ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ ΚΑΙ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

Κρυπτογραφία και Πολυπλοκότητα

Factoring $N = p^r q$ for large r

Μελισσάρης Παπανικολάου Νικόλαος
Α.Μ. 09104221
nikolasm@gmail.com

Περίληψη

Σε αυτή την εργασία, οι Boneh, Durfee και Howgrave παρουσιάζουν έναν αλγόριθμο παραγοντοποίησης ακεραίων N της μορφής $N = p^r q$ για μεγάλο r. Όταν $r \approx \log(p)$ ο αλγόριθμος τρέχει σε πολυωνυμικό χρόνο, log(N). Έτσι, έχουμε μια νέα κλάση ακεραίων, οι οποίοι μπορούν να παραγοντοποιηθούν αποτελεσματικά. Όταν $r \approx \sqrt{\log(p)}$ ο αλγόριθμος ασυμπτωτικά γρηγορότερος από την μέθοδο ελλειπτικών καμπυλών (ECM) του Lenstra. Τα αποτελέσματα συνιστούν πως η χρήση αριθμών της μορφής $N = p^r q$ πρέπει να χρησιμοποιούνται με προσοχή, ειδικά όταν το r είναι μεγαλύτερο από $\sqrt{\log(p)}$.

1. Εισαγωγή

Τα τελευταία χρόνια αριθμοί της μορφής $N = p^r q$ χρησιμοποιήθηκαν αρκετά στη κρυπτογραφία. Για παράδειγμα οι Fujioke, Okamoto και Miyagutchi στην εργασία τους "ESIGN: an efficient digital signature implementation for smartcards", χρησημοποιούν $N = p^2 q$ για ένα σχήμα ηλεκτρονικού χρήματος. Ο Takagi, παρατήρησε πως η RSA αποκρυπτογράφηση μπορεί να γίνει πολύ πιο γρήγορα, χρησιμοποιώντας $N = p^r q$. Σε αυτές τις εφαρμογές τα p, q είναι περίπου ίδιου μεγέθους και η ασφάλειά τους στηρίζεται στη δυσκολία παραγοντοποίησης του N.

Οι συγραφείς, δείχνουν πως η χρήση αριθμών αυτής της μορφής πρέπει να γίνεται με προσοχή. Συγκεκριμένα, έστω πως οι p,q είναι πρώτοι, μεγέθους 512 bits ο καθένας. Δείχνουν πως η παραγοντοποίηση του $N = p^r q$ γίνεται πιο εύκολη όσο το r μεγαλώνει. Για παράδειγμα όταν το r είναι τάξης του log(p) ο αλγόριθμός τους παραγοντοποιεί το N σε πολυωνυμικό χρόνο ενώ όταν το r είναι της τάξης του $\sqrt{\log(p)}$ ο αλγόριθμός τους παραγοντοποιεί το N πιο γρήγορα από την (τότε) καλύτερη μέθοδο ECM (ελλειπτικών καμπυλών). Τα αποτελέσματα τους δείχνουν πως οι $N = p^r q$ με μεγάλο r είναι ακατάλληλοι για κρυπτογραφική χρήση.

Έστω p,q πρώτοι k bits και $N = p^r q$. Όταν r = k^{ε}, ο αλγόριθμός τους (ασυμπτωτικά) τρέχει σε χρόνο T(k) = $2^{(k^{1-\varepsilon})+O(\log(k))}$ και χρειάζεται "πολυωνυμικό χώρο" (σε log(N)). Έτσι, όταν ε = 1, το N είναι μήκους περίπου k^2bits ο αλγόριθμος παραγοντοποιεί το N σε πολυωνυμικό χρόνο. Για ε = $\frac{1}{2}$ ο αλγόριθμος είναι είναι αποδοτικότερος του ECM. Στη σύγκριση μεταξύ του αλγορίθμου των συγγραφέων και του ECM είναι ενδιαφέρον να μελετήσουμε την περίπτωση ε ≈ $\frac{1}{2}$ δηλαδή r ≈ $\sqrt{\log(\mathrm{p})}$. Δυστυχώς, το N γίνεται ταχύτατα πολύ μεγάλο για να το χειριστεί κανείς οπότε οι πειραματισμοί έγιναν με μικρά p. Το "μεγαλύτερο" πείραμα περιλάμβανε πρώτους p,q των 96 bit και r = 9. Σε αυτή την περίπτωση το N είναι 960 bits. Τα αποτελέσματα δείχνουν πως ενώ ο αλγόριθμος είναι ασυμπτωτικά καλύτερος, ο ECM αποδίδει καλύτερα για τέτοιους μικρούς παράγοντες. Τα πειραματικά αποτελέσματα παρουσιάζονται στην παράγραφο 4.

Το πρόβλημα της παραγοντοποίησης του N = $p^r q$ συνδέεται με την παραγοντοποίηση του N = $p^2 q$. Αποτελέσματα των Peralta, Okamoto αλλά και των Pollard, Bleichenbacher δείχνουν πως ο ECM μπορεί να γίνει αποδοτικότερος όταν εφαρμόζεται σε N = $p^2 q$. Τα αποτελέσματα των συγγραφέων για N = $p^2 q$ περιγράφονται στην παράγραφο 6. Η τεχνική τους χρησιμοποιεί αποτελέσματα της θεωρίας πλεγμάτων της οποίας μια πολύ βασική εισαγωγή γίνεται στην επόμενη παράγραφο. Στην παράγραφο 3 περιγράφεται ο αλγόριθμος παραγοντοποίησης ακεραίων της μορφής N = $p^r q$ για μεγάλο r. Στην παράγραφο 4 αναλύεται η εφαρμογή του αλγόριθμου και δίνονται παραδείγματα ολοκληρωμένων παραγοντοποιήσεων. Στην παράγραφο 5 συγκρίνεται η προσέγγιση των συγγραφέων με υπάρχουσες μεθόδους και περιγράφονται κλάσεις ακεραίων για τους οποίους ο αλγόριθμος είναι η βέλτιστη μέθοδος παραγοντοποίησης.

2. Πλέγματα (Lattices)

Έστω $u_1, \dots, u_d \in \mathbb{Z}^n$ γραμμικώς ανεξάρτητα διανύσματα με $d \leq n$. Ένα lattice L που παράγεται από τα $\langle u_1, \dots, u_d \rangle$ είναι το σύνολο όλων των γραμμικών συνδυασμών των $u_1, \dots, u_d$. Λέμε ότι το lattice είναι πλήρους βαθμού αν $d = n$. Με ${u_1}^*, \dots, {u_d}^*$ συμβολίζουμε τα διανύσματα που παίρνουμε εφαρμόζοντας την ορθοκανονικοποίηση Gram-Schmidt στα διανύσματα $u_1, \dots, u_d$. Ορίζουμε την ορίζουσα του lattice L ως:

$$det(L) := \prod_{i=1}^{d} \|u_i^*\|$$

Αν το L είναι πλήρους βαθμού τότε η ορίζουσα του L είναι ίση με την ορίζουσα του $d \times d$ πίνακα που οι σειρές του είναι τα διανύσματα βάσης $u_1, \dots, u_d$.
Λήμμα 2.1 (LLL) Έστω lattice L που παράγεται από τα $\langle u_1, \dots, u_d \rangle$. Τότε ο αλγόριθμος LLL θα παράγει ένα διάνυσμα u που θα ικανοποιεί:

$$\|u\| \leq 2^{d/2} det(L)^{1/d}$$

Ο αλγόριθμος τρέχει σε χρόνο τετάρτου βαθμού στο μέγεθος της εισόδου.

3. Παραγοντοποιώντας $N = p^r q$

Ο στόχος σε αυτή την παράγραφο είναι να αναπτυχθεί ένας αλγόριθμος που θα παραγοντοποιεί ακεραίους της μορφής $N = p^r q$. Χρησιμοποιείται η εξής σημειολογία:

$$\exp(n) = 2^n$$

Ομοίως, σε όλη την εργασία οι λογάριθμοι που αναφέρονται είναι με βάση το 2.

Θεώρημα 3.1 Έστω $N = p^r q$ με $q < p^c$. Ο παράγοντας p μπορεί να βρεθεί δοσμένων N, r και c από έναν αλγόριθμο με χρόνο εκτέλεσης:

$$exp\left(\frac{c+1}{r+c}\right) \cdot logp \cdot O(\gamma)$$

όπου γ είναι ο χρόνος που χρειάζεται να τρέξει ο LLL σε ένα lattice διάστασης $O(r^2)$ με εισόδους μεγέθους O(rlogN). Ο αλγόριθμος είναι ντετερμινιστικός και τρέχει σε πολυωνυμικό χώρο. Ο παράγοντας γ είναι πολυωνυμικός σε logN. Αξίζει να δούμε μερικά παραδείγματα χρησιμοποιώντας αυτό το θεώρημα. Για λόγους απλότητας, θεωρείται $c = 1$ έτσι ώστε p,q να έχουν περίπου το ίδιο μέγεθος. Θεωρώντας το c ως οποιαδήποτε μικρή σταθερά, δίνει τα παρόμοια αποτελέσματα.

- Όταν c $= 1$ έχουμε $\frac{c+1}{r+c} = O\left(\frac{1}{r}\right)$. Άρα όσο μεγαλύτερο είναι το r, τόσο ευκολότερη γίνεται η παραγοντοποίηση. Όταν r $=$ elogp για σταθερό ε, ο αλγόριθμος τρέχει σε πολυωνυμικό χρόνο.
- Όταν $r \approx log^{1/2}p$ τότε ο χρόνος εκτέλεσης είναι exp$(log^{1/2}p)$. Έτσι, ο χρόνος εκτέλεσης είναι ελαφρώς καλύτερος από τον ECM.
- Για μικρό r ο αλγόριθμος τρέχει σε πολυωνυμικό χρόνο.
- Όταν το c είναι μεγάλο (τάξεως r) ο αλγόριθμος γίνεται εκθετικού χρόνου. Έτσι, ο αλγόριθμος είναι αποτελεσματικότερος όταν p,q είναι ίδιου μεγέθους. Όλες οι κρυπτογραφικές εφαρμογές του N $= p^r q$ που είναι γνωστές κάνουν χρήση p,q ίδιου μεγέθους.

Η απόδειξη του θεωρήματος 3.1 βασίζεται στη δουλειά των Coppersmith kai Howgrave – Graham. Η βασική ιδέα είναι να μαντέψουμε έναν μικρό αριθμό των πιο σημαντικών bits του p και να παραγοντοποιήσουμε χρησιμοποιώντας αυτή την υπόθεση. Όπως φαίνεται, μπορούμε να δείξουμε πως όσο μεγαλύτερο είναι το r, τόσο λιγότερα bits του p χρειάζεται να μαντέψουμε.

Στην εργασία του "Small solutions to polynomial equations and low exponent RSA vulnerabilities" ο Coppersmith δείχνει πως αν δίνονται τα μισά σημαντικότερα bits του p, το $N = pq$ μπορεί να παραγοντοποιηθεί σε πολυωνυμικό χρόνο αν p,q είναι ίδιου μεγέθους. Για να γίνει αυτό δίνει μια κομψή απόδειξη εύρεσης μικρών ριζών πολυωνύμων δύο μεταβλητών στους ακεραίους. Παραδόξως, το θεώρημα 3.1 δεν ακολουθεί τα αποτελέσματα του Coppersmith. Το θεώρημα του Coppersmith σχετικά με τα πολυώνυμα δύο μεταβλητών δεν δίνει εύκολα έναν αποδοτικό αλγόριθμο για παραγοντοποίηση N $= p^r q$. Ο Howgrave – Graham στην εργασία του "Finding small roots of univariate modular equations revisited" βρήκε έναν άλλο τρόπο να πάρει τα αποτελέσματα του Coppersmith για πολυώνυμα μιας μεταβλητής. Μετά, στην "Extending LLL to Gaussian integers" έδειξε πως τα αποτελέσματα του μπορούν να οδηγήσουν στην παραγοντοποίηση του N $= pq$ γνωρίζοντας μόνο τα μισά πιο σημαντικά bits του p, υποθέτοντας πως p,q είναι ίδιου μεγέθους. Στη περίπτωση που p,q είναι διαφορετικού μεγέθους τα αποτελέσματα των Coppersmith και Howgrave – Graham είναι ασθενέστερα, με την έννοια πως χρειάζονται μεγαλύτερο ποσοστό των bits του μικρότερου παράγοντα. Για τη απόδειξη του θεωρήματος 3.1 επεκτείνεται η μονομεταβλητή προσέγγιση. Για λόγους απλότητας θεωρείται πως r και c δίνονται στον αλγόριθμο του θεωρήματος 3.1. Προφανώς αυτό δεν είναι αναγκαίο αφού κάποιος μπορεί να δοκιμάσει όλες τις δυνατές τιμές για r,c μέχρι να βρει τις σωστές.

Παραγοντοποίηση με lattices

Δίνεται $N = p^r q$. Έστω ότι επίσης δίνεται ένας ακέραιος P ο οποίος ταιριάζει με τον p σε κάποια από τα σημαντικότερα bits του p. Δηλαδή $|P - p| < X$ για κάποιο μεγάλο X. Για τώρα, σκοπός είναι να βρούμε το p αν μας δίνονται τα N, r και P. Θεωρείστε το πολυώνυμο $f(x) = (P + x)^r$. Το σημείο $x_0 = p - P$ ικανοποιεί την $f(x_0) \equiv 0 \; mod \; p^r$. Άρα αναζητείται ρίζα της $f(x)$ modulo p^r τέτοια ώστε $|x_0| < X$. Δυστυχώς το p^r είναι άγνωστο. Είναι γνωστό ένα πολλαπλάσιό του όμως, το N.

Δοσμένου πολυωνύμου $h(x) = \sum_i a_i x^i$ ορίζεται $\|h(x)\|^2 = \sum_i \left|a_i^2\right|$. Το παρακάτω δεδομένο είναι το κύριο εργαλείο για την εύρεση του x_0.

Λήμμα 3.2 (HG) Έστω $h(x) \in \mathbb{Z}[x]$ πολυώνυμο βαθμού d και έστω πως

α. $h(x_0) \equiv 0 \; mod \; p^{rm}$ για κάποιους θετικούς ακεραίους r,m με $|x_0| < X$ και
β. $\|h(xX)\| < p^{rm}/\sqrt{d}$

Τότε $h(x_0) = 0$ στους ακεραίους.

Απόδειξη Παρατηρήστε πως

$$|h(x_0)| = \left|\sum a_i x_0^i\right| = \left|\sum a_i X^i \left(\frac{x_0}{X}\right)^i\right| \leq \sum \left|a_i X^i \left(\frac{x_0}{X}\right)^i\right| \leq$$
$$\sum \left|a_i X^i\right| \leq \sqrt{d}\|h(xX)\| \leq p^{rm}$$

Όμως αφού $h(x_0) \equiv 0$ modulo p^{rm} έπεται πως $h(x_0) = 0$.

Το λήμμα 3.2 λέει πως πρέπει να αναζητήσουμε ένα πολυώνυμο $h(x)$ που έχει το x_0 ως ρίζα modulo p^{rm} και η νόρμα του $h(xX)$ είναι μικρότερη από p^{rm}. Έστω $m > 0$ ένας ακέραιος που θα οριστεί μετά. Για $k = 0, \dots, m$ και $i \geq 0$ ορίζουμε:

$$g_{i,k}(x) := N^{m-k} x^i f^k(x)$$

Παρατηρούμε ότι το x_0 είναι ρίζα του $g_{i,k}(x)$ modulo p^{rm} για όλα τα $i \geq 0$ και $k = 0, \dots, m$. Αυτό που πρέπει να βρεθεί, είναι ένας ακέραιος γραμμικός συνδυασμός των $g_{i,k}(x)$ με νόρμα μικρότερη από p^{rm}. Άρα φτιάχνουμε ένα lattice που "απλώνεται" από τα $g_{i,k}(xX)$ και χρησιμοποιούμε τον LLL για να βρούμε ένα short vector σε αυτό το lattice. Αφού βρούμε ένα "αρκετά κοντό" διάνυσμα $h(xX)$, από το λήμμα 3.2 θα προκύπτει πως το x_0 είναι ρίζα του $h(x)$ στο $\mathbb{Z}$. Τότε το x_0 μπορεί να βρεθεί χρησιμοποιώντας γνωστές μεθόδους εύρεσης ρίζας στους πραγματικούς.
Έστω L το lattice που δημιουργείται από τους συντελεστές των:

α. $g_{i,k}(xX)$ για $k = 0, \dots, m-1$ και $i = 0, \dots, r-1$ και
β. $g_{j,k}(xX)$ για $j = 0, \dots, d - mr - 1$.

	1	x	x^2	x^3	x^4	x^5	x^6	x^7	x^8
$g_{0,0}(xX)$	N^3								
$g_{1,0}(xX)$		XN^3							
$g_{0,1}(xX)$	$*$	$*$	X^2N^2						
$g_{1,1}(xX)$		$*$	$*$	X^3N^2					
$g_{0,2}(xX)$	$*$	$*$	$*$	$*$	X^4N				
$g_{1,2}(xX)$		$*$	$*$	$*$	$*$	X^5N			
$g_{0,3}(xX)$	$*$	$*$	$*$	$*$	$*$	$*$	X^6		
$g_{1,3}(xX)$		$*$	$*$	$*$	$*$	$*$	$*$	X^7	
$g_{2,3}(xX)$			$*$	$*$	$*$	$*$	$*$	$*$	X^8

Παράδειγμα lattice που δημιουργείται από τα διανύσματα $g_{i,k}(xX)$

(Lattice για $N = p^2q$ όταν $m = 3$ και $d = 9$. Τα "*" στοιχεία είναι μη μηδενικά των οποίων την τιμή αγνοούμε. Η ορίζουσα του lattice είναι το γινόμενο των στοιχείων της διαγωνίου.)

Οι τιμές των m,d θα προσδιοριστούν μετά. Για να γίνει χρήση του λήμματος 2.1 πρέπει να φράξουμε την ορίζουσα του lattice που προκύπτει. Έστω M ένας πίνακας που οι σειρές του είναι τα διανύσματα της βάσης του L. Παρατηρούμε πως ο M είναι τριγωνικός πίνακας άρα η ορίζουσα του L είναι το γινόμενο των στοιχείων της διαγωνίου του M. Δηλαδή,

$$det(M) = \left(\prod_{k=0}^{m-1}\prod_{i=0}^{r-1} N^{m-k}\right)\left(\prod_{j=0}^{d-1} X^{j}\right) < N^{rm(m+1)/2}X^{d^2/2}$$

Το λήμμα 2.1 εγγυάται ότι ο αλγόριθμος LLL θα βρει ένα short vector u στο L που να ικανοποιεί:

$$\|u\|^d \le 2^{d^2/2}det(L) \le 2^{d^2/2}N^{rm(m+1)/2}X^{d^2/2} \quad (1)$$

Αυτό το διάνυσμα u είναι βασικά οι συντελεστές ενός πολυωνύμου $h(xX)$ που ικανοποιεί $\|h(xX)\| = \|u\|$. Ακόμη, αφού το $h(xX)$ είναι ακέραιος γραμμικός συνδυασμός των πολυωνύμων $g_{i,k}(xX)$ μπορούμε να γράψουμε το $h(x)$ ως ακέραιο γραμμικό συνδυασμό των $g_{i,k}(x)$. Έτσι $h(x_0) \equiv 0\ mod\ p^{rm}$. Για να εφαρμόσουμε το λήμμα 3.2 στο $h(x)$ απαιτούμε:

$$\|h(xX)\| < p^{rm}/\sqrt{d}$$

Ο παράγοντας $\sqrt{d}$ στον παρανομαστή έχει πολύ μικρή επίδραση στους επόμενους υπολογισμούς οπότε για λόγους απλότητας παραλείπεται. Βάζοντας το φράγμα από την (1) στο $\|h(xX)\|$ και αναδιατάσσοντας τους όρους βλέπουμε ότι η συνθήκη αυτή ικανοποιείται όταν:

$$(2X)^{d^2/2} < p^{rmd}N^{-rm(m+1)/2}$$

Υποθέτουμε πως $q < p^c$ για κάποιο c. Τότε $N < p^{r+c}$ οπότε χρειαζόμαστε

$$(2X)^{d^2/2} < p^{rmd-r(r+c)m(m+1)/2}$$

Μεγαλύτερες τιμές του X μας επιτρέπουν να χρησιμοποιήσουμε ασθενέστερες προσεγγίσεις του P οπότε θέλουμε να βρούμε το μεγαλύτερο X που ικανοποιεί το φράγμα. Η βέλτιστη τιμή του m επιτυγχάνεται στο $m_0 = \left\lfloor \frac{d}{r+c} - \frac{1}{2} \right\rfloor$ και μπορούμε να διαλέξουμε d_0 τέτοιο ώστε το $\frac{d_0}{r+c} - \frac{1}{2}$ να απέχει $\frac{1}{2r+c}$ από έναν ακέραιο. Θέτοντας $m = m_0$ και $d = d_0$ και μετά από πολλές πράξεις:

$$X < \frac{1}{2}p^{1-\frac{c}{r+c}-\frac{r}{d}(1+\delta)} \text{ με } \delta = \frac{1}{r+c} - \frac{r+c}{4d}$$

Αφού $\delta < 1$ παίρνουμε το ελαφρώς "χειρότερο" αλλά πιο "ελκυστικό" φράγμα:

$$X < p^{1-\frac{c}{r+c}-2\frac{r}{d}} \quad (2)$$

Όταν το X ικανοποιεί τη (2), ο αλγόριθμος LLL θα βρει ένα διάνυσμα $h(xX)$ στο L που θα ικανοποιεί $\|h(xX)\| < p^{rm}/\sqrt{d}$. Αυτό το short vector θα παράγει το πολυώνυμο $h(x)$ που είναι ακέραιος γραμμικός συνδυασμός των $g_{i,k}(x)$ και άρα έχει το x_0 ρίζα modulo p^{rm}. Αφού όμως το $\|h(xX)\|$ είναι φραγμένο, έχουμε, από λήμμα 3.2, ότι $h(x_0) = 0$ στο $\mathbb{Z}$ και κλασσικές μέθοδοι εύρεσης ρίζας μπορούν να μας δώσουν το x_0. Έχοντας το x_0, παίρνουμε τον παράγοντα $p = P + x_0$.
Το αποτέλεσμα συνοψίζεται στο παρακάτω λήμμα.

Λήμμα 3.3 Έστω δοσμένο $N = p^r q$ και έστω πως $q < p^c$ για κάποιο c. Επιπλέον υποθέτουμε πως P είναι ένας ακέραιος που ικανοποιεί:

$$|P - p| < p^{1-\frac{c+1}{r+c}}$$

Τότε η παραγοντοποίηση του N μπορεί να γίνει σε χρόνο $O((logN)^2 d^4)$.
Έστω $\varepsilon = \frac{c+1}{r+c}$. Προχωράμε ως εξής:

α. For all $k = 1, \dots, (logN)/r$ do:

β. For all $j = 0, \dots, 2^{\varepsilon k}$ do:

γ. Θέτουμε $P = 2^k + j \cdot 2^{(1-\varepsilon)k}$

δ. Τρέξε τον αλγόριθμο του λήμματος 3.3 χρησιμοποιώντας τη προσέγγιση του P.

Το εξωτερικό loop στο μήκος του p δεν χρειάζεται αν το μήκος του είναι γνωστό. Αν το p είναι μεγέθους k bits τότε ένα από τα P που παράγεται στο βήμα (γ) θα ικανοποιεί $|P - p| < p^{1-\frac{c+1}{r+c}}$ οπότε θα ισχύει $|P - p| < p^{(1-e)k}$ όπως χρειάζεται. Άρα ο αλγόριθμος θα παραγοντοποιήσει το N στον απαιτούμενο χρόνο.

4. Εφαρμογή και πειράματα

Η μέθοδος των πλεγμάτων εφαρμόστηκε χρησιμοποιώντας το Maple 5.0 και το NTL (Number Theory Library) πακέτο του Victor Shoup. Το πρόγραμμα τρέχει σε 2 φάσεις. Πρώτα, μαντεύει τα σημαντικότερα bits του p και μετά φτιάχνει το πλέγμα που περιγράφηκε στην παράγραφο 3. Χρησιμοποιώντας τον LLL από το NTL, μικραίνει το πλέγμα της παραγράφου 3, ψάχνοντας για "κοντά διανύσματα" (short vectors). Έπειτα, αφού βρει ένα κοντό διάνυσμα, δίνεται το αντίστοιχο πολυώνυμο στο Maple που υπολογίζει τις ρίζες για σύγκριση με την παραγοντοποίηση του N.

Διάφορες παρατηρήσεις έγιναν κατά την εφαρμογή αυτού του αλγορίθμου. Κατ'αρχάς έχει σημασία η σειρά με την οποία τα διανύσματα της βάσης εμφανίζονται στο πλέγμα που δίνεται στον LLL. Συγκεκριμένα αφού το τελευταίο πολυώνυμο είναι σχεδόν πάντα ίδιου βαθμού με τη διάσταση του πλέγματος, ένας γραμμικός συνδυασμός που δίνει ένα κοντό διάνυσμα πρέπει να περιέχει τα διανύσματα της βάσης που αντιστοιχούν στα τελευταία $g_{i,k}, k = m/2, \dots, m$ και $i = 0, \dots, r$. Φαίνεται πως είναι προτιμότερο να

τοποθετηθούν στην "κορυφή" του πλέγματος, όπου ο LLL θα πραγματοποιήσει πρώτα την ελάττωση των σειρών αφού αυτά μόνα τους είναι πολύ πιθανό να αρκούν για την κατασκευή ενός κοντού διανύσματος. Η βέλτιστη σειρά για τα $g_{i,k}$ είναι $i = r-1, \dots, 0$ και $k = m, m-1, \dots, 0$. Αυτή η τοποθέτηση είχε ως αποτέλεσμα ελαττωμένο χρόνο εκτέλεσης σε σύγκριση με τη φυσική σειρά στην οποία ο LLL έκανε πολύ ώρα μειώνοντας διανύσματα της βάσης που τελικά θα ήταν περιττό.

p	N	r	δοσμένα bits	διάσταση πλέγματος	χρόνος εκτέλεσης
64 bits	576 bits	8	16 bits	49	20 λεπτά
80 bits	1280 bits	15	20 bits	72	21 ώρες
96 bits	768 bits	7	22 bits	60	7 ώρες
96 bits	960 bits	9	22 bits	65	10 ώρες
100 bits	600 bits	5	23 bits	69	11 ώρες

Χρόνοι εκτέλεσης σε Pentium 400MHz

Κάποιος μπορεί να ισχυριστεί πως στην κατασκευή του πλέγματος στην παράγραφο 3 θα μπορούσαν να έχουν χρησιμοποιηθεί δυνάμεις του $(P + x)$ αντί για μεταθέσεις και δυνάμεις του $(P + x)^r$. Ο λόγος που προτιμήθηκε το τελευταίο είναι βασικά η βελτίωση της απόδοσης. Ενώ και οι δύο τρόποι δίνουν πλέγμα με την ίδια ορίζουσα, χρησιμοποιώντας μεταθέσεις και δυνάμεις του $(P + x)^r$ παράγεται ένας πίνακας που μοιάζει "πιο ορθοκανονικός". Δηλαδή, συγκεκριμένοι υποπίνακες του πίνακα από την παράγραφο 3 είναι πίνακες Toeplitz και ευρετικά αυτό κάνει πιο εύκολο για τον LLL να βρει μια καλή βάση. Στη σύγκριση μεταξύ των δύο μεθόδων, η χρήση του $(P + x)^r$ βελτιώνει την ταχύτητα κατά 10%. Τέλος, είναι χρήσιμο να θυμηθεί κανείς ότι σε ένα πλέγμα μειωμένο από τον LLL, το κοντύτερο διάνυσμα u ικανοποιεί:

$$\|u\| < 2^{d/2}\det(L)^{1/d}$$

Εφαρμογές του LLL συχνά προσπαθούν να βελτιώσουν τον παράγοντα $2^{d/2}$ ("fudge factor"). Ωστόσο, όπως δείχνει η ανάλυση στην παράγραφο 3, η δράση του είναι αμελητέα καθώς χρειάζεται μόνο ένα επιπλέον bit του p να γνωστοποιηθεί. Έτσι η υψηλής ποιότητας μείωση που παράγεται με μικρότερο "fudge factor" δεν είναι αναγκαία και οι χρόνοι εκτέλεσης μπορούν να βελτιωθούν σημαντικά απενεργοποιώντας βελτιώσεις όπως η μείωση των Korkin και Zolotarev.

5. Σύγκριση με άλλες μεθόδους παραγοντοποίησης

Επαναδιατυπώνεται το θεώρημα 3.1 για ευκολία στη σύγκριση της μεθόδου των πλεγμάτων με τους υπάρχοντες αλγόριθμους. Έστω $T_\alpha(p)$ μια συνάρτηση που ορίζεται ως εξής:

$$T_\alpha(p) = \exp((logp)^a)$$

Αυτή η συνάρτηση είναι ανάλογη της συνάρτησης $L_{\alpha,b}(p)$ που χρησιμοποιείται για να περιγράψει τον χρόνο εκτέλεσης αλγόριθμων παραγοντοποίησης. Υπενθυμίζεται πως:

$$L_{\alpha,b}(p) = \exp(b(logp)^a(loglogp)^{1-a}$$

Φαίνεται εύκολα πως η $T_\alpha(p)$ είναι λίγο μικρότερη από την $L_{\alpha,1}(p)$. Μπορεί τωρα να διατυπωθεί μια ειδική περίπτωση του θεωρήματος 3.1.

Πόρισμα 5.1 Έστω δοσμένος $N = p^r q$ με p,q ακεραίους μήκους k bits και έστω $r = (logp)^\varepsilon$ για κάποιο ε. Τότε με δοσμένους N,r ένας ακέραιος παράγοντας του N μπορεί να βρεθεί σε χρόνο

$$\gamma T_{1-\varepsilon}(p) = \exp[(logp)^{1-\varepsilon}]\gamma$$

όπου γ είναι πολυωνυμικό σε logN.

Ασυμπτωτική σύγκριση

Έστω p,q πρώτοι μήκους k bits και έστω πως δίνεται $N = p^r q$. Μελετάται ο χρόνος εκτέλεσης διάφορων αλγόριθμων ως προς τα k,r και αναλύεται η συμπεριφορά τους καθώς το r τείνει στο άπειρο. Γράφουμε $r = (logp)^\varepsilon$. Στον παρακάτω πίνακα φαίνονται οι χρόνοι εκτέλεσης τριών αλγορίθμων αγνοώντας πολυωνυμικούς παράγοντες. LFM η μέθοδος παραγοντοποίησης με χρήση πλεγμάτων, ECM η μέθοδος ελλειπτικών καμπυλών και NFS το κλασσικό "κόσκινο" παραγοντοποίησης.

Μέθοδος	Ασυμπτωτικός χρόνος εκτέλεσης
LFM	$\exp((logp)^{1-\varepsilon})$
ECM	$\exp(1.414\cdot (logp)^{1/2}(loglogp)^{1/2})$
NFS	$\exp(1.902\cdot (logN)^{1/3}(loglogN)^{2/3}$

Αφού $N = p^r q$ και $r = k^\varepsilon$ ξέρουμε ότι

$$\log N = r\log p + \log q \geq rk = k^{1+\varepsilon}$$

Ξαναγράφοντας τους χρόνους ως προς k:

Μέθοδος	Ασυμπτωτικός χρόνος εκτέλεσης
LFM	$\exp(k^{1-\varepsilon}) = T_{1-\varepsilon}(p)$
ECM	$\exp(1.414\cdot k^{1/2}(logk)^{1/2}) > (T_{1/2}(p))^{1.414}$
NFS	$\exp(1.902\cdot k^{1+\varepsilon/3}\big((1+\varepsilon)logk\big)^{2/3}\big) > (T_{1+\varepsilon/3}(p))^{1.902}$

Παρατηρήστε πως όταν $\varepsilon = \frac{1}{2}$, τότε και οι τρεις αλγόριθμοι τρέχουν σε χρόνο κοντά στο $T_{1/2}(p)$.

6. Μια εφαρμογή σε ακεραίους της μορφής $N = p^2 q$

Έστω p,q πρώτοι ίδιου μεγέθους. Μπορούμε να χρησιμοποιήσουμε το λήμμα 3.3 για να πάρουμε αποτελέσματα σε "παραγοντοποίηση με βοήθεια" για ακεραίους της μορφής $N = p^r q$ με μικρό r. Τα αποτελέσματα του Coppersmith δείχουν πως όταν N = pq, μια "βοήθεια" που περιέχει τα μισά bits του p αρκεί ώστε να παραγοντοποιηθεί το N. Όταν r = 1, το λήμμα 3.3 δίνει το ίδιο αποτέλεσμα.

Ωστόσο, όταν r = 2 το λήμμα, μας δείχνει πως μόνο ένα τρίτο των bits του p χρειάζονται για να παραγοντοποιηθεί το N. Με άλλα λόγια η "βοήθεια" αρκεί να είναι μεγέθους $N^{1/9}$. Άρα αριθμοί της μορφής $N = p^2q$ είναι πιο ευαίσθητοι σε επιθέσεις που χρησιμοποιούν διαρρέοντα bits του p.

7. Συμπεράσματα

Έδειξαν λοιπόν ότι για κρυπτογραφικές εφαρμογές, ακέραιοι της μορφής $N = p^rq$ πρέπει να χρησιμοποιούνται με προσοχή. Συγκεκριμένα έδειξαν ότι το N παραγοντοποιείται ευκολότερα όσο το r μεγαλώνει. Για παράδειγμα, όταν $r = \varepsilon * \log(p)$ για μία σταθερά $\varepsilon > 0$, το N μπορεί να παραγοντοποιηθεί σε πολυωνυμικό χρόνο. Έτσι, αν p,q είναι πρώτοι μήκους k bits, το $N = p^kq$ μπορεί να παραγοντοποιηθεί από έναν αλγόριθμο σε πολυωνυμικό χρόνο. Ακόμα και όταν $r \approx \sqrt{\log(p)}$ το N μπορεί να παραγοντοποιηθεί σε χρόνο που είναι ασυμπτωτικά ταχύτερος από τις καλύτερες γνωστές μεθόδους.

Τα αποτελέσματά τους δεν δείχνουν πολλά για μικρά r, όπως για την περίπτωση $N = p^2q$. Τα πειράματα έδειξαν πως όταν τα p,q είναι μικρά (κάτω από 100 bits) ο αλγόριθμος δεν είναι πρακτικός και δεν μπορεί να ανταγωνιστεί τον ECM. Υποθέτουν όμως, πως σε μεγάλη κλίμακα, δηλαδή μόλις τα p,q υπερβούν τα 400 bits, ο αλγόριθμος είναι αποτελεσματικότερος του ECM. Παραδόξως, τα αποτελέσματά τους δεν μοιάζουν να ακολουθούν τα αποτελέσματα του Coppersmith στην εύρεση μικρών ριζών σε πολυώνυμα δύο μεταβλητών στους ακεραίους. Αντ'αυτού επεκτείνουμε μια διαφορετική τεχνική (του Howgrave Graham). Είναι χρήσιμο να συγκρίνουμε τα αποτελέσματα τους σε περίπτωση μη ισορροπημένου RSA όπου οι παράγοντες p,q του N είναι πρώτοι διαφορετικού μεγέθους, έστω p πολύ μεγαλύτερο του q. Υποθέτουμε πως το p είναι της τάξης του q^s. Τότε, όσο μεγαλύτερο είναι το s, τόσο περισσότερα bits του q χρειάζονται για την αποδοτική παραγοντοποίηση του N, ενώ είδαμε πως αν $N = p^rq$, όσο μεγαλύτερο είναι το r, τόσο λιγότερα bits του p χρειάζονται. Ένα μειονέκτημα της μεθόδου παραγοντοποίησης με πλέγματα (LFM) είναι πως για κάθε υπόθεση των πιο σημαντικών bits του p, πρέπει να χρησιμοποιηθεί ο αλγόριθμος LLL για να μικρύνει το μέγεθος του πλέγματος που προκύπτει. Ένα ενδιαφέρον ανοιχτό πρόβλημα είναι να κατασκευαστεί μέθοδος που θα τρέχει μια φορά τον LLL και θα δοκιμάζει πολλαπλές υποθέσεις για τα πιο σημαντικά bits του p. Αυτό θα βελτιώσει σημαντικά τον χρόνο εκτέλεσης του αλγορίθμου. Η λύση θα είναι ανάλογη μιας τεχνικής που θα δοκιμάζει στον ECM πολλές ελλειπτικές καμπύλες με τη μία. Ένα άλλο πρόβλημα είναι η γενίκευση του του LFM σε ακεραίους της μορφής $N = p^rq^s$ όπου r,s είναι περίπου ίδιου μεγέθους.