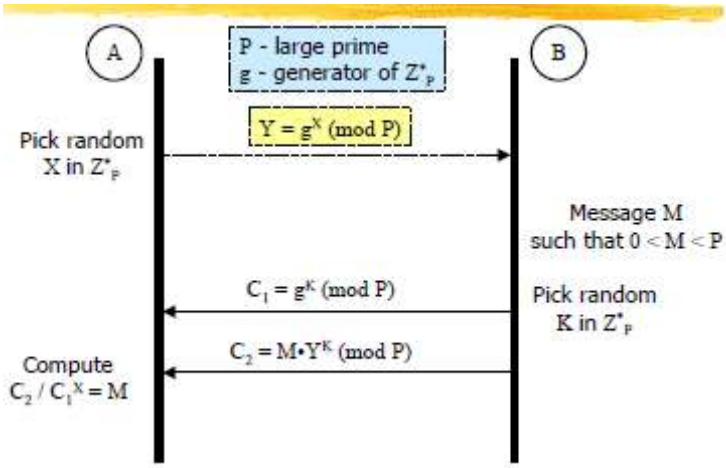


מה אפשר לעשות נגד ההתקפה הזו?

1. קו מוגן – אם אני יודע שהקו בין A וB לא ניתן לשינוי, אז אין בעיה.
2. חתימה דיגיטלית – הצדדים מראש יודעים מידע מסוים אחד על השני.

<u>El-Gamal – הצפנה</u>

1. תשתית מוכרת לכולם –
   a. P ראשוני גדול.
   b. g יוצר של z_P^*



כאשר A רוצה לייצר מפתח ציבורי פרטי, הוא בוחר לעצמו X אקראי מתוך z_P^*. הוא מפיץ את

$$y = g^x \pmod p$$

כאשר מישהו (לדוגמא B) רוצה להצפין לA את הודעה $0 < M < P$, הוא ממציא K אקראי בחבורה.

הוא שולח את $C_1 = g^k \pmod p$, ואת $C_2 = y^k \cdot M \pmod p$.

A מחשב את $\frac{C_2}{C_1^x} = \frac{g^{xk} \cdot M}{g^{kx}} = M$.

נשים לב לדמיון הגדול מול DH (כפל מתפקד כהצפנה, חילוק כפענוח).

איך חותמים (עם El Gamal / DSS):

1. ידוע לכולם P ראשוני גדול , g יוצר.
2. A מגריל X אקראי (פרטי). מדובר בX שמיועד לחתימה.
3. A מפרסם $y = g^x \pmod p$ מפתח ציבורי.

4. עבור הודעה M, מגרילים k אקראי, ושולחים גם את $\sigma = f(M, x, k)$.
5. המקבל מוודא באמצעות y, p, g, σ, M. (פרטים – בשקפים).

כמה מילים על ה k ים האקראיים – למה הם חשובים?
- הצפנה של אותה הודעה פעמיים תיראה שונה.

DSS – Digital signature standard

1. פותח ב1980 בארה"ב כתקן.
2. דומה לאל גמאל, מממש פונקציה שמאפשרת רק חתימה, לא הצפנה.
3. המטרה – שחתימה תהיה נפוצה, אבל שלא יהיה כלי סטנדרטי של הצפנה.

RSA

A רוצה לבנות לעצמו מפתח פרטי וציבורי.

1. בוחר p, q ראשוניים גדולים (לא קשורים).
2. מחשב $N = p \cdot q$.
3. מחשב $\phi(N) = (p-q)(q-1)$ (מספר המספרים שזרים לN).
4. ממציא / מחפש e זר יחסית ל $\phi(N)$.
5. מחשב את d ההפכי של e במודול $\phi(N)$. (כלומר - $e \cdot d = 1 \pmod{\phi(N)}$)

מפתחות:

1. ציבורי (N, e)
2. פרטי (N, d)

(הערה – ניתן להשמיד את $p, q, \phi(N)$).

הצפנה (לA): $C = M^e \pmod N$.

פענוח: $M = c^d \pmod N$.

חתימה(של A): $\sigma = M^d \pmod N$

וידוא: בדיקה שאכן $\sigma^e \pmod N = M$.

למה מתקיים ש $C = M^{ed} \pmod N = M \pmod N$?

$$M^{ed} \pmod N = M^{1+d\cdot\phi(N)} = M \cdot M^{d\phi(N)} = M \cdot \left(M^{\phi(N)}\right)^d = M \cdot 1^d = M$$

הדבר נכון במודולו P, במודולו Q, ולכן גם במודולו N (ממשפט השאריות הסיני).

תכונות:

1. כפליות - $\begin{matrix} M_1 \to \sigma_1 \\ M_2 \to \sigma_2 \end{matrix}$ אזי $M_1^i \cdot M_2^j \to \sigma_1^i \sigma_2^j$. יש כאן בעייתיות בחתימה (אפשר לייצר חתימות על הודעות שלא עברו). זה לא מטריד בהצפנה (כי כל אחד יכול להצפין).
2. אסור ש M^e יהיה קטן מN (כי אז אפשר להוציא שורש e רגיל) $\Leftarrow$ M קטן מסוכן). לעומת זאת, e קטן יאיץ את ההצפנה.

3. נשים לב – אי אפשר לבקש שגם e,d יהיו קטנים (שכן הם הופכיים). לעומת זאת, נרצה שאחד מהם אולי יהיה קטן, למטרת יעילות. נשים לב – אם d הוא קטן , אז אפשר לתקוף את האלגוריתם על ידי חיפוש ממצה. לפעמים נהוג לבחור $e=3$ או $e=2^{16}+1$.

איך פותרים את בעיית הכפליות?

1. RSA LABS הציגו תקן בשם PKCS 1, (Public key cryptor Standard) – שמה שהוא אומר שצריך לא סתם לחתום על M, אלא על $0002(Rand)M(...)$, כדי לפתור את הבעיה הזו. העלאה בריבוע או כפל ישבור את הפורמט. זה גם פותר בעיות של Mים קטנים.

10.12.2007

במחצית הזו נתאר איך מערכות משתמשות בקריפטוגרפיה, לא את המתמטיקה של הקריפטוגרפיה.
השבוע נדבר על אימות אנשים.
אפשר לדבר על כל סוגי האימות של אדם / מחשב כלפי /אדם / מחשב (יש ארבע אופציות).

מטרות האימות:

1. זיהוי - באופן פשוט, לזהות מול מי אני עומד.
2. אימות / וידוא – לוודא שמי שנדמה לי שאני עומד מולו, הוא באמת הוא.
3. הרשאות לפעולות
4. מעקב אחרי פעולות
5. auditing

<u>פרמטרים לבחירה בשיטת אימות וזיהוי</u>

1. זמן זיהוי / אימות
2. עלות מערכת
3. אחזקה
4. רמת דיוק
    a. חיובי – האם כשהיא מקבלת מישהו, מה הסיכוי שתטעה?
    b. שלילי – האם כשהיא דוחה מישהו, מה הסיכוי שטעתה?
5. קלות שימוש
6. יכולת העברה של מזהה
7. החזרת מזהה (האם אפשר לשכפל אותו?)
8. שינוי / תפוגה של מזהה

<u>אימות אנשים</u>

1. משהו שהמשתמש יודע
    a. סיסמא
    b. PIN
    c. מידע אישי
2. משהו שהמשתמש הינו
    a. ביומטרי
    b. פיזיומטרי
3. משהו שהמשתמש מחזיק
    a. מפתח
    b. כרטיס חכם
    c. ציפ
    d. סלולרי

כיום בד"כ משתמשים בשילוב בין השיטות לעיל, ולא אחת בלבד.

<u>משהו שהמשתמש יודע</u>

הבעיה היא שאנחנו רוצים את הדרישות המתנגשות הבאות:

1. קל למשתמש לזכור
2. קשה למתקיף / לנחש / למצוא / לפצח.

האופציות:

1. סיסמאות. בד"כ
    a. מילה באנגלית
    b. שפה דמוית אנגלית
    c. שמות / מספרים חשובים
    d. צירוף של תווים שמורכב ממלעיל
    e. סיסמא קודמת עם מניפולציה פשוטה

f. משהו שקל לניחוש
g. משהו שרשמנו איפשהו
h. סיסמאות דומות / זהות במערכות שונות

הנקודה היא שזה לא יותר מדי אופציות – סד"ג של 10^6. היינו רוצים – 64 – 128 ביטים אקראיים – כלומר 8 – 16 תווי ASCII אמיתיים. אם לוקחים מהתווים הויזואלים בלבד, מדובר על 11 – 20 ויזואליים.
זה מקביל ל20 – 40 תווים הניתנים לקריאה.
מדובר על 30 – 50 תווים שמייצרים משפט.

הפער הזה בין הרצוי למצוי הוא זה שגרם למעבר החוצה מהסוג הזה של אותנטיקציה – הסיבה שנשארים איתו כי זה דבר שמאוד קל / נוח לעבוד איתו.

<u>מורכבות של מערכת סיסמאות</u>
- אלפי משתמשים
- מספר רב של שרתים
- אותם משתמשים במערכות שונות

בטוח שמבין כל אלפי המשתמשים, נמצא לא מעט עם סיסמאות חלשות. ולכן כל מערכת עם סיסמאות ניתנת לפריצה.

בנוסף - הסיסמא (או נגזרת שלה) נשמרת במחשב – האופציות הן:
1. ללא הגבלת גישה באופן גלוי
2. עם הגבלת גישה באופן גלוי
3. עם הגבלת גישה באופן מוצפן (איפה המפתח?)
4. עם / בלי הגבלת גישה באופן של HASH.
5. קובץ מצוי בשרת מבודד

בUNIX, יש קובץ passwd שמכיל HASH של סיסמאות, ומוגן בACL. נשים לב שקריאת קובץ זה לא אמור לתת למתקיף יותר מדי מידע. נשים לב שבמערכות רבות צריך באמת לתת הרשאות גישה להרבה מאוד תהליכים.
מה הבעיה עם זה? Online Dictionary attack. אם למתקיף יש גישת קריאה לקובץ, הוא יכול לבוא עם הרבה HASHים של סיסמאות טיפוסיות, ולבדוק אם אחד מהHASHים האלה נמצא שם.
כדי למנוע את זה, אפשר פשוט למנוע גישת קריאה מהקובץ. מנגנון נוסף שאמור להתמודד עם זה הוא salt.
עבור משתמש A שומרים את $salt_A, H\left(PW_A \,||\, SALT_A\right)$. $salt_A$ הוא מספר אקראי שמגרילים כל פעם שמשנים סיסמא.ברור איך זה מונע את ההתקפה הרלוונטית (צריך לעשות HASH לכל סיסמא עם $2^{|salt|}$).
נשים לב שזה לא מונע התקפת OFFLINE (כי אפשר לנתח את הקובץ עם הsaltים המתאימים).

פרמטרים עבור סיסמאות:
- נוח למשתמש
- עלות טובה
- יכולת העברה – אפשרית
- החזרה – לא אפשרית
- פקיעה – קלה.
- דיוק (חיובי / שלילי) – כמעט 100% (בהנחת פונקציית HASH טובה).
- אחזקה נוחה
- בטיחות – לא מדהימה.

<u>ביומטריה – מה שהמשתמש הינו</u>

- טביעת אצבע
- גיאומטריה של כף יד
- טביעת קול
- זיהוי פנים
- זיהוי קרנית
- זיהוי רשתית
- DNA

נשים לב – אפשר אולי להקליט את המידע הזה ולעשות לו PlayBack.
טביעת אצבע – מודדים תמונה דו מימדית של האצבע. בד"כ מוצאים כמה נקודות פיתול, ושומרים מאפיינים שלהם.
יש לעשות תהליך של רישום – בה הוא לומד את האצבע בכל מיני זויות.
בעיות:

1. לכלוך על האצבע / גלוי
2. פציעה

התקפות:

1. הקלטה וייצור אצבע מפלסטיק מלטקס עם טביעת האצבע שלי
   a. הגנה אפשרית – תמונה תלת מימדית (יותר קשה לזיוף). בדיקת מוליכות חשמלית של האצבע.
   b. בדיקת חיות של האצבע – בודקים טמפרטורה, לחץ דם, ...

גיאומטריית כף יד – מאוד דומה לטביעת אצבע.

<u>בדיקת קרנית</u> – השתמשו בזה בד"כ בכספומטים.

12.12.2007

אימות אנשים – ע"י משהו שהמשתמש מחזיק

- תעודה מזהה
- כרטיס נייר / פלסטיק
- מרכיב חישובי
- מפתח פיזי

כרטיס סיסמאות

- כרטיס זכרון / נייר עם רשימה של N סיסמאות חד פעמיות
  1. המשתמש קורא סיסמא, מוחק מהכרטיס
  2. שולח את מספר הסיסמא, ואת הסיסמא.
  3. השרת מתחזק רשימה של הסיסמאות שהמשתמש עדיין לא השתמש בהם.
  4. יתרונות:
     - סיסמאות חזקות
     - אין בעית האזנה
     - זול מאוד
  5. חסרונות:
     - פגיע לMan in the middle (הוא יתחזה לשרת, וישמור את הסיסמא של המשתמש).
     - זכרון רב בשרת
     - אובדן הכרטיס
     - פריצה לשרת

כרטיס סיסמאות חישובי – S/Key

נשמור - X_0 אקראי. נגדיר $X_i = h\left(x_{i-1}\right)$ (עבור h פונקציית HASH קריפטוגרפית).

1. הלקוח שומר אצלו את X_0 ואת N (מספר הסיסמאות הנדרשות).
2. השרת שומר אצלו את X_N.
3. הלקוח שולח את $X_{N-1} = h^{(N-1)}\left(X_0\right)$ (הוא מחשב את זה מ X_1).
4. השרת מוודא על ידי חישוב האם $X_n \stackrel{?}{=} h\left(X_{n-1}\right)$.
5. הלקוח מבצע - $N \leftarrow N-1$.
6. השרת מבצע $X_N \leftarrow X_{N-1}$.

תכונות:

1. זמן חישוב – לינארי בN. מצד שני, N קטן יגרום לכך שנצטרך להקטין את הכרטיס כמה שיותר.
   a. אפשר כמובן לשמור נקודות ביניים כדוגמת $X_0, X_{\frac{n}{10}}, \ldots$
2. סיסמאות חזקות.
3. פחות זכרון.
4. אין חשש מהאזנה
5. כרטיס חישובי יקר יותר
6. התקפת Man in the middle אפשרית.
7. פריצה לשרת לא בעייתית – אין שם את הסיסמאות / מידע רגיש.

כעת נרצה להתמודד עם התקפות Man in the middle. השיטה תהיה בדרך כלל Challenge Response. כדי שהשרת יהיה משוכנע שהוא אכן מדבר עם A, השרת ירצה להיות אקטיבי. כדי שהשרת ידע שהוא מדבר עם A האמיתי, הוא יתן לA אתגר (C).

Challenge Response סידורי

1. השרת והלקוח שומרים K (פר לקוח)
2. הלקוח שולח A (את שמו)
3. השרת שולח C סידורי
4. הלקוח שולח $R = PW = H(c, K)$
5. השרת מוודא.

K סודי סימטרי בין A ובין S. H פונקציית ערבול. תכונות:

1. מספר סידורי קל לחשב.

גרסה אחרת – הלקוח שולח את הC הסידורי. ההבדל הוא שגם השרת יצטרך לשמור את C , ולצפות לכך שהוא סידורי. מדובר במעין 'אתגר עצמי'. (נשים לב – זה יותר פגיע לMan in the middle). כאן המספר הסידורי צריך להיות פר לקוח. היתרון כאן זה שאין צורך בפרוטוקול דו כיווני.

נדבר על אותו פרוטוקול כאשר $C = time$, ונדבר על שני הכיוונים האפשריים. תאורטית, אם הסינכרון מושלם, לא צריך לשלוח את הזמן. בד"כ, נצטרך לשלוח את הזמן.
נשים לב – אם נעבוד ברמה של מילי שניות, חייבים סינכרון (אי אפשר להסתמך על סינכרון עצמי), ואז נצטרך לעבוד עם granularity של שניה – 5. הסכנה בכך היא כשפותחים את חלון הזמן , אני חשוף לman in the middle בחלון הזמן הזה. אבל אם אתה יכול לתקוף בצורה כזו, אתה יכול פשוט להשתלט על הקו אחרי הצלחת האותנטיקציה.
<u>חשוב לשים לב</u> – כל הפרוטוקולים מוודאים שהלקוח הוא זה שמדבר איתנו בשלב האותנטיקציה. הוא לא מוודא שמישהו ניתק את הלקוח, ומדבר במקומו.

כעת נדסקס את אותו פרוטוקול עם $C = rand()$.
יתרונות – לא צריך סנכרון. כמובן שסינכרון עצמי לא אפשרי (אם הלקוח ישלח את המספר האקראי, הוא יכול להשתמש באחד מהעבר).
חסרון – יקר לייצר מספר אקראי.

<u>One Time Pad – OTP</u>

1. לקוח A מדליק
2. מקליד PIN
3. מקבל מספר אקראי (OTP) המועבר לשרת

$$OTP = f(ID_A, \text{time}, \text{K}_\text{A}, PIN)$$

K_A נשמר בשני הצדדים.
יש גירסאות בהן הPIN משמש לפתיחה ולא נשלח (הגרסה הישנה של הOTP), ויש גירסאות שהוא חלק מבסיס הסיסמא (ואז צריך לשמור את הPIN גם בשרת)
נשים לב שיש צורך בסינכרון זמן – של עשרות שניות בד"כ. אפשר לבדוק חלון זמן אחד קדימה.
יש בו גם סנכרון אדפטיבי – אם רואים שהלקוח מכניס מידע לקראת סוף החלון, אז אפשר לראות את זליגת הזמן שלו, ולהתחיל להתכוונן לחלון הבא. חשוב לשים לב שזה חושף את המערכת ליותר התקפות.
נשים לב – יתרון של שמירת PIN בשרת, זה שהשרת יכול לעקוב אחרי BRUTE FORCE של PINים.

17.12.2007

פרוטוקולי אימות

1. מבוססי סיסמא
2. מבוססי מפתח סימטרי
3. מבוססי מפתח אסימטרי
4. פרוטוקולי אפס-ידיעה

מבוססי סיסמא

1. A שומר סיסמא PW_A.
2. שולח A, PW_A
3. S (השרת) מוודא כי $h(PW_A)$ זהה לHASH המצופה.

תכונות:

1. יתרונות:
   a. פשוט
   b. אין חומרה בלקוח
   c. נוח לשימוש ושינוי
2. חסרונות:
   a. מאזין יכול לגנוב סיסמא (replay attack)
   b. ניתן לנחש / לחפש סיסמאות
   c. יש מידע רגיש בשרת

שימושיות:

1. ערוץ מאובטח
2. סיסמאות חזקות
3. הגנה על השרת

פרוטוקולים עם מפתח סימטרי

מדובר על מחשב מול מחשב.

השרת שומר עבור A את K_{AS}. A שומר גם את K_{AS} (המפתח של A מול S)

מבנה הפרוטוקול הבסיסי:

1. A פונה עם שמו.
2. מקבל C אתגר.
3. עונה $R = f(C, K_{AS})$. המטרה היא שR יהיה קשה לחיזוי. (חשוב להגיד שצורת השרשור של C, K_{AS} היא חשובה – לדוגמא הצורה המתוארת בHMAC היא טובה).

מהו C? בד"כ אחד משלושה: (קוראים לו בד"כ nonce – מחד פעמי)

1. אקראי
2. זמן
3. מונה

מהי f?

1. HASH
   a. חד פעמית.
   b. קשה למצוא התנגשויות
2. MAC

a. קשה למצוא התנגשויות
3. הצפנה
a. הפיכה.

נתייחס לגרסה שונה של הפרוטוקול:
1. A שולח את שמו.
2. S שולח $f\left(C,K_{AS}\right)$, עבור f הפיכה.
3. A שולח את C בחזרה.

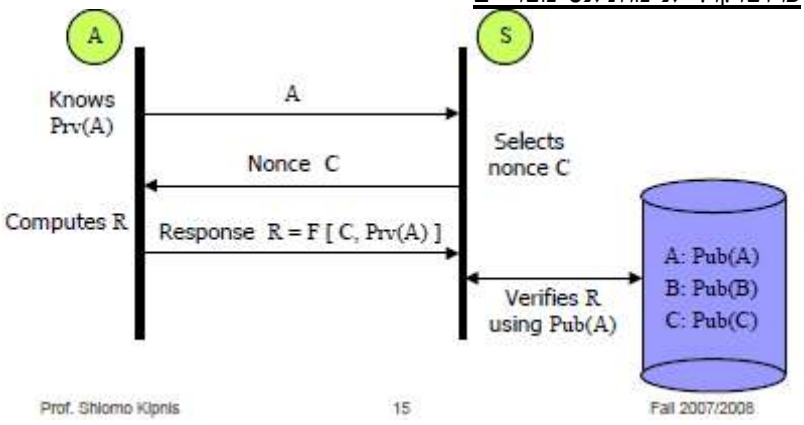
מתקפה: אם C בר חיזוי (בהנחה וראיתי C קודם בסדרה), אז אפשר פשוט לשלוח אותו.

נשים לב – יש גרסאות של הפרוטוקולים המתנהגים כ'שגר ושכח' עבור השרת, ואז זה חוסך לו בזיכרון.
תכונות :
1. יתרונות:
a. אין בעיה של האזנה
b. אין בעיה של חוזק המפתח
2. חסרונות:
a. דורש חומרה בלקוח
b. יש מידע רגיש בשרת (בלקוח תמיד יהיה מידע שמאפשר לו להתחבר לשרת)
c. ניתן להשיג הצפנות / פענוחים (התקפות Known / Chosen plaintext)

<u>פרוטוקול אימות סימטרי עם הודעה אחת:</u>
1. A שולח את $A,H\left(\mathrm{TIME},\mathrm{K}_{\mathrm{AS}}\right)$.

תכונות:
2. אתגר עצמי – זמן.
3. אם מדובר בפונקציית HASH – צריך רזולוציה נמוכה של זמן.
4. (גם אם מדובר בפונקציית הצפנה).

<u>פרוטוקולי אימות אסימטריים</u>



מהי f:
1. 'כמו חתימה' - מישהו נותן טקסט, ואני אחתום עליו
a. הדרך הנפוצה
2. 'כמו פענוח' – מישהו נותן טקסט 'כמו מוצפן', ואני שולח לו את ה'פענוח' שלו. כלומר, $C=g\left(x,pub\left(A\right)\right)$. אני צריך לשלוח את $R=x$.

תכונות:

1. יתרונות:
   a. אין מידע סודי בשרת
2. חסרונות:
   a. מישהו יוכל לסחוט ממני חתימה / פענוח(אם f היא חתימה). (כמובן אם המפתח משמש לעוד דברים).
   b. A מוכיח את זהותו מול מי שפנה אליו - לפעמים זה יתרון, לפעמים חסרון.
   c. לאחר ריצות מרובות, מאזין מקבל מידע על המפתח.

<u>פרוטוקול אפס ידיעה</u>

מה זה אפס ידיעה? זה לעבוד עם פרוטוקולים אסימטריים, אבל מדובר בפרוטוקולים, שגם אחרי מספר רב מאוד של פעמים, לא נדע כלום על המאמת.
הפרוטוקול הראשון התבסס על הוכחת ידיעת איזומורפיזם בין גרפים. המפתח היה האיזומורפיזם.
דרישות:

1. שלמות – A תמיד יוכל להוכיח את זהותו.
2. יציבות – מתחזה לA יכשל בהסתברות $\leftarrow 1$
3. אפס ידע – ריצות מרובות לא מסגירות מאומה על המפתח $PRV(A)$

<u>פרוטוקול ZK מבוסס לוג דיסקרטי</u>

<u>תשתית ידועה:</u> p ראשוני גדול, g יוצר של z_p^*.

<u>מפתחות פרטיים / ציבוריים של A:</u> A בוחר x אקראי ב z_p^*.

<u>פרטי:</u> x הוא המפתח הפרטי.

<u>ציבורי:</u> $y = g^x \pmod p$ הוא הציבורי.

<u>אימות של A מול B :</u>

1. <u>t פעמים בצע:</u>

   a. <u>התחייבות - commitment:</u> A בוחר k אקראי ב z_p^* ושולח לB את
   $$s = g^k \pmod p$$
   b. <u>אתגר - challenge:</u> (לשים לב לסדר!) B שולח $c = {}^{0}\!/_{1}$ אקראי.
   c. <u>Response:</u> A שולח את $R = k + c \cdot x \pmod p$ (כלומר, k או $k + x$).
   d. <u>Verify:</u> בודק האם $g^r =^{?} s \cdot y^c \pmod p$

<u>ניתוח:</u>

- **<u>שלמות:</u>** אם A ביצע את a, c כנדרש, אזי אכן $g^r = k + cx = s \cdot y^c$.
- **<u>יציבות:</u>** המתחזה אינו יודע את x. ראשית, הוא צריך להתחייב. נניח והוא מצפה ש $c = 0$, אז הוא יכול לבחור k אקראי. אם יגיע $c = 0$, הוא יענה עם $r = k$, ואכן הצליח בסיבוב הנוכחי – שכן $g^r = g^k = s \cdot y^0$ הוא תקין. אבל אם $c = 1$ התשובה היחידה שהוא יכול לענות היא $r = k + x$, ואת x הוא לא יודע, ולכן הוא יכשל בהסתברות $\frac{1}{2}$. ההסתברות שיצליח ב t פעמים רצוף היא $\left(\frac{1}{2}\right)^t$.

אם הוא מצפה ש $c=1$, הוא יבחר k אקראי, וישלח $s=\frac{g^k}{y}$. אם $c=1$ הוא אכן יצליח, אבל אחרת, אם $c=0$, אז הדבר היחידי שפותר את המשוואה זה $r=k-x$, והוא לא יודע את x, ולכן בהסתברות של $\frac{1}{2}$ הוא יכשל.

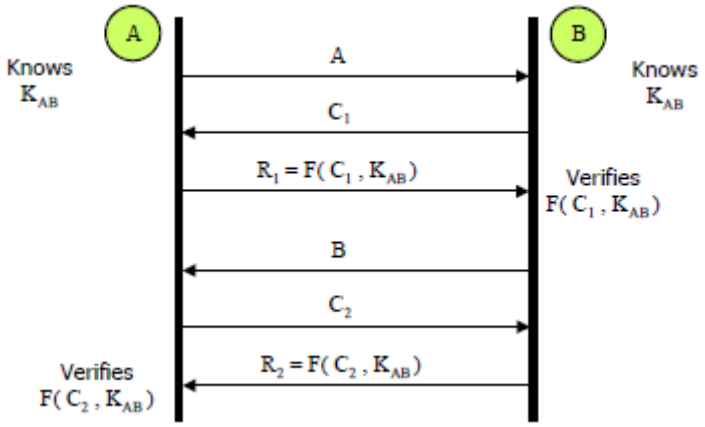
- **<u>אפס ידיעה</u>** - המאזין רואה הרבה שיחות = שלשות של (s,c,r), כאשר s אקראי ב z_p^*, c אקראי ב $0/1$, r אקראי ב $g^r = s \cdot y^c \pmod p$. את אלה בלבד יודע המאזין.
  - **<u>טענה</u>**: כדי לייצר שלשה שעומדת בתנאים אלו לא צריך את x
  - **<u>הוכחה</u>**:
    - בחר r,c אקראי.
    - חשב $s=\frac{g^r}{y^c} \pmod p$.

<u>מסקנה:</u> על כן, כל מספר של שלשות כאלה שמאזין יראה, לא תורם לו כלום. אין דרך להבדיל בין שלשות אמיתיות לשלשות 'מזויפות'. על כן, הקלטת שלשות לא צוברת ידע שלא היה לנו קודם.

19.12.2007

פרוטוקולים לאימות + תאום מפתח

אימות דו צדדי



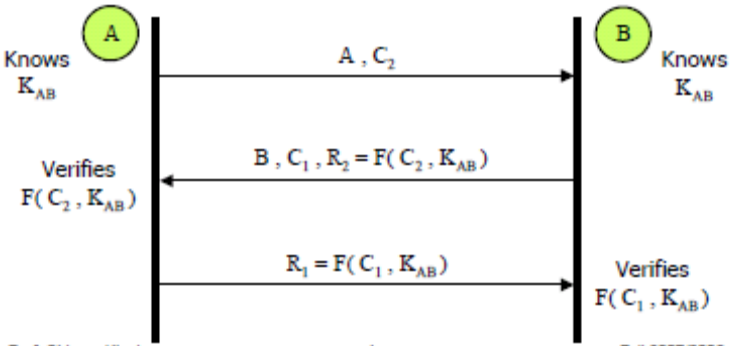
זה בעצם פעמיים אימות חד צדדי.

נשים לב – כאן A מסיים ראשון להוכיח את זהותו.

מי מתחיל? בדרך כלל , מי שיותר רגיש, ירצה שהשני יאמת את עצמו קודם. לכן, בדרך כלל השרת ירצה שהלקוח יאמת את זהותו קודם.

לחילופין, יש מקרים בהם הלקוח יותר רגיש – לדוגמא, כאשר לקוח מדבר מול בנק.

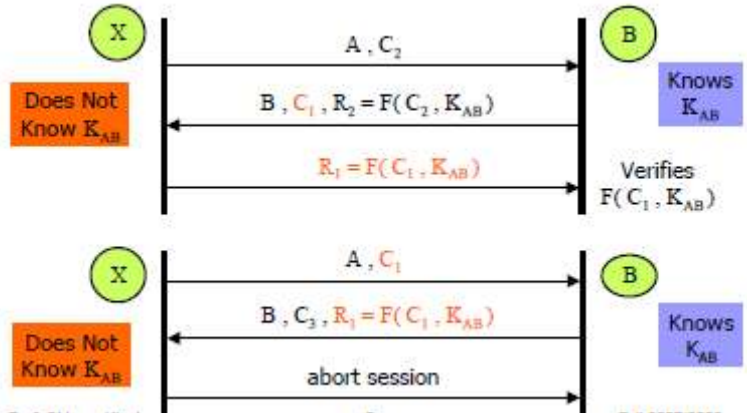
אופציה קומפקטית



בעיות:

1. B שולח את הResponse שלו לA לפני שהוא בטוח שA הוא אכן A. במקרה זה, B מסיים ראשון להוכיח את זהותו.

מתקפה – Reflection attack



מה שקורה כאן בעצם, זה שX משתמש בB בחיבור השני (למטה) כדי להשיג תשובה לSESSION הראשון.

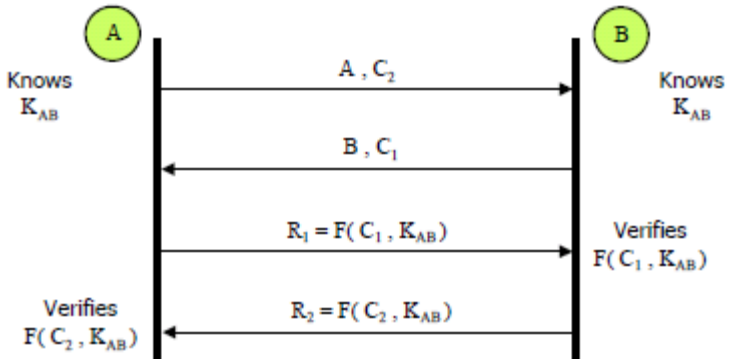
נשים לב: המתקפה מתאפשרת כאשר מדובר בשרת, שאפשר לפנות אליו כמה פעמים. אם הופכים צדדים, אז זה פחות בעיה.

<u>פתרון אפשרי:</u>

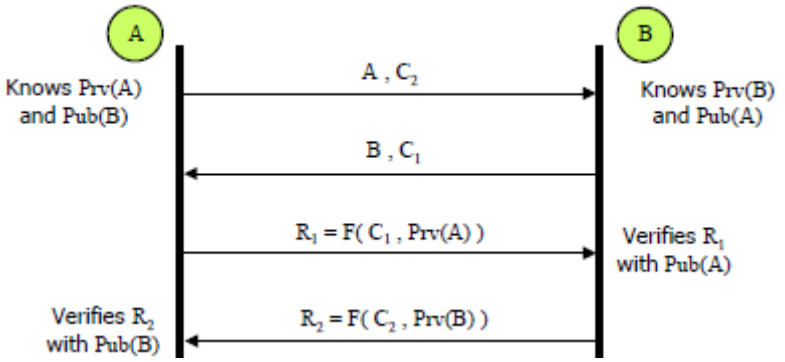
2. להשתמש בשתי גרסאות של f בשני הצדדים.

צמצום שלא מאפשר את המתקפה:

❖ Optimized (4 messages instead of 6 messages), while preserving the order of the authentication:



<u>גרסה א-סימטרית</u>



<u>סיכום אימות</u>

- לא תמיד נצטרך אימות דו צדדי, לפעמים מספיק חד צדדי :
  - לדוגמא, בגלישה באינטרנט – אני רוצה לוודא את זהות השרת ממנו אני קונה. לשרת אין מושג מי קונה ממנו.
  - כאשר אני רוצה לאמת את זהותי מול הבנק (לדוגמא, לעשות פעולות) – אני אאמת את עצמי (בד"כ על ידי סיסמא)

<u>תיאום מפתח</u>

לאחר האימות, נרצה לתאם מפתח, ורק לאחריו – להעביר מידע.

<u>למה צריך?</u>

- לא תמיד יש לי מפתחות המיועדים להצפנה. לפעמים יש לי רק מפתחות פרטיים / ציבוריים, שהם לא נוחים מבחינת קצב עבודה להעברת הרבה מידע. העברת המידע תהיה עם מפתח סימטרי בד"כ, ואותו צריך לתאם.

בד"כ, יהיו לנו מפתחות יחסית קבועים, שישמשו אותנו לפעולות בתדירות נמוכה:

- אימות
- בניית מפתחות שיחה

בנוסף, נרצה מפתחות זמניים, שישמשו לשיחה, ובהם נשתמש בתדירות גבוהה. יש כאן היררכיה של שתי רמות של מפתחות.

<u>החלפת מפתחות בעולם סימטרי</u>

נגדיר - $K_{AB}[i]$ - מפתח בין A,B בזמן i (i בדיד).

נניח כי $K_{AB}[0]$ קיים בשני הצדדים.

סכמה א':

1. העבר את $K_{AB}[i]$ מוצפן על ידי $K_{AB}[i-1]$ אל הצד השני.

סכמה ב':

1. שני הצדדים מחשבים $K_{AB}[i]=h\left(K_{AB}[i-1]\right)$

תכונות:

- חזק קריפטוגרפית
- אם מישהו גנב מפתח אחד, כל העתיד יהיה גלוי בפניו.
- סכמה א' מאפשרת לצד א' ליצור מפתח חלש יחסית, למגננת צד ב'.

סכמה ג' – התקנה פיסית מחודשת

1. מחליפים פעם בX זמן את המפתח פיזית.

תכונות:

- גם העבר וגם העתיד חסוי.

אנחנו נניח שההתקנה היתה פיזית, לצרכי הדיון שלנו.

<u>תאום מפתחות סימטריים בעולם אסימטרי</u>

1. A בוחר K אקראי, ושולח מוצפן לB (באמצעות המפתח הציבורי של B).
   a. B לא יודע שהוא קיבל את המפתח דווקא מA.
   b. פריצה לB נותנת לו מאפשרת לו קריאות לעבר וגם לעתיד.
2. A בוחר K אקראי, שולח מוצפן באמצעות $Pub(B)$ וחתום תחת $\Pr v(A)$.
   a. הערה: לא תמיד יש לA מפתח פרטי.
   b. פריצה לB נותנת לו מאפשרת לו קריאות לעבר וגם לעתיד.

<u>פרוטוקול תיאום מפתח Diffie Hellman מאומת</u>

תשתית:

- p ראשוני גדול
- g יוצר של z_p^*

1. A ממציא x אקראי ושולח g^x , חתום על ידי $PRV(A)$ (כל החישובים הם מודול p כמובן)
2. B ממציא y אקראי ושולח g^y, חתום על ידי $PRV(B)$
3. שניהם מחשבים $k=g^{xy}$ מפתח שיחה.

תכונות:

1. עבר ועתיד חסויים.

<u>העברת סיסמא בין A לB</u>

אופציות:

1. העברת PW_A מA לB ע"ג ערוץ מוצפן ביניהם.
   a. אם מישהו פרץ לB ומצא את המפתח, הוא יכול לחלץ את הסיסמא).

2. פרוטוקול סיסמאות חזק

(מטרה: אימות, בלי להסגיר כלום על הסיסמא, ותיאום מפתח סימטרי)

a. A, B יודעים את $w = h(PW)$

b. A מגריל x אקראי ושולח $E_W\left(g^x \left(\bmod p\right)\right)$ (E פרוטוקול סימטרי)

c. B מגריל y אקראי ושולח את $E_W\left(g^y \bmod p\right)$

d. שניהם מחשבים $k = g^{xy}\left(\bmod p\right)$

e. (מכאן – שלב האימות) -B שולח C_1

f. A שולח C_2

g. A מחזיר $E_K\left(C_1\right)$

h. B מחזיר $E_K\left(C_2\right)$

i. **<u>נשים לב:</u>** השימוש ב E_W לא יוצר פגיעות לכך שהסיסמא היא חלשה, כי מוצפנים איתה רק מספרים אקראיים.

<u>כללי אצבע לשימוש בפרוטוקולי אימות</u>

1. שימוש במפתחות שונים למטרות שונות (הצפנה/ אימות, מידע / מפתח...)
2. שימוש בפונקציות שונות לפעולות שונות
3. תוספת תגים + כיוונים להודעות
4. הוספת nonce להודעות (מונה / זמן / מספר אקראי).
5. שימוש רק בפונקציות אמינות
6. שימוש במפתח / מידע רגיש אחר באופן מינימלי
   a. זמן – לא לשמור אותו בזיכרון לא מוגן מעבר למה שצריך.
   b. מרחב – לא לשמור אותו בזכרון שהוא לא צריך להיות בו.
7. מספרים אקראיים טובים
   a. חומרה ייעודית
   b. תוכנה שמייצרת מספרים פסאודו אקראי (Seed וHASH או פונקציה דומה).

24.12.2007ד

ניהול מפתחות סימטריים

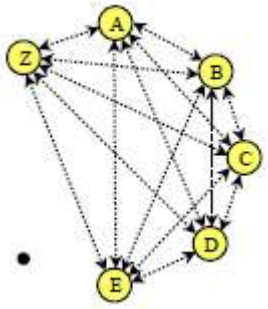
אפשרויות:

1. התקנה פיסית אצל A,B של מפתח k_{AB}
2. העברה באופן מאובטח של מפתח k_{AB} מA לB.
3. גוף שלישי מתקין פיסית אצל A,B מפתח k_{AB}.
4. גוף שלישי מעביר באופן מאובטח מפתח k_{AB} אל A ואל B.

הגדרה: גוף שלישי אמין – TTP – Trusted Third Party.

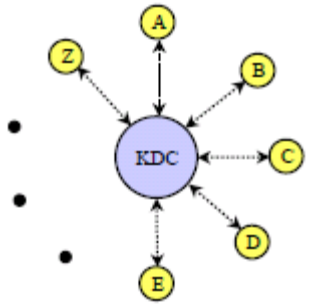
נדבר על KDC – Key disribution center , שזה המנגנון הרלוונטי לעולם הסימטרי.

בעולם בו יש n מחשבים, נרצה שיהיה מפתח $1 \le i \ne j \le n, k_{ij}$. לכן, אם יש בארגון n תחנות, צריך $O(n^2)$ מפתחות תשתיתיים בין תחנות (לא מפתחות שיחה). זה אפשרי כאשר n הוא קטן, ואין צורך בKDC. זה מתחיל להיות קשה - בלתי אפשרי עבור n גדול – צריך לנהל מספר די גדול של מפתחות. היתרון כאן הוא שהמערכת מבוזרת – אין נקודת כשל יחידה – אם מישהו השתלט על A, הוא יכול לראות רק תקשורת נכנסת ויוצאת מA.
במקרה זה, יש גרף מלא – יש מפתח בין כל שניים.



KDC

לכל תחנה $1 \le i \le n$ יהיה מפתח K_i אשר משותף לתחנה ולKDC.



תכונות:

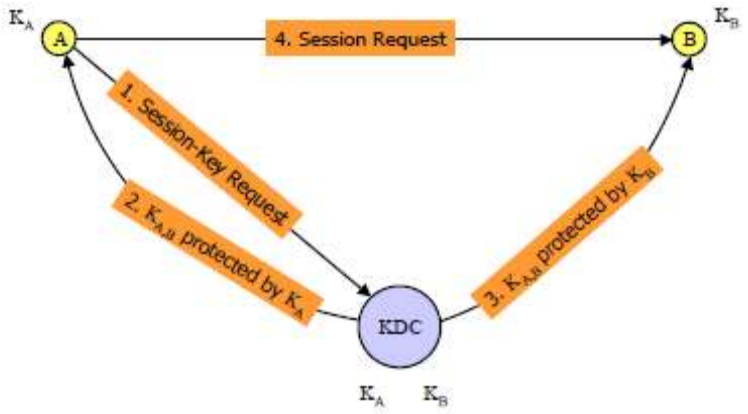
1. ניהול פשוט
2. נקודת כשל יחידה – הKDC
   a. עומס
   b. בטיחות

נניח שA רוצה לדבר עם B (שניהם תחנות בארגון).

1. A פונה לKDC בבקשה לתאם שיחה מול B.

2. הKDC נענה לבקשה של A ושולח לו באופן מאובטח (מוצפן עם K_A) מפתח שיחה עם B - K_{AB}
3. הKDC מעביר לB את מפתח השיחה K_{AB} באופן מאובטח (עם K_B)
4. A וB יכולים ליזום שיחה / לשוחח באופן מאובטח עם K_{AB}.

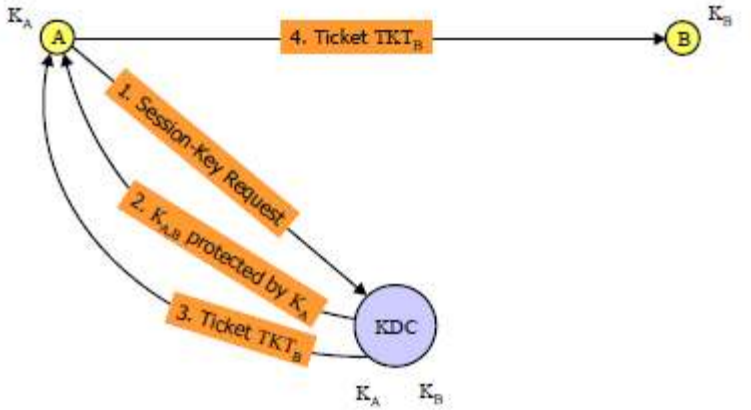
<u>תרחיש Intranet</u>



תכונות:
1. פשוט וברור
2. יש בעיה (לKDC) אם B לא זמין. זה לא תפקידו לרדוף אחרי B
3. לא מתאים לסביבות של שרת / לקוח (A/B)

אפשרי אם B זמין והKDC לא עמוס מדי.

<u>תרחיש אינטרנט</u>



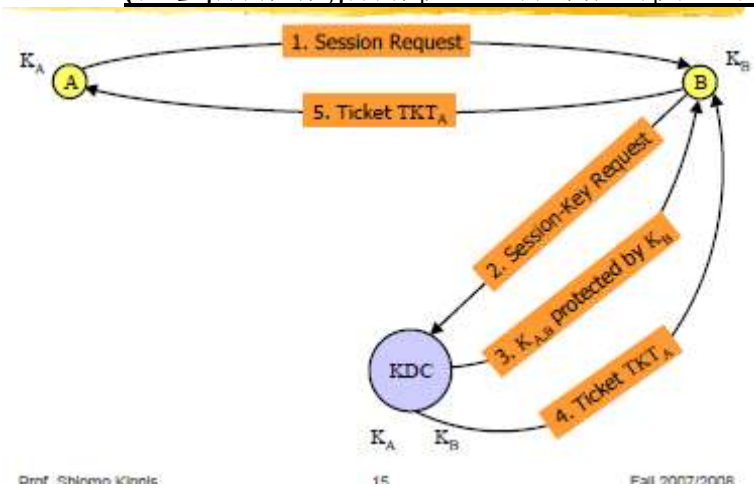
1. A פונה לשירות מול הKDC
2. KDC שולח לA מפתח $K_{A,B}$ מוצפן K_A
3. KDC שולח לA את TKT_B
4. A פונה לB להקמת שיחה עם TKT_B.

מה זה TKT_B ? K_{AB} מוצפן תחת K_B.

תכונות:
1. KDC נותן שירות מיידי ויוצא מהתמונה (כמו שרת טוב)
2. סנכרון בין A לB מובנה לתקשורת A $\leftarrow B$.

תרחיש Extranet

(רוצים רק גישה לשרתי ממשק לארגון, לא לארגון עצמו)



1. A פונה לB בבקשת שירות.
2. B פונה לKDC
3. KDC שולח לB $K_{A,B}$ מוצפן על ידי K_B, ו TKT_A (K_{AB} מוצפן תחת K_A).
4. B חוזר לA עם TKT_A.

יתרונות:

1. אין ללקוח גישה לKDC – רק השרתים צריכים לדעת לדבר עם הKDC.

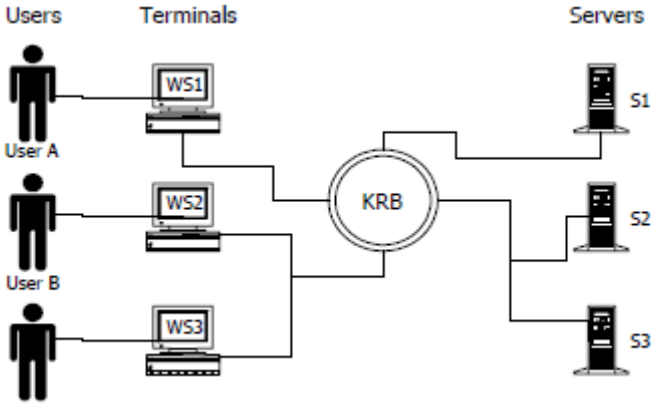
פרוטוקול KDC טיפוסי – אינטרנט / אינטראנט

1. A רוצה לפנות לB. הוא פונה לKDC - שולח A,B,N_1 (Nonce = N) (נשים לב – הוא לא הוכיח את זהותו – אפשר בגרסה אחרת)
2. KDC עונה לA עם $A,B,E_{K_A}\left(A,B,K_{AB},N_1\right)$
3. KDC שולח לA את $TKT_B = E_{K_B}\left(A,B,K_{AB}\right)$
4. A פונה לB:
   a. $A \to B : A,B,TKT_B,N_2,E_{K_{AB}}\left(N_2\right)$
5. אופציונלי – B מוכיח לA את זהותו.
   a. $B \to A : A,B,N_3,E_{K_{AB}}\left(N_3\right)$

26.12.2007

– Kerberos

1. שנות ה80
2. פותח בMIT
3. לחוות של תחנות עבודה

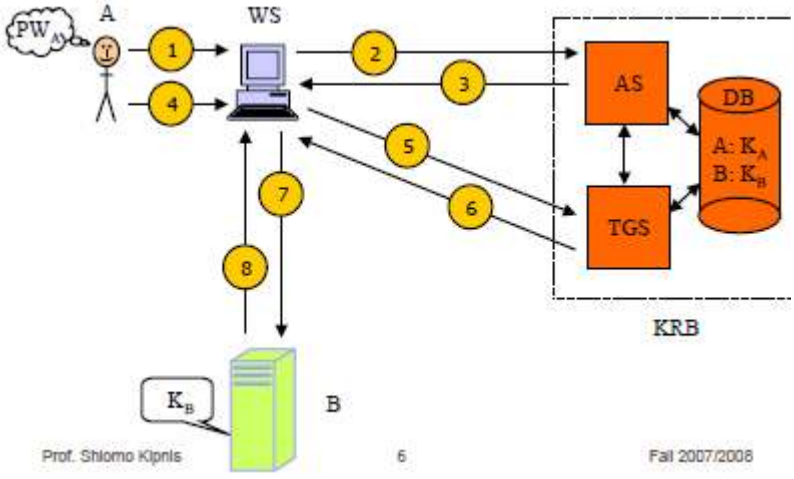


מכילה:

1. Authentication
2. (היתה אמורה להכיל:
   a. Authorization
   b. audit

והיא היתה אמורה לאפשר Single Signon – כלומר, אתה מתחבר פעם אחת לכל השירותים , ולא כמה פעמים. הרעיון הוא לבצע Login לרשת כולה, וזאת על ידי שרת הKerberos, שהוא אחראי לכל הLoginים ברשת. כיום תמיכה בו היא נהפכה די סטנדרט.

כיום יש שתי גרסאות מסחריות – 4 ו5. נדבר בעיקר על V_4.



תצורת העבודה:

1. למשתמש A יש סיסמא, PW_A , ויש מפתח מול A, K_A
2. לשרת בארגון, B, יש מפתח K_B

החצים(על פי מספר)

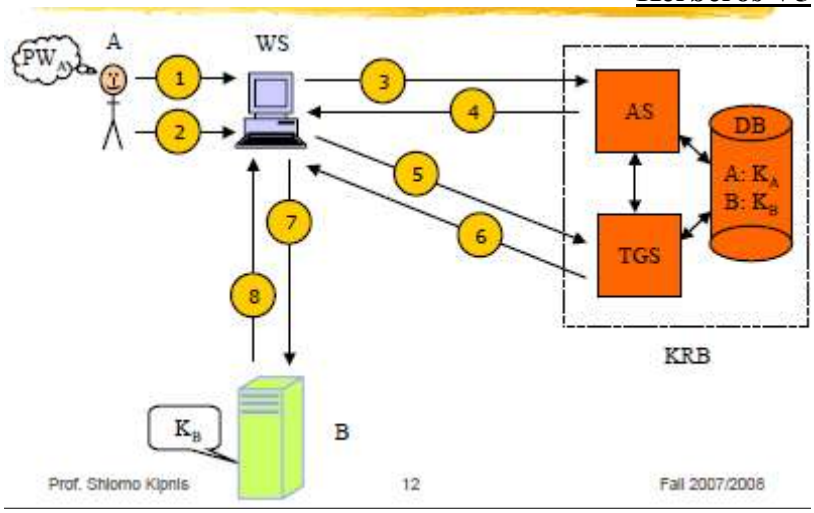
שלבים 1 – 4, מול הAS – Authentication server – אותנטיקה לרשת.

שלבים 5 – 8, מול ה TGS – Ticket Granting Server– יצירת קשר מול שרת.

1. המשתמש אומר לתחנת העבודה (WS) את שמו. מעכשיו נדבר על האובייקט $A+WS$ יחדיו.

2. A+WS פונים לAS ושולחים $\langle A, WS, TGS, TS_1 \rangle$ (TS_1 = Time Stamp 1, TGS = שמו של TGS). אין כאן הוכחת זהות, יש כאן פרטים מזהים.
3. AS עונה ל $A+WS$ עם NC_{A+WS}. NC = Network Credentials.
כאשר $NC_{A+WS} = E_{K_A}\left(\langle K_{A,TGS}, TGS, TS_2, LT_2, TKT_{TGS} \rangle\right)$
$TKT_{TGS} = E_{K_{KRB}}\left(\langle K_{A,TGS}, WS, TGS, TS_2, LT_2 \rangle\right)$. LT_2 = כמה זמן תקפים הCredentials שלנו. K_{KRB} = מפתח בין שרתי הKERBEROS השונים. $K_{A,TGS}$ = מפתח שיחה שטוב מעכשיו למשך LT_2 זמן.
4. $A+WS$ מפענחים ושומרים את NC
   a. A מקליד סיסמא PW_A
   b. WS מחשב $K_A = h(PW_A)$ ומוחק את PW_A מהזכרון (למרות שההבדל לא מהותי, הסיסמא יותר רגישה מ K_A).
   c. WS מפתח את NC_{A+WS} ושומר את השדות הרלוונטיים + TKT. האותנטיקציה כאן היא בזה שהצלחתי לפענח משהו בעל משמעות. מוחקים את K_A. המידע הרגיש שיש כרגע רלוונטי רק למשך LT_2.
5. A+WS פונים לTGS בבקשת שירות מול שרת S, ושולח $\langle S, TKT_{TGS}, auth_3 \rangle$ כאשר $auth_3 = E_{K_{A,TGS}}\left(\langle A, WS, TS_3 \rangle\right)$.
6. TGS עונה ל $A+WS$ עם $SC_{A+WS,S}$ (Service Credential). נשים לב – אם אסור למשתמש A לגשת לשרת S, הוא לא יחזיר תשובה מן הסתם.
כאשר $SC_{A+WS} = E_{K_{A,TGS}}\left(\langle K_{A+WS,S}, S, TS_4, TKT_S \rangle\right)$
$TKT_S = E_{K_S}\left(\langle K_{A+WS,S}, A, WS, TS_4, LT_4 \rangle\right)$. WS מפענח את השדות הרלוונטיים.
7. A+WS פונים לS עם $\langle TKT_S, auth_5 \rangle$ כאשר $auth_5 = E_{K_{A,S}}\left(\langle A, WS, TS_5 \rangle\right)$.
8. S מפענח את הTKT, משווה מול השדות של $auth_5$, ועונה $auth_6 = E_{K_{A,S}} = (S, TS_6)$

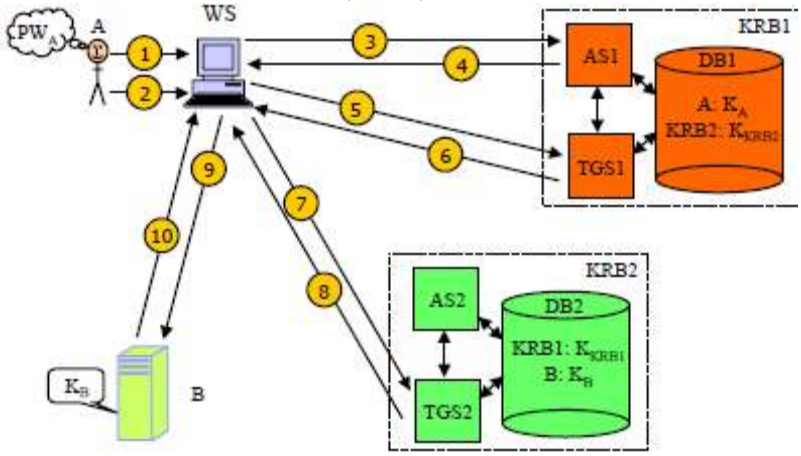
Kerberos V5



שינויים מול גרסה קודמת:

- אימות משתמש מוקדם (כבר בשלב ראשון)

- אי הצפנה כפולה של TKT – הTKT היה מוצפן פעמיים, והיה מועבר לא מוצפן (כי הוא כבר מוצפן בעצמו).
- PostDating, Proxy, Forward
  a. Forward – להעביר זכות שימוש מתחנה לתחנה / ממשתמש למשתמש.
  b. Proxy – לעבוד עם תחנה שמדברת איתי בשם תחנה אחרת.
  c. PostDating – לעבוד במשך הרבה זמן (שבועיים לדוגמא) עם אותו login. הרעיון הוא איפשור הנפקת כרטיס לתאריך עתידי, או כרטיס שניתן לחדשו.
- Realm – 'יקומים'
  a. יש לכל ארגון שירותי Kerberos משלו.
  b. KERBEROS 5 מאפשר ליצור קשר בין עולמות שונים

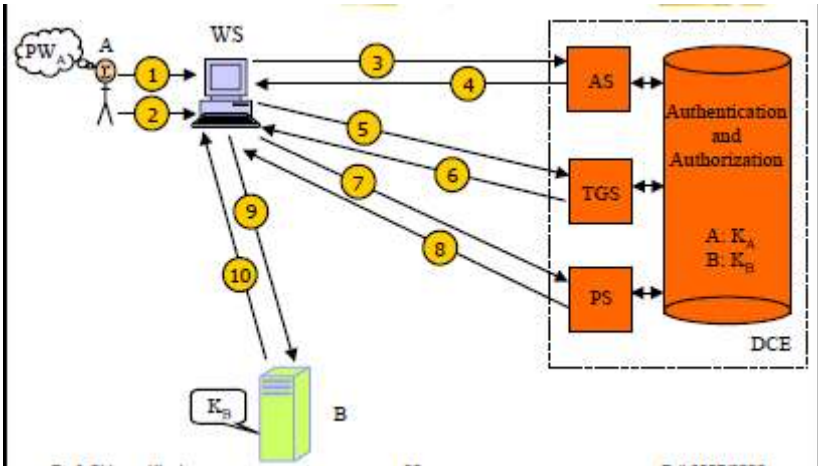


  c.
  d. הרעיון הוא שA מבקש מהTGS שלו מפתח לTGS של Kerberos2. כמו כל שרת אחר... לא פונים ל AS_2 , כי אי אפשר לעשות LOGIN לשם.

-

DCE

הרחבה למטרות הרשאות



יש כאן שרת נוסף בשם PS, Privilege server, שנותן לך רשימת הרשאות לשרת אליו אתה מבקש גישה.

31.12.2007

ניהול מערכות עם מפתחות אסימטריים

N משתמשים. לכל משתמש i יש $PUB(i), PRV(i)$.

סוגיות:

1. בניית מפתחות
2. הפצת מפתחות
3. ביטול מפתחות

בניית מפתחות

אפשרויות:

1. A בונה מפתח פרטי / ציבורי. שומר את הפרטי, ומפיץ (נלמד בהמשך) את הציבורי.
   a. בעיה – צריך כושר חישובי מסוים.
   b. מתאים לעולם בו אני סומך רק על עצמי.
2. A פונה לאחד מהרבה שרתים, שמסוגלים לבנות מפתחות ציבוריים / פרטיים. השרת נותן לA את הפרטי ושוכח אותו.
3. יש שרת אחד מיוחד (TTP) לארגון שרק הוא בונה מפתחות פרטיים /ציבוריים. הוא נותן את הפרטי לA (ולא בהכרח שוכח אותו).
   a. למה לא לשכוח? לצורכי גיבוי / ארכיון /... . מתאים לעולם אינטראנטי.

הצפת מפתחות (ציבוריים):

1. מפיצים:
   a. בעל המפתח (A)
   b. שרת הפצה
2. צורת התקנה
   a. התקנה פיזית
   b. התקנה באתרי אינטרנט מסוימים (כמו דפי זהב)
   c. שליחה לכל דורש לemail / מסמך / ...
      i. בעיה – איך אפשר לדעת שזה אני זה ששלחתי? Man in the middle יכול להחליף. זה רלוונטי גם לסעיף b (בפרסום / בקבלה).
         1. פתרון אפשרי – קבלת אותו מפתח מריבוי מקורות מקטין את הסיכוי שהוא מזויף.
   d. גישה לשרת TTP מיוחד והורדה ממנו.

ביטול מפתחות (פרטי כמובן)

1. למה?
   a. נגנב
   b. פוצח
   c. פג תוקפו
      i. עזבתי את הארגון
      ii. הזמן שהוגדר למפתח
2. איך?
   a. על ידי A
   b. על ידי שרת
   c. משתמש שרוצה לשוחח עם A צריך לבדוק את תוקף $PUB(A)$

תעודות / אשרות – Certificates

מה המטרה? לוודא שהמפתח הציבורי של A הוא אכן של A.

מה מכילה תעודת מפתח ציבורי:

1. שם

2. זיהוי
3. המפתח הציבורי שלי
4. חתימה.

נשים לב – בד"כ לא פונים לשרת החותם כדי לוודא (כמו שלא מתקשרים למשרד החינוך לבדוק שתעודת בגרות היא אותנטית). סומכים על החתימה של הארגון.
נשים לב – אני חייב להכיר את החותם (תעודת בגרות של משרד החינוך הזימבבואי לא ניתנת לוידוא על ידי).

מה הישויות הרלוונטיות:
1. Certificate Authority – הרשות שמנפיקה תעודות.
2. ישויות A,B,C בעלות מפתח ציבורי ופרטי.
3. מאגר מידע המכיל:
   a. $A:Cert(A),B:Cert(B),...$
   b. מה זה $Cert(A)=\underbrace{\langle A,Pub(A)\rangle}_{\text{signed by CA - using Prv(CA)}}$

איך מוודאים מסמך חתום בתצורה זו? באמצעות $Pub(CA)$.

איך יותקן את ה$Pub(CA)$? כנראה פיזית. אין ברירה. מן הסתם נרצה מפתח מאוד מאוד חזק שם.

<u>Certificateים - איך</u>
<u>בניית מפתחות</u>
1. A בונה Pub/Prv
2. שרת פונה Pub / Prv
3. שרת שקשור לCA יבנה את הPub/Prv

נשים לב – זה חשוף לMan in the middle.
ואז...
<u>בניית Cert</u>
4. A פונה ל CA (או לשרת שקשור אליו) ומבקש Certificate
   - איך?
     - באמצעות חברות שעושות את זה בתשלום (Verisign לדוגמא). איך מתקשרים איתם?
       - אימייל
         - זה חלש מאוד מן הסתם, והCA יתן בד"כ Certificate שמסומן כחלש.
       - אימות פיזי חזק
         - ואז מן הסתם יתקבל Certificate חזק יותר.

<u>הפצת Certificate</u>
5. בד"כ לא CA, אלא:
   - שרת שמחובר לCA
   - או A בעצמו
   - נשים לב – זה לא פעולה רגישה.

<u>ביטול Certificate</u>
- למה?

- גניבה / פריצה / ... של הPrivate
- עזיבת ארגון...

מי עושה?

1. לא A (לא חתום)– כי אחרת פתחנו פתח לDenial of service מטורף.
   a. כן בסדר אם A חתום (כלומר, השתמש במפתח הפרטי שלו המתאים למפתח הציבורי בCertificate).
2. A פונה לCA באופן חתום ומבקש זאת , והCA חותם על ביטול.
3. CA יוזם את זה (כי הוא לא סומך יותר על A) וחותם על זה.

כאשר הCA מבטל Certificate הוא בד"כ מפרסם את המבוטלים אחתלX זמן. הדבר מתבצע באמצעות מנגנון הפצת ביטוליים, CRL = Certificate Revocation List.

B שצריך לשוחח עם A צריך לבדוק את הCRL המתאים.

<u>פורמט X509 Certificate</u>

1. #Ver - כיום 3, עוברים ל4.
2. Serial number – מספר סידורי אצל המנפיק.
3. Signature Param – לדוגמא, אם הCertificate חלש / חזק, עם המפתח הוא המפתח הסודי יותר / פחות, ... – טכניקת החתימה של הCA.
4. Certificate Issure – מנפיק.
5. Not Before – ממתי תקף
6. Not After – עד מתי תקף
7. Subject Details – שמו של A (שעבורו אנחנו חותמים). בד"כ שם על פי X509.
8. Subject Public Key – המפתח הציבורי של A.
9. Extensions – מידע נוסף לגבי A (לדוגמא, כתובות פיזיות של A, מספר טלפון, ...). מכיל:
   a. שמות אלטרנטיביים.
   b. שימוש של המפתח הציבורי הרלוונטי– לדוגמא, להצפנה / אותנטיקציה / חתימה /...
   c. CRL Distribution Point – איפה יהיה רשום אם הCertificate יתבטל.
10. Signature – חתימה באמצעות PRV(CA) באמצעות האלגוריתם והמפתח הציבורי המתוארים בSignature Params. החתימה חלה על הכל חוץ מהחתימה.

9.1.2008

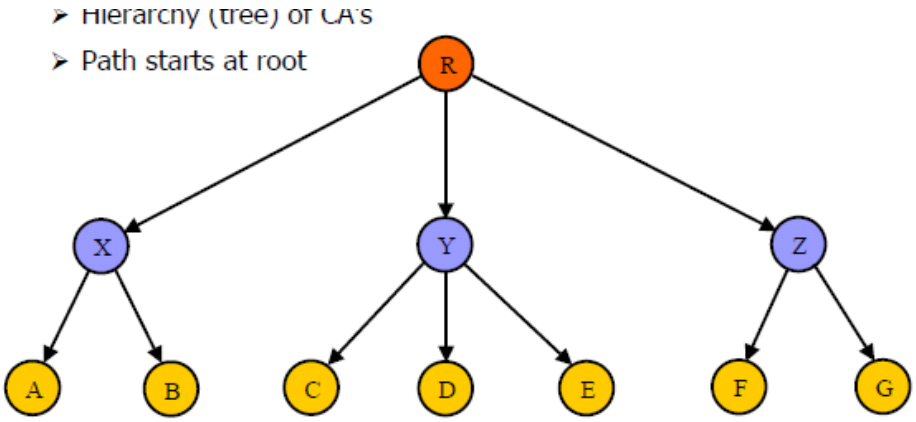
נדבר היום על:

- ניהול אמון במערכות PK – Trust management
- תשתיות PK (PKI)

המציאות היא שאין CA אחד לכל העולם. בנוסף, רוצים CA לתחומים שונים.

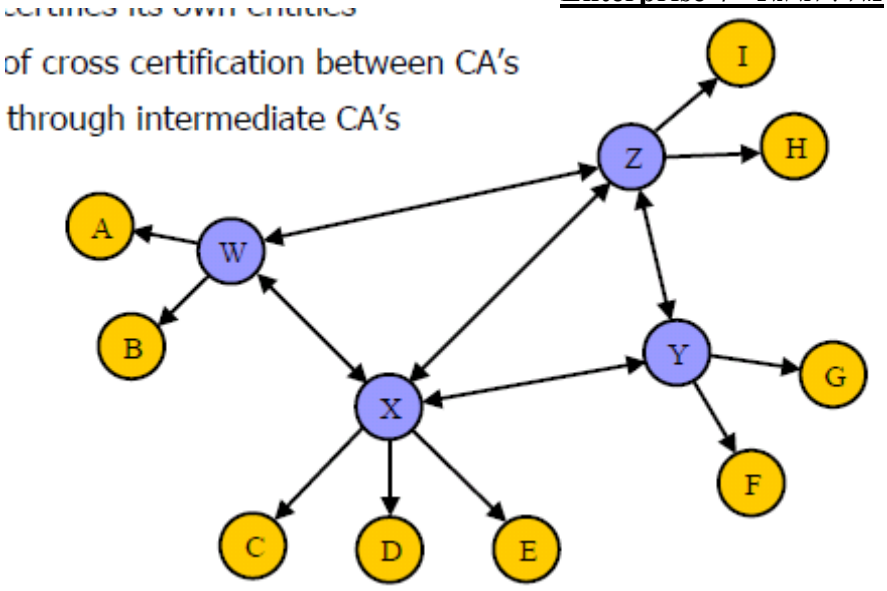
מה האופציות?

מודל היררכי



- לכל עלה יהיה PK של הRoot.
- כל CA מנפיק תעודות לבנים שלו.
- אם A רוצה לדבר עם F, לקבל את הPK שלו, נצטרך $R\langle\langle Z\rangle\rangle, Z\langle\langle F\rangle\rangle$ כאשר $\alpha\langle\langle\beta\rangle\rangle$ - α מנפיק תעודה עבור β. כל שרשרת כאן מתחילה בשורש. לזה נקרא מסלול Certificate.
- המודל לא באמת הושמש בעולם. מושמש בתוך ארגונים.

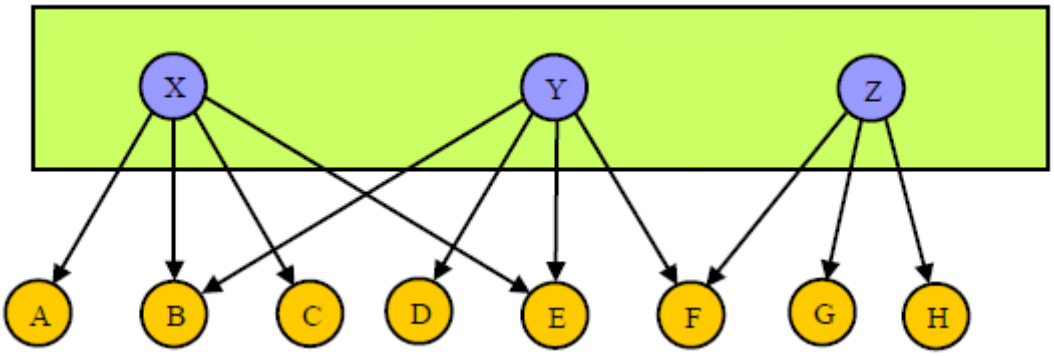
מודל ארגוני / Enterprise



- יש ארגונים , שלכל אחד מהם יש CA (X,W,Y,Z). יש ביניהם יחסי trust (לא בהכרח בין כולם לכולם).
- כל ארגון מנפיק תעודות ללקוחותיו. בין הארגונים יש cross Certificate

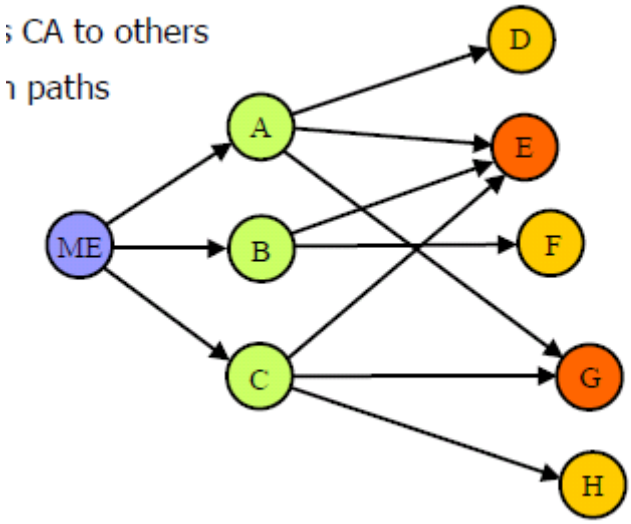
- אם A רוצה להגיע לD, הוא יצטרך $X\langle\langle D\rangle\rangle, W\langle\langle X\rangle\rangle$ (או מסלול אחר, אם קיים – יכולים להיות כמה מסלולים). כאן צריך מנגנון של Path Exploration – למצוא את המסלול , ואת הטוב ביותר מביניהם (לא כל המסלולים נולדו זהים).
- Cross Certificate בין שני CA
  - מכילים בד"כ אינפורמציה על סוג הקשר בין הCA.

<u>מודל שטוח</u>



- יש קבוצת CAים <u>אצלי</u>(בירוק למעלה) – אני מכיר כמה CAים שאני סומך עליהם, וכל CA, מנפיק תעודות עבור לקוחותיו.
- בשימוש ב:
  - בBrowserים באינטרנט (כאן כל העלים הם האתרים). הם מותקנים מראש בBrowser עם התקנתו.

<u>מודל אישי</u>



- הרשת לעיל – Web of trust
- בשימוש בPGP.
- אני מרכז העולם (ושומר את רשימת המפתחות של מי שאני סומך עליהם).
- אני מחליט אם להאמין במישהו שנמצא לא ברשימה שלי, אם הוא מספיק קרוב.
- אני קובע מי בסביבת האמון שלי.

<u>תשתיות Public Key (PKI = Public Key Infrastructore)</u>

מה זה תשתית?

- שמושי באופן נרחב
- נגיש

- פשוט
- אמין
- סטנדרטי
- שקוף

<u>שירותי PK</u>

- רשות המדיניות - מדיניות הארגון (נקרא PA)
- רשות cert – הCA – בונה Certificate וחותם. מחזיק את המפתח הפרטי.
- רשות רישום (RA) – Registration Authority. אליו פונים כשרוצים תעודה. הוא אחראי לוודא שאני זה מי שאני מתיימר להיות. הוא יפנה לCA ויבקש לחתום.
- מאגר cert – הDB ששומר אותם.
- גיבוי מפתחות – בעיקר פרטיים. כמובן שחשוב לשמור את זה היטב.
- אחזור מפתחות (פרטיים)– תהליך ההוצאה מהגיבוי.
- עדכון מפתחות (פרטיים) -
- ארכיב (מפתחות פרטיים) / היסטוריה – לדוגמא לשם וידוא חתימה על מסמכים שנחתמו בעבר.
- ניהול הCertificates

<u>מנגנוני ביטול Cert / החייאה של Cert</u>

<u>גישה א':</u>לפרסם רשימה שחורה של Cert מבוטלים (CRL = Certificate Revocation List). יותר בשימוש.
<u>גישה ב':</u> לפרסם רשימה של Cert פעילים (CAL = Certificate Activiation List)

<u>(X509) CRL</u>

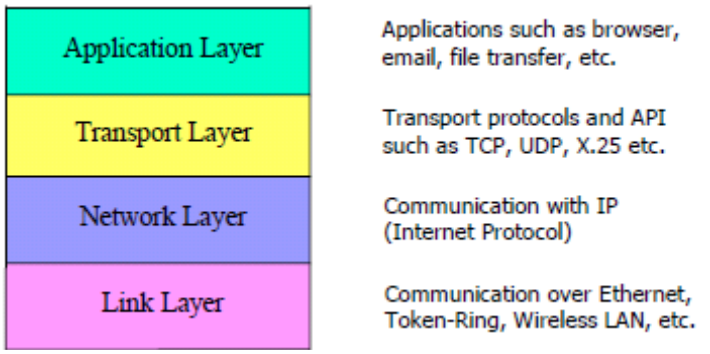
Version
Signature Parameters
Issuer
Issue Date
Next CRL Update
.....   <u>List Of Revoked Certificates</u>   Serial Number and Revocation Date   .....
Extensions
Signature

1. Complete CRL – הCRL מכיל את כל רשימת המבוטלים (עד שפג תוקפו של הCertificate).
2. Distribution Points – יש פיזור של הCRL – לדוגמא – כל אלו שמתחילים ב1, בשרת מסוים, וכיו"ב. או שבCertificate עצמו רשום איפה יהיה רשום אם הוא יבוטל.
3. Delta CRL – רשימה מלאה יוצאת פעם בX זמן, וכל $\frac{x}{n}$ מוציאים delta. במקרה זה – בשביל לברר צריך את הcomplete האחרון, ואת כל הדלטות מהאחרון.

בכל מקרה, הCRL צריך להיות חתום.

14.1.2008

אבטחה ברמת הרשת – IPSEC



באיזה מקום נוסיף את האבטחה?

יש יתרונות וחסרונות לכל מקום. כמובן שאבטחה ברמה הפיזית לא תספיק כדי לאבטח קניות אינטרנטיות.
החסרון של לעבוד ברמת האפליקציה הוא שכל ישום צריך לממש את זה מחדש.

תכונות IP

1. חד כיווני (שולח / מקבל)
2. לא אמין - לא מוודא קבלה – best effort, שגר ושכח
3. כתובות IP V4 (32 ביט). לעיתים חסרות משמעות.
4. סדר הגעה של חבילות לא מובטח (בשל ניתוב שונה).
5. קל להאזין / לשתול חבילות IP.

שנות ה90 – התחילו לפתח את IPSEC

1. זיהוי מקור
2. אימות מידע
3. סודיות מידע
4. שמירה על סדר
5. מניעת replay
6. הרשאות גישה
7. פרטיות מסוימת
8. תאימות מלאה עם IP (נתבים יוכלו לנתב בלי להבין IPSEC).

IPSEC

מסמכים:

Arch - ארכיטקטורה
Authentication – ***AH***
הצפנה – ***ESP***
קודי שגיאות וכיו"ב. – ***DOI***
Internet Key Exchange – מנגנון החלפת מפתחות. – ***IKE***

ארכיטקטורה

- שולח ומקבל מחזיקים מבנה נתונים בשם Security Association – SA. לדוגמא, מפתחות הצפנה, מספרים סידוריים. הם צריכים לתאם את הSA בדרך פיזית / IKE.
- שולח / מקבל מחזיקים כמה מאגרים:
  - Security Asociation DB – SAD (SA מול כל יעד/ מטרה)
  - SPD - מדיניות.

יש לIPSEC שני מצבים:

Transport Mode

לקחת את חבילת הIP המקורית, לשנות טיפה את הHEADER, והמידע נשאר כמו שהוא.
כאן מגינים על החבילה לא כולל הHEADER.

Tunnel mode

עוטפים את כל החבילה המקורית (כולל הHEADER) בHeader חדש. כאן מגנים על כל החבילה המקורית, כולל הHEADER שלה (וכך משיגים גם פרטיות).

מבנה חבילת IP

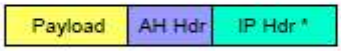
Ver	Hdr Len	Type of Service	Total Length
Identification		Flags	Fragment Offset
Time to Live	Next Protocol	Header Checksum	
Source IP address			
Destination IP address			
Options			Padding
Payload			

1. Ver – גרסה
2. header len – בשל הoptions
3. tos – למען QOS.
4. total len – כולל הPayload.
5. fragment offset – האם התבצעה פרגמנטציה?
6. TTL – כדי שהחבילה לא תסתובב בלי הפסקה בעולם .
7. next protocol – איזה פרוטוקול מכיל הpayload.

AH – פרוטוקול האימות

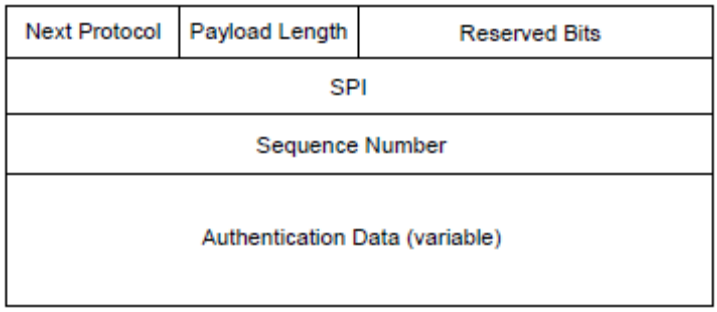
Original IP packet:

Transport Mode AH packet:



Tunnel Mode AH packet:

מה יש בAH Header?



1. Security Parameters Index – SPI – מציין באמצעות איזה SA מטבלת הSAD של המקבל נארזה החבילה .
2. Sequence number –מספר סידורי של חבילה בזרם המדובר.
3. Auth Data – הקוד המאמת של החבילה.
4. Next Protocol – כי הNext protocol המקורי של IP נהפך לAH. לכן זה הNext protocol של המידע. נשים לב – אם מדובר בTunnel, הnext protocol הוא IP – כי זה מוזרם לשכבת הIP בחזרה.

<u>מה מאמת הAH? את כל הPayload, כולל הHEADER של IP (שלפניו), והHEADER שלו (חוץ מהAuth Data עצמו, ודברים שעלולים להשתנות בדרך).</u>

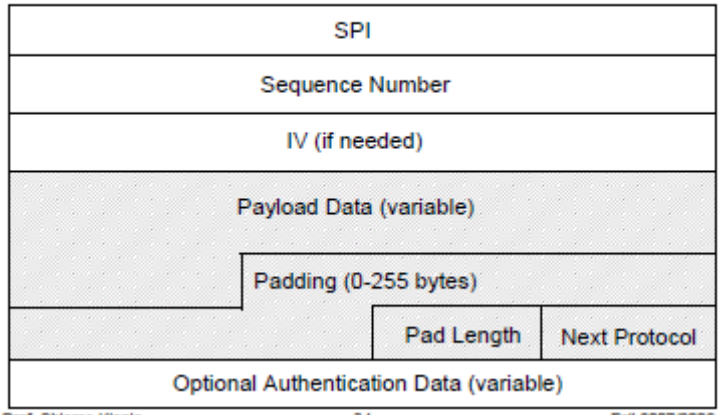
Encapsulated Security Payload – **<u>ESP</u>**

Original IP packet:

Transport Mode ESP packet:

Tunnel Mode ESP packet:

The ESP Header + Payload + Trailer:



- SPI – אינדקס של SA (אצל המקבל) בתוך הSAD.
- Seq num - כמו קודם
- IV – למטרות הצפנה

- Payload – בתלוי בתצורת עבודה (המידע / כל החבילה המקורית)
- next protocol – כמו קודם.
- Padding – גם כדי להסתיר את הגודל המקורי, גם בשביל Block size (IP או הצפנה).
- Auth data – הוא נותן גם אותנטיקציה... ☺ (למרות שיש AH שאפשר להשתמש בו בנפרד). אבל היא לא חזקה כמו AH. זה כי היא לא חלה על הHeader.

16.1.2008

'האם אנו חוזים בסופה של האקדמיה בישראל?'

Security Association – SA

מה הוא מכיל?

- מאונדקס עם SPI – 32 ביט שנקבעים על ידי הנמען (מזהה של הSA).
- כתובת מקור
- כתובת נמען
- פרוטוקול (AH / ESP)
- תצורת פעולה (Transport / Tunnel)
- מס' סידורי בזרם הנוכחי (32 ביט)
- חלון Anti Replay.
- Life time
  - זמן (או)
  - כמות חבילות

מתואם על ידי IKE (Internet Key Exchange) או ידנית.

מנגנון מניעת Replay

בעיקר אחריות של המקבל (אבל גם של השולח, שצריך לתחזק מספר סידורי). המקבל מתחזק **חלון** של Sequenceים שהוא מצפה לקבל.

הוא מסמן לעצמו איזה חבילות (Sequence) כבר התקבלו.

עבור הודעה שמתקבלת, יתכנו 3 אפשרויות – היא בתוך החלון של הSequenceים המצופים, לפניו / אחריו.

אם היא לפניו – נזרוק.

בתוך החלון – אם כבר נתקבל – נזרוק. אחרת, אם היא תקינה (מבחינת אימות / הצפנה – AH / ESP), אז מקבלים אותם, ומסמנים.

אם היא אחריו – בודקים את AH / ESP – דוחים את החבילה.

אם תקין – מקבלים, ומזיזים חלון בהתאם (**שהיא תהיה אחרונה בחלון**).

למה זה מונע Replay? ברור.

נשים לב: אם יתבצע Re-transmission ברמה הגבוהה יותר (TCP), אזי זה יהיה Sequence number חדש מבחינת IPSEC.

Security Association DB – SAD

Security Policy DB – SPD

SAD - טבלה של SA. רלוונטית בעיקר בצד המקבל.

SPD – טבלת מדיניות. רלוונטית בעיקר בצד השולח .

SAD – טבלה מSPI (אינדקס) לSA. נזכור – המקבל מחליט עם איזה SA לעבוד.

SPD – טבלה של מדיניות (בעיקר של השולח) – מכילה:

1. מצביע לSA
2. rule – מה3 הבאים:
   a. drop – אל תיתן לחבילה לעבור.
   b. pass – תעביר כמו שהיא (בלי IPSEC)
   c. apply – תפעיל IPSEC. כאשר זה מסומן, חייב להיות מצביע לSA.
3. selectors
   a. src
   b. dest
   c. destination port (נשים לב – של TCP / UDP!!! – יש כאן התבוננות ברמות מעל)

d. source port

המעבר על הטבלה הוא מלמעלה למטה – כאשר החוק הרלוונטי הוא הראשון שנתפס (בד"כ האחרון יהיה all).

למה יש אפשרויות לכמה Srcים שונים: אם הGateway ממש Ipsec.

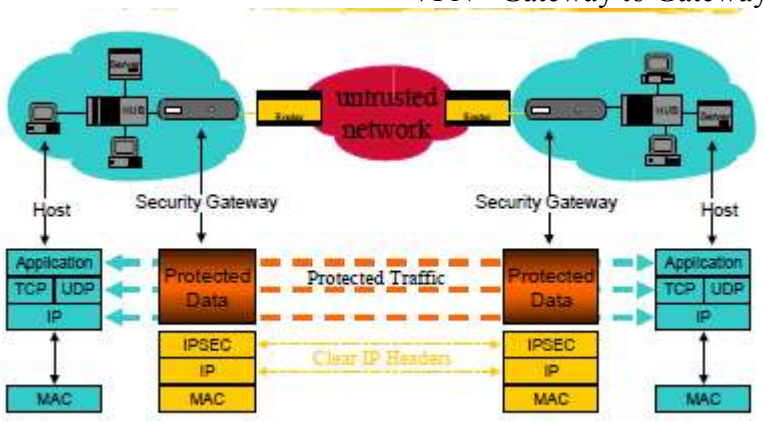
יתכן וגם הצד המקבל יממש SPD (אם הוא Gateway).

צורות ישום IPSec

Point to point

1. יעבוד בד"כ בתצורת Transport (לא כל כך עוזר – כי הוא בעיקר מתמים את ה src / dest, אבל זה בכל מקרה ישאר אותו דבר).

VPN - Gateway to Gateway



1. תצורת tunnel. מספק פרטיות.
2. נשים לב – אם ברשת לדוגמא יש אותנטיקציה בין כל התחנות(בTransport כמובן), אז מעל זה יהיה את ההצפנה / אותנטיקציה של הVPN.

סיכומון לגבי IPSEC

- בIP אין הצפנה / אותנטיקציה, לכן יש IPSEC.
- שקוף יחסית לרמות שאינן IP.

IKE

Phase 1 – תיאום SA של IPSEC – פעם אחת בזמן רב (בין כל זוג)

- תיאום SA
- תיאום מפתח (DH)
- אימות הצדדים (אסימטרי)
- הסתרת זהויות
- העברת Certificates

Phase 2 – תיאום SA של השיחה (כל זרם). – כל זרם.

- תאום מפתח זרם
  - סימטרית / אסימטרית (בהתאם לתכונות שאנו רוצים).

21.1.2008

אבטחה ברמת Transport

- אבטחת שיחה (TCP)
- דו כיווני
- אימות מקורות (?)
- סודיות תוכן
- אימות תוכן
- שמירה על סדר / אי משלוח חוזר / שליחת חבילות אבודות – מתבצע ברמת הTCP

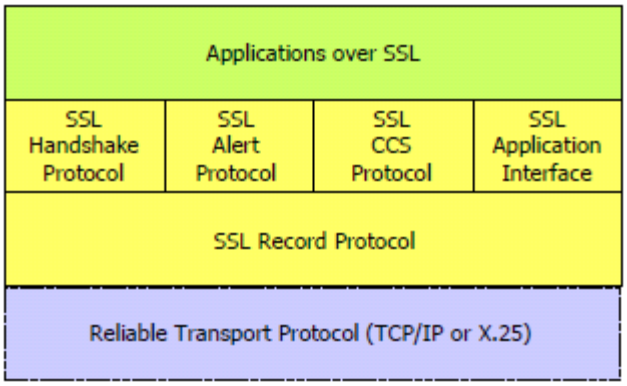
SSL

- מיועד לE-Commrence
- הלקוח יוזם שיחה מול השרת
- יש הרבה לקוחות מול שרת. הלקוחות מזדמנים.
- אי אפשר לצפות מלקוח להחזיק Certificate (ולכן הוא לא יאומת קריפטוגרפית – יאומת אפליקטיבית מעל הSSL בד"כ). יש מוד של SSL (לא נפוץ) שהלקוח גם מאומת באמצעות Certificate, לא מתבצע בישומים אינטרנטיים.
- לשרת יש Certificate (ולכן הוא מאומת קריפטוגרפית) – לכן האימות ברמת הSSL בלבד הוא של השרת, לא של הלקוח.

קיצורים: SSL = Secure Socket Layer. TLS = Trans Layer Security.

כיום משתמשים בSSL V3 או בTLS 1.0 (TLS מאוד דומה לSSL V3 , אבל יצא על ידי גוף התקינה, IETF, כאשר SSL V3 הוא של Netscape).

ארכיטקטורה:



כלומר –מבחינת האפליקציה, SSL הוא שקוף – הוא מעל רמה 4, ונותן את אותם שירותים שTCP נתן, רק מאובטח.

למה החץ מApplication לTCP ? כי אפשר להשתמש בTCP ישר, לא חייבים לעבוד בSSL.
SSL Alert – מטפל בהודעות (שגיאה, אינפורמטיביות, ...). לדוגמא – אם Certificate לא בתוקף.
Handshake – תאום והקמת השיחה המאובטחת.
CCS – פרוטוקול סינכרון.
כל זה רץ מעל הSSL Record.

מה קורה בהתחלת שיחת SSL?

1. התחלת שיחת TCP.
2. מתנהלת שיחת TCP רגילה ולא מאובטחת.

3. כאשר רוצים לבקש פעולה מאובטחת, יוצרים שיחת SSL (באמצעות HandShake). מה הטריגר לכך? בWEB, לינק.
4. מעל זה מתחילה שיחה מאובטחת.
5. בסיום, חוזרים לשיחה לא מאובטחת. (אפשר לחזור לשיחה מאובטחת שוב אם רוצים).
6. סגירת שיחת SSL (ע"פ הפרוטוקול, נדרש לסגור את הSSL בצורה מאובטחת). למה?
   a. אחרת, אי אפשר לדעת אם הגיעו אלי כל החבילות שאמורות להגיע אלי – כדי למנוע Truncation.

SSL Record Protocol (שמעליו רצים כל שאר הפרוטוקולים של SSL)

- אורז את כל פרוטוקולי הSSL מעליו.

סדר הפעולה (כל פעולה מבוצעת על תוצאת הקודמת):

- מבצע פרגמנטציה (בלי קשר לפרגמנטים של TCP). כל פרגמנט הוא לכל היותר בגודל 2^{14} בתים = KB16.
- מבצע דחיסה (אופציונלית) לפרגמנט
- תוספת של MAC (קוד מאמת) לפרגמנט הדחוס (אם הוא דחוס).
- הצפנה
- תוספת Header

נשים לב – הוא לא יכול לבצע הצפנה / MAC לחבילות הראשונות, ואז פשוט מוותרים על שלבים אלו.
מה יש בHeader?

1. סוג חבילה (TYPE)

הMAC מחושב על החבילה וגם על Length וCounter. ולכן בעצם זה מאומת על ידי MAC, בלי שזה ישלח.
למה מיועד הCounter? לצרכים של Retransmission של חבילות SSL (למרות שיש את זה בTCP).

SSL Alert

מכיל:

2. TYPE - שלו (ALERT)
3. קוד – לדוגמא :
   a. MAC לא תקין
   b. הצפנה לא ברורה
   c. Cert לא בתוקף
   d. Certificate עם חתימה שגויה.
   e. ...

CCS – Change Cipher Spec

1. קוד (כמו TYPE) – מסמן CCS.
2. ID – מזהה אלגוריתמים ומפתחות שעובדים איתם (כמו SPI).

חלק מההודעות הללו יכולות להשלח על ידי השרת, חלק על ידי הלקוח.

הקמת שיחה – מחולקת לשני חלקים – Session וConnection. Session מכיל מספר Connections.

SSL Session

- מזוהה ע"י SessionID – **32 Byte.** נקבע ע"י השרת.
- Peer Certificate - הCertificate של הצד השני.
- Compression algorithm – אלגוריתם דחיסה.
- Cipher Spec (enc, Mac) – באמצעות איזה אלגוריתם הצפנה / MAC עובדים.

- is resumable - האם מותר לחדש את הSession הזה, או שהוא מיועד לConnection אחד בלבד. נקבע ע"י השרת.
- Master Secret – **48 בתים**. – מתואם על ידי שני הצדדים.

כדי להקים את הSession נשתמש בקריפטוגרפיה אסימטרית.

Connection

- Session Id
- Server & Client Seq#'s
- Server & Client Random numbers
- crypto keys
  - ENCRYPTION_WRITE_KEY
  - ENCRYPTION_READ_KEY
  - MAC_WRITE_KEY
  - MAC_READ_KEY
  - IV_WRITE
  - IV_READ

נקים את הConnection באמצעות קריפטוגרפיה סימטרית.

גזירת אלמנטים סודיים

1. הMaster secret נוצא על ידי חישוב HASH על המספרים האקראיים של הלקוח, שרת, ו PMS = Pre Master Secret.
2. כדי ליצור מפתחות אחרים, מבצעים HASH (פונקציה אחרת) על הMaster Secret (MS) ומספרים אקראיים של הלקוח והשרת.

איך נוצר PMS?(Pre Master Secret)

אפשרויות:

1. Diffie Hellman Key Exchane
2. העברתו מוצפן על ידי RSA.
3. שימוש בחומרה יעודית בשני הצדדים (אם יש כמובן).

23.1.2008

על המבחן:

- 12 שאלות, 3 שעות.
- יש 2 – 3 שאלות חישוביות יותר
- 9 – 10 שאלות מורכבות יותר.
- שיעור חזרה – יום ב' 11.2. כדאי לעבור קודם על מבחנים משנים קודמות.
- **חומר פתוח**.

SSL Handshake

מה בתוכנית האמנותית?

1. הזדהות
2. תאום אלגוריתמים
3. אימות שרת
4. אימות לקוח (אופציונלי)
5. תאום מפתחות שיחה
6. חידוש Session קיים (במידת הצורך)
7. הוכחת אימות על תכנים של Hand Shake

סדר הודעות: (שולח :)

1. לקוח– Client Hello
   - Ver – הגרסה המקסימלית שהוא יודע לעבוד איתה
   - Algorithms – למטרות הצפנה, חתימה, תאום מפתח, דחיסה. בד"כ ירשום אותם לפי סדר עדיפויות לכל מטרה.
   - R_C (Random Client)
   - ID
2. שרת– Server Hello
   - Ver – הגרסה שהוא רוצה לעבוד איתה (קטן או שווה לגרסה שהלקוח שלח)
   - Algorithm – אלגוריתמים שהוא בוחר מרשימת האלגוריתמים הנתמכים של הלקוח (להצפנה, חתימה, תאום מפתח, דחיסה...). אם אין התאמה מול דרישת הלקוח, החיבור מתנתק.
   - R_S
   - ID
3. שרת– Server Cert(s)
   - X509 – יתכנו כמה Certificates, במידה והלקוח מכיר רק אחת מהסמכויות החותמות.
4. שרת– Server KE – אופציונלי. בהתאם לשיטת תאום המפתחות שעובדים איתה.
   - אם מדובר בDH – אז נשלח כאן $g^x \bmod p$ של השרת.
   - אם מדובר בRSA – אז ריק.
5. שרת– Server Cert Req – אופציונלי (אם השרת רוצה Cert מהשרת)
   - רשימה של CA מוכרים – כלומר, השרת מבקש מהלקוח Certificate שחתום על ידי אחד מהCAים הנ"ל .
6. שרת– Server Done
   - למטרת סינכרון.
   - נשים לב – בשלב הזה, הלקוח לא יודע עדיין כלום על השרת – שזה אכן השרת. יש כאן הזדהות, תאום אלגוריתמים, ותו לו.
7. לקוח– Client Cert (אופציונלי)
   - Certificates לפי בקשת השרת (אם יש כאלה).
8. לקוח– Client KE

- אם מדובר בDH- אז $g^y \bmod p$
- אם RSA – אז הוא שולח $ENC(PMS)$ - PMS = Pre Master Secret. הוא **ממציא** PMS בגודל 48 ביט, ומצפין אותו באמצעות המפתח הציבורי של השרת.

9. לקוח – Client Cert Verify (אופציונלי)
   - **הלקוח מוכיח את זהותו לשרת**, על ידי כך שהוא חותם על ערכים קודמים בפרוטוקול (לדוגמא, מספרים אקראיים, **אבל לא רק**, למנוע Replay).
10. לקוח – Client CCS
    - ID – עכשיו הלקוח רוצה לעבור לסוויטה המוגדרת לחלוטין באמצעות הID – כי הוא יודע כבר את כל המידע הרלוונטי – כל המפתחות שיגזרו מה Pre Master Secret.
11. לקוח – Client FIN
    - Mac של ערכים קודמים. ההודעה מועברת גלויה (לא מוצפנת / מאומת / כיו"ב). המטרה - למנוע Tempering של ערכים קודמים – לדוגמא, פרוטוקולים נתמכים על ידי הלקוח. אם השרת יגלה שלאו – הוא יבצע Abort.
12. שרת – Server CCS
    - שולח את הID – אמור להיות אותו ID.
13. שרת – Server FIN
    - שולח Mac של ערכים קודמים. (זה מפתח שונה מאשר המפתח שהלקוח משתמש בו, ראה שיעור קודם). – כמו קודם – מניעת Tempering של ערכים קודמים. אם הלקוח יגלה שהMac לא מתאים – הוא יבצע Abort.
14. מכאן – התקשורת מוצפנת.

באינטרנט עובדים בדרך כלל עם RSA, ובלי Certificate של הלקוח.

<u>צורות הפעלת HandShake</u>

- RSA
  - נשים לב – למרות שרק הלקוח שולח את הPMS, הMS נגזר גם מהRandom של השרת, ולכן גם אז, גם השרת תורם לאקראיות מפתחות השיחה.
  - סדר הודעות
    - Client Hello
    - Server Hello
    - Client Cert
    - Server Done
    - Client KE
    - CLIENT CCS
    - Client FIN
    - SERVER CCS
    - SERVER FIN

- DH בלי Auth
  - סדר ההודעות
    - CLT Hello
    - SRV Hello
    - SRV CERT
    - SRV KE
    - SRV DONE
    - CLT KE
    - CLT CCS
    - CLT FIN
    - SRV CCS

- SRV FIN
  - אי אפשר לדעת כאן שאין Man in the middle.
- Dh עם Auth
  - כמו קודם, רק ש
    - SRV KE מאומת ע"י הCert הפרטי של השרת.
    - בServer Cert יהיה כאן מפתח ציבורי שמתאים למפתח חתימות של השרת.
  - כששני הצדדים מאומתים, אז:
    - CLT Hello
    - SRV Hello
    - SRV CERT
    - SRV KE – עם חתימה. נשים לב – אפשר לבצע Replay, אבל אני יודע מה g^x ומה החתימה עליו, אבל לא מה x
    - SRV Cert Req
    - SRV DONE
    - CLT CERT
    - CLT KE
    - CLT CERT VER
    - CCS
    - FIN
    - CCS
    - Fin

<u>חידוש session</u>

- Client hello
- Server Hello
- Client CCS
- Client fin
- Server ccs
- Srv fin

המטרה היא בעיקר הID. אם השרת לא מסכים לחידוש, הוא יענה בHello עם מזהה ה Session החדש, ואז מתחילים מחדש את הפרוטוקול, כאילו זה לא חידוש.