פרוטוקול הצפנה מקצה לקצה (E2EE)

מגיש: ישי ישראל

### **הקדמה**

להלן תיאור הפרוטוקול להבטחת הצפנה מקצה-לקצה (E2EE). פרוטוקול זה מבוסס על חילופי מפתחות והצפנה מודרנית, בהשראת פרוטוקולים כמו Signal, תוך התייחסות להעברת הודעות בזמן שאחד הצדדים אינו מקוון, שימוש במפתחות חד-פעמיים, אימות באמצעות OTP מספרי קצר ופקיעת תוקף של ה-OTP.

### **עקרונות יסוד**

1. **זהות הלקוח:**כל לקוח מזדהה באמצעות מספר טלפון (למשל "+972501234567"). מספר הטלפון חייב להתחיל בסימן "+" ואחריו 8 עד 15 ספרות. מספר יכול להיקלט ידנית על ידי המשתמש או להיות מוגדר מראש כארגומנט בעת הפעלת התוכנה. מספר זה הוא client\_id.
2. **תפקיד השרת (בלתי אמין):**השרת משמש כמתווך להעברת הודעות ומפתחות פומביים (כולל pre-keys), אך אין לו גישה לטקסט גלוי של ההודעות או למפתחות הפרטיים של הלקוח. כעת, בנוסף לשמירה על הודעות ומפתחות, לאחר סיום הרישום השרת והלקוח מחליפים הודעות באופן מוצפן עם session\_key, וכך נמנעת אפשרות למתקפות MITM על פרטי הבקשות.
3. **הצפנה מקצה לקצה (E2EE):**כל הודעה בין שתי לקוחות מוצפנת עם מפתח סימטרי (AES-256-GCM) הנגזר מחילופי מפתחות ECDH. המפתחות מבוססים על בסיס מפתחות זהות ומפתחות חד-פעמיים. לאחר סיום הרישום, גם הבקשות עצמן (כמו שליחת הודעה, בקשת הודעות, בקשת מפתחות של משתמש אחר) מוצפנות באמצעות session\_key שהוסכם בין הלקוח לשרת, וזאת על מנת למנוע התקפות MITM. בקשות אלה גם נחתמות בשימוש במפתחות ECDSA.
4. **OTP וערוץ מאובטח חד-פעמי:**בתחילת הרישום השרת מנפיק OTP קצר (6 ספרות) בעל תוקף מוגבל (למשל, 120 שניות). ה-OTP נשלח בערוץ מאובטח חד-פעמי. ה-OTP משמש לאימות השלב הראשוני שבו הלקוח מאמת את מפתח השרת ושניהם מבצעים חילופי מפתחות ליצירת session\_key. אם ה-OTP פג לפני השימוש, יש לבקש OTP חדש.
5. **מפתחות:**
   * **מפתח זהות (Identity Key):** מפתח X25519 לטווח ארוך (256 ביט). מפתח זה נשאר אצל הלקוח, והחלק הפומבי נשלח לשרת בסוף הרישום.
   * **מפתח חתימה (Signing Key):** מפתח ED25519 הנגזר מהמפתח הפרטי של מפתח הזהות. מפתח זה משומש לחתימות ההודעות בין לקוחות לשרת על מנת להבטיח Integrity.
   * **מפתחות חד-פעמיים (Pre-Keys):** הלקוח מעלה מספר מפתחות X25519 חד-פעמיים לשרת (לדוגמה 10). בעת שליחת הודעה למקבל לא מקוון, השולח משתמש באחד ה-pre-keys של המקבל כדי ליצור מפתח משותף מבלי שהמקבל יהיה מחובר.
   * **מפתח ציבורי של השרת:** השרת מחזיק מפתח X25519 ציבורי קבוע. הלקוח מושך את המפתח ומאמת אותו בעזרת ה-OTP ו-HMAC בשלב מוקדם, כדי להבטיח שאין מתקפת MITM.
   * **מפתח חתימה של השרת:** בדומה למפתח החתימה של הלקוח גם השרת גוזר מפתח ED25519 מהמפתח הפרטי שלו שאיתו הוא חותם על הודעות בין הלקוח לשרת להבטיח Integrity.

### **שלבי הפרוטוקול**

#### **א. רישום (Registration)**

1. **קבלת OTP (זמני):**

הלקוח מבקש OTP מהשרת:

| {  "type": "REQUEST\_OTP",  "client\_id": "<מספר הטלפון>" } |
| --- |

השרת מחזיר OTP בן 6 ספרות (למשל "483920") ותוקפו 120 שניות. השרת גם מציג את ה-OTP בערוץ המאובטח החד-פעמי ללקוח, שנראה כך:

| {  "status": "otp\_provided",  "otp": "<מספר בין 6 ספרות>"  } |
| --- |

אם הלקוח לא ישתמש ב-OTP בזמן, הוא יפוג, ובקשת שימוש ב-OTP לאחר תפוגתו תגרום לשרת להחזיר שגיאה, ויהיה צורך לבקש OTP חדש (כמו בווצאפ).

1. **משיכת המפתח הציבורי של השרת ואימותו:**

הלקוח מבקש את מפתח השרת:

| {  "type": "FETCH\_SERVER\_KEY",  "client\_id": "<מספר הטלפון>" } |
| --- |

השרת מחזיר את המפתח הציבורי שלו (X25519), מפתח החתימות הציבורי שלו (ED25519) ו-HMAC בעזרת ה-OTP. הלקוח מאמת את ה-HMAC. תשובת השרת תראה כך:

| {  "type":"SERVER\_KEY\_RESPONSE",  "server\_long\_term\_pub": "<מפתח ציבורי של השרת>",  "server\_signing\_pub": "<מפתח חתימה ציבורי של השרת>",  "mac\_hex":"<HMAC(OTP, כל ההודעה)>" } |
| --- |

אם ה-OTP פג, תוחזר שגיאה otp\_expired\_or\_invalid.

1. **חילופי מפתחות ברישום:** הלקוח יוצר זוג מפתחות ארעיים ושולח:

| {  "type":"REGISTER",  "client\_id":"<מספר טלפון>",  "client\_eph\_pub":"<מפתח ארעי ציבורי ב-hex>",  "mac\_hex":"<HMAC(OTP, כל ההודעה)>" } |
| --- |

השרת מוודא MAC ויוצר מפתח ארעי משלו, ומחזיר:

| {  "type":"REGISTER\_RESPONSE",  "server\_eph\_pub":"<hex>",  "mac\_hex":"<HMAC(OTP, התגובה)>" } |
| --- |

הלקוח מאמת MAC, מבצע ECDH בין המפתח שלו ושל השרת ומעביר התוצאה ב-HKDF עם ה-OTP כ-salt, וכך מפיק מפתח סימטרי (session\_key) להמשך הרישום. מהרגע הזה כל ההודעות בין הלקוח לשרת מוצפנות בעזרת המפתח המשותף הזה.  
 אם בשלב זה ה-OTP פקע, הלקוח ייאלץ לבקש OTP חדש ולנסות שוב.

1. **סיום הרישום והעלאת מפתחות:**   
   לאחר שהלקוח יצר מפתח זהות לטווח ארוך, עליו לייצר ולהעלות מספר מפתחות חד-פעמיים (pre-keys) לשרת. מספר המפתחות החד-פעמיים משתנה בהתאם לצורך. למשל, אם ידוע שמספר הלקוחות במערכת הוא עד 10, ניתן להעלות 10 מפתחות חד-פעמיים, כך שכל אחד מהם יוכל לשמש לקוח אחר לשליחת הודעה גם כשהלקוח לא מחובר.

כמובן, ניתן להעלות יותר מ-10 מפתחות כדי לתמוך במספר רב יותר של הודעות שמגיעות במקביל או כדי לאפשר לשלוח הודעות גם אם חלק מהמפתחות החד-פעמיים כבר נוצלו. במידה שהמפתחות החד-פעמיים נגמרים, הלקוח יכול לבצע פעולה נוספת להעלאת מפתחות חד-פעמיים חדשים לשרת.

הלקוח שולח לשרת, למשל:

| {  "type": "FINALIZE\_REGISTRATION",  "client\_id": "<מספר טלפון>",  "identity\_pub": "<hex של מפתח הזהות הציבורי>",  "pre\_keys": [  "<hex pre\_key1>",  "<hex pre\_key2>",  ...  "<hex pre\_key10>"  ] } |
| --- |

כך, בעתיד, כאשר לקוח אחר ירצה לשלוח הודעה ללקוח זה בעודו לא מחובר, השרת יוכל לספק לו אחד מהמפתחות החד-פעמיים שטרם נוצלו, להבטחת הצפנה מקצה-לקצה גם ללא נוכחות המקבל.

לאחר שלב זה, הרישום הושלם, השרת מסיר את ה-OTP, והלקוח והשרת יכולים כעת להשתמש ב-session\_key כדי להצפין את כל ההודעות הבאות. מאותו רגע, כל הבקשות (כגון FETCH\_KEYS, SEND\_MESSAGE, RETRIEVE\_MESSAGES וכו') ישלחו בהצפנת AES-256 בעזרת ה-session\_key ויאומתו.

#### **ב. התחלת תקשורת בין לקוחות**

כאשר לקוח A רוצה לשלוח הודעה ללקוח B (אשר עשוי להיות לא מקוון):

1. **משיכת מפתחות של B:**

A שולח לשרת:

| {  "type":"FETCH\_KEYS",  "client\_id":"A\_number",  "target\_id":"B\_number" } |
| --- |

השרת מחזיר את מפתח הזהות של B ואחד ה-pre-keys שלו יחד עם האינדקס של אותו pre-key.

1. **יצירת מפתח סימטרי עבור ההודעה:**  
   A יוצר מפתח לכל הודעה, מבצע מספר ECDH (מפתח זהות A עם זהות B, זהות A עם pre-key של B, מפתח ארעי A עם זהות B, ומפתח ארעי A עם pre-key של B). את כל הנתונים האלה מכניסים ל-HKDF ומפיקים מפתח AES-256-GCM.
2. **הצפנת ההודעה ושליחתה:**A מצפין עם AES-GCM ושולח לשרת:

| {  "type":"SEND\_MESSAGE",  "sender\_id":"A\_number",  "recipient\_id":"B\_number",  "A\_ephemeral\_pub":"<hex>",  "B\_one\_time\_key\_id":"<key fetched at FETCH\_KEYS(B)>",  "iv":"<hex>",  "ciphertext":"<hex>" } |
| --- |

השרת מאחסן הודעה זו עבור B.

#### 

#### **ג. קבלת הודעות (לקוח B לא מקוון)**

כאשר B חוזר ו/או מבקש את הודעותיו, הוא שולח:

| {  "type":"RETRIEVE\_MESSAGES",  "client\_id":"B\_number"  } |
| --- |

השרת מחזיר הודעות מוצפנות. B, שמחזיק את מפתח הזהות ואת ה-pre-key המתאים(ים), מבצע את אותם חילופי ECDH, מפיק את המפתח ומפענח עם AES-GCM את ההודעה. כך גם כשהיה לא מקוון, ההודעה נותרה סודית.

**ד. תהליך התחברות מחדש (Reconnection)**

#### **תיאור תהליך ההתחברות מחדש**

כאשר לקוח מבצע התחברות מחדש לאחר ניתוק, עליו לאמת את זהותו מול השרת. התהליך מתבסס על פרוטוקול של אתגר-מענה (Challenge-Response) באמצעות מפתח חתימה דיגיטלית (Ed25519) או HMAC המבוסס על מפתח הסשן שנשמר.

#### **שלבי תהליך ההתחברות מחדש**

1. **בקשת התחברות מחדש**:

הלקוח שולח לשרת בקשת RECONNECT הכוללת חתימה על מזהה הלקוח (client\_id) באמצעות מפתח החתימה הפרטי (Ed25519). במקרה שבו אין מפתח חתימה, הבקשה כוללת HMAC המבוסס על מפתח הסשן. החתימה הזו גם מבוססת על הזמן שבו נשלחה ההודעה על מנת למנוע Replay attack. פרק הזמן המותר הוא 100 שניות ולאחר מכן פג תוקפה של ההודעה. דוגמה לבקשה:

| {  "type": "RECONNECT",  "client\_id": "+972501234567",  "signature": "abcdef1234567890abcdef1234567890abcdef1234567890abcdef1234567890" } |
| --- |

1. **אתגר מהשרת**: השרת מוודא את תקינות החתימה או ה-HMAC על מזהה הלקוח. במידה והאימות עובר בהצלחה, השרת מייצר אתגר (challenge) שהוא מחרוזת רנדומלית (nonce) ומחזיר אותו ללקוח בהצפנה. דוגמה לתשובת השרת:

| {  "type": "CHALLENGE",  "challenge": "a1b2c3d4e5f6g7h8i9j0k1l2m3n4o5p6" } |
| --- |

1. **מענה הלקוח**:
   * הלקוח משתמש באתגר כדי לייצר תגובה חתומה (signed response) בעזרת מפתח החתימה הפרטי (Ed25519).
   * אם ללקוח אין מפתח חתימה, הוא משתמש במפתח הסשן ליצירת HMAC כתגובה לאתגר.

| {  "type": "CHALLENGE\_RESPONSE",  "client\_id": "+972501234567",  "challenge\_response": "abcdef1234567890abcdef1234567890abcdef1234567890abcdef1234567890" } |
| --- |

1. **אימות התגובה בצד השרת**:
   * השרת מאמת את תגובת הלקוח:
     + אם יש מפתח חתימה ציבורי (Ed25519), הוא מאמת את החתימה על האתגר.
     + אם אין מפתח חתימה ציבורי, הוא בודק את תקינות ה-HMAC באמצעות מפתח הסשן.
   * במקרה של הצלחה, הלקוח מאומת מחדש ומחובר לשרת.

תשובת השרת:

| {  "status": "ok" } |
| --- |

#### **אלגוריתמים בשימוש**

1. **Ed25519**:
   * משמש לחתימה על האתגר מצד הלקוח ואימותו בצד השרת.
   * אלגוריתם מהיר ובטוח לחתימות דיגיטליות.
2. **HMAC**:
   * משמש כחלופה לחתימה דיגיטלית כאשר אין מפתח חתימה.
   * מבוסס על SHA-256 ליצירת תגובה מבוססת מפתח סודי (מפתח הסשן).
3. **Nonce (אתגר רנדומלי)**:
   * נוצר על ידי השרת (os.urandom) כדי להבטיח שהאתגר ייחודי ולמנוע התקפות חוזרות (Replay Attacks).

#### **כיצד הלקוח הלא מקוון משיג זאת?**

הלקוח שומר את המידע הדרוש בקובץ ה-JSON שנוצר לאחר תהליך הרישום:

* + **מפתח החתימה הפרטי (Ed25519)**: משמש ליצירת תגובת החתימה.
  + **מפתח הסשן**: משמש ליצירת תגובת HMAC במקרה שאין מפתח חתימה.
  + **מפתח הסשן נשמר בצורה בטוחה (Hex)**: מאפשר שחזור החיבור.

**אופן הצפנת הודעות בין שרת ללקוח**:

1. **מפתח משותף**:
   * הלקוח והשרת הסכימו על מפתח ECDH באורך 256 ביטים לאחר שלב 3 בהרשמה (השלב שבו הושלם חילוף המפתחות).
   * המפתח המשותף ישמש להצפנת המידע באופן מאובטח.
2. **אלגוריתם הצפנה**:
   * האלגוריתם המשמש להצפנה הוא AES-GCM (מצב Galois/Counter Mode) באורך מפתח 256 ביטים.
   * AES-GCM מבטיח גם סודיות (confidentiality) וגם אימות (authentication) של ההודעות.
3. **מבנה ההודעה**:
   * לכל הודעה ייווסף nonce ייחודי (אורך: 12 בתים) הנוצר באופן רנדומלי.
   * ההודעה תוצפן בעזרת המפתח המשותף תוך שימוש ב-nonce.
   * בנוסף, ייחתם תוכן ההודעה, במידה ונעשה שימוש במפתח חתימה (Ed25519). זה יקרה לאחר שלב 4 ברישום, שבו הוחלפו גם מפתחות החתימה.
4. **תהליך שליחה**:
   * ההודעה המקורית (payload) תחולק לשניים:
     + תוכן ההודעה בפורמט JSON.
     + חתימה של תוכן ההודעה (במידה ומשתמשים בחתימות דיגיטליות).
   * ההודעה כולה תוצפן בעזרת AES-GCM.
   * לאחר ההצפנה, ההודעה תישלח בפורמט הבא:

| {  "iv": "nonce\_hex",  "ciphertext": "encrypted\_payload\_hex" } |
| --- |

1. **תהליך קבלה**:
   * המקבל יפענח את ההודעה בעזרת המפתח המשותף (ECDH) וה-nonce.
   * אם קיים מפתח חתימה, החתימה תיבדק מול תוכן ההודעה שנפענח.
   * במקרה של אימות מוצלח, תוכן ההודעה ייחשף לשימוש.
2. **התאמה למצב לא מחובר**:
   * במידה והלקוח או השרת לא מחוברים, הודעות יישמרו בתור זמני מוצפן, תוך שמירה על מפתחות ההצפנה לשימוש מאוחר יותר.

### **הסבר על אופן שמירת המפתחות על המחשב**

שמירת המפתחות מתבצעת בתוך קובץ JSON שמזוהה לפי מזהה הלקוח (client\_id). הקובץ מכיל מידע רגיש כמו מפתחות פרטיים, מפתחות ציבוריים, מפתח סשן, ואובייקטים נוספים הדרושים לתקשורת מאובטחת.

#### **מבנה קובץ ה-JSON**

* **registered**: מציין האם הלקוח כבר נרשם (ערך בוליאני).
* **otp**: קוד חד-פעמי שהתקבל מהשרת בתהליך ההרשמה (אם קיים).
* **server\_long\_term\_pub**: המפתח הציבורי הארוך טווח של השרת.
* **server\_signing\_public\_key**: המפתח הציבורי לחתימות של השרת.
* **identity\_private**: המפתח הפרטי של הלקוח (X25519).
* **client\_signing\_private\_key**: המפתח הפרטי של הלקוח לחתימות (Ed25519).
* **client\_signing\_public\_key**: המפתח הציבורי של הלקוח לחתימות (Ed25519).
* **pre\_keys\_private**: רשימת מפתחות פרטי-זמניים של הלקוח (X25519).
* **pre\_keys\_public**: רשימת מפתחות ציבוריים-זמניים של הלקוח (X25519).
* **known\_identities**: רשימת זהויות מוכרות של לקוחות אחרים (מפתחות ציבוריים).
* **contact\_pre\_keys**: מידע על מפתחות זמניים של לקוחות אחרים.
* **stored\_undecrypted\_msgs**: הודעות שלא הצליחו לפענח אותן.
* **session\_key**: מפתח הסשן הנגזר מתהליך Diffie-Hellman.

כל המידע נכתב לקובץ JSON ונשמר בתיקייה שבה נמצא הלקוח. הקובץ נוצר באופן אוטומטי בעת ההתחברות הראשונית או כאשר המידע נשמר בעזרת הפונקציה \_save\_state.  
יתרונות **שמירה מקומית בקובץ JSON**:

* **פשטות וקריאות**: פורמט JSON קל לקריאה ולשימוש.
* **שימוש במפתחות פרטיים**: המפתחות הפרטיים נשמרים בצורה גולמית (Hex), מה שמקל על שמירתם ושליפתם בתהליך העבודה.
* **ניידות**: הקובץ ניתן להעתקה למחשב אחר או גיבוי לשימוש חוזר.
* **ביצועים**: אחזור ושמירת מידע מקובץ JSON מהירים יחסית לפתרונות מבוססי בסיסי נתונים.

**שמירת מידע על מפתחות זמניים (pre-keys)**:

* **תאימות עם פרוטוקול Signal**: בחירת מבנה המאפשר שילוב עם תהליכים של חילופי מפתחות בטוחים.
* **תמיכה במשתמשים לא מקוונים**: שמירת pre-keys מייעלת שליחה ללקוחות שלא מחוברים בזמן אמת.

**שמירת מפתח סשן**:

* **אבטחה**: שמירת המפתח בפורמט Hex מאפשרת שמירה מאובטחת עם עיבוד מינימלי.
* **פשטות בתפעול מחדש**: מאפשר ללקוח לשחזר את הסשן לאחר ניתוק.

### **פרטים נוספים**

1. **בחירת האלגוריתמים:**

* Diffie-Hellman (ECDH): מאפשר חילופי מפתחות בטוחים מעל ערוץ שאינו מאובטח, מתאים במיוחד לפרוטוקולים מבוססי הצפנה מקצה לקצה. משמש כבסיס ליצירת מפתח סימטרי משותף, ללא צורך בהעברת מפתח סודי ישירות.
* AES-GCM: נבחר בשל ביצועיו המהירים והעובדה שהוא משלב גם סודיות (confidentiality) וגם אימות (authentication) בהצפנה אחת, מה שמקטין את המורכבות.

1. **אורך ועוצמת מפתחות:**
   * מפתחות זהות ו-pre-keys: X25519 (256 ביט).
   * הצפנה סימטרית: AES-256-GCM (256 ביט).
   * OTP: 6 ספרות (0-9), תקף למשך 120 שניות.
   * HMAC: SHA-256.
2. **מספר pre-keys:** הלקוח מעלה מספר pre-keys (למשל 10 בפרויקט הזה). כאשר הם מסתיימים, יכול להעלות נוספים (אך לא מימשתי את זה כי הנחנו שיש עד 10 משתמשים).
3. **קידוד בינארי:** כל מפתחות ומידע בינארי מקודדים ב-hex.
4. **טיפול בשגיאות:** השרת מחזיר {"status":"error","error":"<reason>"} במקרה של בעיה, כגון OTP שפג תוקף, לקוח לא רשום, או היעדר pre-keys.
5. **אחסון שרת ועמידות:** מומלץ להשתמש במסד נתונים כך שהשרת לא יאבד מידע בעת הפעלה מחדש.
6. **התמודדות עם התקפת אדם-באמצע (MITM):** השימוש ב-OTP וחישוב HMAC מבטיח שמפתח השרת ומפתחות ארעיים יאומתו כראוי בתחילת הרישום.
7. **א-סינכרוניות:** ה-pre-keys מאפשרים ל-A לשלוח הודעה ל-B גם אם B לא מחובר. השרת יאחסן את ההודעה, ו-B יוכל למשוך ולפענח אותה בהמשך.
8. **MAC:** בשלב ההרשמה נעשה שימוש ב-HMAC עם SHA-256 לאימות ושלמות, תוך שימוש ב-OTP כמפתח. מפתח סשן נגזר באמצעות ECDH ו-HKDF עם AES-256 להצפנת הודעות.

### **סיכום**

פרוטוקול זה מאפשר רישום מאובטח באמצעות OTP קצר ותוקף מוגבל (כדי למנוע שימוש ב-OTP ישן), חילופי מפתחות מתקדמים עם ECDH, שמירה על סודיות מלאה באמצעות AES-256-GCM, תמיכה במצב לא מקוון באמצעות pre-keys, ואפשרות לאישורי מסירה. הפרוטוקול עמיד בפני מתקפות MITM הודות ל-OTP ול-HMAC בשלבי הרישום הראשוניים.

**האם ניתן להבטיח זמינות (Availability) במערכת זו?**

במערכת כפי שהיא ממומשת בפרויקט זה, **לא ניתן להבטיח זמינות מלאה** (Availability). הסיבה העיקרית לכך היא שהשרת מהווה רכיב קריטי בתהליך התקשורת: הוא אחראי על שמירת מפתחות ציבוריים, מפתחות חד־פעמיים (pre-keys) והודעות מוצפנות בזמן שהלקוחות אינם מחוברים. אם השרת נופל או אינו זמין, המערכת כולה מאבדת את תפקודה.

#### **מה קורה כשאין זמינות לשרת?**

1. **אובדן הקשר והצורך ב־Login מחדש:**מכיוון שהשרת מאחסן את ה־session\_keys שנוצרו בין הלקוח לשרת, קריסת השרת גורמת לאובדן המפתחות הללו. כתוצאה מכך, הלקוח שינסה להתחבר שוב יידרש לעבור מחדש את כל תהליך ההרשמה – כולל קבלת OTP, חילופי מפתחות, ויצירת session\_key חדש.  
   במימוש הנוכחי, קליינט שינסה להתחבר לשרת לאחר קריסתו ייכשל, שכן המפתחות הדרושים (session\_key) אינם זמינים יותר והוא חושב שהוא מחובר.
2. **אובדן הודעות:**ההודעות המאוחסנות בשרת עבור לקוחות שאינם מחוברים יאבדו במקרה של כשל בשרת, מה שיגרום למידע שלא יגיע ליעדו.