Digital Whisper
Technologic Papers

ראשוני ומחוץ לחוק

מאת גדי אלכסנדרוביץ'

הקדמה

האם מספר ראשוני (או סתם מספר) יכול להיות לא חוקי? בהחלט, וזה הסיפור שאני רוצה לספר היום, שהוא גם משעשע, גם מערב מתמטיקה לא טריוויאלית וגם מערב קצת להטוטי מדעי המחשב. תחילת הסיפור היא עם דיסקי ה-DVD. דיסקים מסחריים מגיעים בדרך כלל עם הגנה כלשהי עליהם, וזו שרלוונטית לנו והייתה פופולרית בתחילת המאה ה-21 נקראת Content Scramble System, או בקיצור CSS.

הרעיון הוא פשוט: התוכן של ה-DVD מוצפן, והמפתח שבעזרתו ניתן לפענח את ההצפנה נמצא על איזור מאוד ספציפי של הדיסק. דיסקים לצריבה שנמכרים בחנויות מגיעים במצב שבו האיזור הזה לא ניתן לכתיבה, או שהצורבים הקנייניים פשוט לא מוכנים לצרוב בו, ולכן גם אם מעתיקים את תוכן הדיסק כמות שהוא, לא ניתן להעתיק את המפתח; זה הופך את ה-CSS להגנה כלשהי בפני שכפולי דיסקים נאיביים (שבהם פשוט מעתיקים את תוכן הדיסק בלי לפענח אותו קודם). בנוסף, CSS מספק יכולת לחלק את נגני ה-DVD ל"איזורי שידור" שונים - נגן שמתאים לאיזור א' לא יודע לקרוא את הסמא שנמצאת בדיסק שנועד לאיזור ב'. אפשר להגיד דברים טובים ורעים על השימוש ב-CSS או באופן כללי על השימוש ב-DRM (קיצור של Digital Rights Management, שבעברית נקרא "ניהול זכויות דיגיטלי" אבל לרוב מעדיפים לתרגם כ-"ניהול זכויות קנין", או בקיצור נז"ק) אבל אני מעדיף להימנע מהדיון הזה כאן. החשיבות של ה-CSS היא בכך שבאופן לא מפתיע בכלל, מהר מאוד צצו אנשים עלומי שם שמצאו דרך לפצח את ההצפנה שלו גם בלי לקרוא את המפתח, ומהר מאוד נכתבה תוכנה בשם DeCSS שעושה בדיוק את זה. באופן לא מפתיע, DeCSS נחשבת לא חוקית במספר מדינות (ושוב, לא ניכנס לדיון על אילו מדינות ומדוע...).

למידע נוסף אודות נושא ה-Content Scramble System, ניתן לקרוא את ארבעת המאמרים שפורסמו במגסרת סדרת המאמרים: "The DVD Content Scrambling System Explained".

ועכשיו (בשנת 2001) נכנס לתמונה בחור בשם פיל קרמודי. את התיאור שלו להתגלגלות הדברים אפשר לקרוא כאן, אבל הסיפור פשוט ומשעשע למדי לטעמי. פיל חשב על דרך שבה הוא יוכל לגרום לקוד של DeCSS להיות זמין באופן פומבי, בכל מקום בעולם, מבלי שהחוק יוכל לעשות משהו נגד זה. מכיוון שהוא אהב להשתעשע עם מספרים ראשוניים, צץ בראשו הרעיון הבא: נניח שנקודד את הקוד של התוכנית באמצעות מספר (בהמשך אסביר בפירוט איך), אבל לא "סתם" מספר אלא מספר שיהיה מעניין - למשל, מספר שיהיה אחד מהמספרים הראשוניים הגדולים ביותר שנתגלו אי פעם - אז המספר הזה, שהוא בעל תכונה "מעניינת" בפני עצמו, בלי קשר לתוכנית שהוא מקודד, יוכנס כלאחר כבוד אל רשימת "המספרים הראשוניים הגדולים ביותר שנתגלו אי פעם" ושם ישכון לבטח באופן חוקי לחלוטין, למרות שהוא מקודד תוכנית לא חוקית. אמר פיל, ועשה. ליתר דיוק, הוא מצא שני מספרים ראשוניים, אחד שמקודד את הקוד של DeCSS ולא היה מספיק גדול כדי להיכנס לרשימה, ושני שמקודד כבר את קוד המכונה הרלוונטי של התוכנית והיה גדול דיו להיכנס לרשימה. מאז כמובן שפותחו שיטות הגנה חדשות על דיסקים, וגם שיטות אחרות להיפטר מ-CSS, אבל המקרה הזה נחקק בזכרון בתור דוגמה בולטת ל"מספר לא חוקי".

הכל מספרים

נתחיל מההתחלה - איך תוכנית מחשב יכולה להיות מספר? לשם כך צריך לזכור איך תוכניות מחשב מיוצגות בתוך המחשב: תוכנית מחשב, כמו כל קובץ אחר במחשב, היא בסך הכל רצף של אפסים ואחדים - "ביטים". מטעמים היסטוריים, נהוג לקבץ את הרצפים הללו לקבוצות של שמונה ביטים שנקראות "בייטים". אפשר לחשוב על כל בייט בפני עצמו בתור ייצוג של מספר בין 0 ל-255, בבסיס בינארי. למשל, הבייט 00001011 מייצג את המספר 11 = 8 + 2 + 1. כפי שאתם מנחשים, קצת מסורבל לייצג בייטים בבסיס בינארי, אז נהוג לייצג אותם בבסיס 16 - "בסיס הקסדצימלי" - שהיתרון שבו הוא שכל ספרה הקסדצימלית ניתנת לחישוב מיידי מארבעה ביטים רצופים. הבייט שלעיל, בבסיס הקסדצימלי, הוא 0B.

האופן שבו רצפים שונים של בייטים מפורשים בתור קבצים הוא עניין שהוא לחלוטין תלוי הקשר - אותו רצף של בייטים יכול להיראות בעל משמעות רבה לתוכנה אחת, וחסר משמעות לתוכנה אחרת. כמה פעמים ניסתם לפתוח קובץ MP3 על ידי עורך תמונות כדי לראות מה יקרה? (לרוב עורך התמונות יצעק שהוא לא מבין כלום; אבל עורכי טקסט לפעמים יצליחו לפתוח את הקובץ ולהציג משהו שנראה כמו ג'בריש). יש ייצוג סטנדרטי פשוט יחסית לטקסט שנקרא ASCII - בייצוג זה כל בייט מייצג תו מסויים, כאשר אותיות לועזיות מתחילות ב-65 ונגמרות ב-122 (עם עוד כמה סימנים באמצע). כפי שקל להבין, אפשר לייצג בשיטה זו רק 255 תווים ולכן בשביל להציג אלפביתים רבים ושונים משתמשים בשיטות קידוד מחוכמות יותר שלא אכנס אליהן כאן.

בואו נסתכל על קובץ טקסט שמכיל את הפסקה הבאה, פסקת הפתיחה של "סוס הים והנגר" של לואיס קארול:

```
The sun was shining on the sea,
Shining with all his might:
He did his very best to make
The billows smooth and bright-
And this was odd, because it was
The middle of the night.
```

אם נפתח את הקובץ עם עורך שיודע לקרוא את הערכים המספריים של הבייטים שלו (עורך כזה נקרא Hex Editor), תחילת הקובץ תיראה כך: 73 20 65 68 54. כאן ה-20 הוא תו ה-ASCII שמייצג רווח, ושאר התווים מייצגים אותיות. זכרו שלא מדובר על ייצוג מספרי אלא על ייצוג הקסדצימלי: כדי לתרגם אותו למספר, הכפילו את הספרה הראשונה ב-16 וחברו לשניה. כלומר, 54 הוא בעצם המספר שמונים וארבע, בעוד ש-68 הוא המספר מאה וארבע. אם נכתוב את הספרות הללו בזו אחר זו, נקבל את 5468 שהוא כבר מספר גדול בהרבה: 8 עוד 6 כפול 16, ועוד 4 כפול 16 בריבוע, ועוד 5 כפול 16 בשלישית, כלומר המספר 21608. ואם ניקח את כל הקובץ של סוס הים והנגר ונחשוב עליו כמספר באופן שתיארתי, נקבל מספר די מפלצתי, בן למעלה מארבעה-מאות ספרות:

```
153058130896577434476956204244500915089507607276649591136639967347927
634085752721215073631336899232176479844024041004598713736540252717051
491125294738203137728828190962600734620137160806953118745283821570893
504447012198054017464958848902345608456153112106477819772373230961058
745461657368290816729833943072852706505548140495454380685559992935237
417096409099380602636039274560979892769401971212873564491309951693079
602940848141358
```

כלומר, גם קבצי טקסט קטנים יחסית יהפכו למספרים מפלצתיים בגודלם, אז תוכניות מחשב, גם קטנות, יתפסו עוד יותר מקום. אז מה עושים? ראשית, זה לא נורא במיוחד אם המספר הוא גדול, כי המטרה של קרמודי הייתה ליצור מספר ראשוני גדול; מצד שני, אם המספר הוא גדול מדי, אי אפשר יהיה לבדוק שהוא באמת ראשוני ולכן הוא יהיה חסר ערך. לכן עדיף להתחיל ממספר קטן ככל האפשר ומקסימום להגדיל אותו בצורה מלאכותית (בהמשך נראה איך).

הדרך שבה אפשר לבצע הקטנה שכזו כאן היא באמצעות כיווץ. כיווץ של קבצים הוא נושא רחב ומרתק לכשעצמו ולא אכנס אליו כעת, אבל זה היה גם בדיוק מה שקרמודי השתמש בו. הוא לקח את קוד המקור וכיווץ אותו בעזרת gzip (תוכנת כיווץ חינמית שנפוצה בלינוקס ומערכות דומות) ואת התוצאה הוא המיר למספר. אם נפעיל את אותה תוכנה על השיר של סוס הים והנגר, נקבל את המספר הבא:

```
355941790546066961506342149389258720867004703259858151842931171763230
166556747517269253437195257820204425589941114036432425909398860764878
800593067151962049148381358503674861857798181563071988203782124545041
```

```
443302544039699025354442584168547347939973801595105557917155490117291
567168064926333899651600682718875560672206876979350834558577630451895
5896909747557749279948 8
```

שהוא בבירור קטן יותר, אם כי לא בצורה יותר מדי משמעותית. בתוכנית מחשב, שבה יש הרבה יותר יתירות (שמות של משתנים שחוזרים על עצמם וכדומה) לרוב כיווץ הוא אפקטיבי יותר.

בואו ניגש לאקשן. הנה פונקציה בשפת רובי שמקבלת כקלט מספר, שנתון כמחרוזת של המספר בייצוג עשרוני שאולי מכילה רווחים (כי הרבה פעמים כשכותבים מספרים גדולים משתילים פנימה רווחים וירידות שורה כדי שיהיה קריא), ממירה אותו לקובץ gz, פותחת אותו ומדפיסה את תוכנו:

```
def read(n)
hex_string = n.delete(" \n").to_i.to_s(16)
hex_string = "0" + hex_string if hex_string.length % 2 != 0
program_gzip = [hex_string].pack('H*')
File.open("program.gz", "w"){|f| f.write(program_gzip)}
puts `gunzip -c program.gz`
end
```

מה קורה כאן? בשורה הראשונה מנקים רווחים וירידות שורה מהמספר וממירים אותו לערך מספרי ומייד לאחר מכן ממירים אותו שוב למחרוזת, הפעם כזו שמייצג את המספר בבסיס 16. מכיוון שרובי לא תוסיף 0 מוביל אם אמור להיות כזה, השורה הבאה מוסיפה אותו בעצמה (הסבירו לעצמכם למה צריך לוודא את זה). השורה הבאה היא הקסם השחור הגדול ביותר כאן - הפקודה pack (שמסיבה לא ברורה חייבת לפעול על מערך) ממירה מחרוזת שמייצגת ערך כלשהו לערך האמיתי שלו, בהתאם לסוג המידע שבמחרוזת. כאן H אומר שהערך הוא מספר בבסיס הקסדצימלי והכוכב אומר לקחת את כל הספרות שבמחרוזת. השורה הבאה כותבת את התוכן לתוך קובץ ה-gz, וזו שאחריה פותחת את הקובץ ומוציאה את הפלט. זה הכל. אתם מוזמנים להריץ את הקוד על המספר שנתתי למעלה (לשם כך תצטרכו לעבוד במערכת הפעלה שבה מותקן gunzip בצורה שבה הוא יכול להיות מופעל כך משורת הפקודה, ואני מנחש שאצל רובכם זה לא המצב...) - תקבלו בחזרה את סוס הים והנגר. ואם תעשו את זה על הראשוני של קרמודי, שאפשר לראות כאן, תקבלו את הקוד של DeCSS.

קידוד תוכניות למספרים הוא רק חלק מהסיפור. אחרי הכל, זה לא רעיון חדש; שימוש מבריק ברעיון הזה נעשה כבר על ידי קורט גדל בהוכחת משפטי אי השלמות שלו - גם שם הרעיון היה לקודד תוכניות מחשב באמצעות מספרים טבעיים, ומכיוון שתוכניות המחשב הללו יודעים לפעול על מספרים טבעיים, הן בעצם אמרו משהו על עצמן. גדל כתב תוכנית מחשב שידעה לקחת מספר כזה, לפענח את הייצוג שלו חזרה לתוכנית מחשב ואז לעשות בה דברים מגניבים. רק שקורט גדל עשה את כל אלו לפני שבכלל היו מחשבים או תוכנות מחשב, ומה שאני קורא לו "תוכנת מחשב" אצלו היה בכלל פסוקים בלוגיקה מסויימת (אבל לדעתי מה שהוא עשה איתם היה בדיוק תכנות מהסוג שתיארתי לעיל). אלן טיורינג לקח את

הרעיונות של גדל ופירמל אותם עבור מודל של מחשב שכבר אי אפשר היה לטעות בו; ומטיורינג עד קרמודי עברו חמישים שנים של מדעי מחשב שבהן הקשר הזה היה ברור. מה שהפך את התעלול של קרמודי למוצלח היה העובדה שהוא לא מצא "סתם" מספר שמייצג את DeCSS, אלא מספר ראשוני גדול מאוד שעושה את זה, ולכן כדאי לומר כמה מילים על ראשוניים והחיפוש אחריהם.

מספר ראשוני, למי שלא מכיר, הוא מספר שמתחלק רק ב-1 ובעצמו. למשל 7, או 97. ראשוניים הם מעניינים ממגוון סיבות, שרובן נובעות מכך שכל מספר טבעי ניתן לכתוב בצורה **יחידה** (עד כדי משהו שלא ניכנס אליו) כמכפלה של ראשוניים, והרבה ניתוחים של מספרים טבעיים מתבססים על לקחת את הייצוג הזה כמכפלה ולעשות איתו דברים, תוך שימוש בכך שראשוניים מקיימים שלל תכונות נחמדות שסתם מספרים לא בהכרח מקיימים. דוגמה חשובה לכך היא שיטת ההצפנה RSA: בשיטה זו מגרילים מספר גדול מאוד (מאות ספרות) n שהוא מכפלה של שני ראשוניים: $n=pq$. מתוך n אפשר לייצר שני מספרים, שבאמצעות אחד מהם אפשר להצפין הודעות ובעזרת השני אפשר לפענח אותן; רק שהדרך היחידה שאנו מכירים כיום למצוא את מפתח הפענוח בהינתן מפתח ההצפנה ו-n היא לדעת את הפירוק של n לאותו זוג ראשוניים p,q. מי שסתם נותנים לו את n ביד (וזה מה שנותנים לכל מי שרוצה להצפין) לא יודע לחשב כלום בסגנון הזה, ולמצוא את הראשוניים שמרכיבים את n זה עניין קשה חישובית. אני חושב שזה עניין מרתק למדי, האופן שבו ההיכרות עם הפירוק של n לגורמים משנה את כל התמונה.

כעת, כדי לייצר את n מלכתחילה לא סביר סתם להגריל מספר גדול ולראות אם הוא מתפרק למכפלה של זוג ראשוניים - גם כי לא בהכרח נצליח להגריל מספר כזה, וגם כי כאמור - אנחנו לא באמת יודעים לפרק את n ואם נצליח לעשות את זה, גם אחרים כנראה יצליחו אז מה השגנו? לכן מה שעושים הוא להגריל מראש את p,q ולכפול אותם. זו דוגמה לצורך ה**מעשי** שלנו להיות מסוגלים למצוא מספרים ראשוניים גדולים. למרות זאת, זה עדיין לא מסביר למה אנחנו מחפשים גם מספרים ראשוניים **ממש גדולים**, גדולים מכל מספר ראשוני שהכרנו עד אליהם. התשובה היא - אין סיבה, אנחנו עושים את זה בעיקר מתוך סקרנות אינטלקטואלית ומתוך רצון לשפר עוד ועוד את רמת האלגוריתמים-מחפשי-הראשוניים שלנו (אותם אלגוריתמים שבסופו של דבר גם עוזרים למערכות כמו RSA להתקיים).

איך בדרך כלל מוצאים ראשוניים גדולים? אפשר היה לחשוב שיהיו אלגוריתמים מתוחכמים שיודעים בדיוק איך לחפש ראשוניים, אבל כרגע אין ממש כאלו. מה שעושים הוא להגריל מספרים גדולים, ולבדוק אם התמזל מזלנו וקיבלנו ראשוני. אפשר גם להגריל מספר גדול ולהתחיל לעבור סדרתית על המספרים שאחריו - משפט מתמטי בשם "משפט המספרים הראשוניים" מבטיח לנו שבשיטה הזו לא יקח **יותר מדי** זמן עד שניתקל בראשוני (כמובן שאפשר לעשות אופטימיזציות כמו לבדוק רק מספרים אי זוגיים). אם כן, לב הבעיה הוא פשוט ב**בדיקה** אם מספר הוא ראשוני או לא. איך עושים את זה?
השלב הראשון שבדרך כלל עושים הוא לנסות ולחלק את המספר הנבדק במספרים ראשוניים קטנים יחסית (מספיק לנסות לחלק בראשוניים כי אם מספר מתחלק על ידי מישהו, הוא מתחלק גם על ידי

ראשוני שמחלק את ה"מישהו"). רוב המספרים שאינם ראשוניים יפלו כבר בשלב הזה. בשלב הבא אפשר להפעיל אלגוריתם הסתברותי דוגמת אלגוריתם מילר-רבין שכבר הזכרתי בעבר. אם המספר הנבדק אינו ראשוני, לאלגוריתם הזה יש סיכוי גבוה מאוד לגלות זאת. לצרכים פרקטיים כמו RSA זה מספיק בהחלט, אבל מנקודת מבט מתמטית קפדנית **זה לא מספיק** כדי להכניס את המספר לאף רשימה, כי אנחנו לא **בטוחים** שהוא ראשוני, רק שאלגוריתם בדיקת הראשוניות האקראי לא הצליח להפיל אותו.

אז השלב הבא הוא הפעלה של אלגוריתם שהוא איטי הרבה יותר מאשר מילר-רבין, אבל אם הוא אומר שמספר הוא ראשוני, אז מובטח לנו שזה אכן המצב. יש אלגוריתם מפורסם כזה: אלגוריתם AKS, רק שזה אלגוריתם מאוד איטי יחסית, ובזמנו של פיל קרמודי הוא בכלל לא היה קיים. אז קרמודי השתמש באלגוריתם מהיר יותר, שהוא בעל התכונה הבאה: אמנם, לא בטוח שהוא יסיים את ריצתו בזמן סביר על כל מספר, אבל אם הוא עוצר על מספר ואומר שהמספר ראשוני, אז **מובטח** לנו שהמספר אכן ראשוני. האופן שבו האלגוריתם עושה את זה הוא באמצעות אובייקט מתמטי לא טריוויאלי שנקרא "עקום אליפטי", ומכאן שם האלגוריתם: Elliptic Curve Primality Prover, או ECPP. זה אלגוריתם מגניב למדי וגם לא מסובך יותר מדי, בהינתן כמה שכל הנושאים של עקומים אליפטיים הם לא טריוויאליים, ואני מקווה לכתוב עליו פוסט מתישהו.

עכשיו, כדאי להעיר שקצת שיקרתי בתחילת הפוסט כדי למשוך קוראים. הרשימה שאליה הראשוני של קרמודי הוכנס היא "רשימת המספרים הראשוניים הגדולים ביותר שהראשוניות שלהם הוכחה בעזרת ECPP". אפשר לראות את הרשימה הזו כאן; המספר של קרמודי כבר לא שם, כמובן, בכל זאת עברו עשר שנים מאז. הסיבה לכך שיש הגיון בלדבר על רשימה כמו זו ולא פשוט על רשימת "הראשוניים הגדולים ביותר שהוכחו, נקודה" היא שעבור ראשוניים **מצורה מסויימת** יש אלגוריתמים יעילים יותר לבדיקת ראשוניות. הדוגמה שאני מכיר והיא כנראה החשובה ביותר היא מבחן לוקאס-לאמר למספרי מרסן. מספר מרסן הוא מספר מהצורה $2^p - 1$ כאשר p הוא ראשוני; מבחן לוקאס-לאמר מאפשר לבדוק בצורה יחסית יעילה האם מספר כזה הוא ראשוני, ובשל כך הראשוניים הגדולים ביותר שראשוניותם הוכחה הם מספרי מרסן. זה כמובן יפה מאוד לכשעצמו אבל לא ממש יעיל עבור אלגוריתמים כמו RSA (שבהם חשוב שהראשוניים שמגרילים יהיו מספרים לא ידועים במיוחד; מספרי מרסן ראשוניים אין יותר מדי) ולכן ראוי לתת מקום של כבוד גם לאלגוריתמים "כלליים" לבדיקת ראשוניות, כמו ECPP.

עכשיו אפשר להשלים את התיאור של מה שקרמודי עשה בפועל. ראשית, הוא לקח את המספר שמייצג את קובץ ה-gz של התוכנית; נסמן אותו ב-x. כעת, אין סיבה מיוחדת להניח ש-x יהיה ראשוני, ובטח לא שיהיה בסדרי הגודל שקרמודי מעוניין בהם. הנקודה היא שאפשר לשנות את המספר הזה ולקבל קובץ gz שהוא אמנם שונה אבל שקול, מבחינה זו שההבדל היחיד הוא שיש בסוף שלו עוד קצת ג'יבריש שממנו gz

יודע להתעלם. למי שזה מרגיש להם כמו "רמאות" - ובכן, ראשית כל זכרו שאנחנו מדברים כאן על משהו מעשי לחלוטין, ולכן השאלה החשובה היחידה היא האם זה עובד (וזה עובד, עם הקוד שנתתי לעיל). שנית, גם בתחומים תיאורטיים לגמרי כמו תורת החישוביות יש חשיבות ליכולת להוסיף ג'יבריש למשהו מבלי לשנות את משמעותו. דוגמה נפלאה לכך שהראיתי בעבר בבלוג היא משפט לדנר בתורת הסיבוכיות, שאומר (בערך) שלא ניתן להכריע את בעיית $\mathrm{P=NP}$ באמצעות שיטת הלכסון; שם לב העניין היה לבנות שפה בעל תכונות מוזרות, שנבעו מהוספה של ג'יבריש בכמות מאוד מאוד מסויימת (הפונקציה שמתארת את הכמות הזו הייתה הדבר המורכב ביותר בהוכחה) למילים "חוקיות" של שפה סטנדרטית כלשהי. בקיצור - ג'יבריש זה טוב.

על ידי **הארכה** של קובץ ה-gz באמצעות הוספת בייט לסופו מכפילים את הערך של המספר שמייצג את התוכנית כולה פי 256 (כי אנחנו מוסיפים שתי ספרות הקסדצימליות; אם היינו מוסיפים שתי ספרות עשרוניות בתחילת מספר קיים היינו מכפילים אותו פי 100) ואפשר לשים ערך כרצוננו בבייט האחרון כדי להגדיל עוד טיפה את ערכה המספרי של התוכנית. במילים אחרות, אנחנו מסוגלים לקודד את התוכנית על ידי המספר $256X + a$, לכל $0 \leq a \leq 255$. זה נתן לקרמודי הרבה מספרים לבדוק את ראשוניותם. בסופו של דבר זה לא הצליח, אז הוא פשוט הוסיף עוד בייט ובדק את המספרים מהצורה $256^2X + a$. איך הוא בדק אותם? באמצעות תוכנת קוד פתוח שנקראת OpenPFGW; על פי התיאור שלו, היא ראשית כל מבצעת בדיקה נאיבית של חלוקה של המספר שאותו בודקים בהמון ראשוניים קטנים, ואז מריצה מבחן ראשוניות אקראי כלשהו (אני מנחש שמילר-רבין, אבל לא בטוח; יש עוד אלגוריתמים נאים). בצורה הזו קרמודי סינן את המספרים הבעייתיים והגיע למספר שבודאות מאוד מאוד גדולה אכן היה ראשוני; הרצה של ECPP עליו (במימוש של תוכנה בשם Titanix) סגרה את הסיפור.

המסקנה מכל הסיפור הזה היא כפולה. ראשית, שבכל מתכנת מצוי מתמטיקאי פסיכי שרק מחכה להזדמנות לתקוף (זה כמובן שקר); ושנית, שלפעמים נושאים מגניבים במתמטיקה הופכים למגניבים הרבה יותר כאשר מנצלים אותם כדי לעשות דווקא ל-DRM.

המאמר פורסם לראשונה בבלוג "לא מדויק" של גדי, בקישור הבא:
http://www.gadial.net/2012/12/07/illegal_prime/

על המחבר

גדי אלכסנדרוביץ'. יליד 1982 בעל תואר ראשון במתמטיקה ותואר שלישי במדעי המחשב מהטכניון.
כותב בבלוג "לא מדויק" - http://www.gadial.net.