# מבוא להצפנה ואבטחת מידע

## מבוסס על הרצאותיו של פרופ' שלמה קיפניס

תשס"ח סמסטר א'
דינה זליגר

# תוכן עניינים

Please read the following important legal information before reading or using these notes. The use of these notes constitutes an agreement to abide by the terms and conditions below, just as if you had signed this agreement.

**A. THE SERVICE.**

The following notes ("The service") are provided by Dina Zil's Notes-Heaven ("Notes-Heaven").

**B. DISCLAIMER OF WARRANTIES; LIMITATION OF LIABILITY.**

Notes-Heaven does not endorse content, nor warrant the accuracy, completeness, correctness, timeliness or usefulness of any opinions, advice, content, or services provided by the Service.

YOU AGREE THAT USE OF THE SERVICE IS ENTIRELY AT YOUR OWN RISK. THE SERVICE PROVIDED IS PROVIDED "AS IS," WITHOUT WARRANTY OF ANY KIND. NOTES-HEAVEN EXPRESSLY DISCLAIMS ALL WARRANTIES OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION: ANY WARRANTIES CONCERNING THE ACCURACY OR CONTENT OF INFORMATION OR SERVICES. NOTES-HEAVEN MAKES NO WARRANTY THAT THE SERVICE WILL MEET YOUR REQUIREMENTS, OR THAT THE SERVICE WILL BE ERROR FREE; NOR DOES NOTES-HEAVEN MAKE ANY WARRANTY AS TO THE RESULTS THAT MAY BE OBTAINED FROM THE USE OF THE SERVICE OR AS TO THE ACCURACY OR RELIABILITY OF ANY INFORMATION OBTAINED THROUGH THE SERVICE. YOU UNDERSTAND AND AGREE THAT ANY DATA OBTAINED THROUGH THE USE OF THE SERVICE IS DONE AT YOUR OWN DISCRETION AND RISK AND THAT YOU WILL BE SOLELY RESPONSIBLE FOR ANY DAMAGE TO YOUR GPA.

NEITHER NOTES-HEAVEN NOR ANY OF ITS PARTNERS, AGENTS, AFFILIATES OR CONTENT PROVIDERS SHALL BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF USE OF THE SERVICE OR INABILITY TO GAIN ACCESS TO OR USE THE SERVICE OR OUT OF ANY BREACH OF ANY WARRANTY.

**C. INDEMNIFICATION.**

You agree to indemnify and hold Notes-Heaven, its partners, agents, affiliates and content partners harmless from any dispute which may arise from a breach of terms of this Agreement. You agree to hold Notes-Heaven harmless from any claims and expenses, including reasonable attorney's fees and court costs, related to your violation of this Agreement.

**D. OWNERSHIP RIGHTS.**

The materials provided by the Service may be downloaded or reprinted for personal use only. You acknowledge that the Service contains information that is protected by copyrights, trademarks, trade secrets or other proprietary rights, and that these rights are valid and protected in all forms, media and technologies existing now or hereafter developed. You may not modify, publish, transmit, participate in the transfer or sale, create derivative works, or in any way exploit, any of the Content, in whole or in part. You may not upload, post, reproduce or distribute Content protected by copyright, or other proprietary right, without obtaining permission of the owner of the copyright or other proprietary right.

**E. NO COPYING OR DISTRIBUTION.**

You may not reproduce, copy or redistribute the design or layout of this service, individual elements of the design, Notes-Heaven logos or other logos appearing on this service, without the express written permission of Notes-Heaven, Inc. Reproduction, copying or redistribution for commercial purposes of the service is strictly prohibited without the express written permission of Notes-Heaven, Inc.

If you have any questions about this statement or the practices of this service you can contact

Dina Zeliger
dinazil @ notes-heaven.com

# מבוא לאבטחת מידע

## מדוע בעיית אבטחת המידע כל כך מורכבת?

העולם שבו אנחנו חיים מורכב מאוד, וגרוע עוד יותר – הוא מתפתח בקצבים עצומים. אנחנו מוקפים בטכנולוגיות שונות כמו מחשב אישי, טלפון, מכשירים סלולאריים, GPS, אינטרנט ועוד רבות. כולן עובדות בצורה ממוחשבת, בדרך זו או אחרת, ונגישות ממקומות שונים. השאיפה שלנו היא שהטכנולוגיות האלה גם יוכלו לתקשר אחת עם השנייה, אלא שכל טכנולוגיה היא עולם ומלואו ואנשים שונים מפתחים טכנולוגיות שונות ואין סטנדרט (כי אי אפשר לקבוע תקן למשהו שמתפתח כל כך מהר). אז אם יש $n$ מכשירים שצריכים לדעת לדבר זה עם זה צריכים להיות $n^2 - n$ ממשקים שונים שמאפשרים לכל מכשיר לדבר עם כל מכשיר אחר. לכל מכשיר גם יכולות לקרות תקלות שונות ומשונות...

## הגדרת אבטחת מידע

בלשון עממית, הרעיון הוא *למנוע מדברים רעים לקרות* (שכסף לא ייעלם מהבנק, שלא יפרצו למחשב, שהדיסק לא ייפול), אבל זה כמובן בלתי אפשרי בצורה מוחלטת. אז נמתן את ההגדרה להיות *הקטנת נזקים מדברים רעים*. אבל בזמן שאנחנו מונעים מהדברים הרעים לקרות צריך לאפשר לדברים הטובים לקרות.

## אתגרים שהאבטחה צריכה להתמודד איתם

1. אבטחה של מגוון מערכות שונות
2. אבטחת ממשקים בין מערכות שונות
3. קיום צרכים ומטרות שונים
4. המתקיפים בד"כ מחפשים את החוליה הכי חלשה בשרשרת האבטחה ולכן אנשי אבטחה צריכים לאבטח את כל החוליות
5. האבטחה לא צריכה לפגוע בביצועים ובשימושיות של המערכות
6. העלויות של האבטחה צריכות להיות סבירות – אין טעם להשקיע כמות כסף מסוימת באבטחה של משהו שעולה פחות...

## בעיות אבטחה

ניתן כמה דוגמאות לבעיות של אבטחת מידע:

1. **אובדן מידע** – למשל אם היה קובץ והוא נעלם לפתע פתאום
2. **גניבת מידע** – למשל אם יש קובץ סודי אבל לעוד מישהו יש גישה אליו
3. **השחתת מידע** – למשל אם יש קובץ שסומכים עליו אבל למעשה יש בו שגיאות
4. **אי נגישות למידע** – למשל אם השרת נופל ואי אפשר לגשת לדף אינטרנט
5. **מניעת גישה** (**Denial-of-service**) – למשל אם מישהו מונע אקטיבית גישה למידע
6. **הפצת מידע לא חוקית** – למשל eMule

7. **התחזות, גניבת זהות** – למשל אם מישהו מזייף תעודת זהות ובורח עם הכרטיס אשראי למקסיקו
8. **הונאה**

## מונחים

1. **האזנה** (**Eavesdropping**): כשיש שני צדדים שמדברים וחושבים שאף אחד לא שומע אותם אבל למעשה יש מישהו שמאזין. אם לא מדובר בשיחה סודית אז בד"כ זה לא יוצר בעיה, ואילו אם השיחה סודית הצפנה בד"כ פותרת את בעיית ההאזנה. כאשר מדברים על האזנה מתכוונים בד"כ לקווי תקשורת.
2. **הצצה** (**Snooping**): דומה להאזנה אלא שהגישה למידע היא פיסית ולא דרך קווי תקשורת. למשל, קריאת אי מייל של מישהו אחר או השמת חיישן במקלדת. הצפנה לא עוזרת נגד הצצה משום שהבעיה היא בד"כ בהרשאות גישה.
3. **התעסקות** (**Tampering**): כניסה למערכת ושינוי דברים בה. בד"כ הכוונה היא לשינויים פיסיים.
4. **התחזות** (**Spoofing**): שימוש בכתובת או תעודה מזהה של מישהו אחר כדי להזדהות. כדי למנוע את זה יש צורך במנגנונים של אימות זהות.
5. **חסימה** (**Jamming**): חסימת גישה לשרת או ערוץ תקשורת ע"י העמסה גדולה. אפשר לעשות מעקב של הנכנסים למערכת ולחסום את המטריד, אלא שמטרידים בד"כ מקצועיים ויודעים להסוות את עקבותיהם...
6. **החדרת קוד** (**Code injection**): וירוסים, רוגלות (spyware), תולעים, סוסים טרויאניים וכו'.

נסכם את המונחים בטבלה:

| | תיאור | אמצעים | איומים |
|---|---|---|---|
| **האזנה** | השגת מידע מבלי לשנות אותו | נתבים, שערים, תפיסת חבילות | ניטור מידע שעובר ברשת: שמות משתמש וסיסמאות, מספרי כרטיסי אשראי, אי מיילים והודעות אחרות, ניתוח תעבורה |
| **הצצה** | השגת מידע מבלי לשנות אותו | מעבר על מסמכים בדיסק או בזיכרון: שימוש בהרשאות לגיטימיות, פריצה למערכת, גניבת מחשבים, מעקב אחרי לחיצות מקשים | השגת מידע רגיש כמו קבצים עם מספרי כרטיסי אשראי או שמות משתמש וסיסמאות |
| **התעסקות** | שינוי או הריסה של מידע שמור | שימוש בהרשאות מבפנים או פריצה למערכת מבחוץ | שינוי רשומות כמו ציונים, חובות, תיקים רפואיים, מחיקת עקבות הפריצה, שתילת סוסים טרויאניים |
| **התחזות** | התחזות למשתמשים או מחשבים אחרים במטרה לקבל הרשאות גישה | גניבת חשבונות, ניחוש סיסמאות, זיוף כתובת אי מייל, כתובת IP | זיוף הודעות, קבלת מידע, מניעת שרות |
| **חסימה** | נטרול מערכת או שירות | העסקת השרת בדרכים לגיטימיות רבות ויצירת עומס יתר תוך התחזות לכתובות שונות כדי להימנע ממעקב | השתלטות על כל המשאבים של המכונה המותקפת, גרימה לכיבוי השרתים |
| **החדרת קוד** | החדרת קוד עוין לביצוע על שרת עם הרשאות גבוהות והדבקה של שרתים אחרים | וירוסים (קבצי הרצה שמופצים ע"י אי מייל, דיסקים נגועים, macros), תולעים (משתכפלים באינטרנט) | כל העולה על הדעת... |

## סוגי בעיות

הבעיות מתחלקות לשני סוגים עיקריים:

1. **בעיות טבעיות**: אלה בעיות שאין לנו שליטה עליהן. למשל, נפילת מתח, אסון טבע, פגם בדיסק. לבעיות כאלה יש סיכויי מופע וניתן לחשב את הנזק הצפוי. בעיות טבעיות בעיקר יכולות לגרום לאיבוד מידע ולנזק לחומרה. לכן ההגנה היא בד"כ ע"י מערך גיבויים.
2. **בעיות יזומות**: אלה בעיות שיותר קשה להעריך את סיכוי המופע שלהן ואת הנזק הצפוי. הנזק הוא גם לא תמיד כספי (למשל במקרה של האזנה לקו תקשורת צבאי). כאשר מדובר בבעיות יזומות דברים חסרי ערך בד"כ לא נמצאים בסיכון משום שהם לא מעניינים אף אחד. וכשבאים לאבטח משהו חשוב צריך להעריך את הנזק (הכלכלי, פוליטי, צבאי ועוד) שייגרם ולתת משאבים בהתאם.

דווקא עם בעיות יזומות אפשר בד"כ להתמודד בצורה יותר אקטיבית: שימוש בסיסמאות, אמצעי אימות, הגבלת גישה, firewalls וכן הלאה.

בשני סוגי הבעיות ביטוח יכול לעזור למזער את הנזקים.

## סוגי פורצים

נדרג את הפורצים מה"טובים" אל ה"רעים":

1. **חוקרים אקדמיים** – מטרתם לחקור איך לפרוץ מערכת ואיך להגן עליה מפני פריצות. הם עובדים בצורה חוקית לגמרי ובד"כ מפרסמים את התוצאות שלהם לטובת הכלל. מטרתם אקדמית בלעדית.
2. **יועצי אבטחה** – אלה אנשים שעובדים עבור חברות בצורה חוקית ומטרתם לשבור את מערך ההגנה של החברה כדי למצוא את הנקודות הפגיעות בו ולתקנן. ההבדל בינם לבין האקדמאיים הוא שהם עושים את זה בתשלום ולא מפרסים את התוצאות מחוץ לארגון שהם עובדים עבורו.
3. **פורצים עצמאיים** – קבוצות או בודדים שלפעמים אפילו מזדהים ומדי פעם מפרסמים כלים או דרכים לפרוץ מערכות. מטרתם לדווח לעולם על תקלות אבטחה במערכות והרבה פעמים הם רואים את עצמם כפעילים חברתיים. האקרים עצמאיים נמצאים בתחום האפור מבחינת חוקיות. הם לא עובדים בצורה חוקית אבל המטרה שלהם היא לא רעה.
4. **חובבנים** – פורצים למערכות סתם בשביל הכיף, הרצון להטריד והעניין. פרופיל טיפוסי: זכרים בגילאי 16-12 עם קוקו ☺
5. **ארגוני פשע** – פורצים לצורך מטרות כלכליות. מדובר בתשתיות ארגוניות שלמות שגונבות מיליארדי דולרים באופן שיטתי. יש הרבה פחות ארגוני פשע מחובבנים אבל הנזק שהם גורמים הרבה יותר גדול!

הפורצים שמדאיגים אותנו הם בעיקר החובבנים וארגוני הפשע. מול חובבנים לא מאוד קשה להתמודד (יש כל מיני patch-ים שאפשר להוריד) אבל יש כל כך הרבה מהם שאי אפשר להתמודד עם כולם. ארגוני הפשע הם יותר בעיתיים ובד"כ מאוד קשה לתפוס אותם. אם כבר מוצאים מישהו בד"כ מדובר בדגי רקק...

## סוגי פריצה

| | מוטיבציה | משאבים | סיכונים |
|---|---|---|---|
| **פוליטי** | סיבות פוליטיות, ריגול תעשייתי, ריגול צבאי, לוחמת מידע | כמעט בלתי מוגבלים | תלוי במדינה |
| **מסחרי** | אינטליגנציה עסקית, השגת יתרון עסקי, גרימת הפסדים | מוגבלים ע"י הרווח הפוטנציאלי | תביעה משפטית |
| **סוציאלי** | מטרה פוליטית או סוציאלית, אידיאולוגיה | קבוצות ממסדיות יכולות להיות בעלות משאבים רבים, לקבוצות שונות יכולה להיות תמיכה ציבורית | כמעט אין |
| **פיננסי** | בצע כסף, גניבת משאבים | ליחידים אין משאבים רבים ואילו לפשע מאורגן כמובן יש | נחשבים נמוכים |
| **פרטי** | עובדים לא מרוצים, סקרנות, אתגר אינטלקטואלי, ונדליזם | משאבים מועטים אבל זו קהילה גדולה | כמעט ואין כשחוצים את תחום השיפוט |

# הנדסת אבטחת מידע

הנדסת אבטחה הוא תחום שמטרתו לדון בארכיטקטורה של מערכת אבטחה ולא בפרטים הקטנים שלה.

## מסמך תכנון

מסמך התכנון מתאר את מה שהמערכת עושה, את המטרות שלה וכן הלאה. זהו חלק חשוב מאוד ממערכת האבטחה ובד"כ מושקע הרבה מאוד זמן בכתיבת מסמך התכנון. מאחר שהדברים הם דינאמיים מסמך התכנון יכול להשתנות ולכן נכתב מחדש כל כמה שנים.

כשבונים ארגון חשוב תהיה תמונת עולם כללית. זה גם חשוב ללקוחות ומונע תביעות משפטיות מיותרות. כל מה שהמערכת אמורה לספק או לא לספק צריך להיות מתואר במסמך התכנון. ככה כולם מודעים לכל הסכנות.

מסמך התכנון כולל בין השאר את הפרטים הבאים:

- **מדיניות**: בארגונים שונים המדיניות היא שונה. למשל באוניברסיטה רוב הדברים הם גלויים ואילו בצבא רוב הדברים הם סודיים. צריך להגדיר כל מיני דברים לגבי המדיניות. למשל:
  - מדיניות פתוחה או סגורה
  - שירות בתשלום או חינם
  - האם יש גישה למידע של אחרים
  - האם המידע חסוי או גלוי
- **איומים**: אלה הדברים שהזכרנו בפרק הקודם – האזנה, פריצה, גניבה וכו'. כמובן, כל ארגון נתון לסוגים שונים של איומים. כמו כן, בארגון יכולות להיות הרבה תת מערכות ולכל אחת מהן איומים שונים ואבטחות שונות. למשל, בית חולים יהיה מאוד מוטרד אם יפרצו לתיקים האישיים של החולים ולא אם יפרצו למאגר מידע על מחלות.
- **מטרות**: כמובן, לכל ארגון יש מטרות משלו ולתת מערכות יכולות להיות מטרות שונות. ניתן דוגמאות למטרות אפשריות:
  - **סודיות של מידע** (שאף אחד לא יראה את הקובץ ללא הרשאה) – כמובן זה לא חייב להיות נכון לכל המידע, כמו במקרה של בית חולים ומאגר של מידע על מחלות.
  - **אימות מידע** (שהקובץ שאנחנו רואים בוודאות לא שונה בדרך) – לפעמים המידע צריך להיות אמין, למשל מחירים של מניות באתר הבורסה.
  - **זיהוי ישות** (לזהות באופן ראשוני מי הישות שאנחנו עומדים מולה) – בעזרת מצלמה למשל, או לפי שם או טביעת אצבע.
  - **אימות ישות** (לוודא שמי שהישות אומרת שהיא זו האמת) – זה שונה מזיהוי ישות בכך שאם מישהו אומר שהוא משה זה כמובן לא אומר שזה נכון.
  - **אימות מקור מידע** (לוודא מי באמת שלח את המידע ללא כל קשר לתוכן שלו) – זה שונה מאימות מידע בכך שכאשר מאמתים רק את המידע זה לא מעניין אותנו מה המקור שלו אלא רק אמיתותו. למעשה, בד"כ עושים אימות גם לתוכן המידע וגם למקור.
  - **אי-התכחשות** – מנגנון שבו חתימה מחייבת לא רק בפנינו אלא גם בפני גוף שלישי. למשל, אם קיבלנו חוזה חתום לא נרצה שהחותם יוכל לחזור בו.
  - **זמינות מידע**
  - **זמינות מערכת**

- **פרטיות מידע** – בשונה מסודיות מידע הכוונה היא למשל שמה שאנחנו קונים במכולת הוא לאו דווקא סודי אבל אנחנו לא רוצים כל העולם יידע את זה.
- **הכלת מידע** – למשל רמת סיווג, שכל אחד יוכל לראות רק את מה שמותר לו לפי הסיווג, או שקובץ סודי ביותר לא יגיע לרשת שהסיווג שלה הוא רק סודי.
- **אנונימיות** – יכולת לעשות עסקאות מבלי להזדהות. למשל, אם מתישהו הבחירות יתבצעו דרך האינטרנט צריך יהיה מנגנון אנונימיות.

- **ישויות**: הגופים הפעילים במערכת (משתמשים, תחנות עבודה, gateways, שרתי אבטחה). למשל באוניברסיטה אלה יכולים להיות המרצים, הסטודנטים, המזכירות...
- **עצמים**: גופים סבילים שאינם פעילים באופן עצמאי אלא פועלים עליהם. למשל קובץ, פריט מידע שנשלח באוויר, מערכת מחשבים, מחשב בודד.
- **הרשאות**: פירוט הפעולות המותרות לכל אחת מהישויות על כל אחד מהעצמים.
- **תקלות**: ברור שאי אפשר לאבטח את המערכת מכל בעיה באשר היא. לכן צריך להגדיר מהן התקלות שעלולות לצוץ ואינן מטופלות ע"י מערך האבטחה.
- **התאוששות**: מנגנונים נוספים לטיפול בדברים שלא כיסינו. בד"כ מדובר בטיפול מחוץ למערכת. למשל, אם בניין לא אבטח את עצמו נגד רעידת אדמה הוא יכול לרכוש ביטוח וזה יעזור בהתאוששות מרעידת אדמה פוטנציאלית. ניתן כמה דוגמאות להתאוששות:

| תקלה | התאוששות |
|---|---|
| אובדן מידע (למשל אם הדיסק נפל) | גיבוי – למשל במקרה של RAID זה משהו שקורה ברקע ואנחנו נהנים ממנו אם קורית תקלה. בוודאי יש חשיבות לתדירות הגיבוי. אם מדובר במשהו חשוב אולי נעשה גיבוי כל יום.<br>עוד צורה של גיבוי היא הדפסת דפים. זה פחות נוח מהאופציה הדיגיטלית כמובן. |
| חוסר נגישות (למשל אם כרטיס אשראי לא עובד) | שירותים אלטרנטיביים (כמו כסף מזומן במקרה של כרטיס אשראי לא פעיל) |
| נזק כללי | ביטוח |
| כל השאר | יש דברים שפשוט מכריזים שלא עושים כלום אם הם קורים |

## שיטות כלליות של אבטחה

1. **מניעה** – זה כמובן הדבר הרצוי אבל אי אפשר למנוע את כל הדברים הרעים בעולם. מנגנונים שונים מונעים במידת מה חדירות למערכת. למשל, מנעול, אנטי-וירוס, חומת אש (firewall), הרשאות, שומר ועוד.
2. **זיהוי** – מנגנוני זיהוי בד"כ מזהים חדריה אחרי שהיא קרתה. למשל, מצלמה, אזעקה, איתורן, קבצי log. הדברים האלה אמנם יכולים להרתיע פושעים אבל הם לא מונעים מהם להיכנס פיסית.
3. **התאוששות** – גיבוי, יתירות, ביטוח וכו'.

## שכבות אבטחה

הדרך הכי טובה להגן על מערכת היא להגן עליה בשכבות. השכבות שנציג הן לא מושלמות. הן יותר דומות לקליפה של בצל – יש דרך להיכנס אבל ככל שיש יותר שכבות היא יותר מסובכת.

1. **מודעות משתמש**: אם הארגון לא מודע לכך שיש בעיות אז כבר שום דבר לא יעזור לו. בד"כ ארגונים מודעים לקיום בעיות ומגנים על עצמם.

2. **הגנה פיסית**: צריך להגן קבצים, סיסמאות, מפתחות וכן הלאה. אם הם נגישים לכולם הם לא שווים הרבה.
3. **הרשאות גישה**: זו הרחבה של הגנה פיסית. זו הגנה אלקטרונית לקבצים במחשב שלא לכל המשתמשים תהיה גישה לכל הקבצים.
4. **קריפטוגרפיה** (אם צריך): אם המפתחות מוגנים יש טעם לדבר על הצפנה, אחרת כל אחד יכול לפענח את הטקסטים המוצפנים.
5. **ניטור** (**Monitoring**): זה למעשה סוג של זיהוי.
6. **גיבוי**: הגיבוי חשוב בעיקר למקרה שמידע הולך לאיבוד או שחלק מהמערכת נפל ואז אפשר להשיג מידע ממקום אחר.
7. **יתירות**: סוג של גיבוי שבו יש כמה מערכות ולא רק קבצי גיבוי (למשל RAID).
8. **הטעיה**: יש מערכות שנראות מערכות אמיתיות ויותר קל במידה מסוימת לפרוץ אליהן. אז הפורץ מגיע אליהן במקום אל המערכת האמיתי. המטרה במלכודת הזו היא כפולה: לבזבז זמן לפורץ ולאפשר לתפוס את הפורץ ולהבין איך הוא הצליח להיכנס למערכת. דוגמה למערכת הטעיה היא Honey Pots.
9. **התקפה**: לגרוף נזק לפורץ. זה קצת בעייתי כשמדובר בפשע מאורגן. אם מרביצים לגנגסטר הוא עלול להביא את כל החברים הגנגסטרים שלו עלינו וזה לא נעים. אז את ההתקפות צריך לעשות בצורה מאוד מבוקרת.
10. **הגנה משפטית**: זו הגנה דווקא מפני הלקוחות ולא מפני הפושעים...

## עקרונות אבטחה

- **Least privilege** – ביצוע פעולות עם הישות בעלת ההרשאה הנמוכה ביותר שעדיין מסוגלת לבצע את הפעולות. זה מקטין את הנזק במקרה של תקלה. למשל אם נכנסים ב-superuser בלינוקס וכותבים delete all לא יישאר לנו כלום ואילו אם עושים את דרך חשבון של משתמש מסוים זה ימחק רק את הקבצים שלו.
- **Trusted components** – בכל מערכת יש רכיבים שסומכים עליהם. צריך לזהות אותם, לאפיין אותם ולהיות משוכנעים שהם באמת בסדר. כמובן צריך לבצע את הבדיקות בתדירות מסוימת בהתאם לרגישות הרכיב. למשל מקלדת זה משהו שאנחנו לרוב סומכים עליו בעיניים עצומות, אבל הייתה איזו חברת מקלדות שהסתבר שהמקלדות שלה שלחו את הסיסמאות שהקלידו עליהן...
- **Simple design** – מזה הרבה שנים העולם הבין שמערכות פשוטות הן טובות יותר.
- **Paranoia** – אם יש משהו שווה בארגון שלנו תמיד ירדפו אחרינו. לכן אנשי אבטחת המידע תמיד צריכים להיות בפרנויה מסוימת – לבדוק סיסמאות מדי פעם, לבדוק את הקונפיגורציה של ה-firewall וכן הלאה.
- **No perfection** – צריך להבין שלעולם לא נוכל להגיע לשלמות ותמיד יהיו פרצות או משהו שמתקלקל. צריך להגדיר מהו המקום שבו מפסיקים להגן וכמה כסף מוכנים להוציא על האבטחה.

## אבטחה פיסית

המערכות מורכבות בסופו של דבר מרכיבים פיסיים וגם הם נמצאים בסיכון:

- חוטים שדרכם זורם מידע אנלוגי מפיצים קרינה שאפשר לקלוט ולפענח. לכן צריך להגן על הכבלים שלא ניתן יהיה לחוש את הקרינה מהם. כמובן גם הצפנת המידע שעובר דרכם תשיג מטרה דומה.

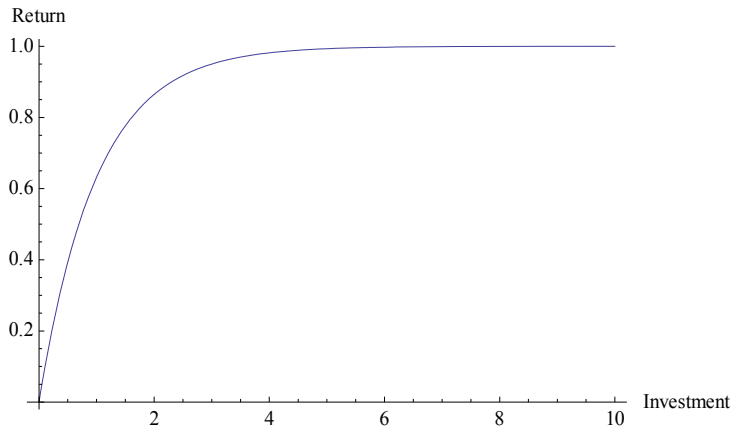
- אפשר לפרוץ לרכיבי חומרה ולהוציא את המידע שיש עליהם. פורצים מיומנים יודעים לקחת מעגל מודפס ולפענח את הפעולה שלו. אז אם האלגוריתם הסודי שלנו נמצא על שבב צריך להגן עליו. חברות בימינו מייצרו שבבים יותר ויותר מוגנים. למשל, יש שבבים שמוגדרים להרוס את עצמם ברגע שפותחים אותם.
- מסכים רגילים עם רובה אלקטרונים פולטים קרינה שאפשר לקלוט ושלחזר את המידע שנמצא על המסך. מהבחינה הזאת מסכי LCD הרבה יותר בטוחים.
- כשמוחקים קובץ מדיסק הוא לא באמת נמחק אלא רק נמחקת הרשומה שלו ברשימת הקבצים על הדיסק. גם אם כותבים אפסים על כל מקום בקובץ אפשר עדיין לזהות מה היה שם במקור. לכן, כדי למחוק קובץ בצורה בטוחה צריך לכתוב על המקום כמה פעמים מידע אקראי.

## אלגוריתם לניהול סיכונים

1. הערכת שווי הנכסים
2. זיהוי איומים ובעיות
3. הערכת הנזקים והסיכויים שלהם להתרחשות
4. הערכת תוחלת הנזקים
5. סקירת מנגנוני אבטחה
6. הערכה של הורדת נזק עם מנגנונים מסוימים
7. החלטה ובחירת מנגנונים

## החזר לעומת השקעה באבטחה

כפי שכבר אמרנו כמה פעמים, לא כדאי להשקיע באבטחה של מערכת יותר מהסכום שהיא שווה. כמו כן, אם אין לנו שום סוג של אבטחה, הוספת מנגנוני אבטחה מגדילה את ההחזר, אבל בוודאי יש גבול להחזר שאפשר לקבל מההשקעה.



# קריפטוגרפיה

המילה **קריפטוגרפיה** מורכבת משתי מילים. *קריפטו* שפירושה סתר ו*גרפיה* שפירושה כתיב. קריפטוגרפיה הוא תחום בן אלפי שנים ובמידה מסוימת הקריפטוגרפיה של היום היא כמו הקריפטוגרפיה של פעם.

שימושים של קריפטוגרפיה:

- סודיות מידע
- אימות מידע
- אימות מקור
- אי התכחשות
- תיאום מפתחות
- הקמת שיחה
- עסקאות בטוחות

הקריפטוגרפיה לא יכולה לפתור את כל הבעיות, לפחות לא בצורה ישירה. למשל, את הבעיות הבאות אי אפשר לפתור באמצעות קריפטוגרפיה:

- אובדן מידע
- השחתת מידע
- גיבוי
- וירוסים
- מניעת שרות

לפני שנתחיל לדון בשיטות קריפטוגרפיה יש לתת את הדעת על ההבדל שבין קריפטוגרפיה ל**סטגנוגרפיה**. קריפטוגרפיה מתעסקת בהסתרת תוכן ואילו סטגנוגרפיה מתעסקת בהסתרת עצם קיום המידע. יש חשיבות לתחום הזה משום שלפעמים גם עובדת קיום התקשורת יכולה ללמד משהו את התוקפים. למשל, אם בין שתי חברות לא קשורות פתאום מתחילה תקשורת מאוד עמוסה, אפילו אם המתקיפים לא יודעים את התוכן שלה, הדבר יכול להיות מחשיד. אולי למשל הן מתכננות שיתוף פעולה כלשהו או שאחת קונה את השנייה. במקרה זה אולי היה רצוי להסתיר את קיום התקשורת.

דוגמה לסטגנוגרפיה היא הסתרת מסרים בתוך תמונה. הצופה הנאיבי (או לא נאיבי) לא יודע שתמונה מסתירה בתוכה מסר, ואילו האדם שהמסר מיועד לו יכול בעזרת כלי מתאים לפענח את ההודעה.

## קריפטוגרפיה קלאסית

### צפני החלפה

כאמור, הקריפטוגרפיה היא תחום בן אלפי שנים. למשל בספר ירמיהו מופיעה המילה *ששך*. ברור מן ההקשר שמדובר בממלכה כלשהי אלא שלא ידועה ממלכה מאז בשם כזה. מסתבר שמדובר בבבל ו*ששך* זו למעשה הצפנה של בבל שמתקבלת ע"י החלפת האותיות באופן הבא:

$$\begin{aligned} ת &\leftrightarrow א \\ ש &\leftrightarrow ב \\ &\vdots \\ ל &\leftrightarrow כ \end{aligned}$$

צופן זה נקרא **אתב"ש** והוא דוגמה ל**צופן החלפה** – צופן שבו כל אות מוחלפת באות אחרת.

### צופן קיסר

צופן קיסר מתקבל ע"י הזזת האותיות באופן מעגלי, כלומר למשל באנגלית פונקצית ההצפנה היא

$$E(M) = (M + a)\ mod\ 26$$

כאשר $a$ הוא גודל ההזזה. בוודאי פונקצית הפענוח היא

$$D(C) = (C - a)\ mod\ 26$$

למשל עבור $a = -3$ מתקבלת ההצפנה הבאה:

$$M:\ \ a\ b\ c\ d\ e\ f\ g\ h\ i\ j\ k\ l\ m\ n\ o\ p\ q\ r\ s\ t\ u\ v\ w\ x\ y\ z$$
$$E(M):\ x\ y\ z\ a\ b\ c\ d\ e\ f\ g\ h\ i\ j\ k\ l\ m\ n\ o\ p\ q\ r\ s\ t\ u\ v\ w$$

ואז למשל $E("ATTACK\ FROM\ EAST") = "XQQXH\ COLJ\ BXPQ"$.

האמת היא שזה בדיוק מה שעושים בהצפנה מודרנית – מחליפים אות באות, אלא שבוודאי עושים את זה בצורה מתוחכמת יותר.

נשים לב שכאשר משתמשים בצופן קיסר הסוד הוא למעשה האלגוריתם. ברגע שאלגוריתם ההצפנה נופל בידי האויב אין לו שום בעיה לפענח כל מסר שיגיע. עולה מכאן שאין טעם לשמור את האלגוריתם בסוד. ואכן מהר מאוד העולם הבין שצריך לשנות את פני הדברים.

### צופן קיסר מוכלל

הרעיון דומה מאוד לצופן קיסר רגיל אלא שעכשיו נשתמש במשפחה של פונקציות הצפנה. לכל $k$ נגדיר פונקצית הצפנה עם פונקציה פענוח מתאימה:

$$E_k(M) = (M + k)\ mod\ 26$$
$$D_k(C) = (C - k)\ mod\ 26$$

בדוגמה מצופן קיסר למעשה פונקצית ההצפנה הייתה $E_{-3}$.

כעת ניתן לספר לכולם שאלגוריתם ההצפנה הוא צופן קיסר מוכלל ואילו הפרמטר $k$ הוא הסוד ונקרא לו **מפתח**. היתרון של שיטה שבה הסוד הוא המפתח ולא האלגוריתם הוא שהאלגוריתם הוא יציב. ניתן להשתמש בו שנים רבות ורק להחליף מפתח מדי פעם. הרי להחליף מפתח הרבה יותר קל מלהמציא אלגוריתם חדש. זה הרבה יותר רלוונטי לשיטות הצפנה מתקדמות יותר, שבהן ההצפנה נעשית ע"י חומרה ואז קל וחומר שקשה יותר להחליף חומרה מאשר מפתח.

זה דומה למנעול של דלת. לכל הדלתות יש בדיוק אותו אלגוריתם – צילינדר. אבל הסוד הוא המפתח ויש אותו רק למי שהדלת שייכת לו. היתרון הוא שאת המכניזם אפשר לבנות פעם אחת (כלומר את האלגוריתם) ומפתח אפשר להחליף מדי פעם וזה הרבה יותר זול מלהחליף את האלגוריתם.

בכל אופן, נחזור לצופן קיסר המוכלל. ברור שזה לא אלגוריתם חזק במיוחד. אחרי **חיפוש ממצה** של 26 ניסיונות לכל היותר כל מתקיף בעל קמצוץ אינטליגנציה יוכל לפענח את המסר ולגלות את המפתח. הבעיה היא ש**מרחב החיפוש** קטן מדי. כמובן, כדי שניתן יהיה לשבור את הצופן יש צורך בטקסט בעל לפחות שתי אותיות. סביר שגם שתי אותיות לא יספיקו. מעניין לבחון מהי כמות הטקסט המינימאלית שדרושה כדי לשבור את הצופן...

### צופן "כל התמורות"

ראינו שהבעיה עם צופן קיסר המוכלל הייתה שמרחב החיפוש היה קטן מדי. נשים לב שהצופן היה למעשה תמורה על האותיות של הא"ב האנגלי ומשפחת פונקציות ההצפנה שדיברנו עליה הייתה תת קבוצה של קבוצת התמורות על 26 אותיות. אז נרחיב את משפחת פונקציות ההצפנה להיות כל התמורות. נמספר את כל התמורות מ-1 ועד $26!$ ואז $E_k(M) = \pi_k(M)$ כאשר $\pi_k$ היא התמורה ה-$k$ במספור שלנו. כאן מרחב החיפוש הוא כבר הרבה יותר גדול – $26! \approx 4e26$. בימינו ניתן לפצח את זה עם מחשב בזמן סביר, אבל כמובן לפני אלפי שנים זה היה בלתי ניתן לשבירה ע"י חיפוש ממצה.

ובכל זאת יש שני חסרונות בולטים בשיטה הזאת:

1. השפה היא לא אקראית ולאותיות שונות יש תדירות מופע, לזוגות אותיות יש תדירות מופע וכן הלאה. לכן כשמקבלים טקסט מוצפן ע"י תמורה כל מה שצריך לעשות הוא **ניתוח תדירויות** של האותיות בטקסט. זה מקטין בהרבה את מרחב החיפוש. למשל האותיות הנפוצות ביותר בשפה האנגלית הן E ו-T. אז שתי האותיות הנפוצות ביותר בטקסט המוצפן הן כנראה E ו-T... יש גם סטטיסטיקות של זוגות אותיות, שלשות וכן הלאה. למעשה זוהי בעיה של כל **הצפנים המונואלפביתיים** – צפנים שבהם אות מסוימת תמיד מוצפנת בעזרת אותו הסימן. צפנים כאלה תמיד ניתן לשבור בעזרת ניתוח תדירויות, אם יש לנו מספיק טקסט מוצפן כדי לאסוף סטטיסטיקות מדויקות.
2. השיטה מאוד לא פרקטית! קשה למספר את כל הפרמוטציות בצורה יעילה וכמובן אי אפשר לשמור רשימות פרמוטציות באורך !26. המשימה גם נהיית הרבה יותר קשה אם אנחנו רוצים לא להכליל חלק מהפרמוטציות במרחב שלנו. למשל פרמוטציית הזהות היא בוודאי לא רצויה, וכך גם כל פרמוטציה שמחליפה רק בין שתי אותיות. אפשר לחשוב על עוד הרבה דוגמאות לפרמוטציות לא מוצלחות.

### *בחירת תמורה טובה*

לתמורה טובה יש מאפיינים רבים. השניים החשובים הם ששתי הודעות דומות יוצפנו באופן שונה מאוד וששני מפתחות דומים יצפינו את אותה ההודעה באופן שונה מאוד.

התמורות שמעוניינים בהן בד"כ הן אקראיות, אבל זה לא פרקטי לאנדקס הרבה תמורות. לכן בד"כ משתמשים בתמורות פסאודו-אקראיות. אלה תמורות אשר מיוצרות ע"י מנגנונים דטרמיניסטיים אל בכל זאת הן נראות אקראיות...

## צופן Flayfair

כדי להתמודד עם בעיית הצפנים המונואלפביתיים נציג שיטה שבה אותה האות לא תמיד מוצפנת בעזרת אותו סימן. אז נציג **צופן פוליאלפביתי** שבו אותה האות יכולה להיות מוצפנת ע"י סימנים שונים.

נסדר את הא"ב בטבלה בגודל $5 \times 5$ באופן הבא: נבחר מילה ארוכה למדי שאין בה חזרות של אותיות ונרשום אותה בטבלה. לאחר מכן נבחר אות אחת להשמיט (יש יש מקום רק ל-25 אותיות) ונרשום את שאר האותיות שלא הופיעו במילה בטבלה לפי הסדר. למשל, נבחר את המילה MONARCHY ונשמיט את האות J[1]:

| | | | | |
|---|---|---|---|---|
| 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 |

ההצפנה עובדת על זוגות אותיות. מחלקים את הטקסט שרוצים להצפין לזוגות אותיות ומצפינים כל זוג באופן הבא: כל זוג אותיות בהכרח נמצא או בשורה אחת, או בטור אחד, או בשורה שונה ובטור שונה.

[1] למעשה בד"כ משמיטים את האות J

1. אם שתי האותיות באותה שורה, כל אות מוחלפת באות שלימינה כאשר עבור האות הימנית ביותר האות השמאלית ביותר נחשבת האות שלימינה. בדוגמה שלמעלה CH הופך ל-HY, HB הופך ל-YD ו-CD הופך ל-HC.
2. אם שתי האותיות באותו הטור, כל אות מוחלפת באות שמתחתיה כאשר האות הכי עליונה נחשבת האות שמתחת לאות הכי תחתונה. בדוגמה שלמעלה RD הופך ל-DK, DT הופך ל-KZ ו-RZ הופך ל-DR.
3. אם שתי האותיות נמצאות בשורות שונות ובטורים שונים האותיות יוצרות מעין מלבן ואז כל אות מוחלפת באות הנגדית שבאותה שורה.

כעת צריך להכין את הטקסט שעומדים להצפין. נניח שאנחנו רוצים להצפין את המשפט ATTACK JAPAN. ראשית, האות T מופיע ברצף פעמיים וזה משהו שאנחנו רוצים להימנע ממנו אז את כל האותיות הכפולות נפריד ע"י X או Z. למשל, ATXTACK JAPAN. שנית, השמטנו את האות J מהצופן, לכן את כל המופעים של האות J נחליף ב-I ונקבל ATXTACK IAPAN. כעת אפשר להתחיל להחליף. החלוקה לזוגות היא AT XT AC KI AP AN.

1. AT יוצרות מלבן ולכן מוחלפות ב-RS:

| | | | | |
|---|---|---|---|---|
| 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 |

2. XT יוצרות מלבן ולכן מוחלפות ב-ZS
3. AC יוצרות מלבן ולכן מוחלפות ב-MB
4. KI נמצאות באותה שורה ולכן מוחלפות ב-EK

| | | | | |
|---|---|---|---|---|
| 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 |

5. AP יוצרות מלבן ולכן מוחלפות ב-OS
6. AN נמצאות באותה שורה ולכן מוחלפות ב-RA

סה"כ ההצפנה של ATTACK JAPAN היא RSZSMBEKOSRA.

ברור שכדי לפענח טקסט מוצפן צריך לעשות את אותו התהליך אלא שהחלפת האותיות נעשית בכיוון ההפוך – אותיות מאותה שורה מוחלפות באותיות משמאלן ואותיות מאותה עמודה מוחלפות באותיות שמעליהן. במקרה שהאותיות בשורה שונה ובטור שונה ההחלפה היא כמובן סימטרית.

לשיטה הזאת שלושה יתרונות מרכזיים:

- פשוט מאוד ליישום מכאני
- יש הרבה מאוד מפתחות פשוטים
- מאחר שמצפינים זוגות של אותיות זה מטשטש את תדירות המופעים

כאמור, לכאורה פתרנו כאן את הבעיה של הקוד המונואלפביתי. אכן האות A הופיעה בטקסט המקורי 4 פעמים והוצפנה בשלושה סימנים שונים. אלא שאם עושים ניתוח תדירויות על זוגות אותיות גם אז ניתן לשבור את הקוד. באותו אופן אפשר להרחיב את הקוד לפעול על שלשות אותיות וגם אז ע"י

הסתכלות על כל שלשות האותיות האפשריות וניתוח תדירות ההופעה שלהן ניתן לשבור את הקוד. ובכל זאת, צופן Playfair חזק יותר מצופן רגיל משום יש הרבה פחות הבדלים סטטיסטיים בין ההופעות של זוגות אותיות ועוד פחות הבדלים בין הופעות של שלשות אותיות וכן הלאה.

### צופן Hill

נבחר פרמטר כלשהו $m$ ונבחר מטריצת מפתח אקראית $K \in M_m(\mathbb{Z}_{26})$. אז עבור טקסט באורך $m$ שאנחנו רוצים להצפין פונקציה ההצפנה היא מכפלת המטריצה בהודעה:

$$E_K(M) = E_K(p_1 \dots p_m) = K \begin{pmatrix} p_1 \\ \vdots \\ p_m \end{pmatrix}$$

אם אורך ההודעה גדול מ-$m$ נחלק את ההודעה לחלקים וכל חלק נצפין בנפרד. אם אורך ההודעה קטן מ-$m$ נרפד אותה באיזשהו אופן.

בוודאי כדי לפענח נרצה לכפול את הטקסט המוצפן במטריצה ההפוכה ולכן מטריצת המפתח $K$ צריכה להיות הפיכה.

בדרך הזו האידיאל הוא לקחת $m$ מאוד גדול אבל זה לא פרקטי כי לכפול מטריצות מאוד גדולות לוקח הרבה זמן. גם כאן, כמו קודם ניתן לעשות ניתוח תדירויות כאשר מסתכלים על כל $m$-יות האותיות שיכולות להיות, אלא שככל ש-$m$ נהיה יותר ויותר ההבדל בין התדירויות קטן ויש צורך בהרבה יותר טקסט מוצפן כדי לקבל תוצאות חד משמעיות. עוד פרט שכדאי לשים לה שאליו הוא שאם אנחנו לא יודעים את גודל מטריצת המפתח גם אין אפשרות לעשות ניתוח תדירויות.

### צופן Vigenère

ניצור טבלה שיש בה את כל צפני קיסר האפשריים:

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

כדי להצפין טקסט כלשהו בוחרים מילת מפתח. האות ה-$i$ במילת המפתח משמשת אינדקס לשורה בטבלת Vigenère שבאמצעותה נצפין את האות ה-$i$ בטקסט. אם מילת המפתח קצרה מדי

משתמשים בה באופן מעגלי. למשל נצפין את ATTACK FROM EAST באמצעות מילת המפתח NOTESHEAVEN:

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $M$: | $A$ | $T$ | $T$ | $A$ | $C$ | $K$ | $F$ | $R$ | $O$ | $M$ | $E$ | $A$ | $S$ | $T$ |
| $key\ index$: | $N$ | $O$ | $T$ | $E$ | $S$ | $H$ | $E$ | $A$ | $V$ | $E$ | $N$ | $N$ | $O$ | $T$ |
| $E(M_i)$: | $N$ | $H$ | $M$ | $E$ | $U$ | $R$ | $J$ | $R$ | $J$ | $Q$ | $R$ | $N$ | $G$ | $M$ |

גם כאן ניתן לומר שלמעשה מדובר בקוד מונואלפביתי ביחד לכל הצירופים האפשריים של אותיות באורך של מילת המפתח, אלא ששוב, ככל שמילת המפתח ארוכה יותר קשה יותר לעשות ניתוח תדירויות. למעשה, אם המפתח הוא באורך המידע אז הצופן בלתי ניתן לפענוח. כמו כן, כמו במקרה של צופן Hill, מאחר שאנחנו לא יודעים את אורך מילת המפתח אנחנו בבעיה...

### צפני הזזה

עד עכשיו דיברנו על צפני החלפה שבהם מחליפים כל אות בסימן אחר לפי כלל מסוים. בצופן הזזה פשוט מחליפים את הסדר של האותיות בהודעה לפי תמורה מסוימת שמהווה את המפתח. למשל, עבור התמורה

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 6 & 4 & 1 & 3 & 2 & 5 \end{pmatrix}$$

ההצפנה של DINAZIL תהיה LIADNIZ. כדי לפענח רק צריך להחיל את התמורה ההפוכה.

ברור שעבור הודעה באורך $n$ מרחב החיפוש הוא $n!$. גם אין הבטחה שתמורת המפתח היא התמורה היחידה אשר נותנת טקסט בעל משמעות בפענוח.

את האלגוריתם הזה כמובן כדאי לממש בחומרה, שהרי אז כל $n$ אותיות מוצפנות במחזור שעון אחד.

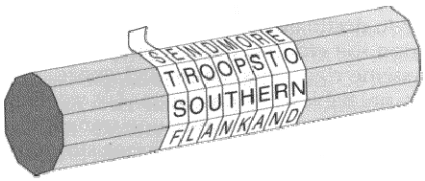
באלגוריתם הזה יש שלוש בעיות:

1. התמורה לא שוברת את הסטטיסטיקה של השפה (כך גם אפשר לגלות שנעשה שימוש בצופן הזה)
2. ככל ש-$n$ גדול יותר התוכנה או החומרה מסובכות יותר.
3. כמו במקרה של צופן "כל התמורות" קיימת בעיה של תיוג המפתחות והשמטת חלק מהם מאוסף המפתחות הקבילים.

מצד שני היתרונות הם שככל ש-$n$ גדל מרחב החיפוש גדל מהר מאוד וקשה לשבור את הצופן ע"י חיפוש ממצה.

### Spartan Scytale

בספרטה הייתה טכנולוגיה מתקדמת לייצור תמורות על טקסטים. היה להם מוט עץ בעל מספר פאות שכורכים סביבו רצועות עור או קלף כמו בציור:




השולח כותב את ההודעה לאורך פאות המוט על גבי העור או הקלף, ואחר כך מתיר את הרצועה, שכעת נראה שהיא נושאת רשימה של אותיות חסרות משמעות. ככה ההודעה מעורבלת. כדי לפענח את ההודעה, המקבל פשוט מלפף את הרצועה סביב גליל בעל קוטר זהה ואותו מספר פאות כמו זה שהשתמש בו השולח.

במקרה הזה ה"מפתח" של האלגוריתם הוא הקוטר ומספר הפאות.

השיטה הזו היא למעשה זהה לרישום המסר בשורות טבלה כלשהי (כאשר גודל הטבלה הוא הסוד) ולאחר מכן שכתוב שלו מחדש לפי העמודות. למשל,

ATTACK AT DAWN FROM SOUTHWEST →

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

→ AKWMHTANSWTTFOEADRUSCAOTT

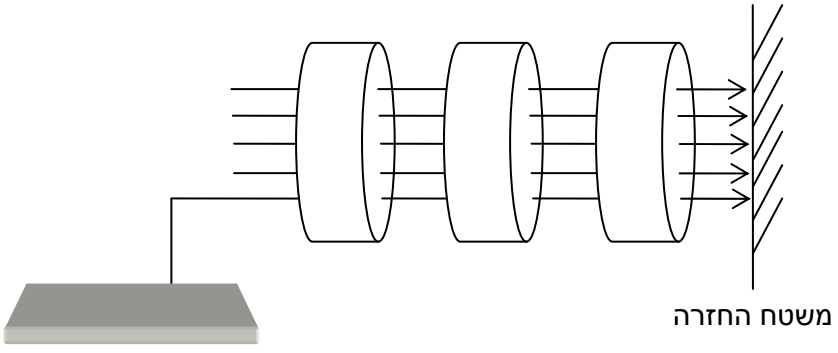
אפשר לשפר את האלגוריתם הזה ע"י הוספת סוד נוסף – תמורה על העמודות. במקום לכתוב את העמודות לפי הסדר אפשר להפעיל עליהם תמורה קודם. במקרה שלנו יש רק 5 עמודות וחיפוש ממצה בוודאי ישבור את הצופן בקלות (אלא אם לא יודעים את גודל הטבלה).כדי להגדיל את מרחב החיפוש אפשר לעשות את התהליך הזה כמה פעמים – בכל שלב לכתוב את הטקסט שהתקבל בשלב הקודם בשורות הטבלה, לבצע פרמוטציה על העמודות ואז לייצר טקסט חדש. עבור $k$ שלבים מרחב החיפוש הוא $(5!)^k$.

נשים לב שאם בטבלה יש $25$ תאים אז בכל שלב אנחנו יוצרים איזו תמורה עליהם. אז יכולנו לפתור את הבעיה ע"י חיפוש ממצפה על $25!$ תמורות אלא שזה מספר מאוד גדול ואם $k$ מספיק קטן אז $(5!)^k < 25!$ ועדיף לנו לנסות להתחקות אחרי תהליך ההצפנה.

### אניגמה

אניגמה היא מכונה מכאנית אשר הומצאה בגרמניה בתחילת המאה ה-20. היה בה שימוש מסחרי החל משנות ה-20 של המאה ה-20 אבל היא מפורסמת בעיקר בגלל השימוש שלה ע"י הנאצים במלחמת העולם השנייה.

למכונה יש שלושה גלילים ועל כל גליל 26 פינים לכניסה ו-26 פינים ליציאה אשר מחוברים בתמורה קבועה כלשהי. כל אחד מהגלילים יכול להסתובב ויש לו 26 מצבים. כל סיבוב של גליל נותן תמורה מסוימת ולכן סה"כ שלושת הגלילים יכולים לייצר $26^3$ תמורות לכל היותר, שהרי יכול להיות שחלק מהקונפיגורציות נותנות את אותה התמורה. אבל סביר להניח שכל התמורות האפשריות מתקבלות.




כדי להפעיל את המכונה כל מה שהיה צריך לעשות היה להקליד את המסר שנועד להצפנה על המקלדת ולכל אות הייתה נדלקת מנורה מתחת לאות שהייתה אמורה להצפין אותה. כאשר מקלידים אות אחד האות מהמקלדת היה עובר דרך כל הגלילים, מגע למשטח המחזיר וחוזר חזרה אל המקלדת אל אות כלשהי. שם היה נדלק אור והיו יודעים מהי ההצפנה. אחרי כל לחיצה הגלגלים היו מסתובבים ואחרי $26^3$ לחיצות המכונה חוזרת למצב ההתחלתי.

כדי לפענח צריך לעשות בדיוק אותו הדבר שהרי המכונה פועלת באופן סימטרי בגלל המשטח המחזיר. אם כשלוחצים על $x$ נדלקת האות $y$ אז בלחיצה על $y$ תידלק האות $x$ (בהנחה שהמכונה נמצאת באותו מצב כמובן).

ברור שהמכונה מייצרת קוד החלפה פוליאלפביתי, שהרי בגלל סיבוב הגלגלים אותה האות לא בהכרח תוצפן תמיד ע"י אותו סימן.

המפתח של האלגוריתם הזה הוא המצב ההתחלתי של הגלילים, ולכן יש $26^3$ מפתחות אפשריים וזה לא מספיק לצרכים צבאיים. לכן לגרמנים היו כל מיני שכלולים. בין השאר הם השתמשו בחמישה גלילים במקום בשלושה. פיסית במכונה היו משתמשים בשלושה אבל אז חלק מהמפתח היה אילו מחמשת הגלילים נמצאים בשימוש ומה הסדר שלהם במכונה. עם כל השכלולים שלהם הם הגיעו למרחב חיפוש בגודל כ-$10^{20}$.

כדי לעשות חיפוש ממצה על מרחב חיפוש כל כך גדול, אפילו אם בודקים כל מפתח בשנייה זה ייקח משהו כמו 13 שנים לבדוק את כל המפתחות. בוודאי זה לא מה שעשו במהלך המלחמה. האנגלים עבדו עם 60 מחשבים במקביל וע"י ניתוח של הקודים הם הצליחו בסופו של דבר לשבור את השיטה ולפענח את כל ההודעות של הגרמנים בזמן קצר.

## הצפנה מושלמת

עד עכשיו ראינו שיטות הצפנה שהיו קשות לשבירה. בזמן שהשתמשו בהן הן היו קשות מאוד לשבירה אבל בכל זאת זה היה אפשרי. שיטות כאלה נקראות **קריפטוגרפיה מותנית** (או **חישובית**). **קריפטוגרפיה לא מותנית** היא קריפטוגרפיה שלא תלויה בכושר החישוב של היריב, בכמות המחשבים, בכמות הטקסט או בכמות הזמן שמשקיעים בשבירה. שאלה מעניינת היא האם קיימת שיטת הצפנה שאי אפשר לשבור אותה, גם אם יינתנו זמן ומשאבים אינסופיים. למרבה ההפתעה התשובה היא חיובית.

לפני שנדון בשיטות הצפנה מושלמות יש להגדיר מה זה. אינטואיטיבית, משמעות מונח זה היא שיריב המיירט את ההודעה המוצפנת לא יכול להשיג כל מידע ביחס להודעה המקורית באמצעות ההודעה המוצפנת (ולכן, יירוט ההודעה לא תורם לו דבר). אפילו אם המתקיף יעשה חיפוש ממצה על כל מרחב החיפוש הוא עדיין לא יוכל לדעת מהי ההודעה המקורית. פורמאלית, ההגדרה היא שלכל הודעה מקורית $M$ ולכל הודעה מוצפנת $C$ ההסתברות ש-$C$ היא ההצפנה של $M$ שווה להסתברות לקבל את $M$ כהודעה מקורית, או בכתיב מתמטי $Pr(M) = Pr(M|C)$.

### הצפנה חד-פעמית – One-Time-Pad

נניח שיש לנו הודעה באורך $N$ ביטים שאנחנו רוצים להצפין. נבחר מפתח $K$ אקראי גם הוא באורך $N$ ביטים ונגדיר פונקציות הצפנה ופענוח באופן הבא:

$$E_K(M) = M \oplus K$$
$$D_K(C) = C \oplus K$$

כאשר $\oplus$ היא פעולת XOR של ביטים. נשים לב שההצפנה והפענוח אכן פונקציות הופכיות משום ש-$M \oplus K \oplus K = M$ לכל $M$.

נטען שאם המפתח סודי ומשתמשים בו רק פעם אחת אז ההצפנה הזו לא ניתנת לשבירה. מאחר שהמפתח אקראי לכל ביט במפתח יש $50\%$ סיכוי להיות $0$ ו-$50\%$ סיכוי להיות $1$. נניח שקיבלנו הודעה מוצפנת כלשהי. נסתכל על ביט כלשהו בהודעה. הביט המתאים במפתח הוא או $0$ או $1$ בהסתברות שווה. לכן הביט המקורי בהודעה המקורית הוא או $0$ או $1$ בהסתברות שווה. אז קיבלנו שלכל אחת מ-$2^N$ ההודעות האפשריות יש בדיוק אותה סיכוי מופע. אפילו חיפוש ממצה לא יעזור לנו במצב הזה. היינו רוצים לומר שנעשה חיפוש ממצה ונמצא את הטקסט בעל המשמעות, אלא כמו שכבר נאמר קודם כשדיברנו על צופן "כל התמורות" אין מניעה שנקבל כמה הודעות פוטנציאליות בעלות משמעות. לכל אחת מהן יש סיכוי זהה להיות ההודעה המקורית. לכן זוהי הצפנה מושלמת.

בשיטה הזו יש שני חסרונות עיקריים:

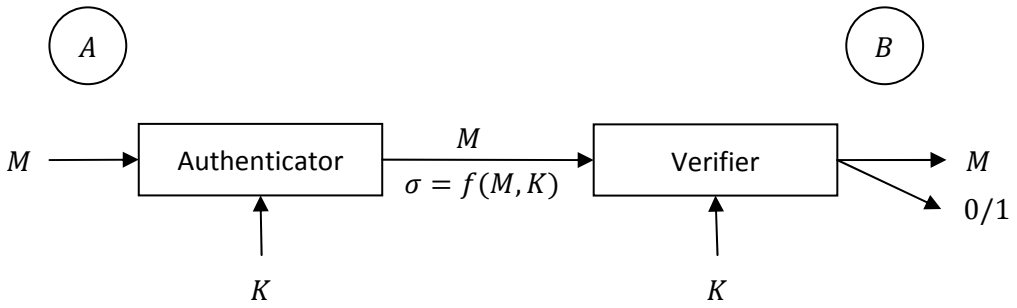
1. השיטה היא אגן מושלמת רק במקרה שהמפתח אקראי באמת. אבל ייצור מספרים אקראיים באמת הוא לא פשוט.
2. המפתח חייב להיות באורך ההודעה ועבור הודעה ארוכה זה דורש ייצור ותיאום מפתחות אורכים מה שיכול להיות לא פרקטי כל כך.

כעת, מדוע המפתח הוא חד-פעמי? ברגע שיש לנו הרבה הודעות שהוצפנו ע"י אותו מפתח אפשר להתחיל לחפש רצפים של אותיות שחוזרות על עצמן, לנחש מהן המילים האלה ולהתקדם ככה.

### אימות חד-פעמי – One-Time-MAC

כאשר מדברים על אימות הכוונה היא לא להסתרת תוכן ההודעה אלא שמירה על המקוריות שלה. כלומר, המטרה שלנו היא לוודא שההודעה תקינה ושאף אחד לא שינה אותה.

באופן כללי מה שקורה הוא ש-$A$ שולח הודעה ל-$B$ יחד עם משהו שנקרא חתימה. ל-$B$ יש יכולת לוודא שהחתימה של $A$ היא אמיתית.




אלגוריתם One-Time-MAC עובד באופן הבא: יש ראשוני גדול $p$ כך שההודעה שרוצים לשלוח מקיימת $0 < M < p$. לשני הצדדים $A$ ו-$B$ יש מפתח סודי חד-פעמי ואקראי $K = (a, b)$ כך ש-$1 \leq a, b < p$. פונקצית החתימה היא

$$\sigma = f(M, K) = (aM + b)\ mod\ p$$

ההודעה יחד עם החתימה עוברים למוודא ואם החישוב שלו זהה לחתימה שהגיעה סימן שההודעה תקינה.

מדוע זה עובד? אנחנו רוצים להראות שלמתקיף אין שום דרך לגלות את המפתח אפילו אם הוא ראה הודעה כלשהי יחד עם החתימה שלה.

ראשית, נשים לב שלפני שהמתקיף ראה איזשהו זוג $(M, \sigma)$ הוא יכול רק לנחש את המפתח בהסתברות $\frac{1}{(p-1)^2}$. זאת כמובן תחת ההנחה שהמפתח אקראי לגמרי.

כעת, נניח שהמתקיף ראה איזשהו זוג $(M, \sigma) = (M, aM + b)$. המפתח מורכב משני חלקים. בהינתן $b$ המתקיף יכול לחלץ מהמשוואה את $a$ המתאים. אבל מאחר שכל $b$ סביר באותה מידה $\frac{1}{p-1}$ זאת גם ההסתברות שבהינתן זוג כזה המתקיף יוכל לנחש את המפתח.

אז גם מראיית חתימה שלה הודעה כלשהי למתקיף אין שום דרך לדעת מהו המפתח הנכון, בלאו הכי לחתימה רנדומאלית שהוא יבחר יש סיכוי של $\frac{1}{p}$ להיות נכונה.

כעת, נשים לב שאם המתקיף רואה שתי הודעות שהוצפנו בעזרת אותו המפתח הוא יכול לחלץ אותו בקלות מהמשוואות:

$$\sigma_1 = aM_1 + b$$
$$\sigma_2 = aM_2 + b$$

אלה שתי משוואות בשני נעלמים בשדה $\mathbb{Z}_p$ ואנחנו יודעים שקיים להן פיתרון.

כאן החשיבות של החד-פעמיות של המפתח.

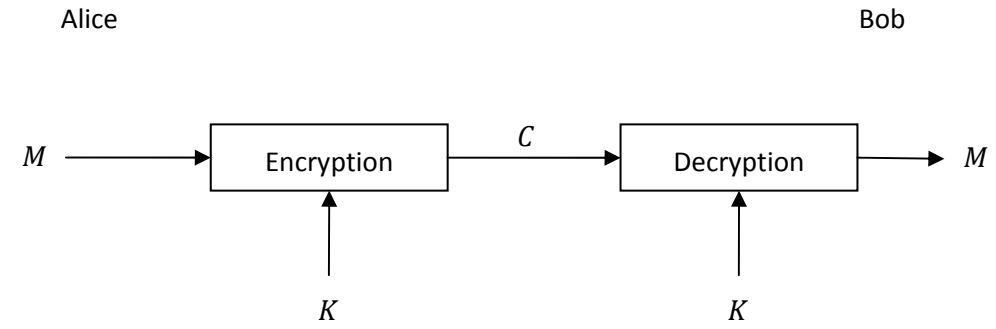
## קריפטוגרפיה מודרנית

העולם של הקריפטוגרפיה המושלמת הוא לא פרקטי כי המפתחות צריכים להיות מאוד ארוכים (כאורך ההודעות) ואקראיים באופן מושלם. הם גם חד-פעמיים. כל זה מקשה מאוד את השימוש בשיטות שהראנו קודם. לכן הקריפטוגרפיה המודרנית היא חישובית – אנחנו יוצאים מנקודת הנחה שליריב אין מספיק זמן או משאבים כדי לשבור את הצופן.

## קריפטוגרפיה סימטרית

בעולם ההצפנה הסימטרית ההנחה היא שלשני אנשים שרוצים לתקשר בצורה מוצפנת יש מפתח מתואם זהה $K$ שבעזרתו הם יכולים להצפין הודעות ולפענח הודעות מוצפנות או לאמת הודעות שנשלחו. בד"כ לשני הצדדים קוראים אליס $(A)$ ובוב $(B)$ ולצד המתקיף קוראים איב (מלשון Eavesdropper).

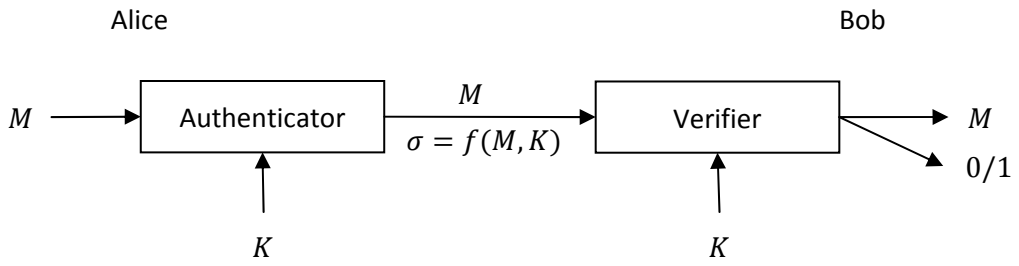
### הצפנה סימטרית




בד"כ אלגוריתם ההצפנה ואלגוריתם הפענוח דומים מאוד או אפילו זהים.

הבעיה עם המודל הזה היא שאליס ובוב צריכים איכשהו לתאם מפתחות, אבל במציאות יכול להיות שהם בכלל לא מכירים אחד את השני או חיים בצדדים שונים של העולם. חוץ מזה ככה לכל שני אנשים שרוצים לדבר אחד עם השני בצורה מאובטחת צריך להיות מפתח משלהם. אז אם יש $n$ אנשים וכל שניים מהם רוצים יכולת לדבר בצורה מאובטחת יש צורך ב-$n(n-1)$ מפתחות שונים.

### חתימה סימטרית




ההנחה היא שהעולם זדוני ורוצה לזייף את החתימה של אליס עם ההודעות שהיא שולחת לבוב לכן המפתח צריך להיות סודי וידוע רק לאליס ולבוב. כמובן, יש כאן אותה הבעיה כמו בהצפנה – אליס ובוב צריכים איכשהו לתאם מפתחות וגם מספר המפתחות הדרושים גדל ריבועים עם מספר האנשים שרוצים לתקשר אחד עם השני.

יש כאן עוד בעיה שאין בהצפנה היא שבוב יכול לזייף את החתימה של אליס, הרי לשניהם אותו מפתח. כך נוצר מצב שבו בוב יכול לחתום על מסמכים בשם אליס וזה משהו שאנחנו רוצים להימנע ממנו.

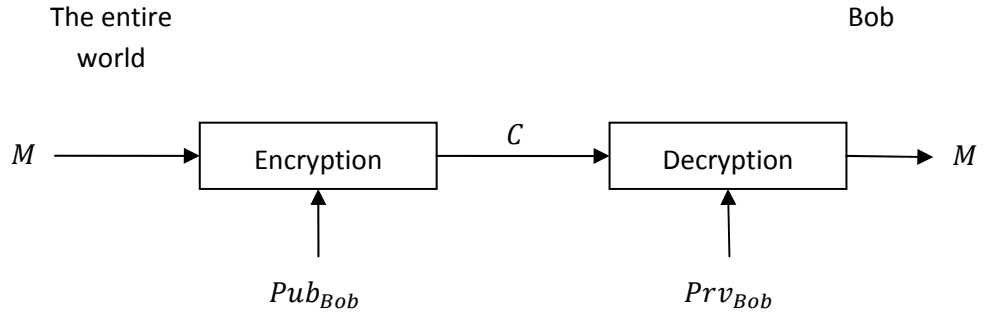
## קריפטוגרפיה אסימטרית

בעולם האסימטרי לכל ישות יש זוג מפתחות – מפתח ציבורי ומפתח פרטי. המפתח הציבורי משמש את שאר העולם כדי לשלוח הודעות ולאמת הודעות שהתקבלו ואילו המפתח הפרטי משמש לפענוח הודעות ולחתימה על הודעות יוצאות.

זה פותר את הבעייתיות של מספר המפתחות במקרה של קריפטוגרפיה סימטרית. כאן לאדם יש רק שני מפתחות וכל העולם יכול לשלוח לו הודעות מוצפנות והוא יכול לשלוח הודעות חתומות וכל אחד יכול לוודא את החתימה.

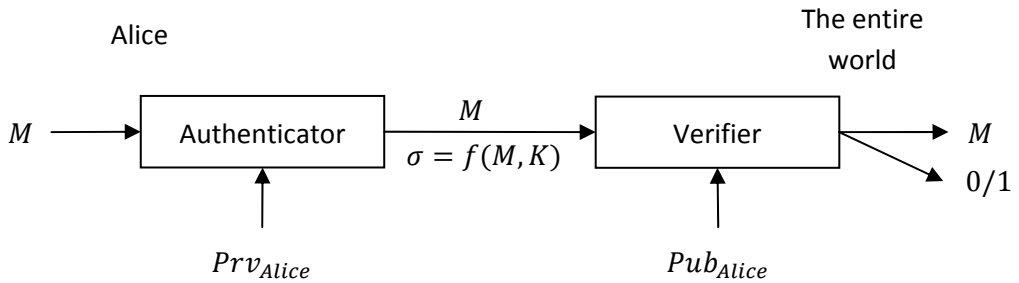
מצד שני, עדיין לא ברור איך מתבצעת הפצת המפתחות ציבוריים.

### הצפנה




כל מי שרוצה לשלוח הודעה לבוב יכול להשתמש במפתח הציבורי של בוב וכדי לפענח בוב צריך להשתמש במפתח הפרטי שלו. ההנחה היא כמובן שקשה מאוד לפענח הודעה מוצפנת בלי המפתח הפרטי ושקשה מאוד לגלות את המפתח הפרטי מתוך ידיעת המפתח הציבורי.

### חתימה




כמו במקרה של ההצפנה המטרה היא שרק אליס תוכל לחתום על ההודעות שלה אבל כל אחד יוכל לוודא את החתימה שלה. שוב, המפתח הפרטי צריך להיות קשה לגילוי בהינתן המפתח הציבורי והחתימה צריכה להיות קשה לזיוף בהינתן המפתח הציבורי.

כאן פתרנו את הבעיה של המקרה הסימטרי שבו בוב יכול לזייף את החתימה של אליס. במודל האסימטרי, אליס יכולה לשלוח הודעות לבוב ולחתום עליהם. בוב יכול לוודא את החתימה שלאליס אבל הוא עצמו לא יכול לחתום בשמה.

### קריפטוגרפיה סימטרית לעומת קריפטוגרפיה אסימטרית

| | קריפטוגרפיה סימטרית | קריפטוגרפיה אסימטרית |
|---|---|---|
| **ניהול מפתחות** | יש צורך בתיאום מפתחות בין כל שני אנשים | יש צורך רק בהפצת מפתחות ולא תיאום |
| **מחיר** | יקר יותר מהמודל האסימטרי כי כל שני אנשים צריכים מפתח סודי משלהם | לכל ישות יש רק שני מפתחות |
| **אי התכחשות** | אי אפשר להשיג אי התכחשות כי אליס יכולה לטעון שבוב חתם במקומה | ניתן להשיג אי התכחשות כי רק לישות עם המפתח הפרטי יש אפשרות לחתום על הודעות |

### אלגוריתמים טובים ומפתחות טובים

יש כל מיני אלגוריתמים קריפטוגרפים. למשל:

- אלגוריתמים כלליים ידועים
- אלגוריתמים שפותחו ע"י חברה
- אלגוריתמים שפותחו ע"י הממשלה
- אלגוריתמים שפותחו ע"י יועץ אבטחה
- אלגוריתמים שפיתחנו בעצמנו

בד"כ רצוי להשתמש באלגוריתמים הכלליים הידועים. מאוד קשה לפתח אלגוריתמים קריפטוגרפיים וסביר שאם נמציא משהו תוך כמה ימים זה יתגלה כלא טוב. האלגוריתמים הידועים נבחנו במשך עשרות שנים ע"י מאות חוקרים. אלה אלגוריתמים חישוביים והדרך הכי טובה להיות בטוחים בהם היא שמאות ואלפי אנשים ניסו לפרוץ אותם ולא הצליחו (או לפחות לא סיפרו שהצליחו).

כל האלגוריתמים מתבססים על מפתח שהוא הסוד. מפתח טוב הוא לא קצר מדי (כי אז אפשר לעשות חיפוש ממצה) ולא ארוך מדי (כי אז זה לא פרקטי כל כך). כמו כן, רצוי שהמפתח יהיה אקראי לגמרי כי זה מקשה מאוד את הניחוש.

### דחיסה לעומת הצפנה

קובץ דחוס נראה לא ברור אבל זוהי לא הצפנה, שהרי ברגע שאלגוריתם הדחיסה ידוע אין בעיה לשחזר את הקובץ. בדחיסה אין מפתח ולכן ברגע שהאלגוריתם הוא לא סודי אין שום אלמנט של סודיות בדחיסה.

בעניין הדחיסה רק צריך לשים לב שאם רוצים גם לדחוס וגם להצפין קודם צריך לדחוס את הקובץ ורק אח"כ להצפין אותו. יש לזה שתי סיבות:

1. דחיסה מתבססת מבחינה מסוימת על מציאת חוקיות כלשהי בטקסט. אם ההצפנה טובה היא מערבלת את הטקסט לגמרי והוא נראה אקראי. טקסט אקראי אי אפשר לדחוס.
2. אחרי הדחיסה הקובץ מאבד מהחוקיות שלו במה שקשור לסטטיסטיקות על השפה, ולכן אחרי ההצפנה אי אפשר להשתמש בניתוח תדירויות כדי לשבור את הצופן.

### סוגי התקפות

- התקפות לא קריפטוגרפיות (אלה ההתקפות שבד"כ עושים היום משום שהאלגוריתמים הקריפטוגרפים הם חזקים למדי):
  - פריצה למחשב או לחדר
  - מציאת מפתח או סיסמה

- ניחוש מפתח או סיסמה

- התקפות קריפטוגרפיות (חוזק האלגוריתם נמדד מול התקפות מסוג זה):
  - התקפת **known ciphertext** – התקפה שבה ידוע הטקסט המוצפן. לא בלתי סביר להניח שזהו תמיד המצב. הרי אם אף אחד בכלל לא מאזין אין טעם להצפין. נשאלת כאן השאלה אם ניתן להבין את המפתח מהטקסט המוצפן בלבד או לפחות את הטקסט הגלוי. למשל במקרה של צופן קיסר זה בוודאי אפשרי.
  - התקפת **known plaintext** – התקפה שבה ידוע גם הטקסט המוצפן וגם הטקסט המקורי ובה למעשה מנסים למצוא את המפתח.
  - התקפת **chosen plaintext** – התקפה שבה למתקיף יש אפשרות לבחור את הטקסט שאותו מצפינים. כל צופן הצפנה מונואלפביתי לא עמיד בפני התקפה כזאת כי ע"י בחירת כל אותיות הא"ב ניתן לקבל מילון לפענוח!

האלגוריתמים המודרניים בד"כ לא חשופים להתקפות קריפטוגרפיות.

## סוגי צפנים מודרניים

יש שני סוגים מרכזיים של צפנים שהעולם המודרני משתמש בהם:

1. **Block ciphers** – המפתח הוא בלוק שבד"כ קבוע לתקופת זמן ארוכה. ההודעה מחולקת לבלוקים בגודל קבוע וכל בלוק מצפינים בנפרד ע"י אותו המפתח. אם אורך ההודעה לא מתחלק באורך הבלוק מוסיפים ריפוד.
2. **Stream ciphers** – מסתכלים על ההודעה כרצף ביטים שמגיע כל הרצף ולא מחולק לבלוקים. כל ביט מקודד בנפרד ובד"כ טרנספורמציית ההצפנה משתנה מביט לביט.

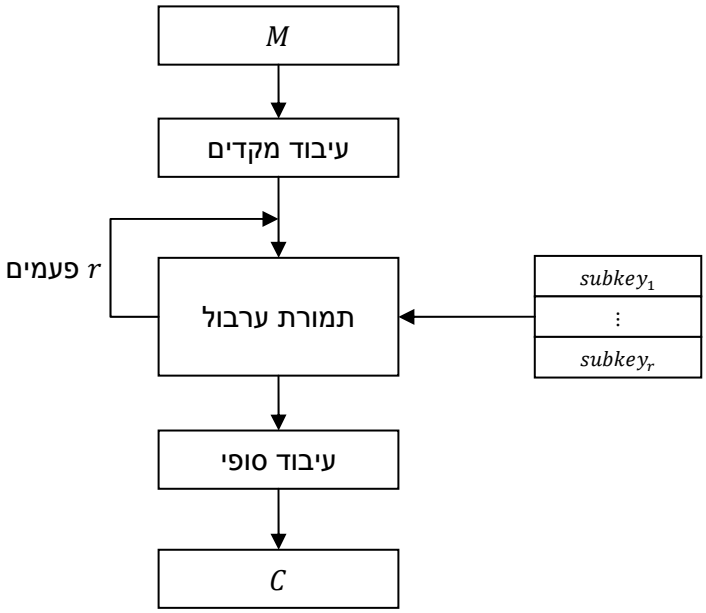
### צפני בלוק

כפי שאמרנו, בצופן בלוק מחלקים את ההודעה לבלוקים ומצפינים כל חלק בנפרד באמצעות אותו המפתח. זה קצת דומה לצופן מונואלפביתי אלא שכאן לכל בלוק יש הצפנה קבועה ולא לכל אות. אז בהינתן מספיק הודעות מוצפנות אפשר לעשות ניתוחים סטטיסטיים. גדלים אופייניים של בלוק טקסט גלוי הם 64, 128 או 256 ביטים. בד"כ הבלוק המוצפן שווה באורכו לבלוק הגלוי. אפילו אם אנחנו עובדים עם בלוקים בגודל 64 ביט, יש $2^{64}$ בלוקים אפשריים. אז אם נסתכל על זה כצופן מונואלפביתי למעשה אנחנו צריכים לבחור אחת מתוך $2^{64}!$ התמורות הקיימות ואז אורך המפתח הוא

$$\log(2^{64}!) \approx 2^{64}\log 2^{64} = 64 \cdot 2^{64} = 2^{70}$$

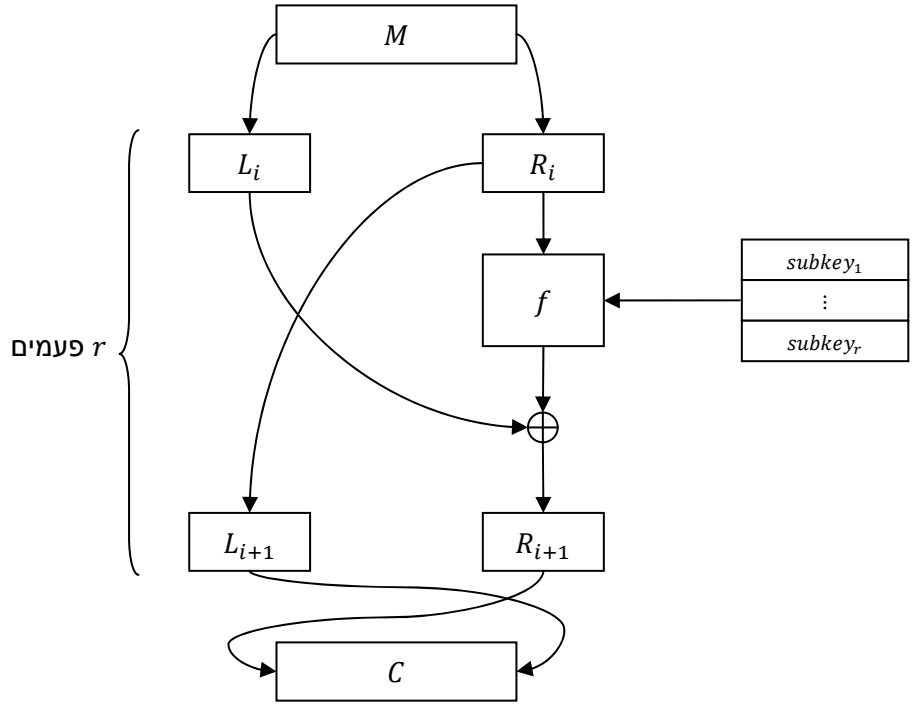
וזה מספר עצום! בד"כ אורך המפתח הוא 64, 128, 192 או 256 ביט וזה אומר שלא עובדים עם כל התמורות. אבל כפי שכבר אמרנו מספר פעמים לא כל התמורות הן טובות. בחירת עולם תמורות קומפקטי אך יעיל היא קשה!

מבנה כללי של צופן בלוקים הוא כזה:




יש $r$ איטרציות של ערבולים שעושים על הבלוק ובכל איטרציה משתמשים במפתח שונה שנגזר מהמפתח של האלגוריתם. האיטרציות למעשה מרכיבות את התמורות אחת על השנייה ובעזרת קבוצת תמורות בסיסיות אפשר באמצעות הרכבה ליצור מרחב תמורות הרבה יותר גדול. מטרת הערבול המקדים והסופי היא לייעל את ההצפנה אבל לא תמיד עושים אותם.

### סכמת פייסטל




מחלקים את ההודעה לשניים. את החלק הימני מעבירים ישירות לרגיסטר שמאלי חדש ואילו את החלק השמאלי מעבירים אחרי סדרה של פעולות לרגיסטר ימני חדש. את רצף הפעולות הזה מבצעים $r$ פעמים ולבסוף מאחדים את הרגיסטרים לתוצאה בסדר הפוך.

הפענוח עובד בדיוק באותו אופן אלא שמכניסים לסכמה את $C$ כקלט ומשתמשים במפתחות בסדר הפוך.

מה שאנחנו רואים כאן הוא לא אלגוריתם אלא רק סכמה כללית. הסכמה לא אומרת לנו מהם אורכי הבלוקים, לא מהו $r$ ולא מהי הפונקציה $f$. למעשה כל פונקציה תתאים לסכמה, אלא שלא כל הפונקציות יעילות באותה מידה. למשל אם הפונקציה היא הפונקציה הקבועה $f \equiv 0$ אז התוצאה היא היפוך הצדדים של ההודעה בהתאם לזוגיות של $r$ וזה משהו שאנחנו כמובן לא מעוניינים בו.

מה שלמעשה יוצר את ההצפנה הוא פעולת ה-XOR ולכן היינו רוצים שתוצאת הפונקציה $f$ תהיה אקראית ככל הניתן, בדומה למה שקורה באלגוריתם One-Time-Pad. אבל זה כמובן בלתי אפשרי כי מדובר במכונה דטרמיניסטית, אז עושים את המקסימום האפשרי.

## Data Encryption Standard

בשנת 1975 פורסם Data Encryption Standard (בקיצור DES) שהיה תוצאה של עבודה משותפת של חברות אמריקאיות גדולות ואפילו ה-NSA הכניסו לו כמה שיפורים. מעניין לציין שהדרישה של גוף התקינה האמריקאי הייתה שהאלגוריתם יהיה יעיל בחומרה אבל איטי בתוכנה, וזאת משום שבזמנו ראו בהצפנה סוג של כלי נשק. ייצור תוכנה קשה יותר וגם קשה יותר להעתיק אותה, ואילו על תוכנה הרבה יותר קשה לשלוט. היום זה כמובן פחות רלוונטי אבל לפני 30 שנה הייתה לזה משמעות רבה.

DES למעשה נותן פירוט מלא לאלגוריתם שמבוסס על סכמת פייסטל. גודל הבלוק המוצפן הוא 64 ביט, מספר האיטרציות הוא 16 וגודל המפתח הוא 64 ביט. למעשה גודל המפתח הוא 56 ביט משום שכל ביט שמיני משמש לבדיקת זוגיות ולא לצורך בניית התת-מפתחות. כל תת מפתח הוא בגודל 48 ביט.

העיבוד המקדים הוא איזושהי תמורה התחלתית הנתונה ע"י

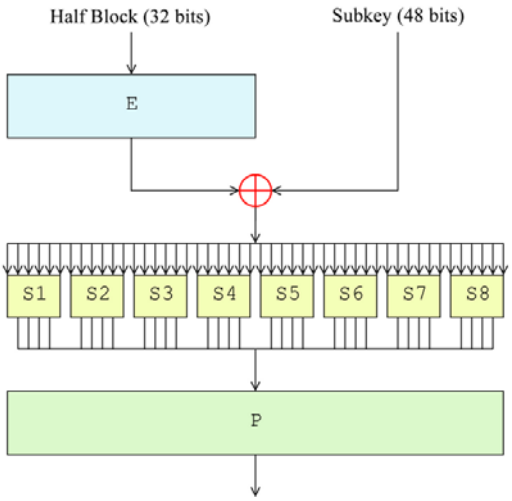
$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 \\ 58 & 50 & 42 & 34 & 26 & 18 & 10 & 2 & 60 & 52 & 44 & 36 & 28 & 20 & 12 & 4 & 62 & 54 & 46 & 38 & 30 & 22 & 14 & 6 & 64 & 56 & 48 & 40 & 32 & 24 & 16 & 8 \end{pmatrix}$$

$$\begin{pmatrix} 33 & 34 & 35 & 36 & 37 & 38 & 39 & 40 & 41 & 42 & 43 & 44 & 45 & 46 & 47 & 48 & 49 & 50 & 51 & 52 & 53 & 54 & 55 & 56 & 57 & 58 & 59 & 60 & 61 & 62 & 63 & 64 \\ 57 & 49 & 41 & 33 & 25 & 17 & 9 & 1 & 59 & 51 & 43 & 35 & 27 & 19 & 11 & 3 & 61 & 53 & 45 & 37 & 29 & 21 & 13 & 5 & 63 & 55 & 47 & 39 & 31 & 23 & 15 & 7 \end{pmatrix}$$

והעיבוד הסופי הוא התמורה ההופכית שלה. העיבוד הזה קיים משתי סיבות:

1. תמורה זה משהו שקל לממש בחומרה בסיבוב שעון אחד אבל בתוכנה זה יכול להיות מאוד איטי ולכן זה עונה על הדרישה של גוף התקינה.
2. הטקסטים ששולחים מורכבים בד"כ ממספרים ומאותיות ולכן קידוד ה-ASCII שלהם לא משתמש בביט השביעי והשמיני. זה נותן איזושהי מבניות לקלט שאולי אפשר לנצל כדי לשבור את הצופן. לכן התמורה ההתחלתית מערבלת את הקלט ובכך משפרת את ההצפנה.

הפונקציה $f$ פועלת באופן הבא:




הפונקציה מקבלת כקלט 32 ביט ומורכבת מארבעה שלבים:

1. הרחבה – הקלט מורחב ל-48 ביטים ע"י שכפול חלק מהביטים.
   אם הקלט הוא ביטים $1, \dots, 32$ הפלט הוא:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 32 | 1 | 2 | 3 | 4 | 5 | 4 | 5 | 6 | 7 | 8 | 9 | 8 | 9 | 10 | 11 | 12 | 13 | 12 | 13 | 14 | 15 | 16 | 17 | 16 | 17 | 18 | 19 | 20 | 21 | 20 | 21 | 22 | 23 | 24 | 25 | 24 | 25 | 26 | 27 | 28 | 29 | 28 | 29 | 30 | 31 | 32 | 1 |

2. ערבול עם התת-מפתח המתאים לאיטרציה בעזרת פעולת XOR.
3. החלפה – הרגיסטר המעורבל מחולק ל-8 חלקים של 6 ביט וכל חלק נכנס ל-S-box אשר מחליף את הקלט בארבעה ביטים חדשים בעזרת הטבלה הנתונה:

| $S_1$ | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 14 | 4 | 13 | 1 | 2 | 15 | 11 | 8 | 3 | 10 | 6 | 12 | 5 | 9 | 0 | 7 |
| 0 | 15 | 7 | 4 | 14 | 2 | 13 | 1 | 10 | 6 | 12 | 11 | 9 | 5 | 3 | 8 |
| 4 | 1 | 14 | 8 | 13 | 6 | 2 | 11 | 15 | 12 | 9 | 7 | 3 | 10 | 5 | 0 |
| 15 | 12 | 8 | 2 | 4 | 9 | 1 | 7 | 5 | 11 | 3 | 14 | 10 | 0 | 6 | 13 |

| $S_2$ | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | 1 | 8 | 14 | 6 | 11 | 3 | 4 | 9 | 7 | 2 | 13 | 12 | 0 | 5 | 10 |
| 3 | 13 | 4 | 7 | 15 | 2 | 8 | 14 | 12 | 0 | 1 | 10 | 6 | 9 | 11 | 5 |
| 0 | 14 | 7 | 11 | 10 | 4 | 13 | 1 | 5 | 8 | 12 | 6 | 9 | 3 | 2 | 15 |
| 13 | 8 | 10 | 1 | 3 | 15 | 4 | 2 | 11 | 6 | 7 | 12 | 0 | 5 | 14 | 9 |

| $S_3$ | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 10 | 0 | 9 | 14 | 6 | 3 | 15 | 5 | 1 | 13 | 12 | 7 | 11 | 4 | 2 | 8 |
| 13 | 7 | 0 | 9 | 3 | 4 | 6 | 10 | 2 | 8 | 5 | 14 | 12 | 11 | 15 | 1 |
| 13 | 6 | 4 | 9 | 8 | 15 | 3 | 0 | 11 | 1 | 2 | 12 | 5 | 10 | 14 | 7 |
| 1 | 10 | 13 | 0 | 6 | 9 | 8 | 7 | 4 | 15 | 14 | 3 | 11 | 5 | 2 | 12 |

| $S_4$ | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | 13 | 14 | 3 | 0 | 6 | 9 | 10 | 1 | 2 | 8 | 5 | 11 | 12 | 4 | 15 |
| 13 | 8 | 11 | 5 | 6 | 15 | 0 | 3 | 4 | 7 | 2 | 12 | 1 | 10 | 14 | 9 |
| 10 | 6 | 9 | 0 | 12 | 11 | 7 | 13 | 15 | 1 | 3 | 14 | 5 | 2 | 8 | 4 |
| 3 | 15 | 0 | 6 | 10 | 1 | 13 | 8 | 9 | 4 | 5 | 11 | 12 | 7 | 2 | 14 |

| $S_5$ | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 12 | 4 | 1 | 7 | 10 | 11 | 6 | 8 | 5 | 3 | 15 | 13 | 0 | 14 | 9 |
| 14 | 11 | 2 | 12 | 4 | 7 | 13 | 1 | 5 | 0 | 15 | 10 | 3 | 9 | 8 | 6 |
| 4 | 2 | 1 | 11 | 10 | 13 | 7 | 8 | 15 | 9 | 12 | 5 | 6 | 3 | 0 | 14 |
| 11 | 8 | 12 | 7 | 1 | 14 | 2 | 13 | 6 | 15 | 0 | 9 | 10 | 4 | 5 | 3 |

| $S_6$ | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 12 | 1 | 10 | 15 | 9 | 2 | 6 | 8 | 0 | 13 | 3 | 4 | 14 | 7 | 5 | 11 |
| 10 | 15 | 4 | 2 | 7 | 12 | 9 | 5 | 6 | 1 | 13 | 14 | 0 | 11 | 3 | 8 |
| 9 | 14 | 15 | 5 | 2 | 8 | 12 | 3 | 7 | 0 | 4 | 10 | 1 | 13 | 11 | 6 |
| 4 | 3 | 2 | 12 | 9 | 5 | 15 | 10 | 11 | 14 | 1 | 7 | 6 | 0 | 8 | 13 |

| $S_7$ | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4 | 11 | 2 | 14 | 15 | 0 | 8 | 13 | 3 | 12 | 9 | 7 | 5 | 10 | 6 | 1 |
| 13 | 0 | 11 | 7 | 4 | 9 | 1 | 10 | 14 | 3 | 5 | 12 | 2 | 15 | 8 | 6 |
| 1 | 4 | 11 | 13 | 12 | 3 | 7 | 14 | 10 | 15 | 6 | 8 | 0 | 5 | 9 | 2 |
| 6 | 11 | 13 | 8 | 1 | 4 | 10 | 7 | 9 | 5 | 0 | 15 | 14 | 2 | 3 | 12 |

| $S_8$ | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 13 | 2 | 8 | 4 | 6 | 15 | 11 | 1 | 10 | 9 | 3 | 14 | 5 | 0 | 12 | 7 |
| 1 | 15 | 13 | 8 | 10 | 3 | 7 | 4 | 12 | 5 | 6 | 11 | 0 | 14 | 9 | 2 |
| 7 | 11 | 4 | 1 | 9 | 12 | 14 | 2 | 0 | 6 | 10 | 13 | 15 | 3 | 5 | 8 |
| 2 | 1 | 14 | 7 | 4 | 10 | 8 | 13 | 15 | 12 | 9 | 0 | 3 | 5 | 6 | 11 |

כדי להשתמש בטבלאות מסתכלים על כל קלט של ששה ביטים כשני ביטים חיצוניים בתוספת ארבעה ביטים פנימיים. הפלט של הטבלה נקבע ע"י בחירת השורה שמתאימה

לביטים החיצוניים והטור שמתאים לביטים הפנימיים, כאשר ספירת השורות והטורים נעשית מ-0 ועד 3 ו-15 בהתאמה. למשל אם הקלט של הקופסה הראשונה הוא 101001 אז הביטים החיצוניים הם 11 שמתאימים לשורה 3 והביטים הפנימיים הם 0100 שמתאימים לשורה 4. אז הפלט יהיה 4 כלומר 0100.

4. 32 הביטים שיוצאים מכל ה-S-boxes עוברים תמורה אחרונה:

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 & 17 & 18 & 19 & 20 & 21 & 22 & 23 & 24 & 25 & 26 & 27 & 28 & 29 & 30 & 31 & 32 \\ 16 & 7 & 20 & 21 & 29 & 12 & 28 & 17 & 1 & 15 & 23 & 26 & 5 & 18 & 31 & 10 & 2 & 8 & 24 & 14 & 32 & 27 & 3 & 9 & 19 & 13 & 30 & 6 & 22 & 11 & 4 & 25 \end{pmatrix}$$

המטרה של ההרחבה והפרמוטציה האחרונה היא ליצור תמורה טובה – תמורה שנראית אקראית – כך שלכל ביט שמשתנה יש השפעה. גם לשינוי קטן בקלט יש השפעה גדולה וגם לשינוי קטן במפתח יש השפעה גדולה. במידה מסוימת זה גורם לכל ביט "להגיע לכל מקום".

מספר האיטרציות של האלגוריתם נבחר להיות 16 אחרי מחקר מעמיק. שום דבר באלגוריתם הזה לא נבחר סתם!

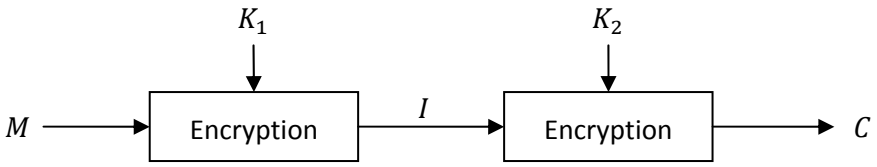
התרומה של ה-NSA לאלגוריתם הייתה בהגדרת הטבלה של ה-S-boxes. במשך שנים היה חשש שה-NSA הכניס את השינויים האלה כדי שיוכלו לפענח תכתובת אזרחית. אבל אחרי 20 שנה עדי שמיר ואלי ביהם פרסמו מאמר שהראה שדווקא השינויים של ה-NSA הכניס היו לטובה ואלה ה-S-boxes הכי טובים שיכולים להיות עבור האלגוריתם הזה.

מה שה-NSA עושו כדי לשמור את הביטחון הלאומי הוא להגביל את גודל המפתח. אין טעם להרוס את האלגוריתם כי במוקדם או במאוחר מישהו יגלה את הפרצה ואז האלגוריתם ישתנה ותהיה להם בעיה. הגבלת אורך המפתח הרבה יותר נוחה. המפתח ארוך מספיק כדי שאזרחים מן השורה לא יוכלו לפצח אותו בעזרת חיפוש ממצה אבל ל-NSA יש הרבה מחשבים חזקים במרתף... ☺

אגב, היום האלגוריתם הזה הוא חלש משום שלא רק ל-NSA יש מחשבים חזקים. האלגוריתם דווקא ממש טוב, המפתח שלו פשוט קצר מדי...

### Double DES

כפי שאמרנו, הבעיה עם DES היא גודל המפתח. כדי לפתור את הבעיה אפשר פשוט לבצע את האלגוריתם פעמיים וכל פעם עם מפתח שונה. למעשה זה משהו שאפשר לעשות עם כל אלגוריתם הצפנה באשר הוא, או אפילו בשילוב של שני אלגוריתמים:




אלא שגם כאן יש בעיה. לכל אלגוריתם הצפנה E האלגוריתם Double E חשוף להתקפה **Meet-in-the-Middle**. נניח שההתקפה היא known plaintext. נעשה חיפוש ממצה של $K_1$. נרשום את כל האפשרויות להצפנה של $M$ ע"י כל המפתחות $K_1$ האפשריים:

$$X_1 = E_{K_1^1}(M)$$
$$\vdots$$
$$X_{2^{|K_1|}} = E_{K_1^{2^{|K_1|}}}(M)$$

כעת נעשה חיפוש ממצפה על $K_2$. נרשום את כל הפענוחים האפשריים ל-$C$ ע"י המפתחות $K_2$ האפשריים:

$$Y_1 = D_{K_2^1}(C)$$
$$\vdots$$
$$Y_{2^{|K_2|}} = D_{K_2^{2^{|K_2|}}}(C)$$

כעת כל מה שנותר לעשות הוא למצוא זוג $(X_i, Y_j)$ כך ש-$X_i = Y_j$. נבדוק את הסיבוכיות של האלגוריתם. כדי לחפש את $K_1$ השתמשנו ב-$O(2^{|K_1|})$ פעולות וכדי לחפש את $K_2$ השתמשנו ב-$O(2^{|K_2|})$ פעולות. כדי למצוא התנגשויות של זוגות צריך למיין אותם אז זה לוקח משהו כמו $O(|K_1|\log|K_1| + |K_2|\log|K_2|)$ ולבסוף מציאת ההתאמות לוקח $O(|K_1| + |K_2|)$. אז בסה"כ ההתקפה לוקחת

$$O\left(2^{|K_1|} + 2^{|K_2|} + |K_1|\log|K_1| + |K_2|\log|K_2| + |K_1| + |K_2|\right)$$

וזה הרבה פחות מחיפוש ממצה של כל המפתחות $K_1, K_2$ שלוקח $O(2^{|K_1+K_2|})$.

צריך רק לשים לב שיכולים להיות הרבה זוגות מתאימים, אבל ככל שנבצע את התהליך הזה על יותר זוגות $(M, C)$ כך גדל הסיכוי שלנו למצוא את המפתח הנכון. בכל שלב אנחנו גם צריכים לבחון רק את המפתחות המועמדים שמצאנו בשלב הקודם אז זה מקטין עוד יותר את כמות החישובים שיש לעשות.

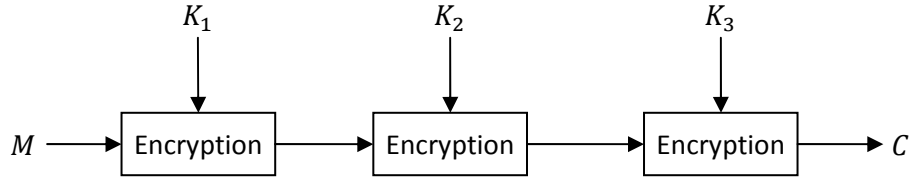
נטען שאם עובדים עם DES אחרי השלב הראשון יש כ-$2^{48}$ זוגות אפשריים. המפתח הוא באורך 64 ביט אבל למעשה משתמשים רק ב-56 מתוכם. אז בהנחה שהאלגוריתם יוצר התפלגות אחידה על $I$ יש שם כ-$2^{56}$ הודעות אפשריות. גם כשעוברים מ-$C$ ל-$I$ מתקבלות כ-$2^{56}$ הודעות. נבדוק עבור $X_1$ את הסיכוי שיש $Y_i$ שמתאים לו. כאשר מחשבים $Y_i$ ספציפי הסיכוי שהוא יפגע בתוך קבוצת ה-$X$-ים הוא $\frac{2^{56}}{2^{64}}$. אחרי $2^{56}$ ניסיונות הסיכוי שנמצא התאמה הוא $2^{56} \cdot \frac{2^{56}}{2^{64}} = 2^{48}$.

עכשיו ננית שנתון לנו זוג חדש $(M', C') \neq (M, C)$. נבדוק אילו זוגות מתוך ה-$2^{48}$ מתאימים לזוג החדש. זוג $(K_1, K_2)$ עובר את המבחן אמ"מ $E_{K_1}(M') = D_{K_2}(C')$. זה קורה בהסתברות $\frac{1}{2^{64}}$ שהרי אם ננית ש-$E_{K_1}(M') = y \in I$ וננית ש-$D_{K_2}$ יוצרת התפלגות אחידה על $I$ אז הסיכוי ש-$D_{K_2}(C') = y$ הוא $\frac{1}{|I|} = \frac{1}{2^{64}}$. לכן הסיכוי שלפחות אחד מהזוגות יעבור את המבחן הוא לכל היותר $\frac{2^{48}}{2^{64}}$ וזה מספר די קטן. לכן סביר מאוד שבשני סיבובים של ההתקפה נמצא את המפתח הנכון.

בעיה נוספת עם אלגוריתמים כפולים כאלה היא שייתכן ש-$E_{K_2}\left(E_{K_1}(M)\right) = E_{K_3}(M)$ ואז לא צריך אפילו להשתמש בהתקפת meet-in-the-middle. מספיק רק לעשות חיפוש ממצה את המפתחות. אבל ספציפית במקרה של DES לא אפשרי ולכן אנחנו לא מודאגים מזה.

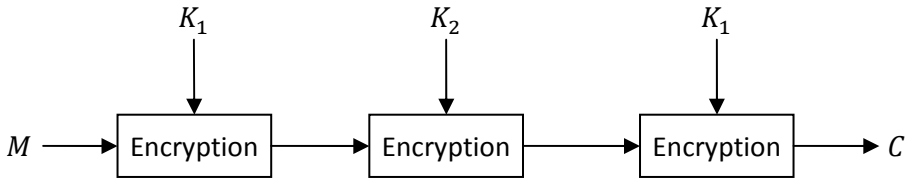
## 3-DES

### *EEE Mode*



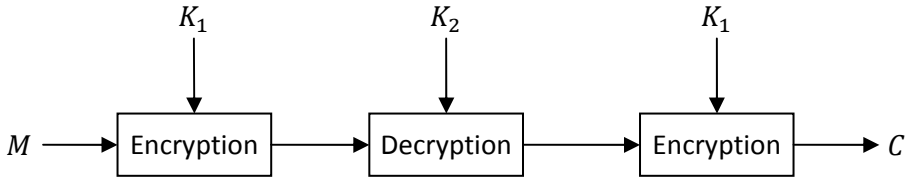

לכאורה יש כאן ביטחון של $2^{|K_1|+|K_2|+|K_3|}$ אבל למעשה ניתן לעשות כאן התקפה דומה ל-meet-in-the-middle ובמקרה של DES ניתן לשבור את זה בזמן $O(2^{2\cdot 56})$: נחשב את כל האפשרויות לזוגות $(K_1, K_2)$ וננסה למצוא את ההתנגשויות בין ההצפנה השנייה לשלישית.

נשים לב שאנחנו משלמים על שלושה מפתחות אבל למעשה הביטחון שמקבלים הוא רק של שניים. אז בד"כ משתמשים באחד מהמפתחות פעמיים:




***EDE Mode***

אפשר לשלב את DES עם 3-DES בחבילה אחת באופן הבא:

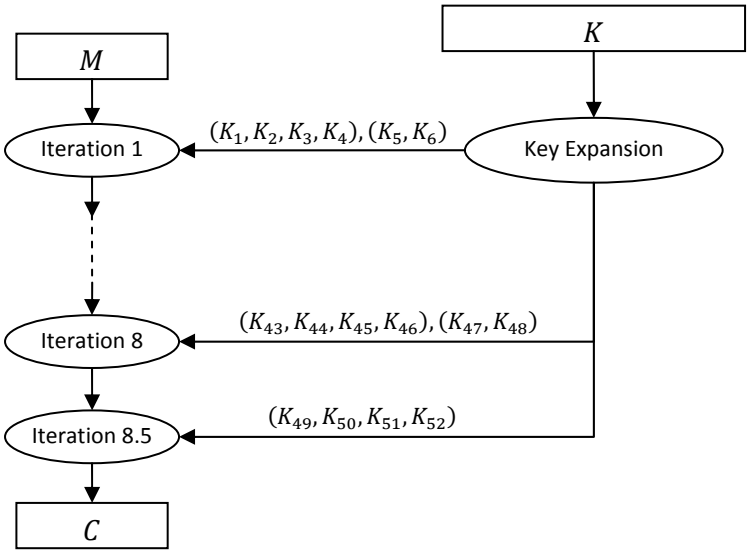



מאחר שההצפנה והפענוח של DES הם זהים עד כדי החלפת סדר המפתחות אפשר להשתמש בסכמה למעלה בשני אופנים. אם $K_1 = K_2$ אז מתקבל סתם DES ואם $K_1 \neq K_2$ מתקבל 3-DES. זה נוח כי זה נותן תאימות לקבצים שונים באותה התוכנה או החומרה.

## International Data Encryption Algorithm

אלגוריתם נוסף שפותח בשנות ה-80 של המאה הקודמת הוא ה- International Data Encryption Algorithm (בקיצור IDEA). הוא לא סטנדרטי באופן פורמאלי אבל בכל זאת משתמשים בו בהרבה מקומות.

האלגוריתם פועל באופן הבא: הבלוקים שנועדים להצפנה הם בגודל 64 ביט והם מוצפנים ע"י מפתח בגודל 128 ביט. האלגוריתם מבצע 8 איטרציות זהות שמורכבות משני חלקים כל אחת ובסוף עוד חצי איטרציה.

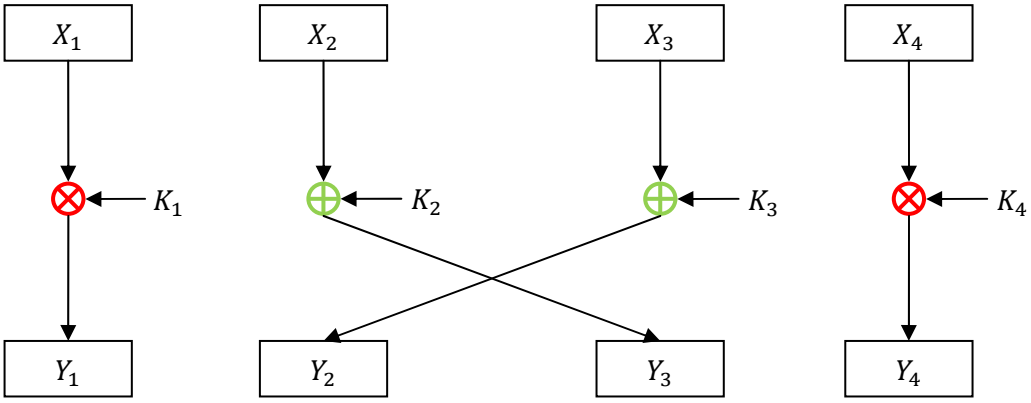



בסיבובים האלגוריתם משתמשת בפעולות XOR, חיבור מודולו $2^{16}$ וכפל מודולו $2^{16} + 1$. בכל סיבוב מלא של האלגוריתם משתמשים בששה תת-מפתחות ובחצי סיבוב האחרון משתמשים בארבעה תת-מפתחות. סה"כ מהמפתח של האלגוריתם יוצרים 52 מפתחות באורך 16 ביט באופן הבא: בכל תחילת איטרציה לוקחים את ששת המפתחות להיות פשוט 96 הביטים הראשונים (משמאל) של המפתח מחולקים לששה חלקים. בסוף האיטרציה מסובבים את המפתח 25 ביטים שמאלה.

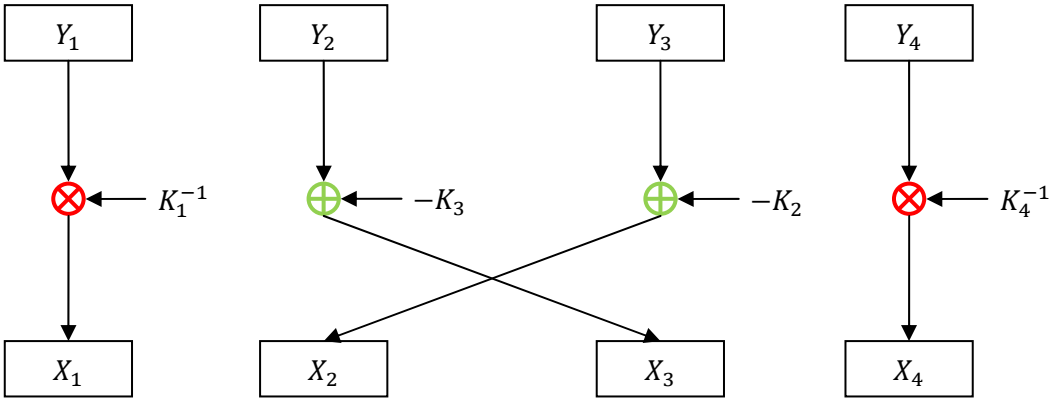
המפתחות של חצי האיטרציה האחרונה מתקבלים באופן דומה בשימוש ב-64 הביטים השמאליים של המפתח המסובב.

כל סיבוב מלא מחולק לשניים. בחלק הראשון משתמשים בארבעה מפתחות ובחלק השני משתמשים בשנים. כל חלק של ההודעה מחולק לארבעה חלקים.

החלק הראשון פועל באופן הבא:

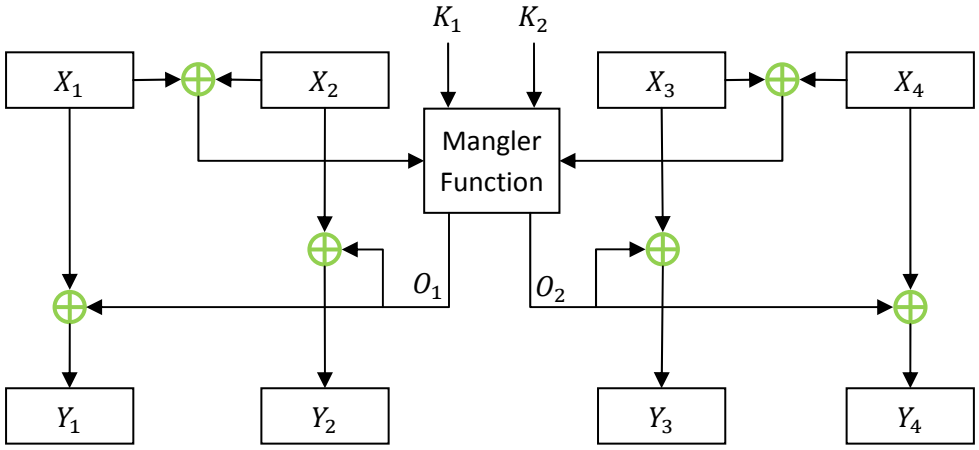



מאחר שפעולת החיבור היא הפיכה וגם כפל מודולו $2^{16}+1$ הוא הפיך (כי $2^{16}+1$ ראשוני) המעגל הזה הוא הפיך. פשוט מכניסים בתור מפתחות את $K_1^{-1}, -K_2, -K_3, K_4^{-1}$ ופועלים באותו אופן.




כמובן את המפתחות ההפכיים אפשר לחשב מראש כדי לחסוך זמן.

החלק השני של הסיבוב פועל באופן הבא:




אפשר לראות שהסיבוב הזה הוא למעשה ההופכי של עצמו עם אותם המפתחות. נשים לב למשל ש-

$$Y_1 = X_1 \oplus O_1$$
$$Y_2 = X_2 \oplus O_1$$

לכן $X_1 \oplus X_2 = Y_1 \oplus Y_2$ ולכן בין אם מצפינים ובין אם מפענחים הפלט $(O_1, O_2)$ של ה- Mangler Function הוא אותו פלט ואז $Y_1 \oplus O_1 = X_1 \oplus O_1 \oplus O_1 = X_1$ ובאותו אופן $X_2 = Y_2 \oplus O_2$.

## Advanced Encryption Standard

Advanced Encryption Standard (או בקיצור AES) הוא סטנדרט חדש של הצפנה שנכנס לתוקף החל משנת 2002 והחליף את DES. האלגוריתם הזה נבחר מתוך כמה מועמדים בעיקר בגלל שיקולים של ביצועים. הוא די קל למימוש וצורך מעט זיכרון.

האלגוריתם פועל על בלוקים בגודל 128 ביט. המפתח יכול להיות בגודל 128, 192 או 256 ביט ומספר הסיבובים של האלגוריתם הוא 10, 12 או 14 בהתאמה לגודל המפתח. מהמפתח יוצרים תת מפתחות בגודל 128 ביט לכל סיבוב.

מאחר שגודל הבלוק הוא קבוע 128 ביט האלגוריתם פועל על מערך ריבועי בעל 16 בתים שנקרא ה**מצב**.

שלבי האלגוריתם:

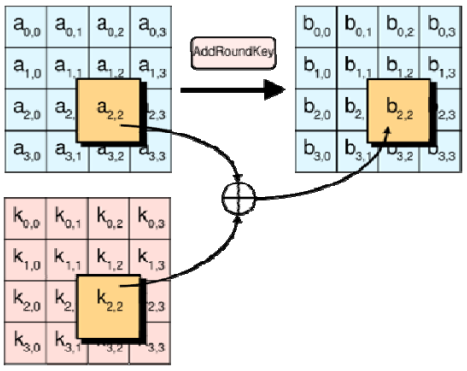
1. הרחבת המפתח (יצירת תת מפתחות לכל סיבוב של האלגוריתם)
2. סיבוב מקדים:
   א. חיבור מפתח סיבוב
3. סיבובים:
   א. החלפת בתים
   ב. סיבוב שורות
   ג. ערבוב עמודות
   ד. חיבור מפתח סיבוב
4. סיבוב סופי:
   א. החלפת בתים
   ב. סיבוב שורות
   ג. חיבור מפתח סיבוב

האלגוריתם הזה הוא הפיך ובעזרת קצת מתמטיקה אפשר להוכיח שהוא חזק, אבל אנחנו ניקח את זה כמובן מאליו. רק נציין שמטרת כל הפעולות, הסיבובים והערבובים היא לערבב היטב את הקלט ולגרום לכך שכל שינוי קטן בטקסט הגלוי או במפתח יביא לשינוי גדול בטקסט המוצפן.

נפרט את התהליכים השונים (פרט להרחבת המפתח).

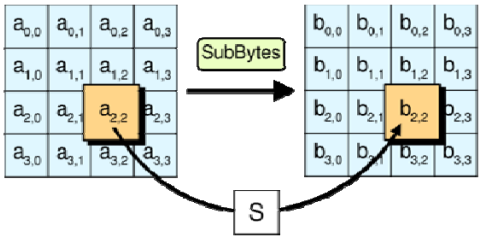
### חיבור מפתח סיבוב

כל בית של מפתח הסיבוב עובר XOR עם המצב:



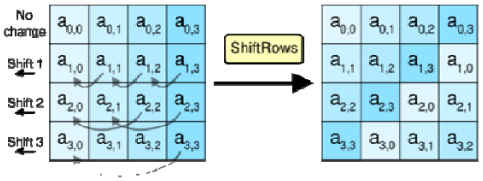
### החלפת בתים

כל בית במצב מוחלף בבית חדש באמצעות טרנספורמציה לא ליניארית שנתונה ע"י S-box:



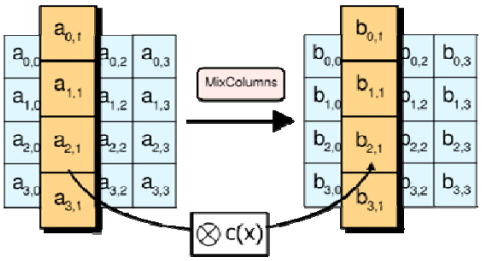

### סיבוב שורות

נמספר את השורות מ-0 ועד 3. במספור זה לשורה ה-i עושים סיבוב שמאלה ב-i בתים:




### ערבוב עמודות

ארבעת הבתים של כל עמודה עוברים טרנספורמציה ליניארית הפיכה:

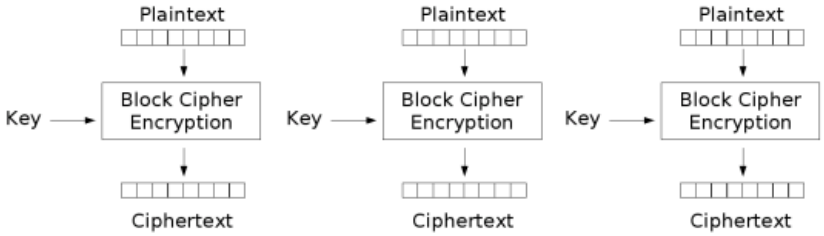



## הצפנת טקסטים ארוכים

עד עכשיו הסתכלנו על הצפנה של בלוק אחד של הודעה. אבל אם ההודעה יותר ארוכה מאורך הבלוק צריך להחליט מה לעשות. נניח שההודעה שאנחנו רוצים להצפין מורכבת מ-$l$ בלוקים $M = M_1 \dots M_l$. נניח שאורך ההודעה מתחלק באורך הבלוק, אחרת אפשר לרפד אותה. כל בלוק יוצפן בנפרד, אבל יש כמה דרכים להצפין את הבלוקים. אנחנו נדון בשלוש דרכים.

### Electronic Codebook

בתצורה של Electronic Codebook (או בקיצור ECB) נצפין כל בלוק של ההודעה באופן בלתי תלוי בעזרת האלגוריתם שלנו ועם אותו המפתח:




אם הבלוק הוא באורך $n$ אז למעשה מדובר בקוד מונואלפביתי עם $2^n$ אותיות בא"ב.

יתרונות:

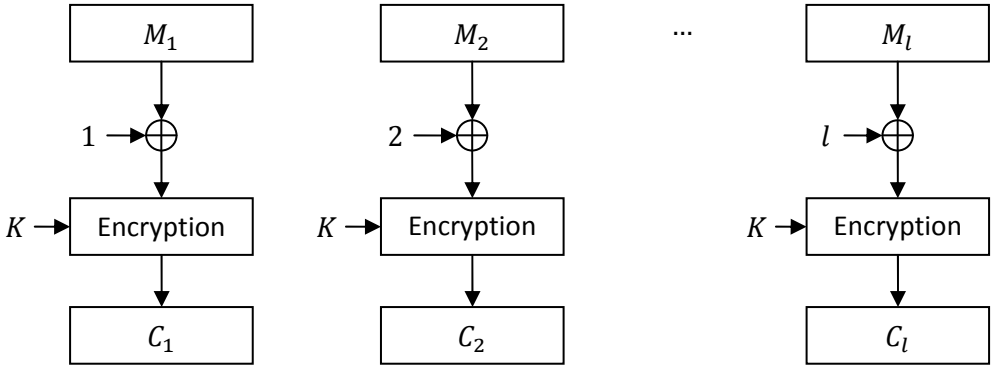
- הצפנה ופענוח מקביליים.
- אפשר להצפין או לפענח רק חלק מהבלוקים, וכך למשל אם חלק מהבלוקים אובד בתקשורת עדיין ניתן להבין חלק מההודעה.

חסרונות:

- בלוקים זהים מוצפנים בצורה זהה מה שפוגע באבטחה כמו בכל קוד מונואלפביתי. מעבר לזה, גם הידיעה שיש שני בלוקים זהים בהודעה יכולה להסגיר מידע – למשל שמשכורת של שני עובדים זהה...
- אין הגנה על סדר ותוכן הבלוקים. מתקיף יכול להחליף אחד מהבלוקים או לשנות את סדר ההודעה ואין שום דרך לגלות את זה.

### Counter

בתצורת Counter (או בקיצור CTR) מנסים להתגבר על הבעיה שבלוקים זהים מוצפנים באותו אופן.




כאן לכל בלוק מתווסף אלמנט שונה ובכך גורם לכך שאפילו אם יש שני בלוקים שונים, בגלל שהמספר הסידורי שלהם זהה ההצפנה שלהם שונה.

יתרונות:

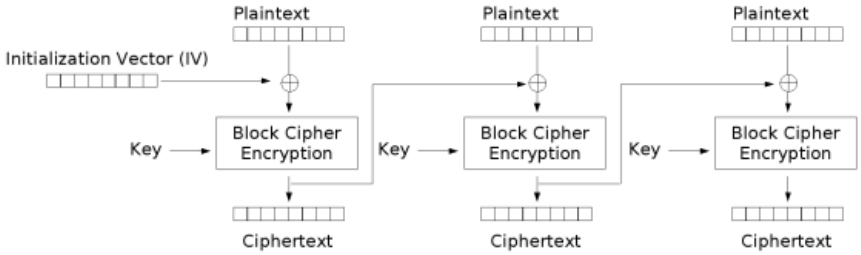
- בלוקים זהים מוצפנים באופן שונה.
- יש הגנה מסוימת על הסדר והתוכן כי אם בלוק כלשהו הושמט ומפענחים את ההודעה מנקודת הנחה שכל הבלוקים הגיעו אז סביר שהפענוח לא יהיה הגיוני. מצד שני, יש סיכוי שהפענוח יהיה הגיוני ולכן ההגנה לא מלאה.
- הצפנה ופענוח מקביליים.

חסרונות:

- דרוש סנכרון חיצוני שאומר איזה בלוק מתאים לאיזה מספר.
- אם עושים XOR עם המספרים כפי שהם אז למשל ההבדל בין 1 ל-5 הוא בביט אחד בלבד וזה נותן פתח להתקפה. לכן בד"כ עושים את ה-XOR עם $h(i)$ כאשר $h$ פונקצית ערבול.

### Cipher-Block Chaining

תצורת Cipher-block chaining (או בקיצור CBC) היא הכי נפוצה כי היא פותרת כמעט את כל הבעיות של שתי השיטות הקודמות שהצגנו.




לפני ההצפנה כל בלוק של ההודעה המקורית עובר XOR עם הבלוק המוצפן הקודם. הבלוק הראשון עובר XOR עם וקטור ראשוני ואקראי IV שנבחר לכל הודעה מחדש. ככה כל הודעה מוצפנת באופן שונה אפילו אם כל הבלוקים זהים לבלוקים של הודעה אחרת.

פונקצית ההצפנה נתונה באופן רקורסיבי ע"י

$$C_0 = IV$$
$$C_i = E_K(M_i \oplus C_{i-1})$$

והפענוח נתון ע"י

$$C_0 = IV$$
$$M_i = D_K(C_i) \oplus C_{i-1}$$

נשים לב שאת IV חייבים לשלוח בתקשורת כי בלעדיו אי אפשר לפענח את ההודעות ולכן הוא בכלל לא חייב להיות סודי. עוד דבר שכדאי לשים לב אליו הוא שאם אפילו ביט אחד משתנה איפשהו במהלך שליחת ההודעה אז הבלוק האחרון $C_L$ ישתנה. לכן אפשר להשתמש בו בשביל אימות. אבל אם כך, צריך לשלוח אותו פעמיים (כדי שאפשר יהיה באמת לבדוק). אבל מישהו יכול בקלות להשמיט את הבלוקים האחרונים ולשלוח את האחרון שנותר פעמיים, ואז ההודעה תיראה תקינה. אז בד"כ שולחים את $C_L$ מוצפן באמצעות איזשהו מפתח $K$.

יתרונות:

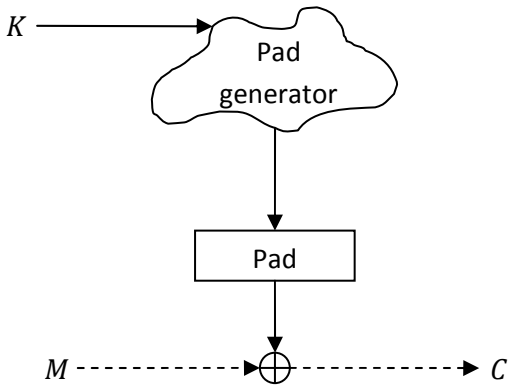
- פענוח מקבילי כי כל $M_i$ תלוי רק ב-$C_i$ וב-$C_{i-1}$ אז אם קיבלנו את כל הבלוקים אפשר לפענח את כולם בבת אחת.
- מאותה סיבה אפשר לפענח רק חלק מהבלוקים אם רוצים.
- בלוקים שונים מוצפנים באופן שונה.
- אין צורך במנגנון סנכרון חיצוני. ברגע שיש שני בלוקים רצופים תקינים אפשר לפענח ואין צורך במנגנון שיגיד לנו מהם המספרים הסידוריים של כל בלוק.

חסרונות:

- ההצפנה סדרתית.

## צפני זרם

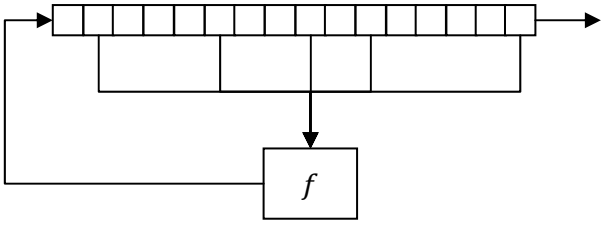
המבנה הכללי של Stream ciphers הוא שההודעה מגיעה כרצף של ביטים ועוברת XOR עם איזה Pad שמיוצר ע"י מכונה סופית דטרמיניסטית שגורמת לו להיראות אקראי:




למעשה זה אמור לשמש כקירוב ל-one-time pad. אנחנו נדון בשלושה מנגנונים כאלה.

## Linear Feedback Shift Register

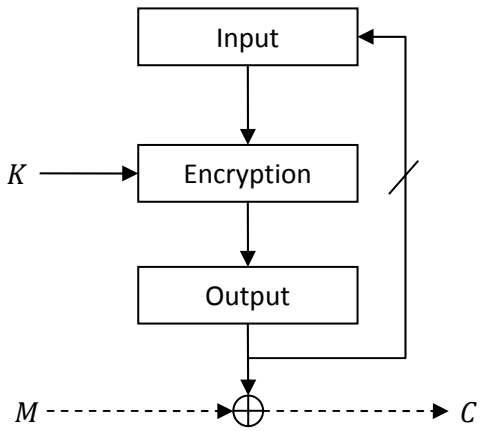
Linear feedback shift register (או בקיצור LFSR) הוא shift register שביט הקלט שלו הוא פונקציה ליניארית של המצב הקודם.




בד"כ הפונקציה שמשתמשים בה היא XOR. הערך הראשוני של הרגיסטר הוא המפתח. לכן למשל לא נרצה שהערך הראשוני יהיה 0 כי אז הוא לא ישתנה אף פעם. אז אם הרגיסטר הוא באורך $n$ אז יש $2^n - 1$ אפשרויות למפתח שאינן 0.

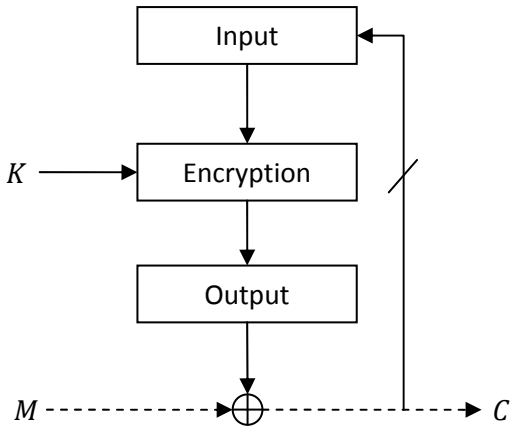
את הביטים שנכנסים לפונקציה אפשר לייצג ע"י פולינום מעל $\mathbb{Z}_2$ שנקרא הפולינום האופייני. אם נמספר את הביטים ברגיסטר מהשמאלי ביותר עד הימני ביותר מ-1 עד $n$ הופלינום האופייני הוא סכום $x^i$ של כל ה-$i$-ים כך שבביט ה-$i$ מופיע 1 בתוספת 1. למשל בציור שלמעלה הפולינום האופייני הוא $1 + x^2 + x^6 + x^9 + x^{14}$. אם הפולינום האופייני לא פריק מעל $\mathbb{Z}_2$ אז בסכמה שמתוארת למעלה הרגיסטר יקבל את כל $2^n - 1$ הערכים האפשריים. רגיסטר כזה נקרא **מקסימאלי**.

## Output Feedback Mode



ב-OFB מתחילים מערך התחלתי IV. מצפינים אותו בעזרת מפתח סודי $K$ והתוצאה משמשת להצפנה של $M$ שמגיע בזרם. בגלל הסימטריות של XOR ההצפנה והפענוח הם זהים. רק צריך לדעת את ה-input הנוכחי כדי להסתנכרן. וזה חיסרון בולט של השיטה – הצורך בסנכרון חיצוני. מצד שני, התהליך שמייצר את ה-output הוא דטרמיניסטי אפשר לייצר הרבה כאלה מראש וזה מייעל את החישוב.

## Cipher Feedback Mode




CFB היא שיטה שמאוד דומה ל-OFB אלא שבמקום להשתמש ב-output בתור בסיס להמשך יצירת ה-pads הטקסט המוצפן משמש לכך. זה אומר שהייצור של ה-pads הוא לא דטרמיניסטי ולא מחזורי שהרי להודעות שמוצפנות אין מחזוריות והמרחב הוא אינסופי. אז בניגוד ל-OFB אי אפשר כאן לייצר pads מראש וזה אולי פוגע בביצועים אבל בדומה ל-CBC יש כאן סנכרון עצמי וזה יתרון חשוב. נשים לב שה-input כאן הוא לא סודי בכלל, שהרי הוא מיוצר באמצעות ההודעה המוצפנת ואנחנו מניחים שכולם יכולים לראות אותה.

ככל שרוחב הפס קטן יותר האלגוריתם בזבזני יותר בחישוב אבל בטוח יותר. אם המשוב הוא של ביט בודד אז אם התקשורת נפלה, ברגע שהיא חוזרת האלגוריתם מסתנכרן מיד, ואילו אם רוחב הפס רחב יותר צריך לשים לב שכדי להסתנכרן צריך להתחיל מרצף ביטים שהוא בדיוק בלוק שלם ולא חצי מבלוק אחד וחצי מבלוק אחר.

# אימות

יכול להיות שנהיה מעוניינים לאמת אחד מהדברים הבאים:

- **ישות** – אימות אחד הצדדים בהודעה
- **מקור** – אימות הישות שממנה ההודעה הגיעה
- **תוכן** – אימות תוכן ההודעה
- **סדר** – אימות סדר ההודעות או התוכן (בעיקר במקרה שההודעות עוברות בתקשורת כי אז מחלקים את ההודעה לחבילות ושולחים כל חבילה בנפרד)
- **זמן** – היינו רוצים לדעת אם הודעה רלוונטית בזמן מסוים (למשל אם ההודעה היא "היום השיעור מבוטל")

יש כמה שיטות לאימות:

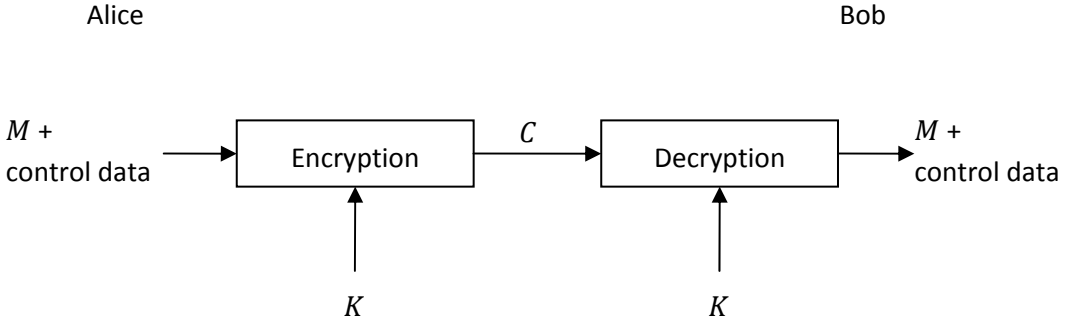
1. **אימות ע"י הצפנה** – אימות של תוכן אפשר להשיג ע"י הצפנה ובתור תוצר לוואי יש גם סודיות. כמובן, אפשר לדבר על הצפנה סימטרית ועל הצפנה אסימטרית.
2. **אימות ע"י פונקצית MAC** (Message Authenticating Code) – כאן השימוש הוא תמיד סימטרי.
3. **אימות ע"י פונקצית ערבול** – פה שוב אפשר לדבר על מקרה סימטרי ומקרה אסימטרי.

### אימות ע''י הצפנה

הצפנה לבד לא נותנת אימות של סדר. כתלות באחוז ההודעות שנחשבות תקינות מתוך המרחב קיים גם סיכוי שהודעה אקראית תפוענח ותיראה תקינה ולכן הצפנה גם לא מבטיחה אימות של תוכן. לכן יש לדון באלגוריתמי אימות נוסף על אלה של ההצפנה.

#### אימות סימטרי

לאליס ולבוב יש מפתח סודי זהה ומתואם $K$. אליס שולחת הודעות לבוב מוצפנות בעזרת המפתח. בוב מפענח אותן ואם יוצא משהו הגיוני אז הוא יודע גם שההודעה הגיע מאליס וגם שהתוכן אמיתי.




אלא שבמה שכתוב כאן יש הרבה פגמים:

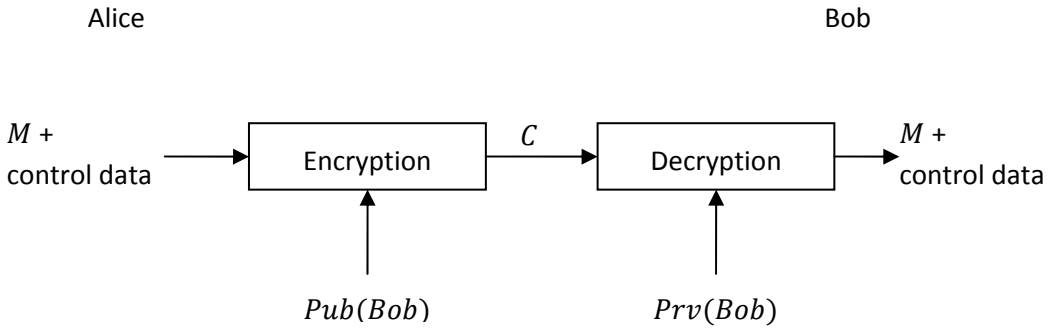
1. התקפת Replay: מישהו יכול לראות את ההודעה כפי שהיא היום ולשלוח אותה שוב מחר. אז בוב יחשוב שאליס שלחה לו הודעה אבל זה לא נכון.
2. באופן דומה, אפשר לקצר הודעות, לשכפל הודעות וכן הלאה. האמת היא שקיצור בצורה הגיונית קשה יותר. אבל בכל אופן אפשר לשלוח את אותה ההודעה פעמיים בלי שום בעיה.

3. הבעיה העיקרית היא שמישהו יכול להמציא משהו אקראי שבמקרה הפענוח שלו בעל משמעות ואז לבוב ממש אין שום בדרך לדעת שזו לא אליס. ואם ההודעות שהם שולחים אחד לשני הן אקראיות אז גם אם יוצא משהו לא בעל משמעות בוב לא יידע שזו לא אליס ששלחה לו את ההודעה. אם ההודעות הן ככלל לא אקראיות אז דווקא אפשר להתמודד עם סוג כזה של התקפה ע"י הוספת מבניות להודעה. ככל שיש להודעה יותר מבניות הסיכוי להמציא משהו אקראי שהפענוח שלו יהיה בעל משמעות ובעל אותה מבניות הנדרשת הוא קטן יותר. זוהי גם מטרת ה-control data. האמת היא שגם אפשר להניח שהסיכוי להמציא משהו אקראי שיפוענח למשהו בעל משמעות הוא קטן.
4. האימות הוא באורך ההודעה, כלומר צריך לפענח את כל ההודעה לפי שמאמתים אותה וזה צורך הרבה זיכרון וזמן.

השיטה מסתמכת על כך שאלגוריתם ההצפנה והפענוח הוא טוב מאוד ושקל לזהות באופן אוטומטי הודעות מפוענחות תקינות. אחרת זו תהיה עבודה מאוד קשה לאמת הודעות שהגיעו.

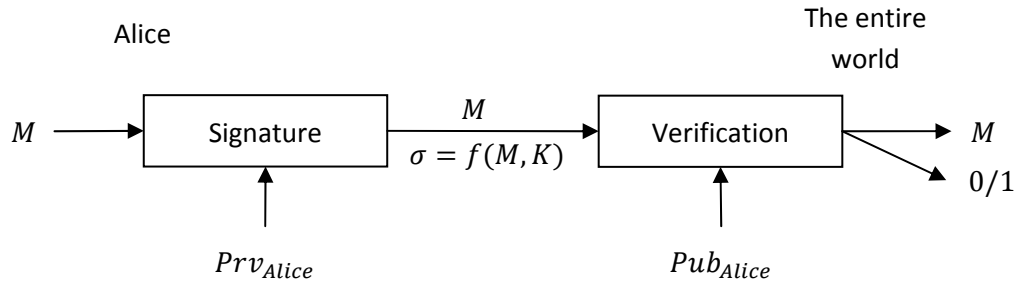
## אימות אסימטרי

ננסה להשתמש באותה שיטה כמו בהצפנה סימטרית אבל בהצפנה אסימטרית:




ברור שזה מגוחך, אין פה לא אימות תוכן ולא אימות מקור. כל אחד יכול לשלוח לבוב הודעות מוצפנות בעזרת המפתח הציבורי שלו שידוע לכל העולם ולבוב אין שום דרך לדעת מי שלח לו ואם זה לא השתנה באמצע.

אז בעולם האסימטרי משתמשים בחתימה דיגיטלית.



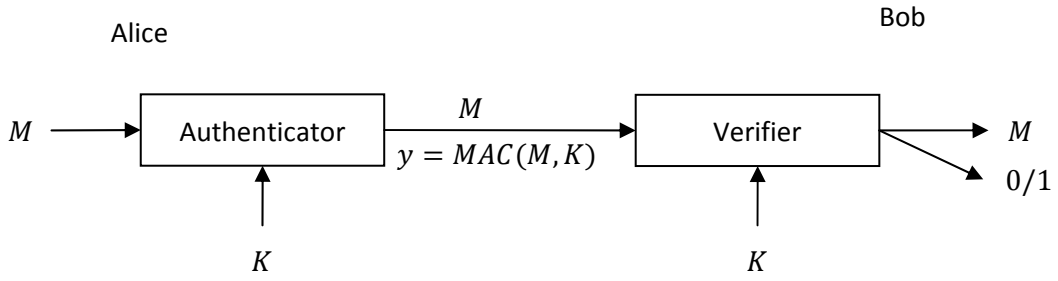

את החתימה יכולה לעשות רק אליס ואילו דווקא את האימות יכול לעשות כל אחד. בצורה הזאת אפשר להשיג גם אימות תוכן (כי בשביל להחליף את ההודעה צריך לדעת לחתום בשם אליס אבל המפתח שלה סודי), גם אימות מקור וגם אי התכחשות (כי אליס היא היחידה שיכולה לחתום על ההודעות שלה)!

גם כאן קיים סיכוי שמישהו ימציא חתימה אקראית והיא תתאים להודעה שהוא שתל במקום ההודעה האקראית. אלא שאנחנו מניחים שמרחב ההודעות האפשריות הוא דליל מאוד וזה אומר שאם מישהו מגריל חתימה אקראית אז הסיכוי שהיא תהיה נכונה הוא קטן מאוד.

נשים לב שבמקרה האסימטרי אנחנו לא משיגים גם הצפנה וגם אימות בבת אחת. צריך להשתמש בשני מנגנונים שונים ובמפתחות שונים.

## אימות ע"י פונקצית MAC

הרעיון דומה מאוד לחתימה דיגיטלית:




לאליס ולבוב יש מפתח סודי מתואר שרק הם יודעים. אליס משתמשת במפתח הזה כדי לחשב פונקצית MAC על ההודעה שהיא שולחת באופן גלוי. בוב מחשב גם הוא את הפונקציה והם התוצאות שלהם מתאימות סימן שתוכן ההודעה והמקור מאומתים.

פונקצית ה-MAC לא צריכה להיות הפיכה. למעשה זה כלל לא רצוי שתהיה הפיכה כי אז ניתן יהיה לפענח את המפתח הסודי.

יש כמה חששות בשימוש בשיטה הזאת:

1. מתקיף שרואה זוג $(M, y)$ יכול לחלץ את המתפח הסודי
2. מתקיף יכול לייצר זוג $(M', y') \neq (M, y)$ חוקי

ראינו שכשמשתמשים ב-One-time-MAC אנחנו לא מוטרדים מהדברים האלה, אבל הבעיה היא שאנחנו רוצים להשתמש ב-Many-time-MAC. אם כך אז יש כמה דרישות שפונקצית ה-MAC צריכה לקיים:

1. פילוג $y$-ים אחיד אקראי
2. הודעות דומות מקבלות $y$-ים שונים
3. MAC קצר
4. MAC רב-פעמי
5. חישוב מהיר של הפונקציה

פונקציות הצפנה הן טובות לייצור MAC כי הן מקיימות את התכונות הנ"ל.

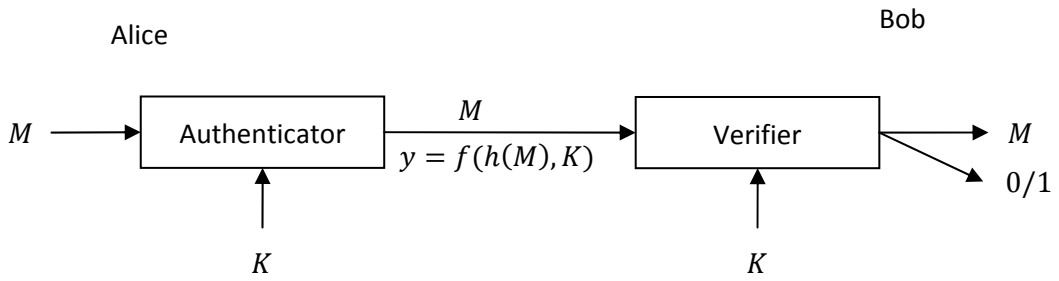
**דוגמאות**:

1. $y = E_K(M_1 \oplus \dots \oplus M_n)$ כאשר $M_1, \dots, M_n$ הם הבלוקים של ההודעה. זאת לא פונקציה טובה. פילוג ה-$y$-ים אמנם אקראי כי זה מה שפונקצית הצפנה עושה אבל יכולות להיות שתי הודעות שונות עם אותו ה-$y$ שהרי אם נשנה את אותו הביט בשני בלוקים ה-XOR של כולם לא משתנה. או לחילופין, אם סדר הבלוקים שונה אין דרך לגלות את זה.
2. נסמן $C_L = E_K^{CBC}(M)$ וניקח $y = (IV, C_L)$. ה-IV אקראי וגם $C_L$ יוצא אקראי מההצפנה אבל בכל זאת אפשר לזייף פה הודעה.

אנחנו רואים שלא מספיק להשתמש רק בפונקצית הצפנה. כדי להגן על הודעה ע"י MAC צריך להוסיף לה עוד הרבה פרטים בשביל בקרה: זמן, מספר סידורי, אורך ועוד. למעשה כל הודעה $M$ צריך להפוך קודם לאוסף $(A, B, Time, Sequence\ Number, Length, Random\ number, M)$ ורק על זה לחתום. זה מוסיף המון פרטים להודעה שלא יכולים לחזור על עצמם ולכן אי אפשר יהיה לעשות התקפת replay למשל.

## אימות ע"י פונקצית ערבול

### אימות סימטרי



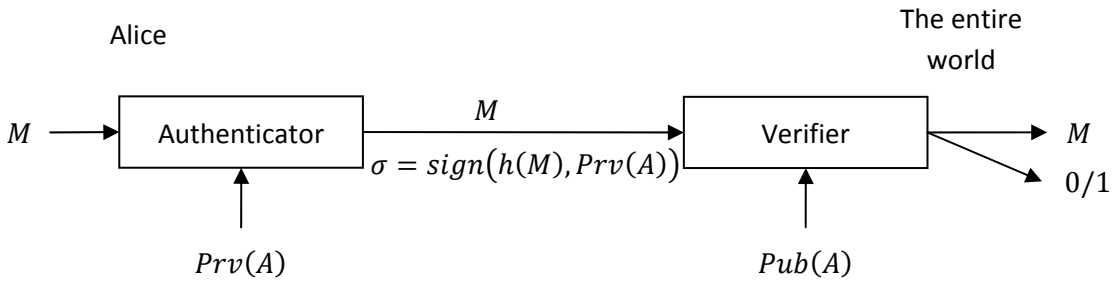

פונקצית הערבול של הודעה היא למעשה איזשהו מאפיין קצר שלה. על המאפיין יחד עם המפתח מפעילים איזושהי פונקציה. למעשה ניתן להסתכל על התהליך הזה כתהליך יצירת MAC, אלא שב-MAC משתמשים בפונקציות הצפנה וכאן לא מצפינים את ההודעה אלא מקצרים אותה.

התהליך הזה נקרא HMAC. הסכמה משתמשת בשלושה פרמטרים: פונקצית ערבול $h$, מפתח סודי $K$ והודעה $M$:

$$HMAC_K(M) = h\left(\left(h(k \parallel M)\right) \parallel k\right)$$

כאשר הסימן $\parallel$ מציין שרשור. התכונה שהמפתח נמצא גם בהתחלה וגם בסוף מונעת הרבה סוגי התקפות – למשל אי אפשר לשנות את קצוות ההודעה.

### אימות אסימטרי

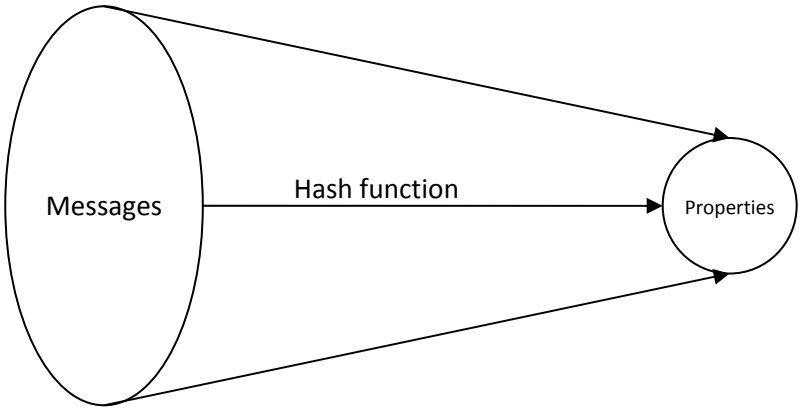



האמת היא שבעולם האמיתי זה מה שנקרא חתימה דיגיטלית. בד"כ לא חותמים על ההודעה במלואה כי זה יכול להיות ארוך מאוד, אלא חותמים את פונקצית הערבול של ההודעה שהיא קצרה יותר. זה לעומת MAC איפה שהמפתח נמצא בשימוש לכל אורך הדרך.

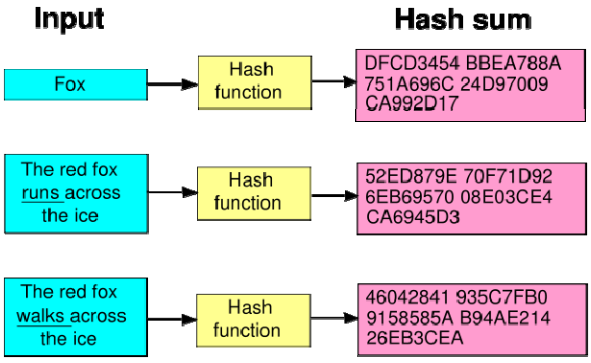
# פונקציות ערבול קריפטוגרפיות

## תכונות רצויות של פונקציות הערבול

המטרה של פונקציות ערבול היא למפות את עולם ההודעות, שהוא תיאורטית לא מוגבל, לעולם המאפיינים שהוא סופי וקטן.




למשל, פונקצית ערבול יכולה לעבוד באופן הבא:




תכונות רצויות של מאפיין:

1. קצר (לפחות ביחס לאורך ההודעות הפוטנציאלי)
2. איכותי, כלומר מפוזר אקראית ובצורה אחידה
3. קשה לזיוף

ברור שהפונקציה היא לא חח"ע, שהרי היא מעבירה עולם ענק לעולם קטן. לכן תהיינה הרבה הודעות שיש להן אותו מאפיין.

מכאן נגזרות התכונות הרצויות של פונקצית ערבול:

1. דרישות אופרטיביות:
   א. $h$ תעבוד על כל קלט
   ב. $h$ תייצר פלט באורך קבוע
   ג. $h$ קלה לחישוב
2. דרישות אבטחה:
   א. **One-way**: בהינתן $y$ קשה למצוא $x$ כך ש-$h(x) = y$. כלומר, קל לחשב את התמונה אבל קשה למצוא מקורות. זה כמובן אפשרי ע"י חיפוש ממצה אבל הרעיון הוא שזה לוקח הרבה זמן.

ב. **Weak collision resistance**: בהינתן $x_1$ קשה למצוא $x_2 \neq x_1$ כך ש-

$$h(x_1) = h(x_2)$$

ג. **Strong collision resistance**: קשה למצוא $x_1 \neq x_2$ כך ש-

$$h(x_1) \neq h(x_2)$$

ההבדל בין דרישה זו לדרישה הקודמת הוא שכאן לא נתון לנו מראש שום $x_1$.

מן הראוי לציין שיכולים להיות מקרים פרטניים שבהם קל למצוא התנגשויות אז הכוונה היא שכמעט לכל המקרים זה קשה.

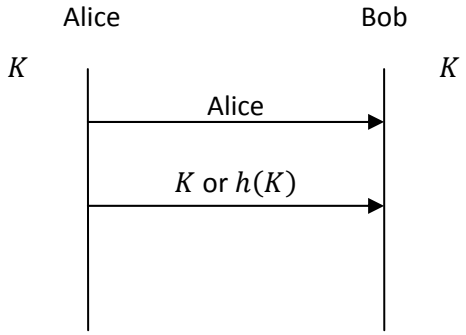
## שימושים של פונקציות ערבול

### אפיון עצמים

נניח ששמרנו קובץ במחשב בעבודה והלכנו הביתה ללילה. הציפייה שלנו היא שהקובץ לא ישתנה עד למחרת בבוקר. אז אפשר לקחת את הקובץ איתנו על דיסקט ולשמור אותו במקום בטוח מתחת לבלטה אבל יכול להיות שהקובץ ענק. אז מה שאפשר לעשות במקום הוא ליצור איזה מאפיין קצר של הקובץ ואותו לקחת איתנו ולשמור מתחת לבלטה. למחרת, אם המאפיין לא השתנה אז בסבירות גבוהה מאוד זה אותו הקובץ.

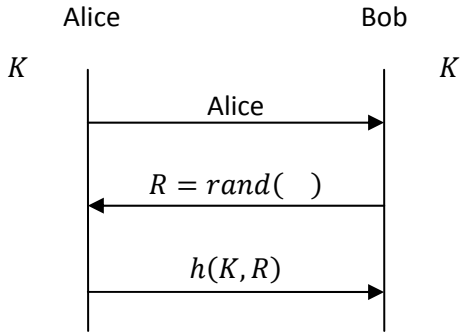
### פרוטוקול אימות

נניח שאליס ובוב מדברים ואליס רוצה לוודא שהיא אכן מדברת עם בוב.




לאליס ולבוב יש מפתח סודי $K$. אליס יכולה לפנות לבוב, להכריז שהיא אליס ולשלוח לו את המפתח. אז בוב יידע שזו אכן היא כי רק שניהם מכירים את המפתח הסודי, אלא שאז המפתח הוא חד-פעמי כי הוא נחשף בפעם הראשונה שמשתמשים בו. אז במקום לשלוח את המפתח $K$ אליס יכולה לשלוח את $h(K)$. זה לא חושף את המפתח אבל עדיין הוא חד פעמי כי כל אחד יכול לקלוט את $h(K)$ בתקשורת ואז לשלוח אותו במקום אליס.

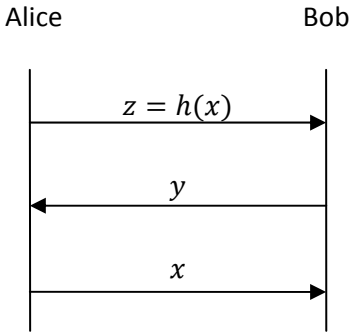
כדי להתגבר על החד-פעמיות צריך להכניס אלמנט אקראי לפרוטוקול:




עכשיו אי אפשר להשתמש ב-$h(K,R)$ יותר מפעם אחת כי בוב כל פעם ישלח לאליס מספר רנדומלי אחר וגם המפתח לא נמצא בסכנה כי קשה מאוד לפענח אותו מתוך פונקצית הערבול.

## פרוטוקול להתחייבות

אליס ובוב רוצים לשחק זוג או פרט דרך הרשת. אבל הם בוודאי לא יכולים לשלוח את הבחירה שלהם בדיוק באותו הזמן. אחד מהן שולח את ההודעה שלו שבריר שניה לפני השני. הם גם לא סומכים זה על זה שלא ישנו את בחירתם לאחר שראו את מה שהשני בחר. אז הם משתמשים בפרוטוקול הבא:



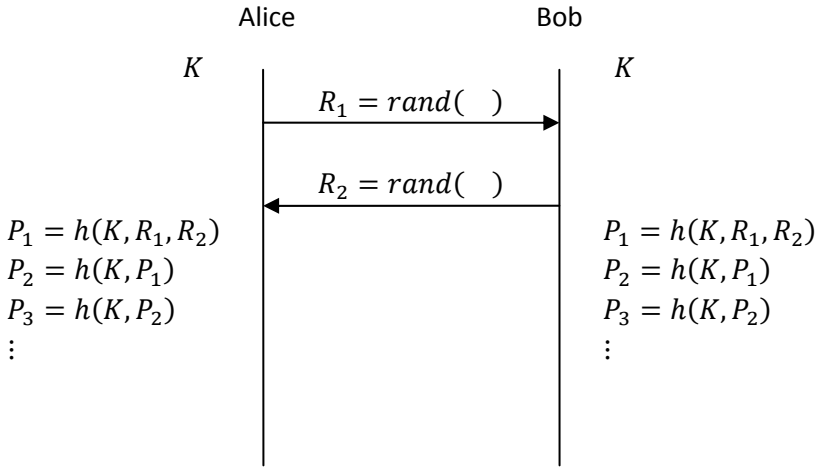

אליס בוחרת $x$ אבל לא שולחת אותו כדי שבוב לא יוכל לשנות את בחירתו. במקום זה היא שולחת את המאפיין של $x$. אז בוב שולח את בחירתו $y$ ולבסוף אליס שולחת את $x$ ובוב יכול לאמת אותו ע"י חישוב $h(x)$. מאחר שקשה לחשב את $x$ מתוך $z$ בוב לא יכול לרמות.

נשים לב שכדי שבוב לא ירמה דרושה התכונה one-way של פונקציות ערבול ואילו כדי שאליס לא תרמה דרושה התכונה strong collision resistance כי אם אליס יכלה למצוא מספר זוגי ומספר אי זוגי שיש להם אותו מאפיין היא תמיד הייתה מנצחת.

## הצפנה

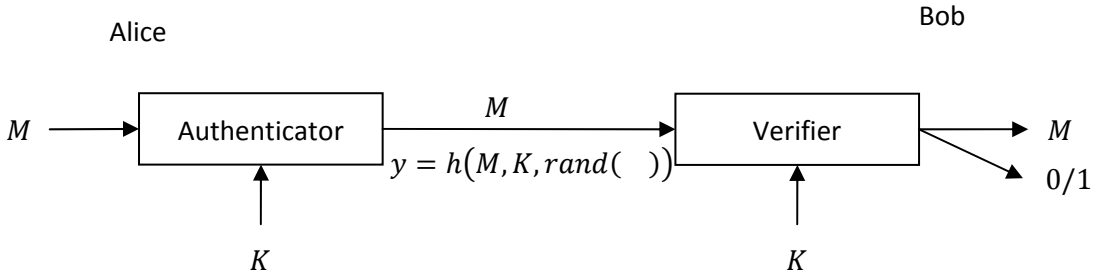
יש אלגוריתמים טובים מאוד להצפנה. למה בכלל נרצה לעשות הצפנה באמצעות פונקציות ערבול? יש לכך שלוש סיבות. הראשונה, אנחנו יכולים וזה נחמד. השניה, יש מקומות שבהם פונקציות הצפנה נחשבות כלי נשק ואסור להשתמש בהן, אבל פונקציות ערבול חוקיות לשימוש. השלישית ואחרונה, הרבה פעמים פונקציות ערבול הרבה יותר מהירות מפונקציות הצפנה אז זה הרבה יותר יעיל.

כדי להצפין בעזרת פונקציות ערבול נשתמש בהן כדי לייצר מלא pads שישמשו אותנו כמו one-time pads. לאליס ולבוב יש מפתח סודי מתואם $K$. הם ישלחו אחד לשני מספרים אקראיים וישתמשו בהם יחד עם המפתח ליצירת ה-pads:



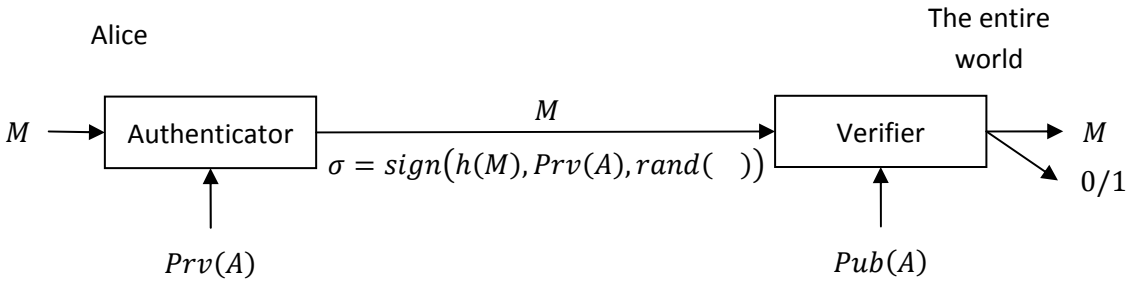

כל ה-pads שונים אבל הם לא לגמרי אקראיים כי לכל $i \geq 1$ $P_{i+1}$ תלוי ב-$P_i$ אבל בגלל התכונות של פונקצית הערבול זה דומה לאקראי.

## MAC




התפקיד של המספר האקראי ב-$y$ הוא שלאותה ההודעה לא יהיה אותו ה-MAC פעמיים ואז אי אפשר יהיה להשתמש בו בשביל התקפת replay.

## חתימה




כאן חותמים על $h(M)$ ולא על $M$ כדי שהחישוב יהיה קצר יותר ו-$rand(\ \ )$ משמש למניעת התקפת replay.

## בניית פונקציות ערבול

מהדרישה שפונקצית הערבול היא strong collision resistant נקבל שהפלט של פונקצית הערבול $h$ לא יכול להיות קצר מדי. שהרי אם אורך הפלט הוא $n$ ביטים מספיק לעבור על $2^n + 1$ קלטים אפשריים ומעיקרון שובך היונים בטוח נמצא התנגשות אחת לפחות.

## התקפת יום ההולדת

למעשה, באופן דומה לפרדוקס יום ההולדת מספיק לבדוק הרבה פחות קלטים כדי למצוא התנגשויות בהסתברות גבוהה.

נניח שמרחב התוצאות הוא בגודל $N$ ונניח שאנחנו עושים ניסיונות אקראיים. נסמן ב-$Q(N,k)$ את ההסתברות שאחרי $k$ ניסיונות לא מצאנו התנגשות. אז מתקיים

$$\begin{aligned} Q(N,k) &= 1 \cdot \frac{N-1}{N} \cdot \frac{N-2}{N} \cdot \ldots \cdot \frac{(N-k+1)}{N} \\ &= \left(1-\frac{1}{N}\right)\left(1-\frac{2}{N}\right) \cdot \ldots \cdot \left(1-\frac{k+1}{N}\right) \\ &\leq e^{-\frac{1}{N}} \cdot e^{-\frac{2}{N}} \cdot \ldots \cdot e^{-\frac{k+1}{N}} \\ &= e^{-\frac{k(k+1)}{N}} \end{aligned}$$

לכן ההסתברות שאחרי $k$ ניסיונות אקראיים יש התנגשות היא לכל הפחות $1 - e^{-\frac{k(k+1)}{N}}$. למשל, אם $k \approx 1.18\sqrt{\mathrm{N}}$ אז ההסתברות להתנגשות היא לפחות $\frac{1}{2}$.

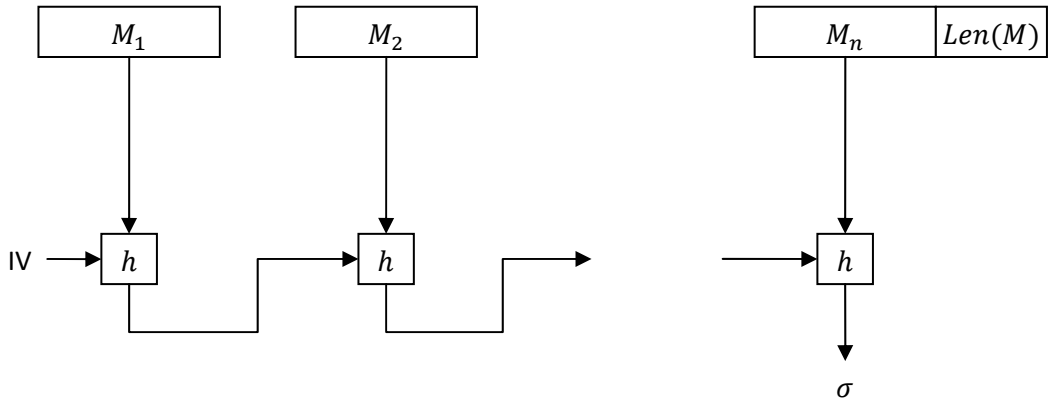
לכן כדי שפונקצית הערבול תהיה מספיק טובה לא רק אורך הפלט צריך להיות ארוך מיכולת החיפוש הממצה אלא גם חצי מאורך הפלט צריך להיות ארוך מיכולת החיפוש הממצה. אם האורך לא מספיק גדול אפשר להשתמש בהתקפת יום ההולדת באופן הבא:

נניח שיש מסמך מקורי שאנחנו רוצים לחתום עליו אבל המזכירה הזדונית רוצה לשתול מסמך מזויף. המזכירה תייצר מלא עותקים של המסמך המקורי ושל המסמך המזויף, שהם כולם זהים סמנטית אבל שונים סינטקטית. למשל, הכנסת רווחים מיותרים משנים את המסמך סינטקטית אבל לא סמנטית. אם מייצרים מספיק עותקים $A_i$ של המסמך המקורי ומספיק עותקים $B_i$ של המסמך המזויף אפשר למצוא שני עותקים כך ש-$h(A_i) = h(B_j)$. כעת המזכירה תחליף את המסמך המקורי $A$ ב-$A_i$ ואנחנו נחתום עליו בלי לחשוב פעמיים – הרי לא נשים לב כלל לשינויים הסינטקטיים שנעשו. ולבסוף במקום לשלוח את $A_i$ עם החתימה המזכירה תשלח את $B_j$ והחתימה תהיה נכונה!

כדי להתגבר על ההתקפה הזאת אפשר להגדיל את אורך החתימה או לחילופין לעשות שינוי סינטקטי קטן במסמך המקורי לפני שחותמים עליו, ואז אפילו אם המזכירה כבר הספיקה להחליף לנו את המסמך המקור $A_i$ במסמך השונה $A_i$ עדיין היא לא תוכל לשלוח את $B_j$ במקום כי החתימה לא תתאים.

בגלל שלא כולם מודעים לדברים האלה ולא יודעים שזה מה שצריך לעשות הקריפטוגרפים הגדילו את אורך החתימות ל-256 ביט.

**דוגמה**: כיום עובדים בפונקציות כמו MD-5, SHA-1, SHA-256 ועוד. שלושת אלה עובדות בצורה דומה. מחלקים את ההודעה לבלוקים בגודל 512 ביט ומייצרים חתימה כך:




להודעה מוספים ריפוד של 64 ביט שבו מופיע מספר הבלוקים של ההודעה. לכן ההודעה הארוכה ביותר האפשרית היא באורך $2^{64}$ בלוקים אבל זה יוצא די הרבה... מטרת הריפוד היא להגן על אורך ההודעה.

# אלגברה ותורת המספרים

## מחלק משותף מקסימאלי

יהי $a,b\in\mathbb{Z}$. **המחלק המשותף המקסימאלי** (**Greatest Common Divisor**) של $a$ ושל $b$ הוא מספר $d=\gcd(a,b)$ כך ש-$d|a$, $d|b$ ולכל $d'\in\mathbb{Z}$ כך ש-$d'|a$ וגם $d'|b$ מתקיים $d'|d$.

אם $\gcd(a,b)=1$ נאמר ש-$a$ ו-$b$ **זרים**.

לכל $a,b\in\mathbb{Z}$ נגדיר את קבוצת הצירופים הליניאריים שלהם $LC(a,b)=\{xa+yb:x,y\in\mathbb{Z}\}$. מסתבר שלכל $a,b\in\mathbb{Z}$ מתקיים $\gcd(a,b)=\min\{x\in LC(a,b):x>0\}$. בפרט זה אומר שתמיד אפשר למצוא $x,y\in\mathbb{Z}$ כך ש-$xa+yb=\gcd(a,b)$. למעשה, $\gcd(a,b)=1$ אמ"מ קיימים $x,y\in\mathbb{Z}$ כך ש-$xa+yb=1$.

## אלגוריתם אוקלידס למציאת מחלק משותף מקסימאלי

אפשר להוכיח שלכל $a,b\in\mathbb{Z}$ מתקיים $\gcd(a,b)=\gcd(b,a\ mod\ b)$. על סמך התכונה הזאת אפשר לחשב בקלות את המחלק המשותף המקסימאלי של כל שני מספרים. למשל,

$$\gcd(345,56)=\gcd(56,345\ mod\ 56)=\gcd(56,9)=\gcd(9,56\ mod\ 9)=\gcd(9,2)=1$$

באופן כללי האלגוריתם נתון ע"י:

function $\gcd(a,b)$

if $b=0$ return $a$

else return $\gcd(b,a\ mod\ b)$

האלגוריתם רץ בזמן $O(\log\min(a,b))$.

קיים גם אלגוריתם אוקלידס המוכלל שבעזרתו אפשר לחשב גם את ה-$x,y$ שעבורם מתקיים $xa+yb=\gcd(a,b)$ בזמן לוגריתמי.

## פונקצית $\phi$ של אוילר

**פונקצית $\phi$ של אוילר** מוגדר להיות $\phi(n)=\#\{a:1\le a\le n\wedge\gcd(a,n)=1\}$, כלומר מספר האיברים מבין $\{1,\dots,n\}$ שזרים ל-$n$.

למשל, $\phi(10)=4$ כי בדיוק המספרים 1,3,7,9 הם זרים ל-10. לכל מספר ראשוני $p$ מתקיים $\phi(p)=p-1$ ואם גם $q$ ראשוני אז $\phi(pq)=(p-1)(q-1)$. באופן כללי, לכל $n\in\mathbb{N}$ מתקיים

$$\phi(n)=n\prod_{p|n}\left(1-\frac{1}{p}\right)$$

למשל, $10=2\cdot5$ ומתקיים $\phi(10)=10\left(1-\frac{1}{2}\right)\left(1-\frac{1}{5}\right)=10\cdot\frac{1}{2}\cdot\frac{4}{5}=4$.

## חבורות

**חבורה** היא זוג $(G,+)$ כאשר $G$ קבוצה ו-$+: G \times G \to G$ פעולה דו-מקומית שמקיימת את התכונות הבאות:

1. **אסוציאטיביות**: לכל $a,b,c \in G$ $(a+b)+c = a+(b+c)$
2. **איבר יחידה**: קיים $e \in G$ כך שלכל $a \in G$ $a+e = a = e+a$
3. **איבר נגדי**: לכל $a \in G$ קיים $-a \in G$ כך ש-$a+(-a) = e = (-a)+a$

למשל $(\mathbb{Z},+)$ חבורה ולכל שדה $F$ $(M_n(F),+)$ חבורה כאשר החיבור הוא חיבור מטריצות. גם $(GL_n(F),\cdot)$ חבורה כאשר $GL_n(F) = \{A \in M_n(F): \det A \neq 0\}$ והפעולה היא כפל מטריצות.

חבורה $G$ היא **אבלית** (או **חילופית**) היא חבורה שבה הפעולה היא קומוטטיבית, כלומר לכל $a,b$ בחבורה $a+b = b+a$. $G$ **ציקלית** אם קיים $g \in G$ כך ש-$G = \{g^i : i \in \mathbb{Z}\}$. במקרה זה מסמנים $G = \langle g \rangle$. למשל, $\mathbb{Z}_n = \langle 1 \rangle$.

## חבורות שאריות

לכל $n \in \mathbb{Z}$ נגדיר את החבורה $\mathbb{Z}_n = (\{0,\dots,n-1\},+_n)$ כאשר החיבור הוא חיבור מודולו $n$. זאת חבורה חיבורית בגודל $n$:

1. ברור שהפעולה מוגדרת היטב כי פשוט לפי ההגדרה לכל $a,b \in \mathbb{Z}_n$ $a+_n b \in \mathbb{Z}_n$.
2. ברור שהחיבור או אסוציאטיבי.
3. איבר האפס הוא איבר היחידה כי $a+_n 0 = a = 0+_n a$.
4. לכל $a \in \mathbb{Z}_n$ האיבר הנגדי הוא $n-a$ כי $(n-a)+_n a = a+_n(n-a) = n \bmod n = 0$.

אי אפשר להגדיר באותו אופן על $\mathbb{Z}_n$ מבנה של חבורה עם פעולת הכפל מודולו $n$. ראשית, ל-0 אין הופכי כפלי. אבל אפילו אם נסתכל על $\mathbb{Z}_n \setminus \{0\}$ עדיין לא תמיד נקבל חבורה. למשל נסתכל על $\mathbb{Z}_{10} \setminus \{0\}$. ל-2 שם אין הופכי כפלי, שהרי מכפלה של 2 בכל מספר אחר נותנת מספר זוגי ולכן לא יכול להיות שהשארית בחלוקה ב-10 תיתן 1.

לעומת זאת, ניתן להגדיר חבורה כפלית $\mathbb{Z}_n^* = (\{x \in \mathbb{Z}_n : \gcd(x,n) = 1\},\cdot_n)$ שמורכבת מכל המספרים שזרים ל-$n$. ברור שאיבר היחידה הוא 1 ושהכפל הוא אסוציאטיבי. רק צריך להראות סגירות של הכפל וקיום הופכי. ברור שאם $a,b$ זרים ל-$n$ אז גם $ab$ זר ל-$n$. ואם $a \in \mathbb{Z}_n^*$ אז קיימים $x,y \in \mathbb{Z}$ כך ש-$xa+yn = 1$ ולכן $xa \equiv 1 (mod\ n)$ ולכן $x \bmod n \in \mathbb{Z}_n$ הוא ההופכי הכפלי של $a$. אבל גם $\gcd(x,n) = 1$ כי $ax+yn = 1$ והרי $a,y \in \mathbb{Z}$.

**משפט פרמה הקטן**: אם $p$ ראשוני אז לכל $a \in \mathbb{Z}$ כך ש-$\gcd(a,p) = 1$ מתקיים

$$a^{p-1} \equiv 1 (mod\ p)$$

נשים לב שמתקיים $a^{p-1} = a \cdot a^{p-2} \equiv 1 (mod\ p)$ ולכן $a^{p-2}$ הוא ההופכי של $a$ ב-$\mathbb{Z}_n^*$. בשביל לחשב את $a^{p-2}$ צריך $O(\log p)$ פעולות. אפשר גם לחשב הופכי בעזרת אלגוריתם אוקלידס המוכלל ואז זה ייקח $O(\log a)$ פעולות.

איבר $a$ בחבורה $G$ ייקרא **ריבוע** אם קיים $b \in G$ כך ש-$a = b^2$. אז $b$ נקרא **שורש** של $a$. במקרה ש-$G = \mathbb{Z}_p^*$ $a$ הוא ריבוע אם $b^2 \equiv a (mod\ p)$. נשים לב שאם $b^2 \equiv a (mod\ p)$ אז גם

$$(p-b)^2 \equiv a (mod\ p)$$

שהרי $(p-b)^2 = p^2 - 2pb + b^2 \equiv b^2 (mod\ p)$. כלומר ב-$G = \mathbb{Z}_p^*$ לכל ריבוע יש שני שורשים!

## בדיקת ראשוניות

למצוא מספר ראשוני קטן זו לא בעיה, אבל בד"כ רוצים למצוא מספרים ראשוניים חדשים באורך אלפי ביטים. בוודאי אם $n$ שלהם אפשר באופן פשוט לבדוק אם הוא ראשוני בזמן $O(\sqrt{n})$. אבל אם $n$ באורך 1000 ביטים זה אומר שצריך $O(2^{500})$ פעולות וזה ממש המון.

נסמן ב-$\pi(n)$ את מספר המספרים הראשוניים עד $n$. ידוע ש-$\pi(n) \approx \frac{n}{\ln n}$. בהנחת התפלגות אחידה בערך $\frac{1}{\ln n}$ מהמספרים שמסביב ל-$n$ יהיו ראשוניים. אז למשל אם $p$ אקראי באורך 1000 ביטים אז בתחום $(p-350, p+350)$ סביר שנמצא מספר ראשוני.

## אלגוריתם בדיקת הראשוניות של מילר ורבין

האלגוריתם של מילר ורבין הוא אלגוריתם הסתברותי שבודק אם מספר נתון הוא ראשוני או לא. אם האלגוריתם מכריז על המספר כלא ראשוני הוא אכן לא ראשוני, ואילו אם האלגוריתם מכריז שהמספר הוא ראשוני אז זה נכון בהסתברות גבוהה שתלויה במספר האיטרציות של האלגוריתם.

בהינתן קלט $n$ אי זוגי האלגוריתם פועל באופן הבא:

1. רושמים את $n-1$ בצורה $n-1 = 2^s \cdot d$.
2. $k$ פעמים:
   2.1. בוחרים $a$ אקראי בתחום $[1, n-1]$
   2.2. אם $a^d \not\equiv 1 (mod\ n)$ וגם $a^{2^r d} \not\equiv -1 (mod\ n)$ לכל $r \in [0, s-1]$ אז המספר פריק
3. המספר כנראה ראשוני

אפשר להוכיח שההסתברות ש-$n$ פריק אבל הוא בכל זאת יעבור איטרציה אחת של האלגוריתם בהצלחה היא לכל היותר $\frac{1}{2}$. אז סה"כ אם האלגוריתם הכריז על מספר שהוא ראשוני זה נכון בהסתברות לכל הפחות $\frac{1}{2^k}$. בד"כ לוקחים $k = 100$ וזה נותן הסתברות טובה מאוד.

אם מממשים את האלגוריתם בצורה הכי ישירה סיבוכיות זמן הריצה שלו היא $O(k \cdot \log^3 n)$ אבל בעזרת מימושים יותר מתוחכמים אפשר להוריד את זה. בכל אופן, זה די יעיל.

# קריפטוגרפיה אסימטרית

## המודל האסימטרי

במודל הקריפטוגרפיה האסימטרית לכל ישות $A$ בעולם יש שני מפתחות: מפתח פרטי סודי $Prv(A)$ ומפתח ציבורי $Pub(A)$ שידוע לכל העולם.

ההצפנה מתבצעת באמצעות פונקצית הצפנה $C = E\big(M, Pub(A)\big)$ וכל אחד יכול להצפין הודעות ל-$A$ אבל רק $A$ יכול לפענח אותן ע"י $M = D\big(C, Prv(A)\big)$.

באופן דומה, רק $A$ יכול לחתום על ההודעות שלו ע"י $\sigma = S\big(M, Prv(A)\big)$ וכל אחד יכול לאמת את החתימה שלו ע"י בדיקת $V\big(M, \sigma, Pub(A)\big)$. כמובן ההודעה עצמה דרושה לצורך האימות.

המודל כפי שהוא לא בהכרח מגדיר הצפנה טובה כי למשל לפי המודל אין מניעה להשתמש ב-$E = id$. גם אם המפתח הפרטי והציבורי זהים זה לא כל כך טוב. אז יש כמה דרישות שהגיוני לבקש:

1. הפונקציות $S, V, E, D$ ידועות
2. הפונקציות $S, V, E, D$ קלות לחישוב בהינתן מפתח מתאים
3. בהיעדר מפתח מתאים קשה מאוד לחלץ את $M$ מ-$C$ או לזייף זוג $(M, \sigma)$
4. ידיעת המפתח הציבורי לא מאפשרת גילוי של המפתח הפרטי אפילו בהינתן זוגות רבים $(M, C)$

## שיטת ההצפנה של Merkle

למעשה מה שהדרישות אומרות הוא שאנחנו מחפשים פונקציה חד-כיוונית שבהינתן "רמז" קל לחשב אותה אבל ללא ה"רמז" זה קשה מאוד. באנגלית זה נקרא **One-Way Trap-Door Function**.

בשנות ה-70 מתמטיקאים התעניינו מאוד במציאת פונקציות כאלה.

נסתכל בבעיית ה-Knapsack. יש $N$ עצמים $1, \dots, N$ שלכל אחד מהם יש משקל שלם חיובי $k_i$. בהינתן רשימה של עצמים קל מאוד לחשב את המשקל שלהם, אבל אם אומרים לנו שיש שק של עצמים שהמשקל הכולל שלו הוא $C$ זה דווקא בכלל לא קל למצוא את רשימת העצמים. נסמן

$$X_i = \begin{cases} 1 & i \text{ is in the sack} \\ 0 & \text{otherwise} \end{cases}$$

אז $C = \sum_{i=1}^{N} X_i k_i$. בסימונים אלה בהינתן $\vec{X}$ קל לחשב את $C$ אבל בהינתן $C$ קשה לחשב את $\vec{X}$.

את הבעיה הזאת אפשר להפוך לשיטת הצפנה. נניח שיש $N$ עצמים ולעצם $i$ יש משקל ידוע $k_i$. נניח שההודעה היא $M = X_1 \dots X_N$ כאשר $X_i \in \{0,1\}$. אז נצפין אותה ע"י $C = \sum_{i=1}^{N} X_i k_i$. אבל ההצפנה הזאת היא לא חח"ע ולכן אי אפשר לפענח. למשל, אם $\vec{k} = (1,2,3,4)$ ו-$C = 5$ אז יש שני פענוחים אפשריים: $(0,1,1,0)$ ו-$(0,0,0,1)$.

לפי Merkle כדי שההצפנה תהיה חח"ע צריך לבחור וקטור $\vec{k}$ שהוא **super-increasing**, כלומר לכל $i$ מתקיים $k_i > \sum_{j=1}^{i-1} k_j$. אם וקטור המשקלות מקיים את זה אז אשר לפענח בצורה חד-ערכית: ידוע ש-$C = \sum_{i=1}^{N} X_i k_i$. כדי למצוא את $\vec{X}$ בכל שלב סוקרים את $\vec{k}$ מסופו לתחילתו ובוחרים את $k_i$ המקסימאלי שעדיין לא גדול מ-$C$. ה-$k_i$ הזה חייב להיות בשק כי אחרת אפילו אם נבחר את כל העצמים הקודמים לא נוכל להגיע למשקל הנחוץ $C$. אז נעדכן $C = C - k_i$ ונמשיך. זה אלגוריתם עם סיבוכיות ליניארית.

עכשיו אלגוריתם ההצפנה הוא הפיך אבל אין בו שום דבר סודי, הרי אמרנו שהמשקל $\vec{k}$ ידוע. Merkle הציע שיטה סודית:

נבחר $m > \sum_{i=1}^{N} k_i$ ונמצא $w$ כלשהו זר ל-$m$. נגדיר את המפתח הציבורי להיות הווקטור

$$\vec{k}^* = \left(w \cdot k_1 (mod\ m), \dots, w \cdot k_N (mod\ m)\right)$$

והמפתח הפרטי הוא $\left(m, w, w^{-1}(mod\ m), \vec{k}\right)$ כאשר $\vec{k}$ הוא super-increasing.

הודעה $M = X_1 \dots X_N$ תוצפן ע"י $C^* = \sum_{i=1}^{N} k_i^* X_i$ ואת זה כל אחד יכול לעשות כי המפתח הציבורי $\vec{k}^*$ ידוע לכולם. בשביל לפענח ראשית נחשב את:

$$\begin{aligned}
C &= (C^* \cdot w^{-1})\ mod\ m \\
&= \left(\left(\sum_{i=1}^{N} k_i^* X_i\right) \cdot w^{-1}\right)\ mod\ m \\
&= \left(\sum_{i=1}^{N} k_i^* X_i \cdot w^{-1}\right)\ mod\ m \\
&= \left(\sum_{i=1}^{N} \left((k_i^* X_i \cdot w^{-1})\ mod\ m\right)\right)\ mod\ m \\
&= \left(\sum_{i=1}^{N} k_i X_i\right)\ mod\ m \\
&= \sum_{i=1}^{N} k_i X_i
\end{aligned}$$

כאשר המעבר נכון בגלל ש-$m > \sum_{i=1}^{N} k_i \geq \sum_{i=1}^{N} k_i X_i$.

עכשיו יש לנו ביטוי $C$ שחושב ע"י וקטור $\vec{k}$ שהוא super-increasing ואפשר לפעול בדרך שתוארה למעלה כדי לפענח את $\vec{X}$.

אם הווקטור $\vec{k}$ הוא באורך $n$ אז הסכום שלו הוא לפחות $1 + 2 + 4 + \dots + 2^{n-1} = 2^n - 1$ אז קשה מאוד לעשות חיפוש ממצה.

כמה חודשים לאחר ש-Merkle פרסם את השיטה שלו עדי שמיר שבר את הצופן...

## חתימה חד-פעמית של Lamport

Lamport הציע מנגנון לחתימה חד-פעמית. הצד החותם ימציא וקטור זוגות $\left((x_1, y_1), \dots (x_N, y_N)\right)$ כאשר כל $x_i$ ו-$y_j$ הם אקראיים וגדולים. זה יהיה המפתח הפרטי. המפתח הציבורי הוא וקטור הזוגות $\left(\left(h(x_1), h(y_1)\right), \dots, \left(h(x_N), h(y_N)\right)\right)$ כאשר $h$ פונקצית ערבול.

בהינתן הודעה $M = b_1 \dots b_N$ החתימה של אליס תהיה $(z_1, \dots, z_N)$ כאשר

$$z_i = \begin{cases} x_i, & b_i = 0 \\ y_i, & b_i = 1 \end{cases}$$

כדי לוודא את הביט ה-$i$ של ההודעה, אם $b_i = 0$ בודקים ש-$h(z_i) = h(x_i)$ ואם $b_i = 1$ בודקים ש-$h(z_i) = h(y_i)$. נשים לב שאם יש ביט בהודעה ששונה, נניח בה"כ ש-$b_1$ מ-0 ל-1 אז החתימה שלו היא $z_1 = x_1$ והרי לפי אלגוריתם הבדיקה נבדוק ש-$h(x_1) = h(y_1)$ אבל בהנחה שפונקצית הערבול טובה הסיכוי שהם שווים הוא מאוד קטן. ואילו אם אף ביט של ההודעה לא שונה אז כל ההשוואות יהיו חיוביות.

החתימה היא חד-פעמית משום שאם שולחים הודעות שונות $M, M'$ עם אותו מפתח זה יחשוף חלק מהזוגות $(x_i, y_i)$. למעשה, זה יחשוף בדיוק את הזוגות שעבורם $b_i \neq b_i'$ ואז בהודעות הבאות אפשר יהיה לשנות את הביטים האלה ואת החתימה בהתאם.

כמובן, השיטה הזאת בכלל לא פרקטית כי החתימה היא באורך ההודעה וזה יכול להיות מאוד ארוך.

## בעיות קשות

כפי שראינו Merkle ניסה לקחת בעיה חישובית קשה ולהפוך אותה לשיטת הצפנה. ספציפית השיטה שלו לא עבדה טוב כל כך, אבל הרעיון ככלל הוא טוב. כיום שיטות ההצפנה מתבססות בעיקר על שתי בעיות קשות.

### בעיית הלוג הדיסקרטי

בהינתן $p$ ראשוני גדול ובהינתן $g$ כך ש-$\mathbb{Z}_p^* = \langle g \rangle$ קל לחשב את $y = g^x \bmod p$ אבל אם נתונים $y, g, p$ כנ"ל קשה לחשב את $x$ שמקיים $y = g^x \bmod p$. נשים לב שאי אפשר לעשות כאן חיפוש בינארי כמו במקרה של החבורה $\mathbb{Z}$ כי ב-$\mathbb{Z}_p^*$ אין סדר.

אז בשימוש בבעיית הלוג הדיסקרטי אפשר להשתמש ב-$x$ כמפתח הפרטי וב-$y$ כמפתח הציבורי.

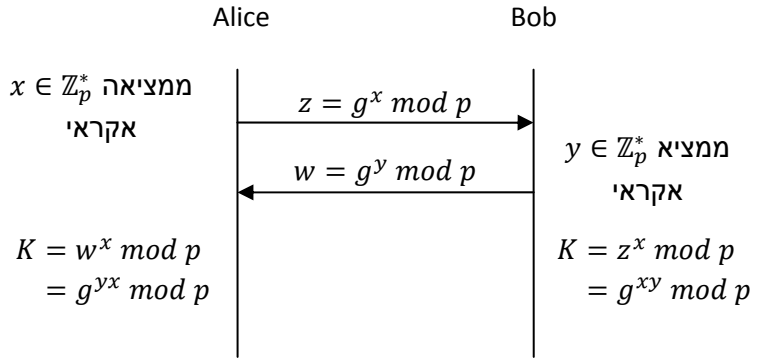
### בעיית הפירוק לגורמים ראשוניים

בהינתן ראשוניים גדולים $p, q$ קל לחשב את המכפלה שלהם $N = p \cdot q$, אבל בהינתן $N$ כזה קשה למצוא את המחלקים הראשוניים שלו. כאן המפתח הפרטי יכול להיות הפירוק לגורמים והמפתח הציבורי יכול להיות $N$.

## אלגוריתם החלפת המפתחות של Diffie-Hellman

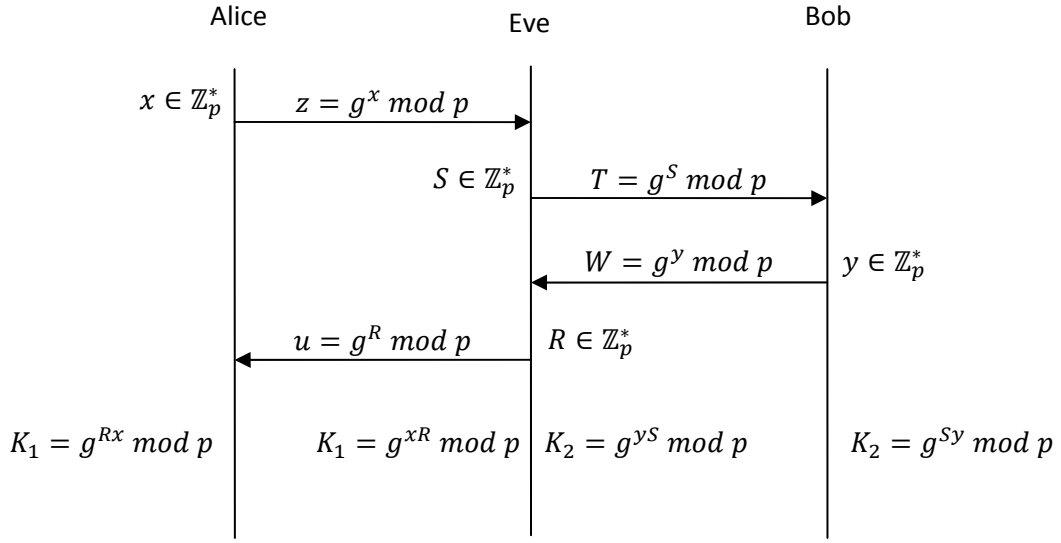
עד עכשיו כשדיברנו על אלגוריתמי הצפנה סימטריים הנחנו שלשני הצדדים שמעוניינים בתקשורת יש מפתח סודי מתואם מראש. אבל יכול להיות ששני הצדדים כלל לא מכירים זה את זה או חיים בצדדים שונים של העולם. מן הראוי לתת את הדעת על האופן שבו הם מתאמים מפתחות כדי שיוכלו לדבר בצורה מוצפנת או לחתום על הודעות.

Diffie ו-Hellman הציעו פרוטוקול (שנקרא לו DH) להחלפת מפתחות. ההנחה הבסיסית היא שיש $p$ ראשוני גדול שידוע לכל העולם וכן ידוע יוצר $g$ של $\mathbb{Z}_p^*$. עכשיו אליס שנמצאת בארה"ב ובוב שנמצא בסין יכולים לתאם מפתח סודי באופן הבא:



כל העולם יכול לדעת את $p, g, z, w$ אבל בגלל שבעיית הלוג היא קשה אף אחד לא יכול לפענח את $x$ ואת $y$ וככה אי אפשר לפענח את המפתח $K$ אבל לאליס ולבוב יש מפתח משותף. למעשה ההנחה כאן שהיא איזו וריאציה על בעיית הלוג הדיסקרטי – בהינתן $z$ ו-$w$ כמו בפרוטוקול קשה לפענח את $x$ ואת $y$.

בפרוטוקול זה יש בעייתיות. הצדדים המשתתפים לא יכולים לדעת עם מי הם באמת מדברים. ואכן, האלגוריתם לא עמיד בפני התקפת man-in-the-middle:




איב יכולה להיכנס לתקשורת שבין אליס לבוב ולדבר עם כל אחד מהם כאילו היא השני. אז אליס ובוב חושבים שהם מדברים אחד עם השני בסודיות גמורה. הם אכן מדברים בסודיות גמורה כי אף אחד בעולם לא יכול לפענח לא את $K_1$ ולא את $K_2$ אלא שהם לא מדברים אחד עם השני, אלא עם איב...

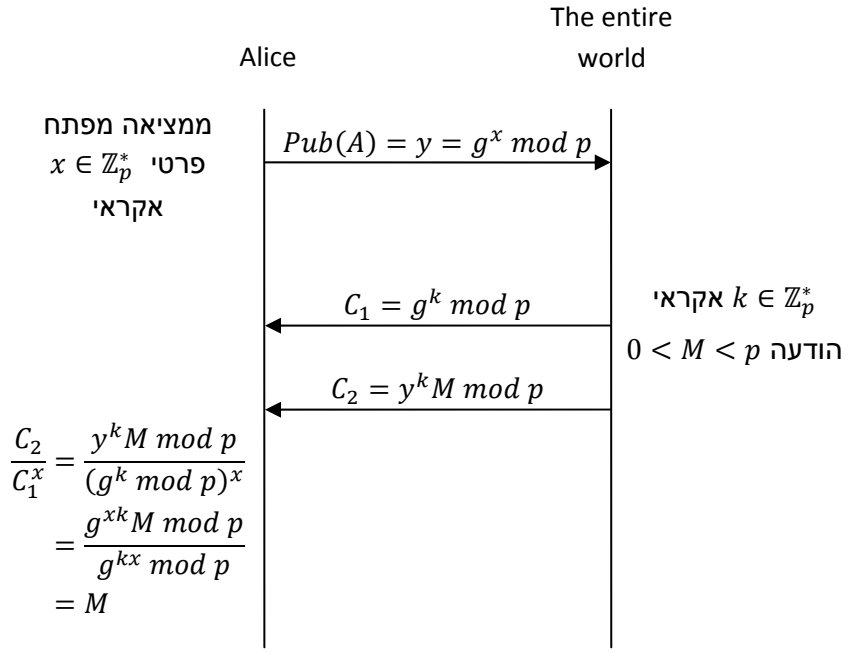
על אף החסרונות שלו האלגוריתם הזה מאוד נפוץ אבל יש לו שתי וריאציות:

1. **קו מוגן** – אם הקו מוגן, לא נגד האזנה, אבל לפחות נגד חיתוך או שרואים שהקו לא נפגע אז אפשר להריץ את האלגוריתם כפי שהוא, שהרי הבעיה היחידה שלו היא שמישהו ייכנס בין אליס לבוב בתקשורת.
2. **DH מאומת** – הצדדים מלכתחילה יודעים משהו אחד על השני כדי שהם ידעו שהם אכן מדברים עם מי שצריך.

## האלגוריתם של El-Gamal

### הצפנה

זהו אלגוריתם הצפנה שמבוסס על בעיית הלוג הדיסקרטי. אליס רוצה ליצור לעצמה מפתח פרטי ומפתח ציבורי. הנחת העבודה היא שיש ראשוני $p$ גדול שידוע לכל העולם וכן ידוע $g \in \mathbb{Z}_p^*$ כך ש-$\mathbb{Z}_p^* = \langle g \rangle$.

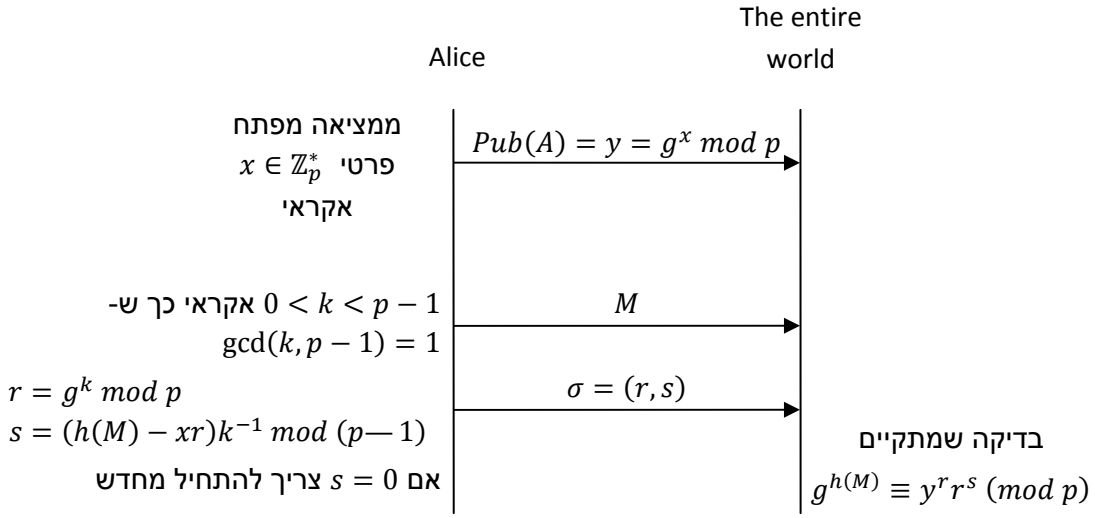



נשים לב שהתנאי ש-$0 < M < p$ הוא הכרחי כי אחרת היא לא תשוחזר בצורה נכונה מאחר שהחישובים נעשים בחבורה $\mathbb{Z}_p^*$.

גם כאן, בדומה לאלגוריתם DH אין שום אלמנט של אימות זהות. לאליס אין מושג מאיפה מגיעה ההודעה. שולח ההודעה יכול להזדהות אבל הזהות שלו לא מאומתת.

### חתימה

שוב הנחת העבודה היא שיש ראשוני $p$ גדול שידוע לכל העולם וכן ידוע $g \in \mathbb{Z}_p^*$ כך ש-$\mathbb{Z}_p^* = \langle g \rangle$. כמו כן נתונה פונקצית ערבול קריפטוגרפית טובה $h$. אליס רוצה לחתום על הודעה ושכל העולם יוכל לוודא אותה.




ה-$k$ האקראי גורם לכך שלאותה הודעה שחתומה פעמיים יש חתימה שנה בכל פעם. זה מונע התקפת replay. נראה רק למה אכן יש אימות. מההגדרה של $s$ נובע שמתקיים

$$sk + xr \equiv h(M) \pmod{p-1}$$

ולכן מהמשפט הקטן של פרמה

$$g^{h(M)} \equiv g^{sk+xr} \equiv g^{xr} g^{sk} \equiv y^r r^s \ (mod\ p)$$

## האלגוריתם RSA

זה אלגוריתם שמשתמש בקושי של בעיית הפירוק לגורמים ראשוניים. כדי לבנות לעצמה מפתח פרטי ומפתח ציבורי אליס פועלת באופן הבא:

- בוחרת ראשוניים גדולים ולא קשורים $p, q$
- מחשבת $n = p \cdot q$
- מחשבת $\phi(n) = (p-1)(q-1) = |\mathbb{Z}_n^*|$
- ממציאה $e$ זר ל-$\phi(n)$
- מחשבת $d = e^{-1}\big(mod\ \phi(n)\big)$

כעת המפתח הציבורי של אליס הוא $(n, e)$ והמפתח הפרטי הוא $(n, d)$ ואפשר להשמיד את $p$, $q$ ו-$\phi(n)$ כדי שאף אחד לא ימצא אותם.

בעזרת המפתחות האלה אפשר להצפין ולחתום (כמובן לא כדאי להשתמש באותם מפתחות גם בשביל ההצפנה וגם בשבילך האימות):

- **הצפנה**: הודעה $M < n$ נצפין ע"י $C = M^e \ mod\ n$. כל אחד יכול להצפין הודעה עבור אליס בשימוש במפתח הציבורי שלה.
- **פענוח**: $M = C^d \ mod\ n$. רק אליס יכולה לפענח את ההודעה המוצפנת באמצעות המפתח הפרטי שלה. אם $M$ אינה כפולה של $p$ אז לפי משפט פרמה הקטן $M^{p-1} \equiv 1 (mod\ p)$ ואז ב-$\mathbb{Z}_p^*$ מתקיים:

$$\begin{aligned} C^d = M^{ed} &= M^{\alpha\phi(n)+1} = M \cdot M^{\alpha(p-1)(q-1)} \\ &= M \cdot (M^{p-1})^{(q-1)\alpha} = M \cdot 1 = M \end{aligned}$$

- **חתימה**: אליס יכולה לחתום על הודעה ע"י $\sigma = M^d \ mod\ n$
- **וידוא**: כל אחד יכול לאמת את החתימה ע"י בדיקה ש-$M = \sigma^e \ mod\ n$

כפי שהאלגוריתם תואר כאן פונקצית החתימה היא כפלית, כלומר

$$\sigma_K(M_1 \cdot M_2) = \sigma_K(M_1) \cdot \sigma_K(M_2)$$

וזאת בעיה כי זה אומר שאם ראינו כמה הודעות עם החתימות שלהן אז אנחנו יכולים לזייף הודעות עם חתימות שנראות אותנטיות. בשביל לפתור את הבעיה הזאת RSA-Labs פיתחו סטנדרטים לשימוש בטכנולוגיה. למשל, לכל הודעה צריך להוסיף מסגרת מסוימת. ככה אי אפשר לכפול הודעות כדי לזייף חתימות.

בעיה נוספת היא שאם ההודעה $M$ היא קטנה מספיק אז $M^e < n$ ואז אפשר פשוט להוציא שורש ולפענח את ההודעה המקורית. במידה מסוימת התוספות שצריך להוסיף לכל הודעה פותרות גם את הבעיה הזאת.

כדי שאי אפשר יהיה לעשות חיפוש ממצה המפתח הפרטי $d$ צריך להיות גדול ומשיקולים של יעילות החישוב כדאי לבחור דווקא $e$ קטן.

# אימות אנשים

במערכת מחשב אפשר לדבר על שני סוגי עצמים: אנשים ומחשבים ומכאן שיש ארבעה סוגי פרוטוקולים:

- אדם מול אדם
- אדם מול מחשב
- מחשב מול אדם
- מחשב מול מחשב

לאימות יכולות להיות כל מיני מטרות, כתלות במערכת. למשל,

- **זיהוי** – כלומר לדעת מול מי אנחנו עובדים. כאן הכוונה היא רק לזיהוי ראשוני. למשל, יכול להיות שאנחנו מסתכלים על תמונה של מישהו ומזהים שזה מישהו שאנחנו מכירים. אבל יכול להיות שיש לו אח תאום...
- **אימות** – להיות בטוחים שמי שאנחנו חושבים שאנחנו מדברים איתו הוא אכן מי שאנחנו מדברים איתו.
- **הרשאות** – לאפשר לכל אחד לעשות כל ורק מה שמותר לו.
- **מעקב** – יצירת log של מה שנעשה. בוודאי נרצה שמידע יהיה מדויק ונכון.

## פרמטרים לבחירת שיטת אימות או זיהוי

יש כל מיני פרמטרים שיש לקחת בחשבון כאשר בוחרים את שיטת העבודה.

- **זמן** – יש מערכות שבהן לא אכפת לנו לחכות כמה שניות (למשל בקנייה בכמה אלפי שקלים) ולפעמים כמה שניות זה יותר מדי זמן (למשל בקניית שלגון).
- **עלות** – יש לקחת בחשבון כמה כסף אנחנו מוכנים להשקיע במערכת והאם מה שאנחנו מגנים עליו שווה את זה.
- **אחזקה** – מהי תדירות ורמת התחזוקה שהמערכת דורשת
- **דיוק** – יש מצבים שבהם הדיוק פחות חשוב ויש מצבים שבהם יותר...
  - **דיוק חיובי**: המערכת מקבלת רק אנשים שמותר להם להיכנס אבל לפעמים דוחה אנשים נכונים. בכל אופן, אנשים שאין להם הרשאות לא נכנסים אף פעם למערכת.
  - **דיוק שלילי**: המערכת לא מקבלת רק אנשים שאין להם הרשאות. כלומר, לפעמים יש אנשים ללא הרשאות שמצליחים להיכנס אבל אף פעם אין אדם שיש לא הרשאה שלא מצליח להיכנס.
- **קלות שימוש** – בדיקת טביעת אצבע היא תהליך פשוט למשתמש ואילו נתינת דם עבור בדיקת DNA זה תהליך קצת פחות נעים...
- **העברת מזהה** – למקרה שרוצים שמישהו יוכל למלא מקום
- **החזרת מזהה** – אם למשל האדם כבר לא עובד בחברה יותר או שעבר לתפקיד אחר. אם המזהה הוא סיסמה אז אי אפשר לקחת אותו חזרה אבל אם זה למשל כרטיס מחולל סיסמאות אז דווקא אפשר.
- **שינוי או תפוגת מזהה** – אם הייתה פריצת אבטחה או סתם כי מדי פעם צריך להחליף. למשל סיסמה אפשר לשנות אבל טביעת אצבע לא (לא באמצעים סטנדרטיים לפחות).

## סוגים של מנגנוני אימות

נדון בשלושה מנגנונים שונים של אימות אנשים. כיום בד"כ עובדים עם שילוב של המנגנונים ולא רק אחד מהם. זה משיג דיוק גבוה יותר ואבטחה טובה יותר.

### משהו שהמשתמש יודע

כאן מדובר בעיקר על סיסמאות, אבל זה יכול להיות גם איזשהו PIN או מידע אישי. הנחת העבודה היא שאפשר לסמוך על השכל של המשתמש שיזכור את הדברים הנחוצים. מזהה אידיאלי יהיה כזה שקל למשתמש לזכור אבל קשה למתקיף לנחש, למצוא או לפצח. הבעיה עם הדרישות האלה היא שהן לא ממש ניתנות להשגה עם אנשים מהדגם הנוכחי. בוודאי כדי שאי אפשר יהיה לנחש את המזהה הוא צריך להיות רנדומאלי וכדי שאי אפשר יהיה פשוט לעשות חיפוש ממצה הוא צריך להיות די ארוך. אבל הרוב המוחלט של בני האדם כיום לא מסוגלים לזכור רצפים אקראיים של סימנים. סיסמה טיפוסית היא מילה באנגלית או דמות אנגלית, מספר טלפון, שם של מישהו, מניפולציה כלשהי על הסיסמה הקודמת. אלה דברים שקל לנחש בד"כ. סביר להניח שאחרי מספיק ניחושים מתקיף יגלה את הסיסמה.

כפי שכבר נאמר, הסיסמה הרצויה היא משהו כמו 100 ביטים אקראיים. אבל זה משהו שאי אפשר לזכור. אם נסתפק בתווי ASCII ויזואליים אז מדובר במשהו כמו 20-15 תווים כאלה. אבל גם רצף תווי ASCII אקראיים אפשר לזכור אז אפשר להמיר את זה במשהו שניתן להגייה ולכן קל יותר לזכור, אבל זה מגדיל את הסיכוי שמתקיף ינחש את הסיסמה. אז אם משתמשים בסיסמה חסרת משמעות אך ניתנת להגייה רצוי שיהיה מדובר ב-40-20 תווים. או לחילופים אפשר להשתמש ב-50-30 תווים שמייצרים משפט, כלומר מילים בעלות משמעות עם קשר תחבירי (למשל "הסוס הוורוד רקד סמבה עם החביתה הכחולה").

בכל אופן, מה שצריך לקחת מהדיוק הזה הוא שהפעם בין הרצוי למצוי הוא מאוד גדול אבל בכל זאת סיסמה זה מאוד נוח למשתמש, אנחנו משתמשים בסיסמאות כל יום אפילו כמה פעמים ביום ואיכשהו העולם לא מתמוטט סביבנו...

#### מערכות סיסמאות

מערכת סיסמאות היא מאוד מורכבת, בעיקר משלוש סיבות:

- יש אלפי משתמשים
- יש מספר רב של שרתים
- אותם משתמשים מופיעים במערכות שונות

בכל המבנה הענק הזה בטוח יש סיסמאות חלשות!!

כדי שמשתמש יוכל להשתמש בסיסמה שלו הסיסמה צריכה להיות ידועה לשרת בדרך זו או אחרת. השרת צריך לשמור את הסיסמה או נגזרת כלשהי שלה. יש כל מיני אפשרויות לשמירה:

1. ללא הגבלת גישה ובאופן גלוי
2. עם הגבלת גישה (כלומר הרשאות) ובאופן גלוי
3. עם הגבלת גישה ובאופן מוצפן (למשל קובץ מוצפן, אבל אז צריך לשמור את המפתח איפשהו...)
4. עם הגבלת גישה ע"י פונקצית ערבול (כאן, בניגוד לאפשרות הקודמת, לא צריך מפתח)
5. שמירת קובץ סיסמאות בשרת שהוא גם מבודד פיסית מעבר להרשאות הגישה

נרחיב על האפשרות שמשתמשת בפונקציות ערבול. יש קובץ סיסמאות ששומר שמות משתמש ואת פונקצית הערבול של הסיסמאות שלהם. הקובץ לא צריך להיות מוצפן כי לפי ההנחה על פונקצית

הערבול קשה למצוא מקורות של ערכים. כל פעם שמשתמש מכניס את הסיסמה שלו המחשב מחשב את פונקצית הערבול של הסיסמה שהוא הקליד ומשווה למה ששמור אצלו.

הסכמה הזאת רגישה ל-**Online Dictionary Attack**. אם למתקיף יש הרשאת קריאה לקובץ הסיסמאות הוא יכול לבוא מוכן עם מלא זוגות $(PW, h(PW))$. ואז תוך כמה שניות הוא יכול למצוא התאמה בין אחת הסיסמאות למשהו שקיים במערכת. נשים לב שאם הוא מוצא התאמה בפונקצית הערבול זה לא אומר שהוא גילה את הסיסמה האמיתית אבל זה לא משנה כי גם המערכת בודקת רק את הזיהוי של פונקצית הערבול.

כדי להתמודד עם הבעיה הזאת ביוניקס משתמשים במנגנון שנקרא **Salt**. Salt הוא פשוט מספר אקראי שמוצמד לכל שם משתמש בכל פעם שהוא פותח חשבון או משנה סיסמה. בקובץ הסיסמאות נמצאות רשומות מהצורה $(username, salt, h(PW \parallel salt))$. בצורה הזו אפילו אם יש הרבה סיסמאות פופולאריות כל אחת מהן יש מופע שונה בגלל שה-salt שונה. זה מקשה מאוד מציאת התאמות, במיוחד אם ה-salt ארוך. אבל תהליך זה לא מונע **Offline Dictionary Attack**. אם למתקיף יש הרבה זמן הוא יכול לבוא מוכן עם כל הסיסמאות ולכל סיסמה הרבה salt-ים שונים.

### יתרונות וחסרונות של סיסמאות

לסיכום ניתן רשימה של יתרונות וחסרונות של שימוש בסיסמאות:

1. יתרונות:
   א. נוח מאוד למשתמש
   ב. עלות נמוכה בד"כ
   ג. אפשר להעביר סיסמה
   ד. אפשר לשנות ולהחליף סיסמה
   ה. בהנחה שאין התנגשויות גם הדיוק החיובי וגם הדיוק השלילי טובים
   ו. אחזקה נמוכה
2. חסרונות:
   א. אי אפשר להחזיר סיסמה
   ב. האבטחה לא מאוד טובה

## משהו שהמשתמש הינו

הכוונה היא לא למשהו שצריך לזכור אלא משהו שנישא עם המשתמש ומאפיין אותו. מדובר במידע די אישי שלאף אחד אחר אין אותו. למשל, טביעת אצבע, גיאומטריה של כף היד, טביעת קול, קשתית, רשתית, פנים, DNA ועוד. בד"כ מדובר באיזה מאפיין ביומטרי.

רצוי לשים לב שהדברים האלה אמנם אישיים אבל בכל זאת חלק מהם ניתנים ל"זיוף". למשל, אפשר להקליט קול ואז להשמיע את ההקלטה למערכת הזיהוי. למערכת שמשתמשת לזיהוי במצלמה אפשר אולי להראות תמונה. אופציה קצת פחות נעימה היא קטיעת אצבע לצורך זיוף טביעת אצבע...

### טביעת אצבע

המחקרים מראים שקשה למצוא שתי אצבעות זהות. אפילו לתאומים שהגיעו מאותה ביצית יש טביעת אצבע שונה (כי קמטי הגוף מתעצבים בשהייה במי השפיר וזהו תהליך סביבתי).

כדי להשתמש בטביעת אצבע לצורך זיהוי צריך להיות תהליך רישום שהו נותנים למחשב כמה תמונות של האצבע. המשתמש שומר את המידע הדרוש במקום שמזוהה עם המשתמש הרלוונטי. לכאורה השיטה נראית נוחה מאוד אבל למעשה יש כאן בעיה גדולה בשימוש: לכלוך על האצבע, לכלוך על הגלאי, פציעה ועוד. כל אלה יכולים לשבש את תהליך הזיהוי.

מלבד בעיה אופרטיבית זו יש פה גם פרצת אבטחה: אפשר לקחת תמונה של טביעת האצבע מכוס קפה או משהו ואז ליצור תמונה שבאים איתה לגלאי. יש גלאים יותר מתוחכמים שבודקים תמונה תלת-ממדית של האצבע בעזרת גלאי שמודד מוליכות. גם לגלאי כזה אפשר לבוא גם דגם של אצבע מפלסטלינה או משהו. אז יש גלאים עוד יותר מתוחכמים שבודקים את החיות של האצבע – טמפרטורה לחץ, דם וכו'. כמובן אם מביאים אצבע אמיתית שום דבר לא יעזור...

### זיהוי כף יד

לכל כף יד יש מבנה גיאומטרי ייחודי – אורך אצבעות, מרחקים בין אצבעות, עובי האצבעות וכו'. הבעיה עם השיטה הזאת היא שהיד יכולה להשתנות עם הזמן וגם המדידות לא מדויקות. אז צריך לתת איזה טווחי ביטחון, אבל זה כמובן מוריד את הדיוק של הזיהוי.

### בדיקת קשתית

קשתית היא החלק הצבעוני בעין שמקיף את האישון. הקשתית יכולה להיות דך לזהות אנשים בעזרת עיבוד תמונה. למשל, את האישה האפגאנית מהתמונה המפורסמת של נשיונל ג'יאוגרפיק זיהו כ-16 שנים לאחר הפעם הראשונה שהיא צולמה בעזרת ניתוח של הקשתית. דוגמה שקצת יותר קשורה לאבטחה היא כספומט. אפשר לשים מצלמה מעל הכספומט ובזמן שמקישים את הקוד הסודי המצלמה מצלמת את העין ובודקת התאמה. אז יש כאן בדיקה משולשת: גם כרטיס, גם קוד סודי וגם בדיקת קשתית, אז זה די יעיל.

### בדיקת רשתית

הרשתית היא שכבת תאים דקה באחורי גלגל העין. הרשתית מאוד מאפיינת אנשים ואפשר לצלם אותה בעזרת אור אינפרא-אדום. הדיוק הוא די טוב אבל הבדיקה היא קצת חודרנית ולא נוחה למשתמש.

### מאפיינים של זיהוי ביומטרי

1. עלות גבוהה יותר מאשר בשימוש בסיסמה
2. יש צורך באחזקה של כל המכשירים
3. נוחיות למשתמש תלויה בשיטה
4. אי אפשר להעביר את המזהה שזה יכול להיות גם יתרון וגם חיסרון כי מצד אחד קשה יותר לגנוב או לזייף אבל מצד שני לפעמים צריך להעביר אבל אי אפשר
5. אין צורך להחזיר את המזהה כי הוא הולך יחד עם הבן-אדם
6. למקרה שיש שינויים אפשר להירשם כל כמה שנים מחדש. מאפיינים ביומטריים לא משתנים בקצב מאוד גבוה בד"כ.

## משהו שהמשתמש מחזיק

מדובר באיזשהו פריט שהמשתמש יכול להחזיק אצלו ומשמש אותו לזיהוי מול המערכת. למשל:

1. תעודה מזהה (לא מן הנמנע שבעתיד מחשב יוכל לבדוק תעודה מזהה)
2. כרטיס נייר או פלסטיק
3. מרכיב חישובי כלשהו
4. מפתח מכאני

### כרטיס סיסמאות לא חישובי

הסכמה הרעיונית של שימוש בנוסחה היא שמשתמש מזדהה ומעביר את הסיסמה שלו:


אלא שהסיסמה לא מוגנת ואפשר פשוט להאזין לה בתקשורת ולכן היא חד-פעמית. אפשר לתת למשתמש פתק עם רשימת סיסמאות חד-פעמיות והמשתמש יקליד כל פעם סיסמה אחרת. כמובן הרשימה הזאת צריכה להופיע גם בשרת. כשהסיסמאות נגמרות צריך לייצר רשימה חדשה. נשאלת פה גם השאלה איך השרת מסונכרן עם המשתמש לגבי הסיסמה שהוא עומד להשתמש בה. אז אפשר למשל להשתמש בסיסמאות פשוט לפי הסדר, או שאפשר לתת לכל סיסמה מספר סידורי ואז המשתמש ישלח כל פעם זוג של מספר סידורי והסיסמה שמתאימה לו.

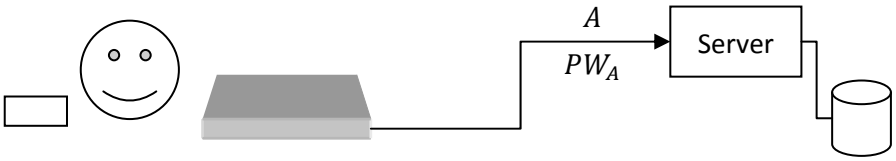
כרטיס הסיסמאות יכול להיות ויזואלי או ממוחשב, אבל בכל אופן אין לו שום רכיב חישובי. כל מה שהוא צריך זה יכולת לזכור דברים.

תכונות של השיטה הזאת:

- יתרונות:
  - הסיסמאות יכולות להיות חזקות כי המשתמש לא צריך לזכור אותן בע"פ
  - אין בעיית האזנה כי הסיסמאות חד-פעמיות
  - מחיר נמוך (זיכרון בלבד)
- חסרונות:
  - אפשר לעשות התקפת man-in-the-middle: מתקיף יכול להתחזות לשרת, לקבל את הסיסמה מהמשתמש, להודיע לו שאין שירות ואז להתחבר לשרת האמיתי בעצמו בעזרת הסיסמה שהוא כרגע קיבל.
  - צריך הרבה מאוד זיכרון בשרת כי כמו שכבר אמרנו יש אלפי משתמשים
  - אם הכרטיס נאבד זו בעיה גדולה
  - אם פורצים לשרת וגונבים את קובץ הסיסמאות זו בעיה עוד יותר גדולה (למרות שעל זה אפשר להתגבר באמצעות שמירת פונקצית ערבול ולא הסיסמאות עצמן)

### כרטיס סיסמאות חישובי

כרטיס סיסמאות יכול להיות לא סתם זיכרון אלא גנראטור של סיסמאות.




לכל משתמש $A$ פועלים באופן הבא: בוחרים $x_0$ אקראי גדול ומחשבים שרשרת סיסמאות $x_0,..,x_{N+1}$ כך ש-לכל $0 \le i \le N$ כאשר $h$ פונקצית ערבול. בכרטיס של המשתמש שמורים $N$-ו $x_0$ וכל מה שהכרטיס יודע זה לחשב את פונקצית הערבול בצורה מהירה. כדי להזדהות המשתמש שולח לשרת את שמו ואת $x_{N-1} = h^{(N-1)}(x_0)$. לשרת יש מסד נתונים שבו רשום $x_N$ אז ע"י הפעלת $h$ פעם אחת הוא מוודא שהסיסמה הייתה נכונה. כעת בכרטיס מעדכנים $N = N - 1$ ובשרת מעדכנים $x_N = x_{N-1}$ וכך נוצרת סיסמה חדשה לשימוש.

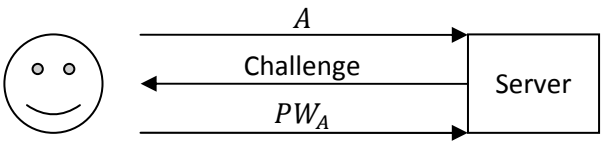
כרטיס זה מאפשר שימוש ב-$N$ סיסמאות שונות חד-פעמיות. הסכמה עובדת בגלל התכונות של פונקצית הערבול. נשים לב שמשתמשים בסיסמאות מהסוף להתחלה כי בהינתן $x_i$ כל אחד יכול לחשב את $x_{i+1} = h(x_i)$.

תכונות של השיטה הזאת:

- יתרונות:
  - אין חשש מהאזנה
  - הסיסמאות חזקות
  - הזיכרון הדרוש הן בשרת והן בכרטיס מועט
  - השרת מחזיק מידע לא רגיש (גניבת סוף השרשרת היא לא מסוכנת)
- חסרונות:
  - יותר יקר מכרטיס זיכרון בלבד
  - אפשר לעשות התקפת man-in-the-middle כמו במקרה של כרטיס סיסמאות לא חישובי
  - ככל ש-$N$ גדול יותר חישוב הסיסמה לוקח יותר זמן (אבל מצד שני זה אומר שצריך לחדש בתדירות נמוכה יותר). אפשר להתמודד עם זה ע"י שמירת נקודות ביניים בחישוב.

### Challenge Response

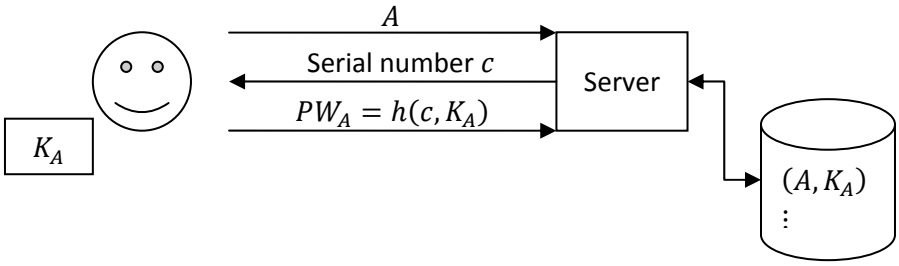
בשתי הוריאציות של כרטיס סיסמאות לא הצלחנו לפתור את בעיית התקפת man-in-the-middle. פרוטוקול אפשרי הוא **Challenge Response** (בקיצור CR). אחרי שהמשתמש מזדהה השרת שולח לו "אתגר" ורק לאחר מכן המשתמש שולח סיסמה שתלויה ב"אתגר" שניתן לו.




האתר יכול להיות מספר סידורי, זמן או מספר אקראי. הסיבה לכך היא שהם חד-פעמיים בהסתברות גבוהה מאוד.

#### CR סידורי

לכל משתמש $A$ יש מפתח סודי $K_A$ שידוע לשרת. בכל פניה של המשתמש לשרת השרת שולח לו מספר סידורי והמשתמש שולח סיסמה מתאימה שמחושבת באמצעות פונקצית ערבול.




לסכמה הזאת יש שלושה יתרונות עיקריים:

- מספר סידורי הוא קל לייצור ולכל שרת יכול אפילו להסתפק במונה אחד
- אין צורך בסנכרון בין השרת ללקוח
- הקלטת זוגות $(c, h(c, K))$ לא מטרידה בגלל ההנחה על פונקצית הערבול

וריאציה על הפרוטוקול הזה היא שדווקא המשתמש שולח אתגר $c$ סידורי לשרת. זה פועל באותו אופן אלא שכאן יש צורך בסנכרון עם השרת והשרת צריך לדעת שהמספר הסידורי הזה לא הופיע עדיין. אם השרת לא שומר את הנתונים האלה יש סכנת replay. אם יש המון לקוחות (וזה בד"כ המצב) אז זה מסובך לשמור לכולם מידע סנכרון, כי צריך מונה לכל לקוח, אבל היתרון הגדול הוא שהשרת רק מוודא ולא עסוק בפעילויות נוספות.

### CR מבוסס זמן

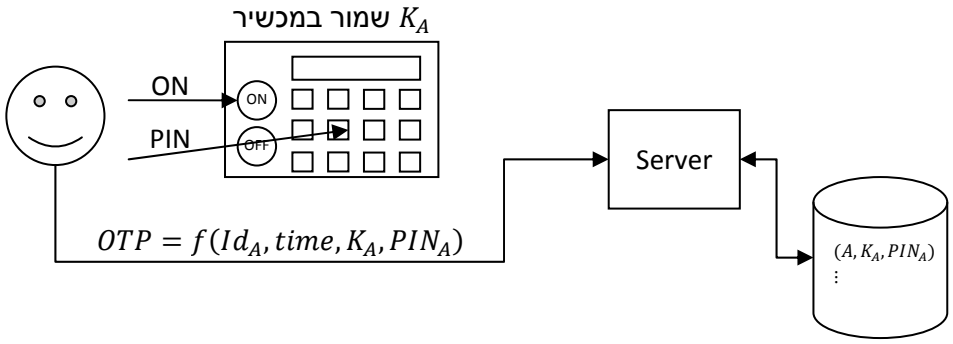
ברמת העיקרון הרעיון זהה לרעיון של אתגר סידורי, אלא שהאתגר הוא פשוט הזמן הנוכחי. אם יש סנכרון מושלם בין השרת ללקוח אז תיאורטית אין צורך בכלל לשלוח את האתגר כי שניהם יודעים מה השעה. אבל זה בד"כ לא המצב ולכן נשאלת השאלה מי צריך לשלוח את השעה למי – השרת ללקוח או הלקוח לשרת? אם אין סנכרון השרת חייב להיות זה ששולח את השעה משום שאחרת מתקיף יכול מראש להתחזות לשרת ולקבל ממשתמש רשימה של זוגות $\big(t, h(t,K)\big)$ ואח"כ להשתמש בהם מול השרת אמיתי. אבל אפילו אם השרת הוא זה ששולח את הזמן עדיין יש בעייתיות בשימוש משום שהתקשורת לוקחת זמן. אם עובדים ברמת דיוק גבוהה יותר מזמן התקשורת הלקוח לא יצליח לעולם להיכנס למערכת. לכן חייבים לפתוח חלון זמן מספיק גדול שלוקח בחשבון את זמן התקשורת. בזמן הזה לכאורה המערכת חשופה להתקפת man-in-the-middle אבל זה לא באמת איום משום שהמתקיף לא יכול לייצר סיסמאות נכונות בעצמו, אז כל מה שהוא יכול לעשות זה להעביר את התקשורת הלאה וזה לא כזה נורא...

### CR מבוסס מספר אקראי

כאן רק השרת יכול לשלוח מספר למשתמש, כי אחרת אין דרך לדעת שהמספר הוא באמת אקראי ושלא השתמשו בו עדיין. הבעיה היא שייצור מספרים אקראיים היא יקר וזה במיוחד בעייתי לשרת שאולי צריך לטפל באלפי לקוחות בדקה. אז אפשר לעבוד עם שרשראות פסאודו-אקראיות אבל זה פותח את המערכת להתקפת man-in-the-middle כי אם המתקיף יצליח להתחקות אחרי תהליך ייצור "המספרים האקראיים" אולי הוא יצליח להוציא מהמשתמש הרבה סיסמאות מראש ואז להשתמש בהן.

חשוב לציין שבכל שלוש השיטות שראינו עד כה השרת באמת יודע עם מי הוא מדבר רק בזמן של האימות. אם לאחר מכן מישהו נכנס באמצע התקשורת לשרת אין שום דרך לדעת על זה.

### ה-OTP האוניברסיטאי




לכל משתמש $A$ יש סיסמה $K_A$ ששמורה במכשיר ה-OTP והוא כלל לא יודע אותה. מה שהוא צריך לדעת כדי להפעיל את המכשיר הוא PIN. יש שתי אפשרויות:

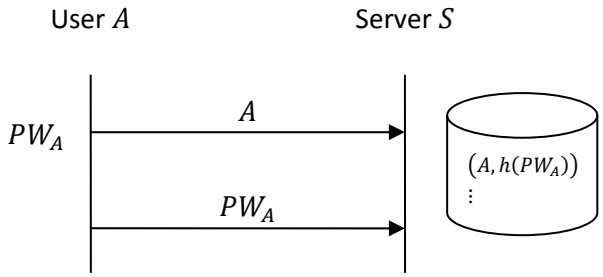
- ה-PIN משמש לפתיחת המכשיר ואז הוא צריך להיות שמור רק במכשיר
- ה-PIN משמש חלק מהבסיס לסיסמה ואז הוא צריך להיות שמור גם בשרת

כשנותנים את המכשיר למשתמש מסנכרנים את הזמן שלו עם הזמן של השרת. אבל עם הזמן הסנכרון הזה יכול נהרס. אם זה קורה באופן עקבי השרת יכול ללמוד את קצב הזליגה של הכרטיס ואז זה עדיין עובד. אבל אם הזליגה היא כאוטית אי אפשר ללמוד את זה. כמובן, ככל שהמערכת גמישה יותר היא יותר רגישה להתקפות. אם השרת מקבל הרבה מאוד סיסמאות שגויות מאיזשהו משתמש זה כנראה מצביע על בעיה כלשהי. אז זה עולה בזיכרון אבל אפשר לעשות מעקב אחרי כל הכרטיסים ובכך להימנע לפחות מחלק מההתקפות.

# פרוטוקולים של אימות

לכל אחד מהפרוטוקולים שנדון בו יש מאפיינים יחודיים משלו ובתכנון מערכת צריך לחשוב מה הפרוטוקול המתאים ביותר לצרכים ולאילוצים הקיימים.

## פרוטוקולים שמבוססים על סיסמאות




פרוטוקולים כאלה נמצאים בשימוש כל יום – דואר אלקטרוני, כניסה לחשבון הבנק ועוד.

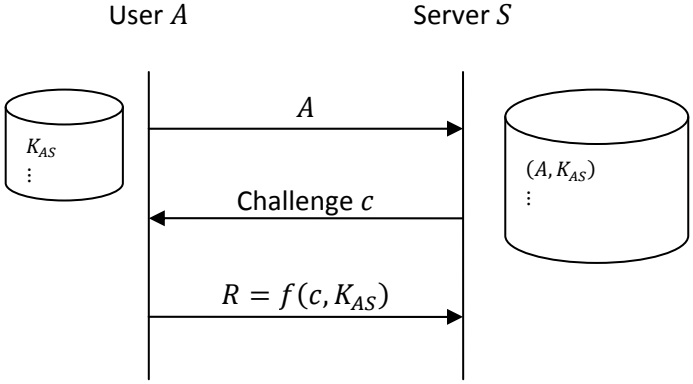
המאפיינים של הפרוטוקול:

- יתרונות:
  - פשוט וזול
  - אין צורך בחומרה בצד של המשתמש
  - נוח לשימוש ולשינוי
- חסרונות:
  - מאזין יכול לגנוב סיסמה
  - ניתן לנחש סיסמה
  - יש מידע רגיש בשרת

אם הערוץ מאובטח (כמו למשל ב-SSL) אין שום בעיה להשתמש בפרוטוקול הזה. אפשר גם להשתמש בסיסמאות חזקות. זה מקשה את הניחוש אבל לא פותר את בעיית ההאזנה אם הערוץ לא מאובטח. סיסמאות חזקות הן גם פחות נוחות למשתמש.

אם השרת מוגן אז זה גם לא מטריד כל כך שיש עליו מידע רגיש.

## פרוטוקולים שמבוססים על מפתחות סימטריים



לכל משתמש יש רשימה של מפתחות עבור כל אחד מהשרתים שהוא מתחבר אליהם ולכל שרת יש רשימה של המפתחות של המשתמשים שלו.

בעת כניסה של משתמש לשרת השרת שולח למשתמש אתגר. זה גורם למעשה למפתח שלו להיות חד-פעמי ומונע התקפת replay. המשתמש בתגובה שולח מענה שתלוי הן באתגר והן במפתח הסימטרי שלו עם השרת. המענה צריך להיראות אקראי.

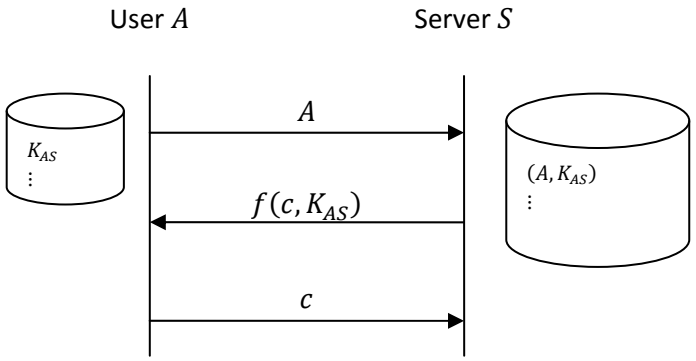
האתגר יכול להיות אחד האתגרים שדיברנו עליהם קודם – מספר אקראי, זמן או מונה. באופן כללי מדובר ב-**nonce** – משהו שלא חוזר על עצמו.

הפונקציה $f$ יכולה להיות למשל פונקצית ערבול, פונקצית MAC או פונקצית הצפנה.

המאפיינים של הפרוטוקול הזה:

- יתרונות:
  - אין בעיה של האזנה והתקפת replay כי האתגר אינו חוזר על עצמו
  - אין בעיה של חוזק מפתח כי הוא ארוך ואקראי
- חסרונות:
  - יש צורך בחומרה מיוחדת שיכולה לחשב את הפעולות הדרושות אצל המשתמש
  - אצל הלקוח יש מידע רגיש
  - בשרת יש מידע עוד יותר רגיש (מפתחות של כל המשתמשים ולא רק של אחד מהם)
  - אם למשל $f$ היא פונקצית הצפנה אז ע"י התחזות לשרת ושליחת אתגרים למשתמש ניתן לעשות chosen plaintext attack

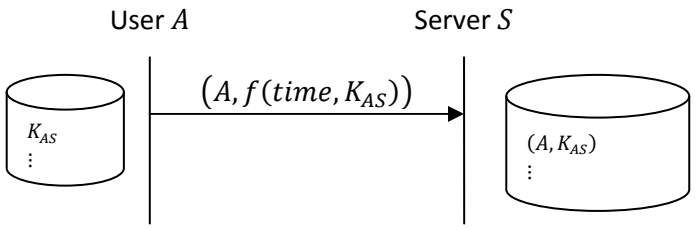
וריאציה של הפרוטוקול הזה היא שהשרת שולח למשתמש הצפנה של האתגר והמשתמש צריך לפענח אותו בעזרת המפתח הפרטי שלו.




בוריאציה הזאת יש בעיה אם האתגר הוא בר חיזוי כמו במקרה של זמן או מונה סידורי, כי אז אפילו בלי לדעת את המפתח אפשר פשוט לשלוח חזרה את התשובה הנכונה. לכן במקרה זה האתגר חייב להיות אקראי ולכן השרת חייב לזכור את האתגר שהוא השתמש בו.

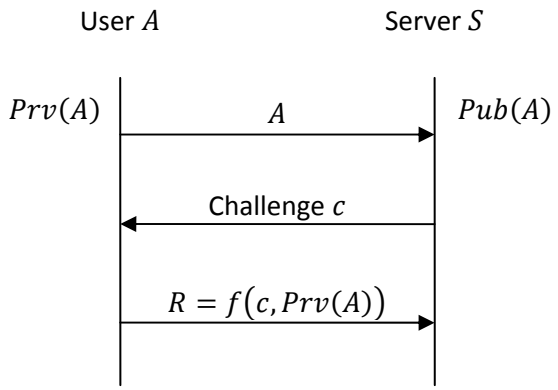
היתרון של הוריאציה הזאת הוא שאפשר להשיג בה מידה של זיהוי של השרת. אם באתגר יש איזושהי מבניות בנוסף למספר האקראי אז למתחזה יש סיכוי נמוך מאוד להמציא משהו אקראי שהפענוח שלו יהיה בעל המבניות הנכונה. אם המשתמש מפענח את ההודעה ומתקבל משהו בעל משמעות אז יש יותר סיכוי שהוא אכן מדבר עם השרת הנכון.

אם לשרת ולמשתמש יש שעון מסונכרן אפשר להשתמש בפרוטוקול אימות עם הודעה אחת בלבד:



כאן הזמן הוא מעין אתגר עצמי. אלא ששוב צריך לקחת בחשבון את הזמן התקשורת ובחלון הזמן הפתוח של השרת הוא יכול להיות חשוף להתקפת replay.

## פרוטוקולים שמבוססים על מפתחות אסימטריים




הפונקציה $f$ יכולה להיות או חתימה או פענוח. בעזרת המפתח הציבורי של $A$ השרת יכול לוודא את החתימה של המשתמש או להצפין את ההודעה כדי לבדוק שהתקבל האתגר הנכון.

כעת, מתקיף יכול לסחוט מהמשתמש חתימה או פענוח של מסמכים סודיים. הוא פשוט יתחזה לשרת וישלח למשתמש מסמך שהוא רוצה שהוא יחתום עליו או הודעה שהוא רוצה לפענח. לכן לכל משתמש חייבים להיות מפתחות שונים להצפנה, לחתימה ולאימות!

בניגוד לפרוטוקול הקודם כאן אין למשתמש שום דרך לדעת שהוא מדבר עם השרת הנכון. המשתמש יכול להוכיח את זהותו מול כל מי שפונה אליו. היתרון הוא שאפשר להוכיח זהות מול כל העולם והחיסרון הוא שהמשתמש לא יודע עם מי שהוא מדבר.

מאחר שהזיהוי נעשה הרבה פעמים כל פעם בעזרת אותו המפתח זה יכול להסגיר משהו על המפתח הפרטי. לכן בד"כ לא משתמשים בפרוטוקולים האסימטריים בצורה הפשטנית שתיארנו.

## פרוטוקולי אפס-ידיעה

כאן גם נעשה שימוש במפתחות אסימטריים אבל המטרה היא לפתור את הבעיה שאם מריצים את הפרוטוקול הרבה מאוד פעמים המידע עלול להסגיר משהו על המפתח הפרטי של המשתמש.

הדרישות מפרוטוקול אפס-ידיעה:

1. **שלמות** – המשתמש תמיד יוכל להוכיח את זהותו
2. **יציבת** – מתחזה ייכשל בהסתברות גדולה מאוד
3. **אפס-ידיעה** (**zero-knowledge** או ZK) – ריצות מרובות לא יסגירו מאומה על המפתח הפרטי של המשתמש

ניתן דוגמה לפרוטוקול כזה שמבוסס על בעיית הלוג הדיסקרטי. הנחת העבודה היא שידוע $p$ ראשוני גדול וידוע גם יוצר $\mathbb{Z}_p^* = \langle g \rangle$. המפתח הפרטי של משתמש $A$ הוא $x \in \mathbb{Z}_p^*$ אקראי והמפתח הציבורי הוא $y = g^x \bmod p$.

האימות של $A$ מול $B$ נעשה באופן הבא:

1. $t$ פעמים:
   1.1. **Commit**: $A$ בוחר $k \in \mathbb{Z}_p^*$ אקראי ושולח ל-$B$ את $s = g^k \bmod p$
   1.2. **Challenge**: $B$ שולח $c \in \{0,1\}$ אקראי

1.3. **Response**: $A$ שולח את $r = (k + cx)\ mod\ (p-1)$

1.4. **Verify**: $B$ בודק ש-$g^r = (s \cdot y^c)\ mod\ p$

נוכיח שהפרוטוקול מקיים את התכונות הדרושות:

1. שלמות: צריך להראות שהמשתמש האמיתי יוכל לעבור את כל השלבים של הבדיקה בהצלחה. בוודאי אין לו בעיה לבצע את ה-commit וה-response. נראה רק שהבדיקה של $B$ תניב תוצאה חיובית: נניח ש-$k + cx = q(p-1) + r$. אז ב-$\mathbb{Z}_p^*$ מתקיים:
$$g^r = g^{k+cx-q(p-1)} = g^k \cdot g^{cx} \cdot (g^{p-1})^{-q} = g^k \cdot (g^x)^c \cdot 1 = s \cdot y^c$$
2. יציבות: המתחזה לא יודע את $x$. את שלב ה-commit אין לו בעיה לבצע כי אין צורך ב-$x$. רק ב-response הוא צריך בהמת להוכיח שהוא יודע משהו ועד אז הוא יכול לעשות מה שהוא רוצה. יש שתי אפשרויות:

א. המתקיף מצפה שהאתגר יהיה $c = 0$. אז הוא יבחר $k$ כלשהו ואם באמת יגיע אתגר $c = 0$ המענה שלו יהיה $r = k$ וזה גם המענה הנכון. אבל זה קורה רק בהסתברות $\frac{1}{2}$. אם האתגר האמיתי יהיה $c = 1$ רק $r = k + x$ יכול לעמוד במבחן אבל את $k$ המתקיף כבר בחר ואת $x$ הוא לא יודע. לכן הוא יכול לנחש אותו רק בהסתברות $\frac{1}{p-1}$. במקרה זה הסיכוי של המתקיף להצליח הוא
$$\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot \frac{1}{p-1} = \frac{1}{2} \cdot \frac{p}{p-1}$$

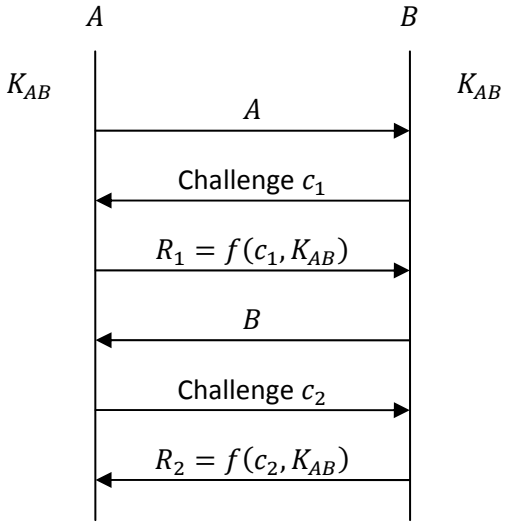
ב. המתקיף מצפה שהאתגר יהיה $c = 1$. אז הוא יבחר $k$ כלשהו וישלח את $s = \frac{g^k}{y}$. אז אם האתגר אכן יהיה $c = 1$ הבדיקה תעבור בהצלחה כי הוא ישלח את $r = k$:
$$s \cdot y = \frac{g^k}{y} \cdot y = g^k = g^r$$
אבל אם האתגר יהיה $c = 0$ אז הערך היחיד שיעמוד בבדיקה הוא $r = k - x$ והמתקיף שוב נמצא בבעיה כי הוא יכול רק לנחש את $x$ בהסתברות $\frac{1}{p-1}$. ושוב כמו במקרה הקודם ההסתברות של המתקיף להצליח היא $\frac{1}{2} \cdot \frac{p}{p-1}$.

אז אנחנו רואים שזה לא משנה מה המתקי; חושב שיקרה, בכל אופן הסיכוי שלו להצליח הוא $\frac{1}{2} \cdot \frac{p}{p-1}$. אז הסיכוי שהוא יעבור $t$ מבחנים הוא $\left(\frac{1}{2} \cdot \frac{p}{p-1}\right)^t$. $p$ אמנם גדול אבל הוא נתון מראש ולכן ההסתברות הזאת שואפת לאפס.

3. אפס-ידיעה: החשש הוא שאחרי ריבוי שיחות אפשר לעשות איזה ניתוח ולמצוא את $x$. נניח שמאזין רואה הרבה שלשות מהצורה $(s, c, r)$ כאשר $s, r \in \mathbb{Z}_p^*$ אקראיים, $c \in \{0,1\}$ אקראי ו-$g^r = (s \cdot y^c)\ mod\ p$. בשביל לייצר שלשות כאלה בכלל לא צריך לדעת את $x$ כי אפשר פשוט לבחור $r$ ו-$c$ אקראיים כנ"ל ולחשב את $s = \frac{g^r}{y^c}\ mod\ p$ שיוצא אקראי. זה אומר שראיית שלשות כמו למעלה לא תורמת לגילוי $x$ וזה בדיוק אפס-ידיעה.

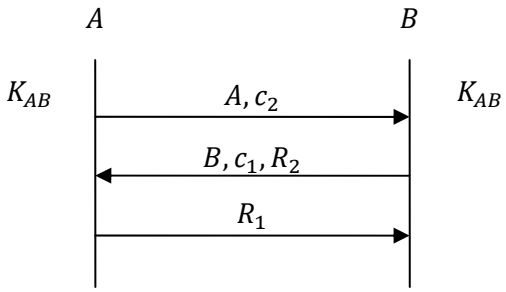
## אימות דו-צדדי

עד כה דיברנו על פרוטוקולים שבהם משתמש מוכיח את זהותו בפני שרת. אבל אם שתי ישויות מתקשרות הן יכולות בקלות להוכיח את זהותן אחת לשנייה:

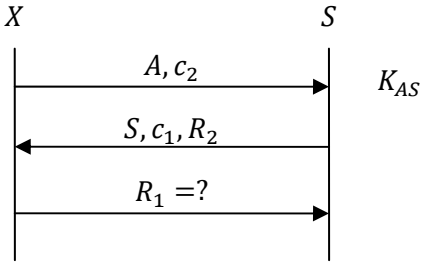



זאת פשוט הרחבה של פרוטוקול האימות שמבוסס כל מפתחות סימטריים שראינו קודם. ראשית $A$ מוכיח את זהותו בפני $B$ ואח"כ $B$ נוכיח את זהותו בפני $A$. בד"כ הצד הרגיש יותר ידרוש שהצד השני יאמת את עצמו קודם. בד"כ אם $B$ הוא שרת הוא בטח ירצה ש-$A$ יאמת את עצמו ראשון. אבל יש פעמים שבהם דווקא הלקוח הוא הרגיש יותר. למשל אם $B$ הוא הבנק הלקוח ירצה להיות בטוח שהוא מדבר עם הבנק לפני שהוא ממשיך הלאה.

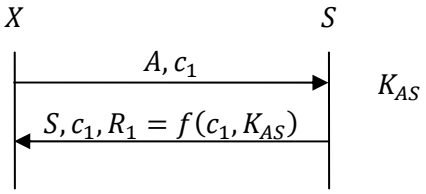
לכאורה ניתן לצמצם את הפרוטוקול למעלה באופן הבא:




אז חסכנו חצי מכמות ההודעות, אבל שני הפרוטוקולים לא שקולים. בפרוטוקול השני צד $B$ מוכיח את זהותו לפני צד $A$. את הפרוטוקול הזה אפשר לתקוף ב-**Reflection Attack**. נניח שמתקיף $X$ מתקיף את השרת $S$. ראשית הוא יפתח בשיחה כך:



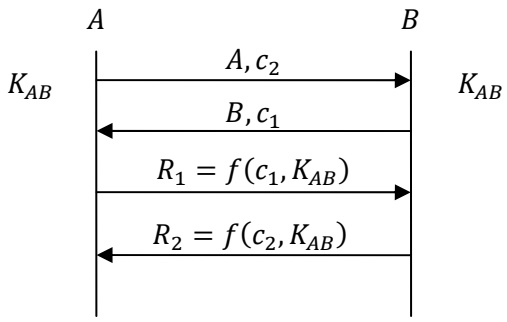

המתקיף לא יודע את $K_{AS}$ ולא לא יכול לחשב את $R_1 = f(c_1, K_{AS})$ ולתת מענה לשרת. במקום זה הוא יפתח בשיחה נוספת:



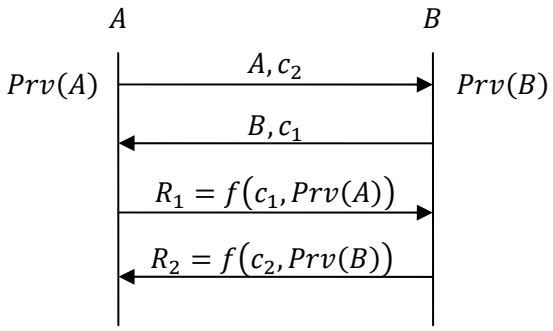
אבל עכשיו השרת בדיוק הסגיר בפני המתקיף את התשובה שהוא היה נחוץ לה בשיחה הקודמת. בשלב הזה המתקיף יכול לסגור את השיחה השנייה, לענות לשרת במענה $R_1$ הנכון ולהמשיך לעשות את זממו.

הסיבה שהדגמנו את ההתקפה הזאת עם שרת היא שבד"כ לקוחות יכולים לפנות לשרת כמה פעמים ולנהל איתו כמה שיחות בו-זמנית. זה משהו שלא קורה בד"כ בין שני אנשים.

בכל זאת יש הרגשה שיש בפרוטוקול המקורי איזו יתירות. אכן אפשר לקצר אותו לארבע הודעות באופן הבא:




לפרוטוקול יש גם גרסה אסימטרית דומה:




הבעיה עם הגרסה האסימטרית היא בהפצת המפתחות. לא תמיד כולם יודעים את המפתחות הציבוריים של כל העולם.

## חוקים לבניית פרוטוקולים טובים

1. שימוש במפתחות שונים לפעולות שונות
2. שימוש בפונקציות שונות לפעולות שונות
3. תוספת תגים ומבניות להודעות
4. הוספת nonce-ים להודעות
5. שימוש רק בפונקציות אמינות (לא להמציא פונקציות מפוקפקות משלנו!)
6. שימוש במפתחות או במידע רגיש באופן מינימאלי
   - זמן – לשלוף מפתח מהמקום המוגן לפרק זמן מינימלי
   - מרחב – לא להשאיר את המפתח במקום לא מוגן כמו זיכרון
7. להשתמש במספרים אקראיים טובים (זה בד"כ כרוך בחומרה ייעודית)

# הפצת מפתחות סימטריים וניהולם

בגדול יש שני סוגי מפתחות: מפתחות קבועים ומפתחות זמניים. המפתחות הקבועים אלה מפתחות שמשתנים בתדירות יחסית נמוכה והם משמשים למשל לאימות או לבניית מפתחות שיחה. המפתחות הזמניים משתנים בתדירות גבוהה ונמחקים בסוף השימוש. למשל, מפתחות שיחה הם זמניים.

העברת המידע מתבצעת בד"כ בעזרת מפתח סימטרי ואותו צריך לתאם. כמובן, יש טעם לתאם מפתחות רק אם בוצע קודם לפחות אימות חד-צדדי.

בעיקרון אם אליס ובוב רוצים שיהיה להם מפתח מתואם יש ארבע אפשרויות:

1. התקנה פיסית של $K_{AB}$ אצל שניהם
2. העברה באופן מאובטח של מפתח $K_{AB}$. ההעברה יכולה להיות פיסית או אלקטרונית. בוודאי העברה פיסית בטוחה יותר אבל היא גם יקרה יותר ונוחה פחות.
3. גוף שלישי אמין (trusted third party) מתקין את $K_{AB}$ פיסית במחשבים של אליס ושל בוב
4. גוף שלישי אמין מעביר (פיסית או אלקטרונית) באופן מאובטח מפתח $K_{AB}$ אל אליס ואל בוב

בד"כ השלב הראשון של בניית מערכת נעשה פיסית כי כפי שנראה בהמשך העברה אלקטרונית כרוכה בכך שכבר יש לאליס ולבוב איזה מידע משותף. אז השלב הראשון הוא יקר יותר אבל לאחר מכן אפשר כבר להשתמש בהעברה אלקטרונית.

## החלפת מפתחות סימטריים בעולם סימטרי

נניח שאליס ובוב רוצים לתאם מפתח והם בכלל לא מעוניינים להשתמש בהצפנה סימטרית. נסמן ב-$K_{AB}[i]$ מפתח בין אליס לבוב בזמן הדיסקרטי $i$. נניח ש-$K_{AB}[0]$ קיים גם אצל אליס וגם אצל בוב. אז יש כמה סכמות שאפשר לפעול לפיהן:

1. אחד הצדדים יעביר את $K_{AB}[i]$ באמצעות $K_{AB}[i-1]$ אל הצד השני. אלא אם מצפינים את אותה הודעה כמה פעמים או שמשתמשים באותו מפתח המון זמן כנראה שאין כאן פרצות אבטחה מבחינת קריפטוגרפיה. הבעיה היא שאם איכשהו מתגלה $K_{AB}[i]$ אז אפשר לפענח כל הודעה שתגיע בעתיד (אבל עדיין אי אפשר לפענח את ההודעות הקודמות).
2. שני הצדדים יחשבו את $K_{AB}[i] = h(K_{AB}[i-1])$ כאשר $h$ פונקצית ערבול טובה. אם הפונקציה באמת טובה אז שוב מבחינת הקריפטוגרפיה אין פה סיכן גבוה. אבל כמו קודם אם מתגלה $K_{AB}[i]$ אז למעשה ידועים גם כל המתפחות העתידיים (אבל לא המפתחות הקודמים כי ההנחה היא שקשה למצוא מקור של פונקצית ערבול). בכל זאת היתרון של הסכמה הזאת על פני הסכמה הקודמת הוא שהיא לא דורשת תקשורת בין הצדדים. אז למשל אליס ובוב יכולים לדעת שכל שבוע הם צריכים להחליף מפתח והם בכלל לא צריכים לדבר אחד עם השני בשביל זה. אז המפתח לא עובר בתקשורת והוא מוגן יותר.
3. פעם בכמה זמן מתקינים שני הצדדים חומרה מיוחדת עם מפתחות. זה יקר אבל בטוח. כך גם ההודעות העתידיות וגם הודעות מהעבר חסויות כי אין קשר בין המפתחות והם גם לא עוברים בתקשורת. כמו כן, אם מישהו גונב מפתח אליס ובוב בטוח ידעו מזה כי ההתקנה היא פיסית ואז הם יכולים לאפס את המערכת.

## החלפת מפתחות סימטריים בעולם אסימטרי

אליס ובוב חיים בעולם אסימטרי אבל הם מעוניינים לתאם מפתח סימטרי כדי לדבר בצורה מוצפנת. יש כמה פתרונות.

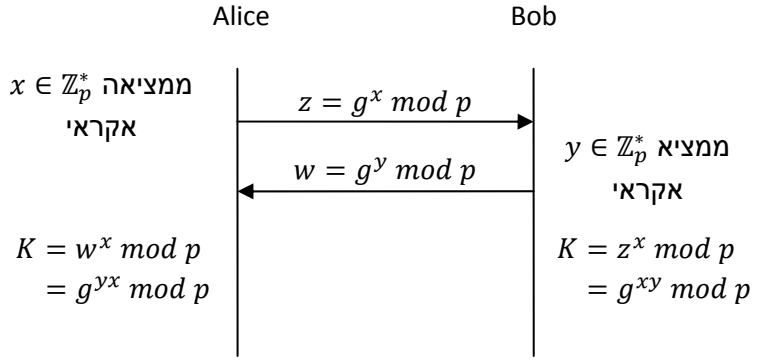
1. אליס בוחרת מפתח $K$ אקראי ושולחת אותו מוצפן לבוב באמצעות המפתח הציבורי שלו. בתרחיש הזה בוב אל יודע שהוא מקבל את המפתח מאליס, אבל אם יש לו דרך לדעת שאכן מדובר באליס זה בסדר.
2. אליס בוחרת מפתח $K$ אקראי ושולחת אותו מוצפן באמצעות המפתח הציבורי של בוב וחתום ע"י המפתח הפרטי שלה. כדי לקצר קצת את תהליך החתימה בד"כ חותמים על פונקצית ערבול של המפתח.

הבעיה עם הסכמה הזאת היא שלא תמיד לאליס יש בכלל מפתח פרטי. למשל אם בוב הוא בנק ואליס היא לקוחה אז סביר שאין לה מפתח פרטי. למעשה לקוחות בד"כ מוכיחים את זהותם מול שרתים בעזרת סיסמה.

בשתי הסכמות אם המפתח הפרטי של בוב מתגלה והוא לא יודע מזה אז גם ההודעות מהעבר וגם ההודעות מהעתיד לא סודיות יותר, כי המתקיף יכול לפענח את מפתחות השיחה הסימטריים שהיו ושיהיו ולפענח באמצעותם את כל התקשורת. אם בוב יודע שגנבו לו את המפתח הפרטי ומשנה אותו עדיין ההודעות מהעבר ניתנות לפענוח אבל ההודעות החדשות יישארו סודיות עד שגם המפתח הפרטי החדש יתגלה.

## פרוטוקול תיאום המפתחות המאומת של Diffie ו-Hellman

הנחת העבודה היא שיש ראשוני $p$ גדול וידוע ש-$\mathbb{Z}_p^* = \langle g \rangle$. כבר ראינו קודם את פרוטוקול תיאום המפתחות DH:



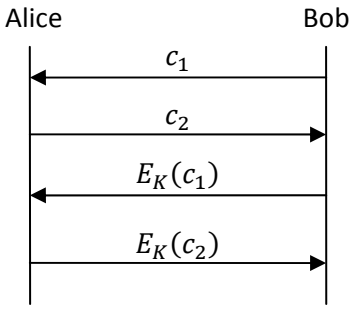

בפרוטוקול הזה גם העבר וגם העתיד חסויים כי בכל שיחה אליס ובוב ממציאים מפתח חדש. אבל אמרנו שבפרוטוקול הזה אין שום אלמנט של אימות ולכן הוא חשוף להתקפת man-in-the-middle.

כדי להוסיף אלמנט של אימות לפרוטוקול החלקים $g^x$ ו-$g^y$ צריכים להיות חתומים ע"י $Prv(A)$ וע"י $Prv(B)$ בהתאמה. הרי בלאו הכי ההודעות נשלחות בצורה גלויה לא מוסתרות. ואם אליס ובוב גם יאמתו את ההודעות שלהם אז למעשה בבת אחת הם גם תיאמו מפתחות וגם אימתו את זהותם. החיסרון הוא שמאחרת שמעורבות פעולות קריפטוגרפיות זה הופך את הפרוטוקול ליקר יותר.

## העברת סיסמאות

כפי שכבר אמרנו הרבה פעמים אימות נעשה בעזרת סיסמה אבל אז צריכה להיות דרך להעביר סיסמאות בצורה מאובטחת. יש שתי אפשרויות:

1. העברת הסיסמה מאליס לבוב על גבי ערוץ מוצפן. אבל אם מישהו מגלה את המפתח הפרטי של בוב אז הוא יכול גם לפענח את הסיסמה של אליס.
2. פרוטוקול סיסמאות חזק. למשל, אם אליס ובוב שניהם יודעים את $w = h(PW_A)$ אליס יכולה לשלוח לבוב את $E_w(g^x \bmod p)$ כאשר $x \in \mathbb{Z}_p^*$ אקראי ובוב ישלח לאליס את $E_w(g^y \bmod p)$ כאשר $y \in \mathbb{Z}_p^*$ אקראי. עכשיו אליס ובוב יכולים לחשב את המפתח $K = g^{xy} \bmod p$. בגלל שמצפינים כאן דברים אקראיים אפילו אם המתקיף ינסה את כל הסיסמאות בעולם הוא לא יוכל לדעת אם הוא ניחש נכון או לא. עכשיו אם אליס ובוב יוכיחו ששניהם יודעים את המפתח הנכון אז בלי שאליס הסגירה דבר על הסיסמה שלה בוב יודע שהוא אכן מדבר עם אליס. בדיקת המפתח היא פשוטה:



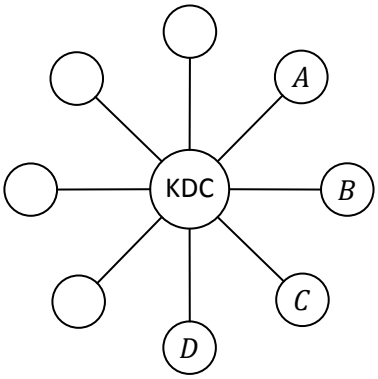

אם ההצפנות משני הצדדים נכונות סימן שהמפתח נכון.

## ניהול מפתחות סימטריים

עד עכשיו דיברנו רק על שתי ישויות אליס ובוב. אבל למעשה אנחנו חיים בעולם שיש בו הרבה ישויות וכולן רוצים לדבר עם כל האחרות. אם יש $n$ ישויות ולכל שתיים מהן $A$ ו-$B$ יש מפתח $K_{AB}$ אז יש סה"כ $n^2 - n$ מפתחות תשתיתיים בין תחנות. מפתחות אלה ישמשו את התחנות לתיאום מפתחות שיחה זמניים. את כל המפתחות האלה צריך לנהל (הוספות, שינויים, הורדות וכן הלאה). אם $n$ קטן אז אפשר לעבוד עם $n^2 - n$ מפתחות, אבל ככל ש-$n$ גדל זה נהיה יותר ויותר קשה. לכן במערכות גדולות משתמשים בד"כ בגוף שלישי אמין שינהל את המפתחות. מצד שני, היתרון הוא שזו מערכת מבוזרת שאין בה נקודת כשל יחידה. אם משתלטים על $A$ רק התקשורת של $A$ נפגעת ואין שום השפעה על התקשורת שבין $B$ ל-$C$.

## Key Distribution Center

ה-**Key Distribution Center** (בקיצור KDC) הוא סוג של גוף שלישי אמין (להלן TTP) שאחראי על ניהול מפתחות בד"כ בעולם סימטרי. הרעיון הוא שכל התחנות מקושרות ל-KDC וכל התקשורת נעשית דרכו.



לכל תחנה $A$ יש מפתח $K_A$ שנשמר ב-KDC. אם מפתח של תחנה כלשהי משתנה רק צריך לשנות אותו ב-KDC ואם מוסיפים תחנה חדשה אז צריך להוסיף אותה רק ברשימה של ה-KDC. גם הורדת תחנה היא פשוטה. אז הניהול במצב הזה הוא כמובן פשוט. החיסרון הוא שה-KDC הופך לצוואר הבקבוק והוא מהווה נקודת כשל יחידה. הוא צריך להיות זמין כל הזמן ולטפל בהמון בקשות. בשביל להקל את העומס הרבה פעמים יש כמה KDC-ים ואז הם לא מהווים צוואר בקבוק בתקשורת. גם האבטחה שלהם הרבה יותר גבוהה מהתחנות הרגילות כי הם מכילים המון מידע רגיש ומהווים גורם קריטי במערכת. מבחינת מחיר יש איזשהו איזון. אמנם ההגנה על ה-KDC היא יקרה מאוד אבל ההגנה על כל התחנות היא זולה יותר מקודם כי עכשיו הן לא מכילות שום מידע רגיש שקשור למפתחות.

נניח שתחנות $A$ ו-$B$ מחוברות ל-KDC, אין ביניהן נתיב תקשורת מאובטח אבל הן בכל זאת רוצות לדבר. הדרך היחידה שלהן לעשות זאת היא דרך ה-KDC:

1. $A$ פונה ל-KDC בבקשה לתאם שיחה מאובטחת עם $B$.
2. KDC נענה לבקשה של $A$ ושולח לו באופן מאובטח בעזרת המפתח $K_A$ מפתח שיחה $K_{AB}$ עבור השיחה העתידית עם $B$.
3. KDC מעביר ל-$B$ את מפתח השיחה $K_{AB}$ באופן מאובטח באמצעות $K_B$.
4. כעת $A$ ו-$B$ יכולים לשוחח באופן מאובטח בעזרת $K_{AB}$. בד"כ ההנחה היא שה-KDC הוא אמין ואפשר לסמוך עליו, אבל אם רמת הפרנויה שלהם גבוהה במיוחד הם יכולים למשל ליצור מפתח שיחה חדש בעזרת $K_{AB}$.

יש שלושה תרחישים כללים שבהם שתי תחנות רוצות לדבר אחת עם השנייה בעזרת ה-KDC.

### תרחיש Intranet

1. $A$ מבקש שירות מ-KDC
2. KDC אולי בודק את זהותו של $A$ ומחזיר לו מפתח $K_{AB}$ מוצפן ע"י $K_A$
3. KDC שולח ל-$B$ את $K_{AB}$ מוצפן בעזרת $K_B$
4. $A$ או $B$ יוזמים פנייה להתחלת שיחה מאובטחת בעזרת $K_{AB}$

לפרוטוקול הזה יש כמה בעיות:

- צריך סנכרון בין $A$ ל-$B$ כי הם לא יודעים באופן טבעי מתי הם יכולים לדבר ומתי לא.
- אם $B$ לא זמין זאת בעיה עבור ה-KDC כי התפקיד היחיד שלו הוא להפיץ מפתחות ולא לחכות לתחנות שיהפכו לזמינות. ב-intranet הנחת העבודה היא שכל התחנות זמינות תמיד ולכן אפשר להשתמש בפרוטוקול הזה.
- מסגרת הפרוטוקול לא מתאימה לסביבות שרת-לקוח. למשל, אם $B$ מדפסת אז $A$ לא ירצה לחכות עד שאפשר יהיה לדבר עם $B$.

### תרחיש Internet

1. $A$ מבקש שירות מ-KDC
2. KDC אולי בודק את זהותו של $A$ ומחזיר לו מפתח $K_{AB}$ מוצפן ע"י $K_A$
3. KDC שולח ל-$A$ כרטיס $TKT_B$ עבור $B$
4. $A$ פונה ל-$B$ עם $TKT_B$ להקמת שיחה בעזרת $K_{AB}$

$TKT_B$ הוא כמו כרטיס כניסה ל-$B$. ה-KDC נותן ל-$A$ את הכרטיס וכש-$A$ יחליט שהוא רוצה הוא יכול להשתמש בו. היתרון הוא ש-KDC נותן שירות ומיד יוצא מהתמונה. מאחר ש-$A$ פנה אליו הוא כנראה זמין ואין בעיות בתקשורת איתו ואילו הזמינות של $B$ בכלל לא משפיעה על KDC. זה מצב קלאסי של שרת-לקוח. הסנכרון בין $A$ ל-$B$ מובנה בתקשורת של $A$ ל-$B$. $B$ מתעורר בדיוק כשצריך אותו.

הכרטיס $TKT_B$ יכול להיות למשל $K_{AB}$ מוצפן ע"י $K_B$. מטרת הכרטיס היא להראות ל-$B$ ש-$A$ קיבל אישור לדבר איתו.

### תרחיש Extranet

1. $A$ פונה ל-$B$ בבקשת שירות
2. $B$ פונה ל-KDC עם כל הפרטים על הפנייה של $A$ אליו
3. KDC פונה ל-$B$ עם מפתח $K_{AB}$ מוצפן ע"י $K_B$
4. KDC שולח ל-$B$ כרטיס $TKT_A$ עבור $A$
5. $B$ שולח ל-$A$ את $TKT_A$
6. $A$ ו-$B$ מקימים שיחה

זה מתאים לתרחישים שבהם מאיזושהי סיבה אנחנו לא רוצים שלקוחות ידברו עם KDC. למשל אם מדובר בבנק ו-$A$ לקוח שרוצה למשוך כסף. אז $B$ הוא למעשה הכספומט – שרת ביניים שמגשר בין הלקוח לבנק עצמו. הלקוח מושהה עד ש-$B$ מסיים לדבר עם KDC, וזה אכן מה שקורה כאשר אנחנו מושכים כסף מהכספומט.

### תרחיש טיפוסי

1. $A$ פונה ל-KDC ושולח לו $(A, B, N_1)$ כאשר $N_1$ הוא nonce
2. KDC עונה ל-$A$ עם $\left(A, B, E_{K_A}(A, B, K_{AB}, N_1)\right)$

כאן מתחילה הוכחת הזהות של ה-KDC כי $A$ יכול לפענח את המסר המוצפן ולבדוק את $N_1$. מצד שני, אם דווקא $A$ הוא המתחזה אז הוא לא יכול לפענח את $K_{AB}$. ה-KDC אמנם לא יידע את זה, אבל המתקיף לא יוכל להמשיך לשלבים הבאים. החשש היחיד מהבחינה הזו היא שמתחזים מעמיסים עם הרשת.

3. KDC שולח ל-$A$ את $TKT_B = E_{K_B}(A, B, K_{AB})$
זה מוצפן ע"י המפתח של $B$ ולכן אף אחד חוץ ממנו לא יוכל לפענח את זה.
4. $A$ פונה ל-$B$ עם כל הפרטים $\left(A, B, TKT_B, N_2, E_{K_{AB}}(N_2)\right)$ כאשר $N_2$ הוא nonce

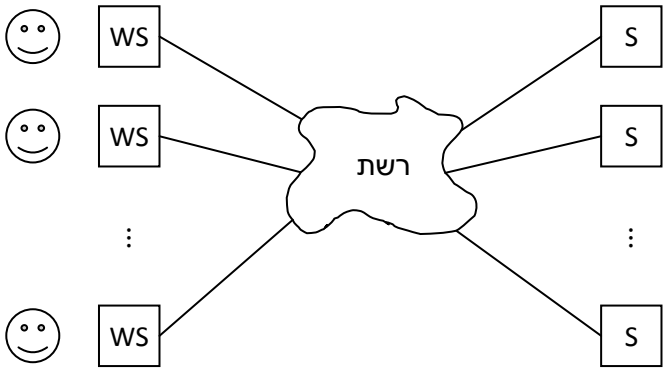
$N_2$ משמש לאימות ש-$A$ הוא אכן $A$. ב-$TKT_B$ כתוב המפתח האמיתי $K_{AB}$ ואם ההצפנה של $N_2$ נכונה אז כנראה ש-$A$ אינו מתחזה כי הוא באמת קיבל את המפתח מה-KDC. נשים לב רק שלא כדאי כאן ש-$N_2$ יהיה סתם מספר רנדומאלי כי אז אפשר לעשות replay attack.

5. $B$ מוכיח את זהותו בפני $A$ ע"י $\left(A, B, N_3, E_{K_{AB}}(N_3)\right)$ כאשר $N_3$ הוא nonce

# קרברוס

## המודל הכללי

בשנות ה-80 המוקדמות התחיל להיכנס לשימוש המודל של הרצה מרחוק. אנשים יכלו להגיע לתחנת עבודה ודרכה להריץ חישובים על שרת מרוחק:




באופן טבעי נשאלת השאלה אין יוצרים פה מודל של אימות לקוח-שרת. למשל אי אפשר להשתמש בסיסמה בצורה ישירה כי היא עוברת ברשת בצורה לא מאובטחת.

המטרה הייתה ליצור כמה מנגנונים:

- **אימות** (Authentication)
- **הרשאות** (Authorization)
- **בקרה** (Auditing)
- **Single sign-on** – ברגע שנכנסים למערכת מיד מקבלים את כל השירותים האפשריים (בניגוד לאינטרנט איפה שאנחנו צריכים להיכנס לכל שירות, כמו דואר אלקטרוני או אתר הבנק בנפרד)

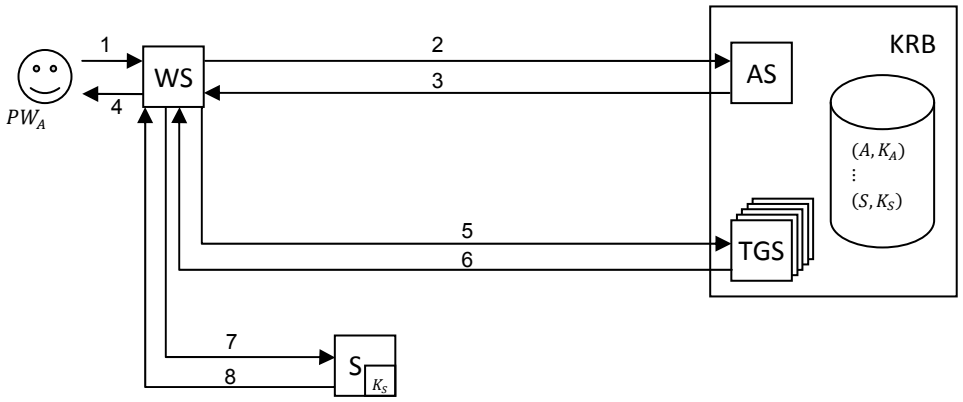
במערכת קרברוס המקורית מומש רק האימות. המודל כלל ארבעה עצמים: משתמש $A$, שרת קצה S, תחנת עבודה WS ומערכת קרברוס KRB.

למערכת קרברוס יש מפתח $K_{KRB}$ שהוא ממש חזק ופנימי למערכת. למשתמש $A$ יש סיסמה $PW_A$ ולשרת $S$ יש מפתח $K_S$. תחנת העבודה לא שומרת שום מידע של אבטחה. בוודאי תחנת העבודה גם לא מכירה את המשתמש כי זה לא הגיוני שכל תחנות העבודה יכירה את כל המשתמשים. זה גם מאוד מקשה את הניהול במקרה שצריך לעדכן משהו.

כל מה שיש בתחנת העבודה הוא תוכנה שיודעת לעבוד בשיטת קרברוס וכמובן גם בשרת צריכה להיות תוכנה שיודעת לעבוד בשיטה הזאת. תהליך הוספת תוכנה למערכת ההפעלה שיודעת לעבוד במודל קרברוס נקרא **קרבריזציה**. כיום המודל מובנה בכל מערכות ההפעלה.

כיום יש לקרברוס שתי גרסאות מסחריות.

## KRB V4




AS הוא ה-**Authentication Server** ותפקידו לתת לאנשים אפשרות להיכנס בכלל לרשת (שלבים 1-4). הוא עוד לא מקשר בין המשתמש לשרת הספציפי שהוא רוצה להיכנס אליו.

TGS הוא **Ticket Granting Server** ומטרתו לתת למשתמש כרטיס שמשמש אותו לכניסה לשרת הרלוונטי. יכולים להיות כמה TGS-ים כאלה כדי להקל על העומס. כדי להיכנס לשרת הרצוי מבצעים את שלבים 5-8.

נפרט את התהליך:

1. משתמש $A$ משתדך עם תחנת העבודה WS הוא מקליד את שמו לתחנה ומעתה אפשר למעשה לדבר על המשתמש ועל תחנת העבודה ביחד $A + WS$.
2. $A + WS$ פונים ל-AS עם ההודעה $(A, WS, TGS, TS_1)$ כאשר $TS_1$ היא חותמת זמן. אין פה עדיין שום אימות. הם רק מכריזים בפני ה-AS מי המשתמש, באיזו תחנה הוא עובד ובאיזה TGS הוא רוצה להשתמש.
3. AS עונה ל-$A + WS$ עם Network Credentials:

$$NC_{A+WS} = \mathrm{E}_{K_A}\left(K_{A,TGS}, TGS, TS_2, LT_2, TKT_{TGS}\right)$$

כאשר $LT_2$ הוא הזמן (lifetime) שאנחנו מרשים ל-$NC_{A+WS}$ להיות בתוקף ו-$TKT_{TGS}$ הוא

$$TKT_{TGS} = E_{K_{KRB}}\left(K_{A,TGS}, WS, TGS, TS_2, LT_2\right)$$

$K_{A,TGS}$ הוא מפתח שקרברוס המציא לשיחה הזאת והוא תקף לזמן $LT_2$.

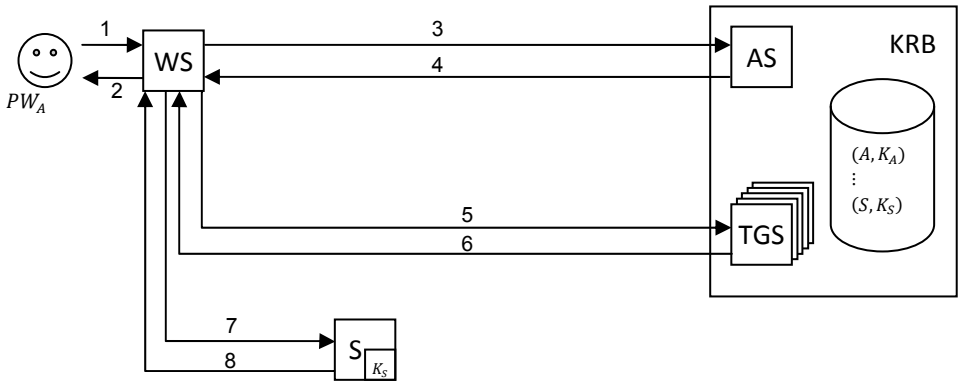
4. $A + WS$ מפענחים ושומרים את $NC_{A+WS}$:
   - $A$ מקליד את הסיסמה $PW_A$.
   - WS מחשב את $K_A = h(PW_A)$ ומוחק את $PW_A$ מהזיכרון (כדי שהסיסמה לא תהיה בסיכון, שהרי היא הרבה יותר רגישה מ-$K_A$).
   - WS מפענח את $NC_{A+WS}$ ושומר את השדות הרלוונטיים בצורה מוגנת ומוחק את $K_A$ מהזיכרון (אם אחרי הפענוח השדות הם בעלי משמעות ובמבנה הנכון סימן שהפענוח נכון והסיסמה שהוקלדה נכונה). המידע הרגיש היחיד שנותר במערכת בשלב זה הוא $K_{A,TGS}$ אבל הוא תקף רק לזמן מוגבל.
5. $A + WS$ פונים ל-TGS בבקשת שירות מול S בעזרת ההודעה $(S, TKT_{TGS}, auth_3)$ כאשר $auth_3 = E_{K_{A,TGS}}(A, WS, TS_3)$.
6. TGS עונה ל-$A + WS$ עם Service Credentials:

$$SC_{A+WS,S} = E_{K,TGS}\left(K_{A+WS,S}, S, TS_4, LT_4, TKT_S\right)$$

כאשר $TKT_S = E_{K_S}\left(K_{A+WS,S}, A, WS, TS_4, LT_4\right)$. $A + WS$ מפענחים את השדות הרלוונטיים.

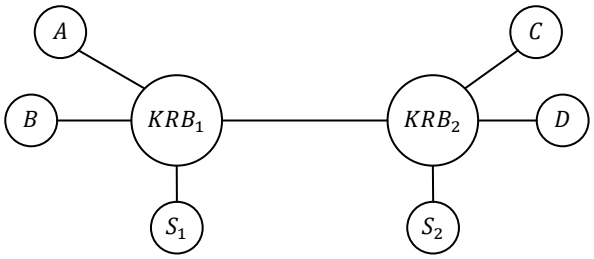
7. $A + WS$ פונים ל-S עם הכרטיס $(TKT_S, auths)$ כאשר $auths = E_{K_{A+WS,S}}(A, WS, TS_5)$
8. S מפענח את $TKT_S$ ועונה עם $auth_6 = E_{K_{A+WS,S}}(S, TS_6)$.

## KRB V5

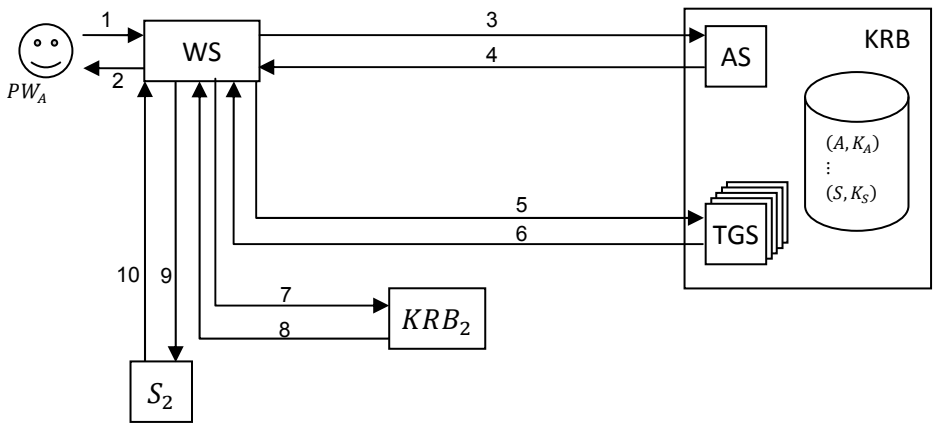



בגרסה החמישית של קרברוס השתנו כמה דברים:

- השתנה סדר ההודעות: אימות המשתמש מתבצע מוקדם – המשתמש מכניס שם וסיסמה ביחד ותחנת העבודה מיד מחשבת את $K_A$ ועושה תהליך אימות ראשוני (בעזרת $TS_1$) כדי לא להעמיס על AS עם מתחזים.
- לא מצפינים פעמיים את $TKT_{TGS}$. לפי הגדרתו הוא מוצפן ע"י $K_{KRB}$ וזה מפתח חזק. לכן אין צורך להצפין אותו שוב בעזרת $K_A$ ולבזבז זמן ומשאבים.
- Forwarding: יש אפשרות להעביר זכות שימוש מתחנה לתחנה וממשתמש למשתמש.
- Proxy: יש אפשרות לעבוד מול תחנה שמדברת בשם תחנה אחרת.
- Postdating: יש אפשרות לעבוד במשך הרבה זמן באותה הכניסה. הרעיון הוא לאפשר הנפקת כרטיס לתאריך עתידי או כרטיס שניתן לחדש.
- תמיכה ביקומים (Realms) שונים: קרברוס כפי שתארנו אותו עובד ברמה הארגונית. אבל לפעמים ארגונים שמריצים קרברוס שונים רוצים לתקשר אחד עם השני. אז לכל ארגון (יקום) יש שרת קרברוס משלו והם לקוחות אחד של השני:

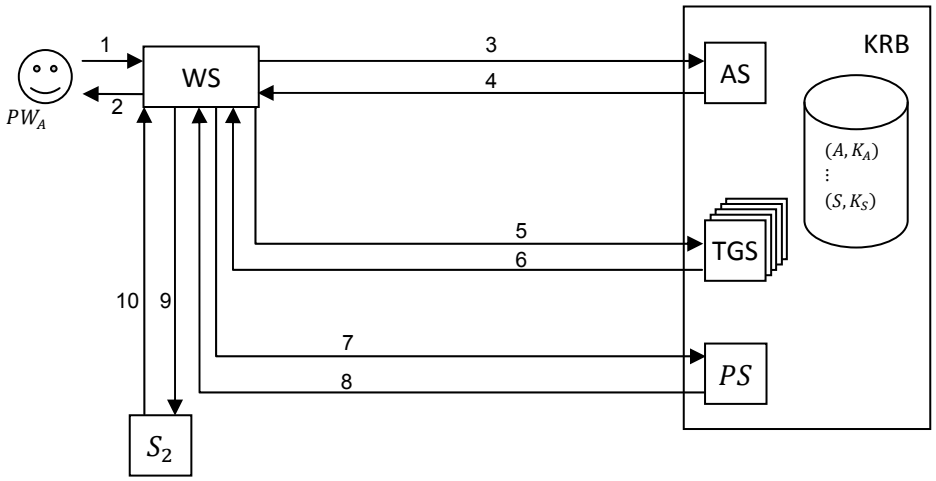



אם $A$ רוצה לגשת ל-$S_2$ הוא נכנס למערכת מול $KRB_1$ ובגישה ל-TGS הוא אומר שהוא רוצה לדבר עם $S_2$ שנמצא ביקום השני. אז $KRB_1$ פונה ל-TGS של $KRB_2$ ומבקש שירות. $KRB_1$ נותן כרטיס ל-$A$ לגשת ל-$KRB_2$ והוא בתורו נותן ל-$A$ כרטיס גישה ל-$S_2$. מבחינת $KRB_2$ $A$ הוא פשוט שרת שמנפיק כרטיסים לשרתים אחרים:



## Distributed Computing Environment

**Distributed Computing Environment** (בקיצור DCE) היא מערכת תוכנה שנותנת מסגרת לפיתוח אפליקציות לקוח/שרת. היא משתמשת מבוססת על קרברוס לצורך אימות ומרחיבה את יכולת ניהול ההרשאות. נוסף לשרתים של קרברוס יש לה שרת **Privilege Server** שנותן רשימת הרשאות לשרת שאליו המשתמש מבקש גישה.



# מערכות מפתחות ציבוריים

עד עכשיו תמיד הנחנו שאנחנו חיים בעולם שבו לכל אחד יש מפתח פרטי וכל שאר העולם יודע את כל המפתחות הציבוריים של כולם. בפרק זה נדון ביצירת העולם הזה.

## בניית מפתחות

יש כמה אפשרויות:

- כל ישות $A$ בונה זוג של מפתח פרטי ומפתח ציבורי ושומרת את המפתח הפרטי לעצמה במקום מאובטח ואילו את המפתח הציבורי מפיצה לעולם. האופציה הזאת מתאימה לעולם שבו $A$ סומך רק על עצמו.
- $A$ לא תמיד בעל יכולת חישובית להמציא לעצמו מפתח אז הוא יכול לפנות לאחד מהרבה שרתים שיודעים לבנות מפתחות. השרת נותן ל-$A$ את המפתח הפרטי ושוכח אותו.
- יש שרת TTP שרק הוא בונה מפתחות והוא נותן ל-$A$ את המפתח הפרטי שלו אבל לא בהכרח שוכח אותו (למשל לצרכי גיבוי). זה בסדר כי ההנחה היא שהשרת הזה אמין.

## הפצת מפתחות

יש כמה אפשרויות:

- $A$ או שרת מיוחד שנועד להפצה הם המפיצים.
- התקנה פיסית אצל המבקשים. זו שיטה בטוחה אבל קשה למימוש כי יכולים להיות הרבה מאוד מבקשים שפזורים בכל רחבי העולם.
- התקנה באתרי אינטרנט כמו דפי זהב וכו'.
- שליחת המפתח לכל דורש בתוספת לדוא"ל, קבצים וכו'. במקרה זה קשה להיות בטוחים שזה אכן המפתח הנכון כי אפשר לזייף מכתבים אבל אם אותו המפתח מתקבל מהרבה מקורות שונים זה מגדיל את האמון שלנו בו. אם מתישהו התגלה חוסר התאמה לא ידוע איזה מהמפתחות הוא המזויף, אבל ברור שיש בעיה.
- גישה לשרת TTP מיוחד וקבלת המפתח ממנו.

## ביטול מפתחות

יש מקרים שבהם יש צורך לבטל מפתח ציבורי. למשל:

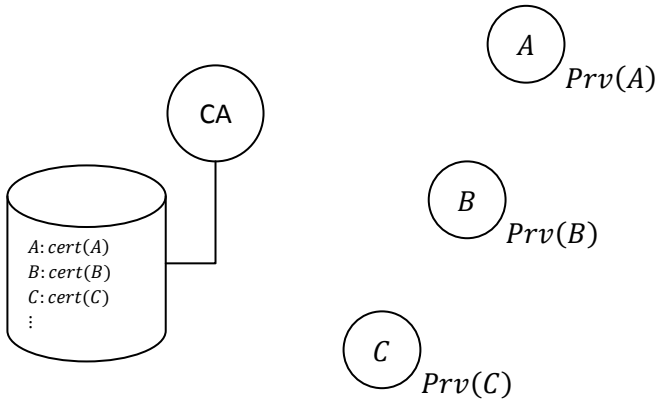
- אם המפתח הפרטי נגנב אין טעם יותר להשתמש במפתח הציבורי המתאים לו
- אם מישהו פיצח את המפתח הפרטי
- אם פג תוקפו של המפתח (למשל, אם המשתמש עזב את הארגון)

במקרה שצריך לבטל את המפתח של $A$ יש שתי אפשרויות. או ש-$A$ יכול להפיץ את האינפורמציה על כך או ששרת יכול להפיץ את זה. בכל מקרה אם משתמש אחר רוצה לשוחח עם $A$ הוא תמיד חייב לבדוק קודם את התקפות של $Pub(A)$.

# אשרות

בדיון של הפרק הקודם לא הצגנו דרך בטוחה לוודא נכונות של מפתח ציבורי. כדי לאמת מפתח ציבורי צריך להציג אשרה שהוא חוקי ואמיתי.

במודל שנציג יש **רשות הנפקת תעודות** (**Certificate Authority** או CA) שאנחנו כמובן מניחים שהיא אמינה והיא אחראית לייצר אשרות.




CA מנפיק את $cert(A) = \left(A, Pub(A)\right)$ חתום ע"י ה-CA בעזרת $Prv(CA)$. אבל עכשיו יש לנו בעיה מעגלית כי איכשהו כל העולם צריך להשיג את $Pub(CA)$ הנכון. אז כאן אין ברירה אלא להתקין את המפתח פיסית בעת ההצטרפות לארגון. ל-CA יש בד"כ מפתחות מאוד חזקים (2000-4000 ביט).

## בניית מפתחות

כמו קודם, יש כמה אפשרויות:

- $A$ בונה זוג מפתחות פרטי וציבורי.
- שרת בונה זוג מפתחות פרטי וציבורי.
- שרת שקשור ל-CA בונה זוג מפתחות פרטי וציבורי. הסיבה שה-CA לא עושה את זה בעצמו היא שאנחנו מנסים לצמצם את הפעילויות של שרתים רגישים. ה-CA אחראי רק על האשרות וזהו. למשל, כדי לקבל אשרה קודם כל שרת אחר יאמת את הזהות ורק אז תינתן גישה ל-CA שייתן את האשרה נחוצה.

## בניית אשרות

יש כל מיני סוגים של תעודות. המחיר והאיכות שלהן תלויים ברמת הגישה הפיסית בעת הנפקת התעודה. למשל, אפשר לקבל מ-VeriSign אשרה פשוט ע"י פנייה בדואר אלקטרוני או בטלפון. ואז כמובן רמת אימות הזהות לא גבוהה במיוחד וגם תעודה כזו לא עולה הרבה כסף. מצד שני לבנקים וחברות גדולות כמובן יש אשרות יותר רציניות. אבל גם לאשרות הפשוטות יש ערך. למשל, אפשר להשתמש בהן כאשר בונים מערכת ורוצים לבדוק אותה או בשביל עסקאות שלא כרוכות בהפסד גדול.

## פורמט X509

אשרות מסוג X509 מכילות את השדות הבאים:

1. **Version number**: הגרסה הנוכחית היא 3
2. **Serial number**: המספר הסידורי של האשרה ברשימות של המנפיק
3. **Signature parameters**: האלגוריתם של החתימה
4. **Certificate issuer**: מנפיק התעודה
5. **Not before**: הזמן שהחל ממנו האשרה בתוקף
6. **Not after**: הזמן שבו פג תוקף התעודה
7. **Subject details**: השם של הישות ואולי פרטים נוספים
8. **Subject public key**: המפתח הציבורי של הישות
9. **Extensions**: עוד פרטים כמו דואר אלקטרוני, אתר, שימוש של האשרה, איפה אפשר לבדוק אם היא בתוקף...
10. **Signature**: החתימה של המנפיק על שדות 1-9 בעזרת האלגוריתם משדה 3

## הפצת אשרות

כדי לא להעמיס על ה-CA לא הוא בד"כ זה שמפיץ את האשרות. בד"כ מדובר בשרת שמחובר ל-CA או $A$ עצמו. לפעמים גם יש שרת הפצה כללי.

## ביטול אשרות

יכולת ביטול אשרות נחוצה מאותן סיבות כמו ביטול מפתחות. יש שתי דרכים לבטל מפתח:

- $A$ עצמו יכול לפרסם הודעה בעיתונות. אבל זה לא טוב כי מישהו יכול להתחזות ל-$A$, אז $A$ צריך לחתום על ההודעה. כאן יש שתי אפשרויות: או ש-$A$ באמת התכוון להודעה או שגנבו לו את המפתח הפרטי וכך חתמו על ההודעה. בשני המקרים לא נרצה להשתמש במפתח שלו יותר.
- $A$ פונה ל-CA בבקשה לבטל את האשרה שלו וה-CA מפיץ את המידע חתום על ידו. ה-CA יכול גם ליזום ביטול של אשרה במקרה שהוא מגלה ש-$A$ רמאי או משהו כזה.

## הפצת שינויים וביטולים

קיימות שתי גישות. לפי הראשונה, נפרסם כל הזמן את רשימת האשרות המבוטלות (**Certificate Revocation List** או בקיצור CRL), ולפי השנייה, נפרסם כל הזמן את רשימת האשרות התקפות (**Certificate Activation List** או בקיצור CAL). הבחירה כמובן תלויה בגודל ובמורכבות. אבל בכל אופן ההפצה נעשית בזמנים דיסקרטיים.

הדיסקרטיות בזמני ההפצה מהווה בעיה כמובן. אם גנבו ל-$A$ את המפתח הוא ירצה שיבטלו את המפתח הציבורי שלו מיד אבל בפועל הוא יצטרך לחכות עד זמן ההפצה הבא. אם בזמן שבין גניבת המפתח להפצה הרשימה $B$ שולח ל-$A$ הודעה הגנב יוכל לפענח אותה.

אם רוצים להיות מאוד בטוחים אפשר להפיץ את השינויים כל שנייה, אבל זה בוודאי לא מציאותי...

## Complete CRL

באינטרנט בד"כ משתמשים ב-CRL. זוהי למעשה מעין רשימה שחורה של אשרות שה-CA מפיץ פעם בכמה זמן. זה מבנה נתונים חתום ושמור יחד עם התעודות. הרי מי שרוצה לראות אשרה של מישהו בוודאי ירצה לבדוק גם ב-CRL. הבעיה היא שזה יכול להיות קובץ ממש גדול ולכן יש מנגנונים שמקלים על ההפצה. נדון בהם בקרוב.

### CRL X509

פורט X509 של ה-CRL דומה מאוד לפורמט של האשרה. הוא כולל בתוכו שמונה שדות שרובם זהים לשדות של פורמט האשרה:

1. **Version**
2. **Signature parameters**
3. **Issuer**
4. **Next CRL Update**: כדי שמי שמשתמש ברשימה יידע אם הוא מסתכל על הרשימה הכי מעודכנת
5. **List of Revoked Certificates**: המספרים הסידוריים של התעודות שבוטלו והתאריך שבוטלו בו
6. **Extensions**
7. **Signature**

## Distribution Points

במקום להפיץ את הרשימה המלאה אפשר לחלק אותה לשרתים שונים ועל כל אחד לשמור רשימות קטנות יותר. זה גם מקל את העומס על השרתים וגם מקל את החיפוש של המשתמש. בכל אשרה כותבים בד"כ באיזה שרת אפשר למצוא את המידע על הביטול שלה. וכך מי שרוצה לוודא שאשרה עדיין בתוקף יודע באיזה שרת לחפש ושם הרשימה כבר קטנה יותר.

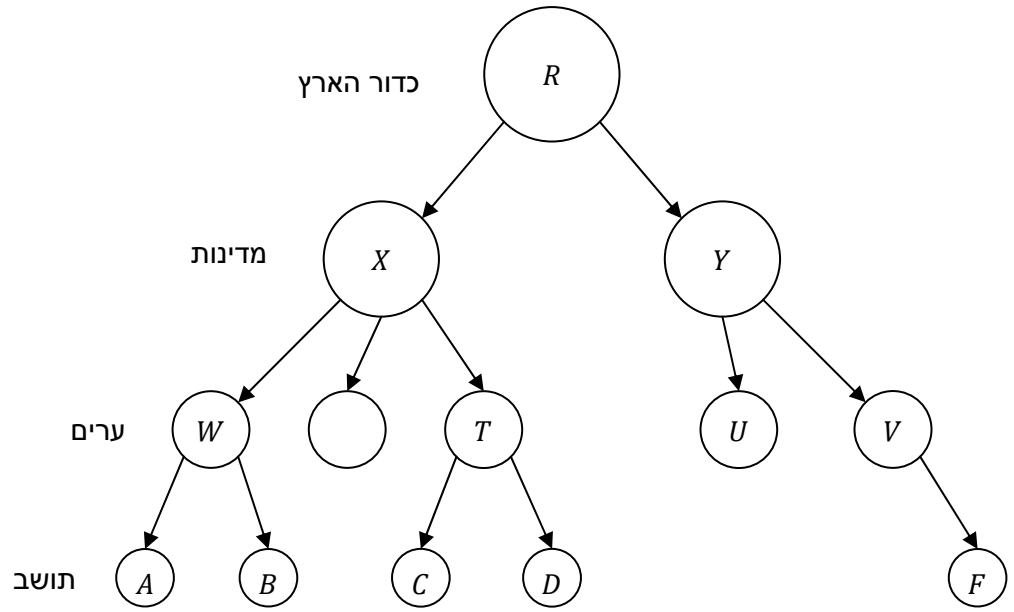
## Delta CRL

עוד דרך להתמודד עם הגודל של הקבצים היא להוריד את הרשימה המלאה בתדירות נמוכה יחסית ובתדירות גבוהה יותר להוריד הפצות שרק מציינות את ההבדלים מההפצה הקודמת. ההנחה היא שבפרקי זמן קטנים אין הרבה שינויי ולכן רשימות השינויים לא גדולות כל כך.

המציאות היא שלא ניתן ליצור CA אחד לכל העולם. יש המון ארגונים ומדינות שונות ולא כולם רוצים לשתף פעולה עם כולם. וגם אם כולם היו חברים שרת אחד לא יוכל להחזיק כמות מידע כל כך גדולה ולטפל בבקשות של כל הישויות על כדור הארץ... לכן יש CA-ים שונים לתחומים שונים. למשל הממשלה יכולה להוות CA לאזרחים שלה והאוניברסיטה יכולה להוות CA לתלמידים ולסגל שלה.

הניהול של המבנה נקרא **Trust Management**.

### מודל היררכי




העלים מסמלים את הישויות ברמה הכי נמוכה וכל הצמתים מהווים CA-ים. לכל עלה יש מפתח ציבורי של ה-CA הראשי $R$ ושל האבא שלו. כל צומת מנפיק אשרות לבנים שלו. כאשר $\alpha$ מנפיק תעודה עבור $\beta$ נסמן זאת ב-$\alpha \gg \beta \ll$.

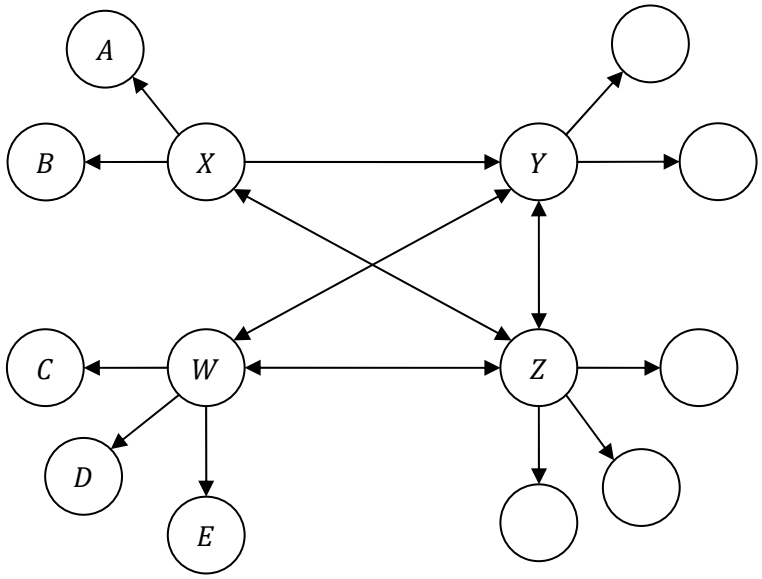
אם $A$ רוצה לקבל את המפתח הציבורי של $F$ הוא צריך לעבור מסלול של תעודות:

$$R \ll Y \gg, Y \ll V \gg, V \ll F \gg$$

למשל, אם $F$ זו חנות פרחים בטוקיו ו-$A$ זה אנחנו ואנחנו ניגשים לאתר האינטרנט שלה ורוצים להזמין מהם פרחים. כנראה שבאתר של החנות כבר שמור $V \gg F \ll$. אבל בשביל לבדוק את האשרה הזאת צריך לדעת את המפתח של $V$. לכן צריך $Y \gg V \ll$ ובאותו אופן צריך גם $R \gg Y \ll$ ומאחר שאנחנו יודעים את המפתח הציבורי של $R$ סיימנו. המסלול הזה נקרא **השגת אשרות**. זה נראה מאוד פשוט אבל למעשה זה די מורכב.

ברור שהמודל הזה לא מציאותי. לא באמת קיים CA אחד ראשי של כדור הארץ. אבל בכל זאת המודל מתאים לארגונים גדולים שיש להם מבנה היררכי. למשל ל-IBM יש מנהלה והיא יכולה להנפיק תעודות לסניפים שלה והסניפים יכולים להנפיק תעודות לעובדי הסניף.

## מודל ארגוני



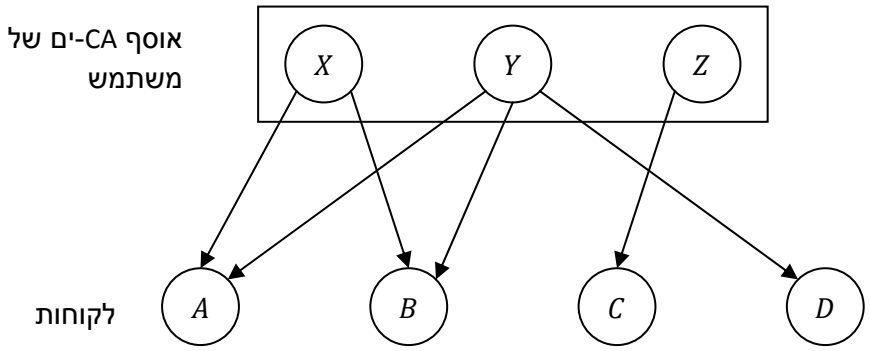

יש ארגונים שיש ביניהם יחסי גומלין ולכל ארגון יש לקוחות. כל ארגון מנפיק אשרות ללקוחותיו ובין הארגון יש **Cross-Certificates**, כלומר הארגונים נותנים אשרות אחד לשני.

במודל הזה יכול להיות יותר ממסלול השגת אשרה אחד. למשל, אם $A$ רוצה לקבל את האשרה של $D$ יש שני מסלולים אפשריים:

$$W \ll D \gg, Y \ll W \gg, X \ll Y \gg$$
$$W \ll D \gg, Z \ll W \gg, Z \ll Z \gg$$

מציאת מסלול נקראת **Path Exploration**. כמובן יש שיקולים שונים לבחירת המסלול (איכות, מחיר, אורך ועוד).

## מודל שטוח

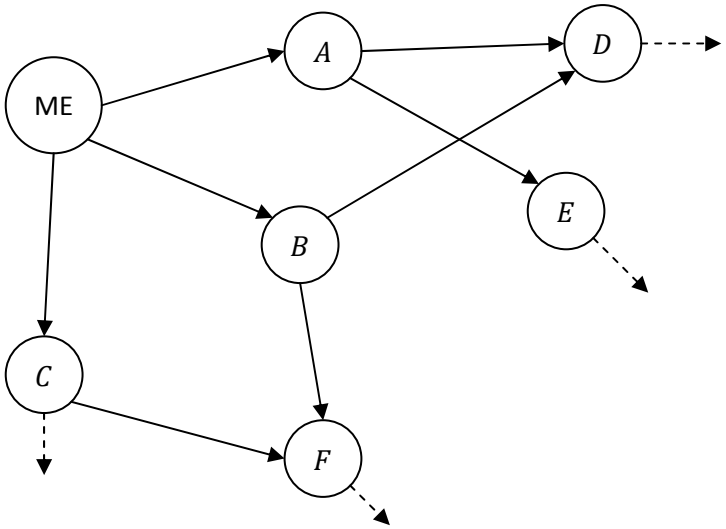



לכל משתמש יש רשימה של CA-ים שהוא סומך עליו וכל CA מנפיק אשרות ללקוחותיו. כך כל משתמש יכול לקבל אשרות על כל משתמש של כל אחד מה-CA-ים שהוא סומך עליו.

זה בדיוק מה שקורה בדפדפני אינטרנט. חלק מהדפדפן זו למעשה רשימה של CA-ים שהוא סומך עליהם. כשגולשים לאתר באינטרנט מחפשים את האשרה שלו דרך ה-CA-ים האלה. הבעיה עם זה היא שהרשימות האלה ניתנות לקונפיגורציה ואפשר לשתול שם CA מזויף.

## מודל Web of Trust

המודל יוצא מנקודת ההנחה שהמשתמש הוא מרכז העולם והוא בונה סביבו שכבות של אמון בדומה ל"חבר מביא חבר":




כל משתמש קובע לעצמו את המדיניות בנוגע לקשרים. למשל אפשר לקבוע שסומכים רק על מי שקרוב אלינו מדרגה שנייה לכל היותר. כך חברים של $D$ למשל לא יהיו ב-Web of trust שלנו.

# תשתיות מפתחות ציבוריים

לפני שנדבר באופן פרטני על **תשתיות מפתחות ציבוריים** (**Public Key Infrastructure** או בקיצור PKI) נציין מהם המאפיינים הכלליים שאנחנו מצפים מכל תשתית:

- שימוש באופן נרחב
- נגישות
- פשטות
- אמינות
- סטנדרטיות
- שקיפות למשתמש

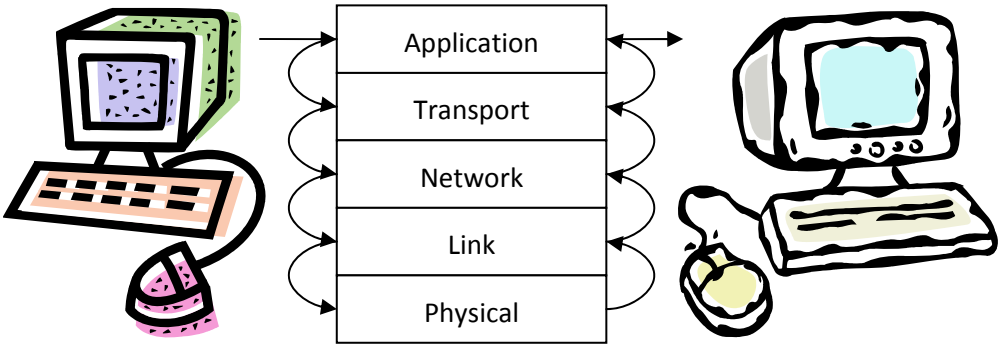
## שירותים שהתשתית נותנת

- **רשות המדיניות** (**Policy Authority** או בקיצור PA): למערכת גדולה צריכה להיות מדיניות – הגדרה של מה מותר ומה אסור, רשימה של מקרים ותגובות, מה אפשר לעשות עם האשרות ועוד. בד"כ עורכי דין אחראים על כתיבת המסמך.
- **רשות אשרות** (ה-CA): הפעולה היחידה והכי חשובה של רשות זו היא בניית האשרות וחתימה עליהן. המפתח הפרטי של הרשות הזאת הוא מאוד רגיש ולכן גם מאוד חזק.
- **רשות רישום**: המחשב או הפקיד שאליו ניגשים, מזדהים ומציגים את הבקשות. זה הגוף שפונה ל-CA כדי לקבל את האשרה אחרי שהוא משוכנע שזהות הלקוח אמיתית.
- **מאגר אשרות**
- **גיבוי מפתחות**: זהו מאגר מאוד רגיש של גיבויים של מפתחות פרטיים (למקרה שמתפח נעלם, למשל). בוודאי מאגר זה אינו נגיש לכולם ומוגן מאוד.
- **אחזור מפתחות**: כשיש צורך אפשר להוציא מפתח פרטי מהגיבוי. כמובן, יש להחליט אין בודקים זהות הרגע שללקוח נאבד המפתח הפרטי שלו ואיך מוציאים את המפתח מהמאגר בצורה מאובטחת.
- **עדכון מפתחות**:המפתחות טובים רק לתקופת זמן מוגבלת. התשתית מאפשרת לעדכן מפתחות, אלא שצריך להחליט איך זה קורה מבלי להטריד את הלקוח ואיך לעדכן את המפתחות אוטומטית.
- **ארכיון**: מאחר שמפתחות משתנים עם הזמן צריך לשמור את ההיסטוריה כדי שיהיה אפשר לפתוח ולוודא מסמכים שנוצרו לפני הרבה זמן.
- **ניהול אשרות**: לידה, הפצה, ביטול וכו'.

# אבטחה ברמת הרשת

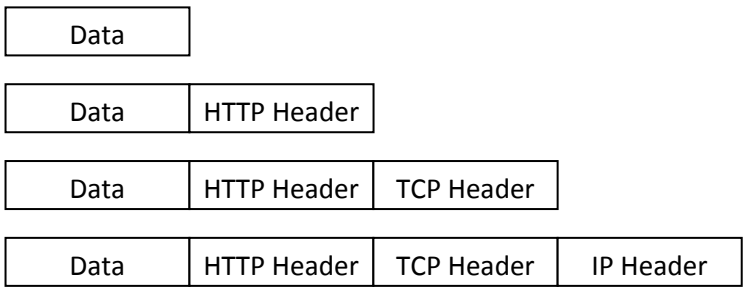
## מודל השכבות של הרשת

במערכת תקשורת יש היררכיה של פרוטוקולים:




כאשר שני אנשים מתקשרים זה עם זה דרך הרשת הם עובדים מול תוכנה (שכבת Application). המידע מחלחל דרך הפרוטוקולים עד שהוא עובר לרמת החומרה וזורם בכבלים לצד השני. שם המידע עובר חזרה לתוכנה והאדם בצד השני יכול לראות אותו.

כששולחים מידע ברשת למעשה יוצרים **חבילות** מקוננות של מידע לפי השכבות. החבילות אלה נקראות **packets**. למשל, אם שולחים דרך HTTP המידע עובר את השינויים הבאים:




וכן הלאה...

נפרט קצת על השכבות השונות:

1. **Physical**: זו הרמה האנלוגית ולא נדבר עליה למרות שניתן לאבטח מידע ברמה האנלוגית.
2. **Link**: זו רמה שמטפלת בשירותים של חיבור ספרתי על גבי תווך מיידי (כמו Ethernet). לכל יחידה יש כתובת MAC והמידע יכול לעבור בין יחידות. יש פרוטוקולים לאבטחה ברמת ה-link, אבל אנחנו לא נדבר עליהם. IEEE אחראים על זה.
3. **Network**: כאן כבר האובייקטים לא צריכים להיות מחוברים ישירות והרמה עושה ניתוב של חבילות מידע ברשת. זה נעשה בעזרת כתובות IP הצמתים ברשת.
4. **Transport**: כאן קורה ניהול השיחה (סדר חבילות וכו') כאשר מתעלמים מהמיקום הפיסי של הדברים. למשל TCP או UDP.
5. **Application**: זהו הצד הפונקציונאלי של יישומים. למשל שירותים של HTTP, FTP, SMTP.

נשאלת כעת השאלה באיזו מהרמות צריך לתת את האבטחה. התשובה היא לא חד משמעית ותלויה בצרכים. למשל, אם עושים רק אבטחה פיסית אז לכאורה זה מצוין אבל למעשה זה שירות מאוד מצומצם, שהרי אולי נרצה לאמת גם ישויות לוגיות ולא רק כתובות פיסיות וזה בלתי אפשרי אם עובדים רק ברמה הפיסית. ברמת האפליקציה אפשר לתת הרבה יותר גמישות לאבטחה. אבל מצד שני החיסרון הגדול הוא שכל יישום צריך לממש את האבטחה מחדש וזה לא יעיל...

## פרוטוקול IP

במקור המטרה של פרוטוקול IP הייתה מחקרית לגמרי ולכן לא הייתה בו בכלל אבטחה וקל היה להאזין, לשתול ולזייף הודעות. התכונות המקוריות היו:

- **חד-כיווניות**: השולח הוא רק שולח והמקבל הוא רק מקבל. אם השולח רוצה לקבל ולהפך הם צריכים לפתוח ערוץ תקשורת נוסף.
- **חוסר אמינות**: לא הייתה בדיקת הגעה של חבילות. לשולח אין אינפורמציה אם החבילה הגיע או לא והמקבל בוודאי לא יודע אם קרתה תקלה. זה נקרא **Best Effort**. דווקא ברמת ה-transport אפשר להעביר משוב אבל זה לא חלק מפרוטוקול ה-IP.
- **כתובות IP**: לכל צומת ברשת יש כתובת IP ולפי הכתובת הזאת מנתבים את ההודעות. בגרסה IPv4 הכתובות הן באורך 32 ביט וזו בעיה כי זה לא מספיק כדי לתאר את כל המחשבים שיש כיום בעולם. חוץ מזה לפעמים הכתובות הן לא חד-ערכיות כי למשל ספקיות אינטרנט נותנות כתובות IP דינאמיות למשתמשים שלהן ובזמנים שונים אותה הכתובת יכולה להיות משויכת למחשבים שונים.
- **סדר הגעה**: סדר ההגעה של החבילות (אם בכלל) לא מובטח.
- קל להאזין לחבילות IP או לשתול מזויפות.

## פרוטוקול IPsec

בשנות ה-90 הגיעו להבנה שאי אפשר להמשיך עם פרוטוקול IP ללא שום הגנה וקמה ועדה גדולה שאמורה הייתה לפתח פרוטוקול **IP Security** (בקיצור IPsec) שעומד בדרישות הבאות:

- זיהוי מקור
- אימות מידע
- סודיות מידע (אם רוצים)
- שמירה על סדר חבילות
- מניעת התקפת Replay
- הרשאות גישה (Access Control)
- פרטיות (במידה מסוימת)
- תאימות מלאה עם פרוטוקול IP (כדי שהנתבים הישנים יוכלו להעביר את ההודעות של הפרוטוקול החדש)

## הארכיטקטורה של IPsec

- השולח והמקבל מחזיקים כמה מאגרים:
  - **Security Policy Database** (או בקיצור SPD): מאגר מידע של מדיניויות לגבי החבילות השונות: איזה חבילות יוצאות? אילו שירותים לתת לחבילות? איזה חבילות להצפין? וכו'.
  - **Security Association Database** (או בקיצור SAD): השולח והמקבל מחזיקים מבנה נתונים מתואם שנקרא Security Association (או בקיצור SA) והוא מחזיק בתוכו מפתחות הצפנה, מספרים סידוריים של חבילות וכו'. לכל לקוח יש מאגר SA-ים כאלה שמתאימים לכל ערוץ תקשורת שיש לו. הכי פשוט לתאם אותו בעזרת התקנה פיסית. כשאין אפשרות להתקנה פיסית **Internet Key Exchange** (או בקיצור IKE) אחראי לתיאום מרוחק של SA.
- יש שתי תצורות של העברה:

- **Transport Mode**: רק המידע של החבילה מקבלת אבטחה (כלומר הצפנה או אימות) ואילו המידע שעובר יחד עם החבילה שאחראי על הניתוב שלה לא משתנה.
- **Tunnel Mode**: כל החבילה יחד עם המידע שנוסף לה לצורך המעבר ברשת מקבל אבטחה. למעשה מכניסים את החבילה המקורית לחבילה חדשה ומאובטחת.

## מבנה חבילות IPv4

חבילת IP מורכבת מהמידע שאנחנו רוצים לשלוח ומתוספת שנקראת IP Header.

| + | Bits 0–3 | 4–7 | 8–15 | 16–18 | 19–31 |
|---|---|---|---|---|---|
| 0 | Version | Header length | Type of Service (now DiffServ and ECN) | Total Length | |
| 32 | Identification | | | Flags | Fragment Offset |
| 64 | Time to Live | | Protocol | Header Checksum | |
| 96 | Source Address | | | | |
| 128 | Destination Address | | | | |
| 160 | Options | | | | |
| 160 or 192+ | Data | | | | |

1. **Version**: השדה הראשון ב-header מציין את הגרסה של הפרוטוקול. ב-IPv4 מופיע שם המספר 4.
2. **Header Length**: מספר המילים בנות 32 ביט ב-header בלבד. השדה הזה נחוץ משום שב-IPv4 יכול להיות מספר משתנה של options.
3. **Type of Service**: העדפה של הטיפול בחבילה בדרכה דרך הרשת.
4. **Total Length**: האורך הכולל (בבתים) של החבילה שכולל גם את המידע וגם את ה-header.
5. **Identification**: נועד בעיקר כדי לזהות באופן חד-ערכי את חלקי ההודעה המקורית (לפי שחולקה לחבילות).
6. **Flags**: דגלים שנועדו לבקרה ופירוק של חבילות.
7. **Fragment Offset**: הסטייה של המידע של החבילה הנוכחית מתחילת ההודעה המקורית לפני שחולקה לחבילות קטנות יותר.
8. **Time to Live**: מונע מהחבילה לשוט ברשת במעגלים. הספירה נעשית ב"קפיצות" (Hops) של החבילה בין נתב לנתב. בכל "קפיצה" ערך השדה קטן באחד וכשהוא מגיע לאפס החבילה לא מועברת יותר ע"י הנתבים.
9. **Protocol**: זה הפרוטוקול שנמצא בשימוש בחלק של המידע בחבילה, כלומר למעשה הוא מציין לאיזה פרוטוקול בשכבת ה-transport צריך להעביר את המידע.
10. **Header Checksum**: נועד לבדיקת שגיאות ב-header. הבדיקה נעשית בכל "קפיצה" של החבילה. אם הבדיקה נכשלת החבילה לא מועברת יותר.
11. **Source Address**: כתובת IP של שולח ההודעה.
12. **Destination Address**: כתובת IP של הנמען של ההודעה.
13. **Options**: שדה אופציונאלי עם כל מיני תוספות.
14. **Data** (או **Payload**): זה כבר לא חלק מה-header והוא לא נכלל בשדה של ה-checksum. את תוכן שדה המידע צריך להעביר לפרוטוקול שמצוין בשדה protocol.

## אבטחה ברמת החבילה

יש שני פרוטוקולים של אבטחה ברמת החבילה. **Authentication Header** (בקיצור AH) משיג אימות ואי-התכחשות. ו-**Encapsulated Security Payload** (בקיצור ESP) משיג חשאיות ובמידת הצורך גם אימות.

### Authentication Header

יש שתי אפשרויות בהתאם לתצורת ההעברה:

| | | | | |
|---|---|---|---|---|
| חבילה מקורית | Payload | IP Header | | |
| Transport Mode | Payload | AH Header | IP Header* | |
| Tunnel Mode | Payload | IP Header | AH Header | New IP Header |

הפרוטוקול מוסיף לחבילה header שמכיל את השדות הבאים:

| 0 - 7 bit | 8 - 15 bit | 16 - 23 bit | 24 - 31 bit |
|---|---|---|---|
| Next header | Payload length | RESERVED | |
| Security parameters index (SPI) | | | |
| Sequence number | | | |
| Authentication data (variable) | | | |

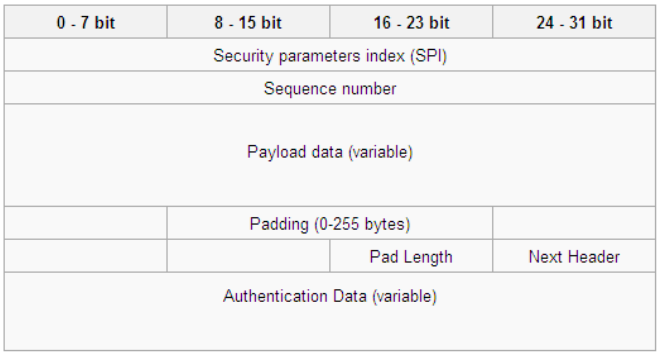
1. **Next Header**: הפרוטוקול בשכבת ה-transport שהמידע בחבילה צריך לעבור אליו.
2. **Payload Length**: הגודל של חבילת ה-AH
3. **RESERVED**: כרגע זה שדה שמלא באפסים והוא שמור לשימושים עתידיים.
4. **Security Parameters Index**: הפרמטרים של ההצפנה אשר בשילוש עם כתובת ה-IP קובעים את ה-SA שבעזרתו מומשה החבילה.
5. **Sequence Number**: המספר הסידורי של החבילה בזרם שנועד למנוע התקפת replay.
6. **Authentication Data**: שדה שמכיל את המידע הדרוש כדי לאמת את החבילה. ה- AH Header מאמת את כל מה שנמצא בחבילה ולא משתנה בדרך.

### Encapsulated Security Payload

כמו קודם, יש שתי אפשרויות בהתאם לתצורת ההעברה:

| | | | | | |
|---|---|---|---|---|---|
| חבילה מקורית | Payload | IP Header | | | |
| Transport Mode | ESP Trailer | Payload | ESP Header | IP Header* | |
| Tunnel Mode | ESP Trailer | Payload | IP Header | ESP Header | New IP Header |

פרוטוקול ESP מספק אימות של המקור, אמינות וחשאיות לחבילה. הפרוטוקול תומך גם באפשרות של הצפנה בלבד או אימות בלבד, אבל שימוש בהצפנה ללא אימות לא מומלץ. בניגוד ל-AH, חבילת ה-IP לא מוגנת ע"י ESP (למרות שב-tunnel mode האבטחה ניתנת לכל חבילת ה-IP הפנימית כולל ה-header הפנימי, ה-header החיצוני לא מוגן).




1. **Security Parameters Index**: מזהה את הפרמטרים של האבטחה ויחד עם כתובת ה-IP מגדיר את ה-SA שהחבילה ממומשת בעזרתו.
2. **Sequence Number**: מספר סידורי של החבילה בזרם שנודע למנוע התקפת replay.
3. **Payload Data**: המידע שמועבר.
4. **Padding**: נועד לצפני בלוקים שצריכים לקבל מידע בגודל מסוים וקבוע.
5. **Pad Length**: מספר הבתים שבריפוד.
6. **Next Header**: מזהה את הפרוטוקול של המידע המועבר.
7. **Authentication Data**: המידע שנחוץ לאימות החבילה (האימות כאן לא חזק כמו האימות של AH כי הוא לא חל על ה-header).

## מבנה הנתונים SAD

כאמור, SA הוא מבנה נתונים שמתואם לכל זוג של שולח ומקבל. הוא כולל בתוכו את הנתונים הבאים:

- **Security Parameters Index** (או בקיצור SPI): אינדקס ייחודי באורך 32 ביט שנקבע ע"י הנמען ומציין את הכניסה ל-SAD שבעזרתה הוא יוכל לפענח חבילות נכנסות.
- **כתובת מקור**
- **כתובת נמען**
- **פרוטוקול**
- **תצורת העברה** (Transport או Tunnel)
- **אלגוריתמים** ומפתחות הצפנה או אימות
- **מספר סידורי בזרם הנוכחי** – כל פעם שהשולח שולח חבילה הוא מקדם את המספר הזה
- **חלון Anti-Replay**
- **Lifetime** – זמן עד התפוגה או מספר חבילות עד התפוגה

ה-SAD זו פשוט טבלה של SA-ים יחד עם האינדקסים שלהם ששמורה גם אצל השולח וגם אצל המקבל כאשר לכל זוג של שולח ומקבל ולכל זרם הודעות ביניהם יש כניסה מתאימה משותפת בטבלה (ה-SA הרלוונטי מותקן באופן פיסי אצל שני הצדדים או מתואם באמצעות IKE). השימוש העיקרי של ה-SAD הוא בצד המקבל כי הוא זה שצריך לפענח את החבילה שהתקבלה. בכל חבילה רשום האינדקס של ה-SA שבעזרתו המקבל יכול לפענח את ההודעה.

## ה-Security Policy Database

ה-SPD הוא למעשה טבלת מדיניות שמשמשת בעיקר את הצד השולח ומגדירה מה צריך לעשות עם כל הודעה. הטיפול בהודעה מוגדר למעשה לפי הפעולה (Rule) וה-SA המתאים:

| Selectors | | | | | | |
|---|---|---|---|---|---|---|
| Source IP | Destination IP | Protocol | Source Port | Destination Port | Rule | SA |
| * | * | * | * | * | | |

יש שלושה חוקים אפשריים:

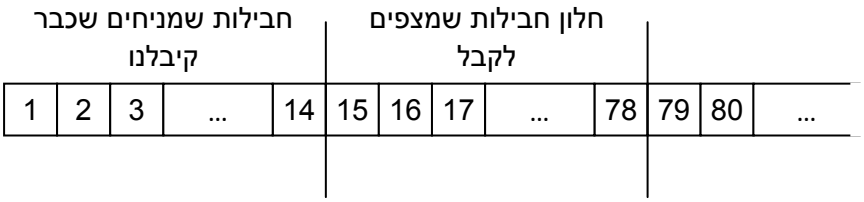
- **Drop**: אסור לתת להודעה לצאת מהמחשב
- **Pass**: מעבירים את ההודעה כמו שהיא
- **Apply**: מפעילים IPsec על החבילה בעזרת ה-SA שנתון בטבלה

הטבלה הזו מתפקדת במידה מסוימת כמו firewall. כשמגיעה חבילה שצריך לשלוח עוברים על כל הטבלה מלמעלה ולמטה ורואים אם כתוב מה המדיניות הראשונה שחלה עליה – כלומר מה צריך לעשות איתה. בד"כ השורה האחרונה בטבלה אומרת מה לעשות אם חבילת שאף אחד מהחוקים הקודמים לא חל עליהן.

טבלה כזו יכולה להיות גם בצד המקבל למקרה שגם הוא צריך לבדוק מה לעשות עם הודעות שמגיעות (אם הוא gateway למשל).

## מנגנון מניעת Replay

השולח לא חושש משום דבר אם הוא עובד לפי החוקים. מי שמוטרד בד"כ הוא דווקא המקבל. אז הוא עובד עם חלון Anti-Replay בגודל קבוע של 64 חבילות (הגודל קבוע כי הרבה פעמים מממשים את זה בחומרה ואז חוצץ קבוע הרבה יותר קל ויעיל למימוש מאשר חוצץ בגודל משתנה).




נניח שמגיעה הודעה עם מספר סידורי $i$. אז יש כמה אפשרויות:

1. $i$ משמאל לחלון – זה אומר שאנחנו שכבר קיבלנו את ההודעה הזאת ולכן אנחנו מתעלמים ממנה.
2. $i$ בתוך החלון – זה אומר שאנחנו מצפים לקבל את ההודעה הזאת.
   - אם כבר קיבלנו אותה מתעלמים
   - אם עוד לא קיבלנו אותה בודקים את ה-AH וה-ESP ורואים אם ההודעה תקינה.
     i. אם היא תקינה מקבלים אותה ומסמנים שקיבלנו.
     ii. אחרת דוחים אותה (אם רוצים אפשר גם לסמן בקובץ לוג שקיבלנו הודעה שלא הייתה תקינה).
3. $i$ מימין לחלון – בודקים את ה-AH ואת ה-ESP
   - אם ההודעה תקינה מקבלים אותה ומזיזים את החלון ימינה במספר הצעדים המינימאלי הדרוש כך שההודעה תהיה בתוך החלון.
   - אם ההודעה לא תקינה דוחים אותה.

ברור שהמנגנון הזה מונע replay. ברגע שהודעה הגיע פעם אחת באופן תקין היא לעולם לא תתקבל שוב, אלא אם מישהו הצליח לשנות את המספר הסידורי שלה אבל זה כבר לא replay.

## Internet Key Exchange

כאשר אי אפשר לסנכרן את ה-SAD פיסית צריך להשתמש ברשת כדי לתאם אותו. **Internet Key Exchange** (או בקיצור IKE) הוא הפרוטוקול שאחראי על זה והוא מורכז משתי פאזות: הראשונה, אחראית על תיאום SA של IPsec והשנייה אחראית על תיאום SA של זרם תקשורת מסוים.

1. פאזה 1 – תיאום SA (לכל זוג משתמשים)
   - תיאום SA
   - תיאום מפתח בעזרת דיפי-הלמן
   - אימות צדדים
   - הסתרת זהויות
   - העברת תעודות
2. פאזה 2 – תיאום SA של השיחה
   - תיאום מפתח זרם (סימטרי או אסימטרי)

## יישום של IPsec

### Peer-to-Peer

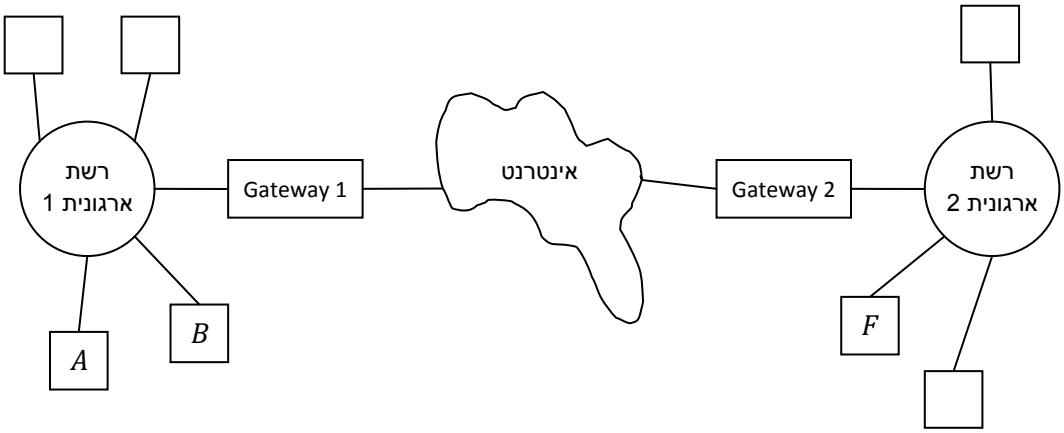
ברשתות תקשורת מחשבים, **רשת עמית לעמית** (**Peer-to-Peer** או בקיצור P2P) היא רשת תקשורת בה כל אחד מהקצוות מתפקד הן כלקוח והן כשרת, וכל אחד מהקצוות מסוגל ליזום או לסיים התקשרות וכן לספק או לדרוש שירותים. בהשוואה לרשת שרת לקוח בה יש הבחנה ברורה בין הצדדים השונים ברשת, ברשת עמית לעמית אין הבחנה כזו, והרשת מתפקדת באופן מבוזר, ללא צורך בתיווך או תלות בגורם מרכזי כלשהו.

רשתות עמית לעמית משמשות בעיקר ברשתות שיתוף קבצים, בהן כל משתמש מאפשר לשאר המשתמשים להוריד קבצים מהמחשב שלו, ומקבל בתמורה גישה לקבצים ממחשבי המשתמשים האחרים.

מאחר שמבחינה לוגית כל חבילה עוברת ישירות בין משתמש אחד לשני אין היגיון בשימוש ב- tunnel mode, שהרי לא משנים את השולח ואת המקבל אז זה בזבוז משאבים.

### Virtual Private Network

**Virtual Private Network** (או בקיצור VPN) היא שיטה להעברת מידע פרטי על גבי תשתית שעיקרה או כולה בבעלות ציבורית או עם גישה ציבורית ובבעלות פרטית. הטכנולוגיה מתבססת על תשתית ציבורית (בדרך כלל רשת האינטרנט) כדי להעביר נתונים בתוך הרשת הפרטית כך שתהיה למשתמש ב־VPN גישה למשאבים ארגוניים גם כאשר אין כבל ישיר המחבר ביניהם.




אם סומכים על הרשת הארגונית אז חבילות שעוברות בתוכה לא צריכות לעבור דרך IPsec (למשל כאשר $A$ מדבר עם $B$). אבל אם רוצים ליצור תקשורת בין רשתות ארגוניות צריך לעבור דרך האינטרנט ולכן צריך להגן (אימות, הצפנה או שניהם) על החבילות.

אם שני הצדדים מריצים IPsec זו לו בעיה אבל בד"כ זה לא המצב (למשל כשחבר בארגון מתחבר לרשת הארגון מהמחשב הביתי). במקרה כזה ה-gateway עושה את זה בשבילו בשימוש ב- tunnel mode: נניח ש-$A$ שולח חבילה ל-$B$. כשהחבילה מגיעה ל-gateway 1 הוא עוטף אותה בתוך חבילה חדשה שהמקור שלה הוא gateway 1 והיעד הוא gateway 2. כשהחבילה מגיעה באופן מאובטח ל-gateway 2 הוא יכול להוציא את ההודעה הפנימית ולהעביר אותה ל-$F$.

אם המדיניות של הארגון היא שכל משתמש מריץ IPsec וכל הודעה צריכה להיות מאומתת אז בתוך הארגון כל הודעה יכולה לעבור ב-transport mode שיש בו רק אימות. אבל ברגע שההודעה מגיעה ל-gateway 1 היא עוברת ל-tunnel mode. כל התהליך הזה שקוף מבחינת $A$ ו-$F$: הם מדברים אחד עם השני כאילו היו בתוך אותה רשת ארגונית.

# אבטחה ברמת ה-Transport

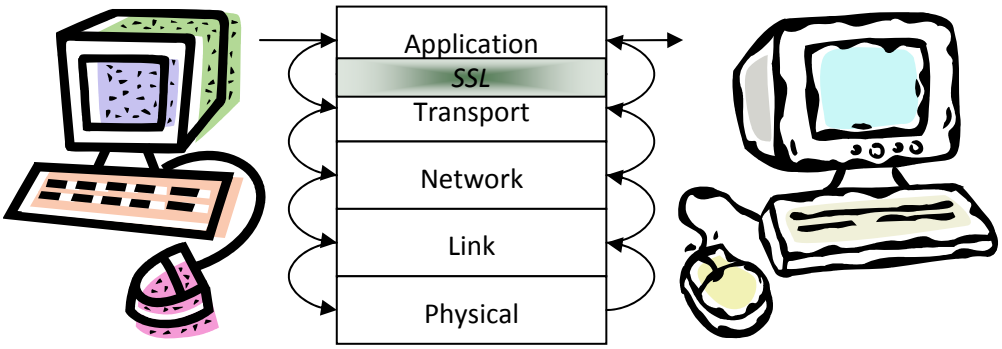
## המטרה של SSL והשימוש בו

ישנם סוגים שונים של אבטחת מידע:

- אבטחת שיחה
- אבטחה דו-כיוונית
- אימות מקורות
- סודיות תוכן
- אימות תוכן
- שמירה על סדר
- אי משלוח חוזר של הודעות
- שליחה של חבילות אבודות

רוב הדברים האלה לא קיימים בפרוטוקול IP. **Secure Socket Layer** (או בקיצור SSL) הוא פרוטוקול אבטחת מידע שפותר את הבעיות האלה. זה פרוטוקול שמאפשר ליישומים להעביר מידע דרך הרשת בצורה שמונעת האזנה, התעסקות וזיוף הודעות. בד"כ רק זהות השרת עוברת אימות ואילו זהות הלקוח נשארת לא מאומתת. פירוש הדבר שרק משתמש הקצה (בן אדם או אפליקציה כמו דפדפן אינטרנט) יודע בוודאות עם מי הוא מתקשר. כדי להשיג אימות הדדי הלקוח בד"כ צריך להציג סיסמה אבל זה נעשה רק אחרי שכבר פתוח ערוץ תקשורת מאובטח והלקוח יכול להעביר בו את הסיסמה שלו ללא חשש.

המקום של SSL בהיררכיית הרשת הוא בין רמת האפליקציה לרמת התעבורה.




כששני צדדים רואים לתקשר בערוץ מאובטח זה נעשה בארבעה שלבים:

1. לקוח ושרת מנהלים שיחת TCP לא מאובטחת
2. ברגע שהם מחליטים שהם רוצים לאבטח את ערוץ התקשורת הם מריצים פרוטוקול SSL Handshake אשר מאמת את הזהויות ומתאם את האלגוריתמים הקריפטוגרפיים
3. אחרי תיאום האלגוריתמים ואימות הזהויות הלקוח והשרת יכולים לנהל שיחה מאובטחת
4. מסיימים את ה-SSL בצורה מסודרת והשיחה שוב לא מאובטחת

עם רוצים שוב לעבור לשיחה מאובטחת חוזרים לשלב 2 וחוזר חלילה.

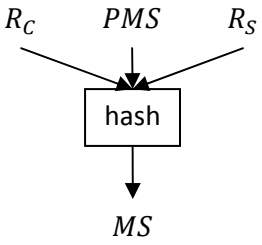
חשוב לציין שקיים פרוטוקול נוסף בשם **Transport Layer Security** (או בקיצור TLS) אשר מאוד ל-SSL מבחינה ארכיטקטונית אבל שונה ממנו במימוש ולכן אין ביניהם תאימות.

## Sessions ו-Connections

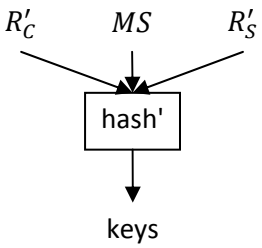
השימוש ב-SSL מחולק לוגית לשתי רמות: **Session** ו-**Connection**.

- Session – נפתח בתדירות נמוכה יותר ויכול לכלול בתוכו הרבה connections. לכל session צריך לקבוע את השדות הבאים:
  - Session ID (32 בתים) – מוענק ע"י השרת כדי שיוכל לעקוב אחרי השיחות שלו
  - Peer certificate – בד"כ רק השרת מציב אשרה ללקוח אבל יכול להיות גם הפוך
  - Compression algorithm – אלגוריתם לדחיסת המידע (בד"כ לא משתמשים בזה)
  - Cipher specification – אלגוריתמים להצפנה ולאימות
  - IS_RESUMABLE – נקבע ע"י השרת ואומר אם ה-session חד-פעמי או בר חידוש
  - Master Secret (MS) – מתואם ע"י שני הצדדים בתקשורת ונשמר אצל שניהם
- Connection
  - Session ID (32 בתים) – מספר ה-session שה-connection משויך אליו.
  - Server and client sequence numbers
  - Server and client random numbers
  - מפתחות קריפטוגרפים:
    - ENC_WRITE_KEY
    - ENC_READ_KEY
    - MAC_WRITE_KEY
    - MAC_READ_KEY
    - IV_WRITE
    - IV_READ

כדי להקים את ה-session משתמשים בקריפטוגרפיה אסימטרית וכדי להקים connection משתמשים בקריפטוגרפיה סימטרית. כדי לייצר את האלמנטים הסודיים משתמשים ב-**Pre Master Secret** (בקיצור PMS ובמספרים אקראיים שממציאים גם השרת וגם הלקוח $R_S$ ו-$R_C$ ומהם מייצרים את ה-**Master Secret**:




כדי לייצר את המפתחות עושים משהו דומה:

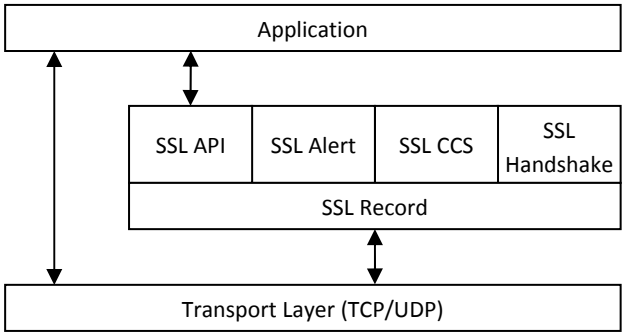



לכל connection מייצרים מספרים אקראיים חדשים. בד"כ כשפותחים session מיד לאחר הפתיחה שלו פותחים גם את ה-connection הראשון. אז ב-connection הראשון משתמשים לצורך החיסכון באותם מספרים אקראיים כמו של ה-session.

ה-PMS יכול להגיע מכמה מקורות:

- אפשר לבנות PMS ע"י DH בעזרת נתונים שגם השרת וגם הלקוח נותנים
- להעביר בצורה מוצפנת ע"י RSA
- להשתמש בחומרה בשני הצדדים (אפשרות זו נמצאת בעיקר בשימוש צבאי)

## הארכיטקטורה של SSL




האפליקציות לא חייבות להשתמש ב-SSL ולכן עדיין יש חיבור ישיר בין רמת האפליקציה לרמת התעבורה. אבל במקרה שיש צורך ב-SSL קיים API שמאפשר לאפליקציה להשתמש בפרוטוקול.

## SSL Record Protocol

SSL Record Protocol הוא פרוטוקול שאורז את כל התקשורת הנעשית במהלך יצירת הערוץ המאובטח. פרוטוקול SSL למעשה מחליף רשומות (records) שעוטפות את המידע שיש להעביר. כל רשומה היא בגודל לכל היותר $2^{14}$ בתים ואפשר לכווץ אותה, להוסיף לה ריפוד, להוסיף לה קוד מאמת (MAC), או להצפין אותה. האפשרויות תלויות במצב של החיבור (למשל את החבילות הראשונות אי אפשר להצפין או לאמת כי עדיין אין שום סוד משותף).

כל רשומה כוללת מידע על אורך ההודעה, סוג החבילה ועוד כמה פרטים טכניים:

| + | Bits 0–7 | 8-15 | 16-23 | 24–31 |
|---|---|---|---|---|
| 0 | Content Type | Version (MSB) | Version (LSB) | Length (MSB) |
| 32 | Length (LSB) | Protocol Message(s) | | |
| ... | Protocol Message (cont.) | | | |
| ... | MAC (optional) | | | |
| ... | Padding (optional) | | | |

## SSL CCS Protocol

Change Cipher Spec הוא פרוטוקול שלמעשה אומר מהו סוג מנגנון ההצפנה (אלגוריתמים ומפתחות) שהולך להיות בשימוש מעתה.

| + | Bits 0–7 | 8-15 | 16-23 | 24–31 |
|---|---|---|---|---|
| 0 | 20 | Version (MSB) | Version (LSB) | 0 |
| 32 | 1 | 1 (CCS protocol type) | | |

## SSL Alert Protocol

SSL Alert הוא פרוטוקול של הודעות שגיאה:

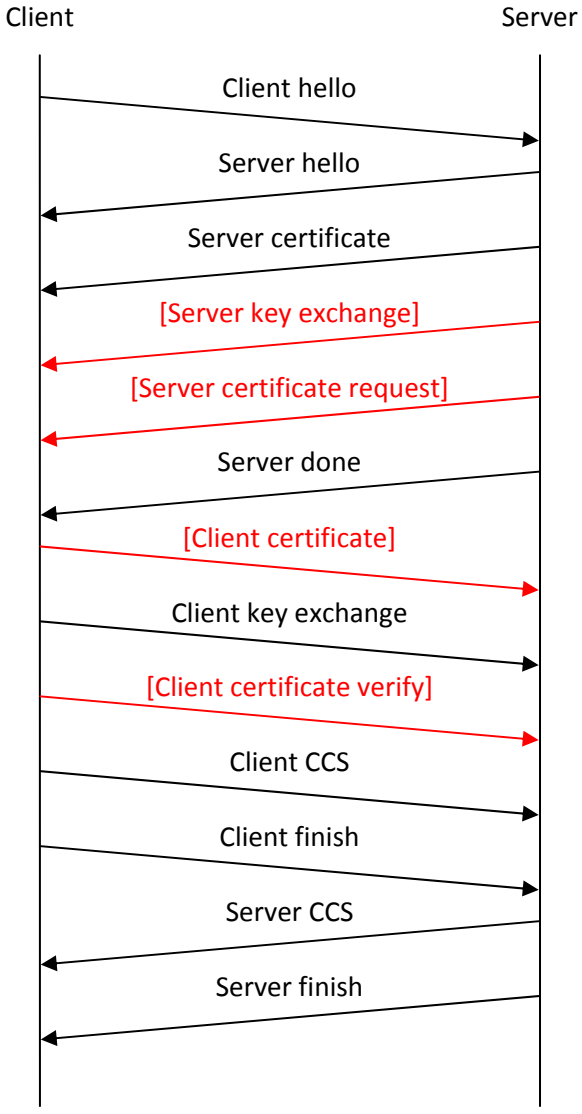
| + | Bits 0–7 | 8-15 | 16-23 | 24–31 |
|---|---|---|---|---|
| 0 | 21 | Version (MSB) | Version (LSB) | 0 |
| 32 | 2 | Level | Description | |

יש כ-25 סוגי שגיאות אפשריים. למשל MAC לא תקין, הצפנה לא ברורה, אשרה לא בתוקף, אשרה עם חתימה שגויה ועוד. סוג השגיאה מופיע בשדה description.

רמת השגיאה היא או **Warning** או **Fatal**. בהתאם לכך, חלק מהודעות השגיאה אפשר להעביר לאפליקציה (למשל אם אשרה לא בתוקף אולי בכל זאת הלקוח יסכים להמשיך לעבוד) וחלק צריך להעביר לשרת (למשל אם החתימה לא שגויה).

## SSL Handshake Protocol

באופן הכי כללי פרוטוקול ה-Handshake מורכב מההודעות הבאות (ההודעות שמסומנות באדום אינן הכרחיות והן תלויות בשיטת החלפת המפתחות ובאימות הלקוח):



1. **Client hello**: הלקוח שולח $(ID, version, algorithms, R_c)$ כאשר:
   a. $ID$ הוא ה-ID של ה-session הנוכחי.
   b. $version$ זו הגרסה של פרוטוקול ה-SSL שהוא מריץ.
   c. $algorithms$ היא רשימה של אפשרויות לאלגוריתמים (הצפנה, אימות, אולי דחיסה והחלפת מפתחות) מסודרת מהעדיפות הגבוהה אל העדיפות הנמוכה.
   d. $R_c$ מספר רנדומלי (nonce).
2. **Server hello**: השרת עונה עם $(ID, version, algorithm, R_s)$ כאשר:
   a. $ID$ הוא ה-ID של ה-session הנוכחי.
   b. $version$ זו הגרסה של SSL שמריץ השרת. אם היא נמוכה מזו של הלקוח אולי הלקוח ירצה לנתק את הקשר.
   c. $algorithm$ זו העדיפות הכי גבוהה של הלקוח לאלגוריתמים קריפטוגרפים שהשרת תומך בה. כמובן המטרה היא להשתמש בעדיפות הכי גבוהה אבל לא תמיד השרתים תומכים בכל האלגוריתמים.

   חשוב לשים לב שבשלב הזה עדיין אין שום דבר סודי בשיחה והיא גלויה לכל המאזינים.
3. **Server certificate**: כאמור, השרת חייב לאמת את זהותו אז הוא שולח אשרה (למשל X509) חתומה ע"י אחד ה-CA-ים שרשומים בדפדפן. לפעמים השרת שולח כמה אשרות למקרה שהלקוח לא מכיר את כל ה-CA-ים.
4. **Server key exchange**: יש שתי אפשרויות לפי האלגוריתם הנבחר.
   a. **DH** – השרת שולח את $g^x \ mod \ p$
   b. **RSA** – אין מה לשלוח וההודעה לא קיימת בכלל
5. **Server certificate request**: השרת שולח ללקוח רשימה של CA-ים שהוא מכיר. זה לא נעשה תמיד כי בד"כ ב-SSL לא דורשים שהלקוח יאמת את זהותו אבל אפשר לקנפג את הפרוטוקול כך שגם הלקוח יואמת.
6. **Server done**: בשביל הסנכרון בין הלקוח לשרת השרת מודיע ללקוח שהוא סיים לשלוח את כל מה שיש לו לשלוח.

   נשים לב שבשלב הזה הלקוח עדיין לא יכול להיות בטוח שמדובר בשרת האמיתי כי אלה הודעות שכולם רואים וכל אחד יכול להתחזות לשרת בשלב זה ע"י replay של האשרות למשל. בינתיים יש פה רק הזדהות ותיאום אלגוריתמים.
7. **Client certificate**: אם השרת ביקש מהלקוח אשרות הלקוח שולח לו את האשרות המתאימות. אם אין ללקוח אשרות מתאימות השרת יכול להחליט אם להמשיך בתקשורת או להפסיקה.

   כמו קודם, גם אם הלקוח מציג אשרות עדיין השרת לא יכול להיות בטוח בזהותו.
8. **Client key exchange**: שוב, יש שתי אפשרויות לפי האלגורתם הנבחר:
   a. **DH** – הלקוח שולח את $g^y \ mod \ p$
   b. **RSA** – הלקוח ממציא מספר אקראי $PMS$ ושולח $E_{Pub(S)}(PMS)$ כאשר $Pub(S)$ הוא המפתח הציבורי של שרת שנמצא באשרה שהשרת הציג ללקוח.

   עדיין הלקוח והשרת אינם בטוחים בזהות הצד השני!
9. **Client certificate verify**: הלקוח מוכיח את זהותו מול השרת ע"י חתימה על ערכים אקראיים קודמים בעזרת המפתח הפרטי שמתאים למפתח הציבורי באשרה שהוא הציג בפני השרת. כעת הלקוח הוכיח את זהותו והשרת יודע בוודאות שהוא מדבר עם הלקוח הנכון. הרי ההנחה היא שלאף אחד אחר אין את המפתח הפרטי של הלקוח אז מי שאינו הלקוח הנכון אינו יכול לחתום בצורה נכונה. כמובן, כדי שאי אפשר יהיה לעשות התקפת replay החתימה צריכה להיות על ערכים אקראיים.
10. **Client CCS**: הודעה זו מודיעה לשרת שמעכשיו התקשורת עוברת לאלגוריתמים שהוסכמו ע"י הלקוח והשרת ומוגדרים ע"י $ID$ כי שניהם יודעים את כל המפתחות הרלוונטיים.
11. **Client finish**: הלקוח שולח MAC של ערכים קודמים כלשהם כדי לציין שהוא סיים.

12. **Server CCS**: השרת שולח $ID$ שאמור להיות אותו אחד כמו זה של הלקוח.
13. **Server finish**: השרת שולח MAC של ערכים קודמים כדי לציין שהוא סיים.

נשים לב שבפרוטוקול הזה עובר מידע שאפשר לקלקל. למשל, מתקיף יכול לראות שלקוח מעוניין בגרסה 3.1 וגם השרת מוכן לזה אבל הוא ינטר את ההודעה שיגיד ללקוח שהשרת מוכן רק לגרסה 2.5. אם הלקוח מוכן להמשיך יש כאן פגם. לכן, למשל כאשר שולחים MAC של ערכים קודמים אחד הערכים האלה יכול להיות הגרסה ואז מתגלה חוסר ההתאמה. כך גם לגבי קבוצת האלגוריתמים שהשרת והלקוח מסכימים עליה. בסוף הפרוטוקול שני הצדדים יודעים שהערכים שיש להם הם הערכים הנכונים והם מתואמים.

כיום עובדים בעולם האינטרנט בעיקר עם RSA וללא כל החלקים האופציונליים. הלקוח מזדהה מול השרת רק באמצעות סיסמאות כאשר החיבור כבר מאובטח.

## תצורות הפעלת SSL Handshake Protocol

לא תמיד צריך להפעיל את הפרוטוקול המלא. ברוב הפעמים אפשר להסתפק רק בחלק מההודעות. כמו כן אפשר לשלוח כמה ערכים בבת אחת.

### RSA

אפשר להסתפק במשלוח ארבעה הודעות:

1. Client hello
2. את ההודעות הבאות ניתן לשלוח יחדיו:
   - Server hello
   - Server certificate
   - Server done
3. את ההודעות הבאות ניתן לשלוח יחדיו:
   - Client key exchange
   - Client CCS
   - Client finish
4. את ההודעות הבאות ניתן לשלוח יחדיו:
   - Server CCS
   - Server finish

נשים לב שגם הלקוח וגם השרת תורמים לאקראיות בעזרת $R_S$ ו-$R_C$, אבל דווקא כאן טמונה רגישות בפרוטוקול. הלקוח לא יכול לייצר מספרים אקראיים יותר מדי גדולים. השרת יכול לייצר מספרים אקראיים גדולים אבל הוא צריך לעשות את זה למיליוני לקוחות בזמנים קצרים. לכן משתמשים במספרים פסאודו-אקראיים וזה אומר שיש ביניהם קורלציה. אם מצליחים להתחקות אחרי המספרים האקראיים אפשר לשבור את האבטחה. כך אכן נשברו הגרסאות הראשונות של SSL.

### Diffie-Hellman ללא אימות

אפשר להסתפק במשלוח ארבעה הודעות:

1. Client hello
2. את ההודעות הבאות ניתן לשלוח יחדיו:
   - Server hello
   - Server certificate
   - Server key exchange ($g^x \ mod \ p$)

- Server done

3. את ההודעות הבאות ניתן לשלוח יחדיו:
   - Client key exchange ($g^y \ mod \ p$)
   - Client CCS
   - Client finish
4. את ההודעות הבאות ניתן לשלוח יחדיו:
   - Server CCS
   - Server finish

כאן ניתן לאמת כל דבר שהשרת שולח ע"י המפתח הפרטי שלו. לכן האשרה שהשרת שולח צריכה להיות זאת שמכילה את המפתח הציבורי שמתאים למתפח החתימה הפרטי של השרת. אז את $g^x \ mod \ p$ השרת יכול לשלוח יחד עם חתימה ואז זהות השרת מתחילה להתאמת קודם.

### Diffie-Hellman עם אימות

אפשר להסתפק במשלוח ארבעה הודעות:

1. Client hello
2. את ההודעות הבאות ניתן לשלוח יחדיו:
   - Server hello
   - Server certificate
   - Server key exchange ($g^x \ mod \ p$)
   - Server certificate request
   - Server done
3. את ההודעות הבאות ניתן לשלוח יחדיו:
   - Client certificate
   - Client key exchange ($g^y \ mod \ p$)
   - Client CCS
   - Client finish
4. את ההודעות הבאות ניתן לשלוח יחדיו:
   - Server CCS
   - Server finish

כמו קודם, כדאי שהלקוח ישלח את $g^y \ mod \ p$ מאומת. ניתן לעשות replay בשליחת ה-$g^y \ mod \ p$ המאומת אלא שזה לא יעזור כי מתישהו שני הצדדים יצטרכו להשתמש ב-$g^{xy} \ mod \ p$ ואז המתקיף ייתקע. זה יקרה מתישהו לקראת סוף הפרוטוקול אבל בכל זאת התרמית תתגלה.

### חידוש Session

בפרוטוקול מעורבות פעולות קריפטוגרפיות רבות שדורשות משאבים מרובים. הלקוח חלש חישובית והשרת עמוס ולכן כדאי לנשות לצמצם את הקריפטוגרפיה למינימום. כאן נכנסים לשימוש ה-sessions וה-connections. במקום לפתוח session חדש כל פעם כדאי לפתוח session אחד ולהוסיף לו connections לפי הצורך.

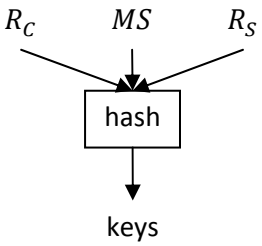
במקרה זה רצף ההודעות הוא כזה:

1. Client hello ($ID$)
2. את ההודעות הבאות ניתן לשלוח יחדיו:
   - Server hello ($ID$)
3. את ההודעות הבאות ניתן לשלוח יחדיו:

- Client CCS
- Client finish

4. את ההודעות הבאות ניתן לשלוח יחדיו:
   - Server CCS
   - Server finish

אם ה-$ID$ שהלקוח שולח כבר קיים אצל השרת והשרת מוכן להכניס לתוכו connection חדש אז משתמשים ב-$MS$, $R_S$ ו-$R_C$ ליצירת מפתחות חדשים:




אם השרת לא מוכן לחדש את ה-session אז הוא שולח ללקוח 0 ואז צריך להריץ את כל הפרוטוקול מחדש. למעשה כל התקשורת פועלת במין לולאה כזאת: הלקוח שולח לשרת את ה-$ID$ של ה-session שהוא מעוניין בו והשרת שולח לו חזרה את ה-session שהוא מוכן לעבוד בו. אם השרת הסכים למה שלקוח ביקש אז הכל טוב ויפה ואפשר לעבור מיד ל-CCS שבו שולחים שוב את ה-$ID$ הנבחר. אם השרת לא מוכן יש שתי אפשרויות:

- הלקוח לא מסכים לזה והשיחה נגמרת.
- הלקוח מוכן להשתמש ב-session שהשרת אמר ואז הם מבצעים את כל הפרוטוקול מחדש וב-CCS משתמשים ב-$ID$ שהשרת החליט עליו.

# מקורות לעיון נוסף

1. הסיכומים שלי מהשיעור
2. הסיכומים של אבי מהשיעור
3. המצגות של פרופ' קיפניס
4. ויקיפדיה
5. האתר של RAS-Labs