
بسم الله الرحمن الرحیم

معرفی

الهام خالقی لوحه سرا

دانشگاه جامع انقلاب اسلامی

رشته: فناوری اطلاعات

استاد: دکتر سید علی لاجوردی

دی ماه ۱۴۰۴

فهرست مطالب فصل ۱۰

توضیحات

مسائل سخت نظریه اعداد

- در فصل قبل، چند مسئله نظریه اعداد را معرفی کردیم از جمله فاکتورگیری حاصلضرب دو عدد اول بزرگ و محاسبه لگاریتم گسسته در گروه‌های خاص که عموماً سخت فرض می‌شوند.
- سختی یعنی فرض بر این است که هیچ الگوریتم کارآمدی برای حل این مسائل وجود ندارد.

مفهوم سختی و پارامتر امنیتی

- با این حال، این مفهوم مجانبی سختی، اطلاعات کمی درباره چگونگی تنظیم پارامتر امنیتی (گاهی "طول کلید") برای دستیابی به یک سطح امنیتی مشخص و عملی به ما می‌دهد.
- درک صحیح این مسئله برای استقرار واقعی سیستم‌های رمزنگاری مبتنی بر این مسائل بسیار مهم است.

توضیحات

⚖️ انتخاب طول کلید خیلی پایین، سیستم را آسیب‌پذیر می‌کند؛ انتخاب خیلی بالا، امنیت را تضمین می‌کند ولی کارایی را فدای آن می‌کند.
تعادل بین امنیت و کارایی یک چالش طراحی اساسی در رمزنگاری است.
🔄سختی نسبی مسائل مختلف نظریه‌اعدادی می‌تواند در انتخاب پایه‌ی رمزنگاری مؤثر باشد.
نه همه‌ی مسائل یکسان هستند؛ برخی از نظر عملی سخت‌تر یا مناسب‌ترند.
❓جستجوی جامع لزوماً بهترین راه‌حل نیست؛ بنابراین طول کلید n به‌معنای مقاومت در برابر 2^nعملیات نیست.
→ در کلید عمومی، الگوریتم‌های هوشمندانه‌تری از brute-force وجود دارند که این رابطه خطی را از بین می‌برند.
🔑 در مقابل، در رمزنگاری کلید متقارن، بهترین حملات تقریباً همان پیچیدگی brute-forceرا دارند.
→ همین تفاوت باعث می‌شود کلیدهای عمومی خیلی طولانی‌تر از کلیدهای متقارن باشند (مثلاً ۲۰۴۸ بیت در مقابل ۱۲۸ بیت).

توضیحات

📚 این فصل الگوریتم‌های غیر-چندجمله‌ای ولی بسیار کارآمدتر از **brute-force** برای فاکتورگیری و لگاریتم گسسته را مرور می‌کند.

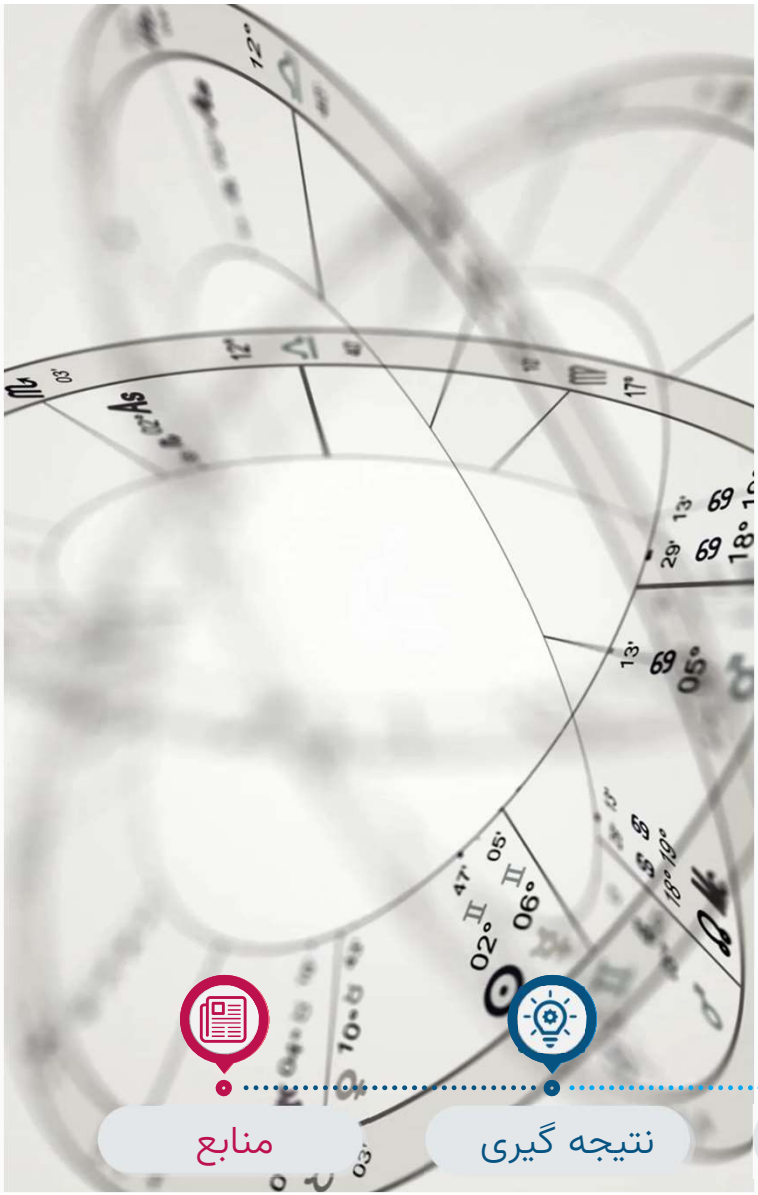
🔍 تمرکز بر ایده‌های سطح‌بالا است و جزئیات پیاده‌سازی نادیده گرفته می‌شوند.

⚛ الگوریتم‌های کوانتومی در اینجا بررسی نمی‌شوند و به فصل جداگانه‌ای واگذار شده‌اند.

🎯 فقط الگوریتم‌های فاکتورگیری و لگاریتم گسسته بررسی می‌شوند، چون بهترین راه‌حل‌های شناخته‌شده برای مسائلی مانند **RSA** و **DDH** نیز از همین راه‌ها عبور می‌کنند.

بخش اول

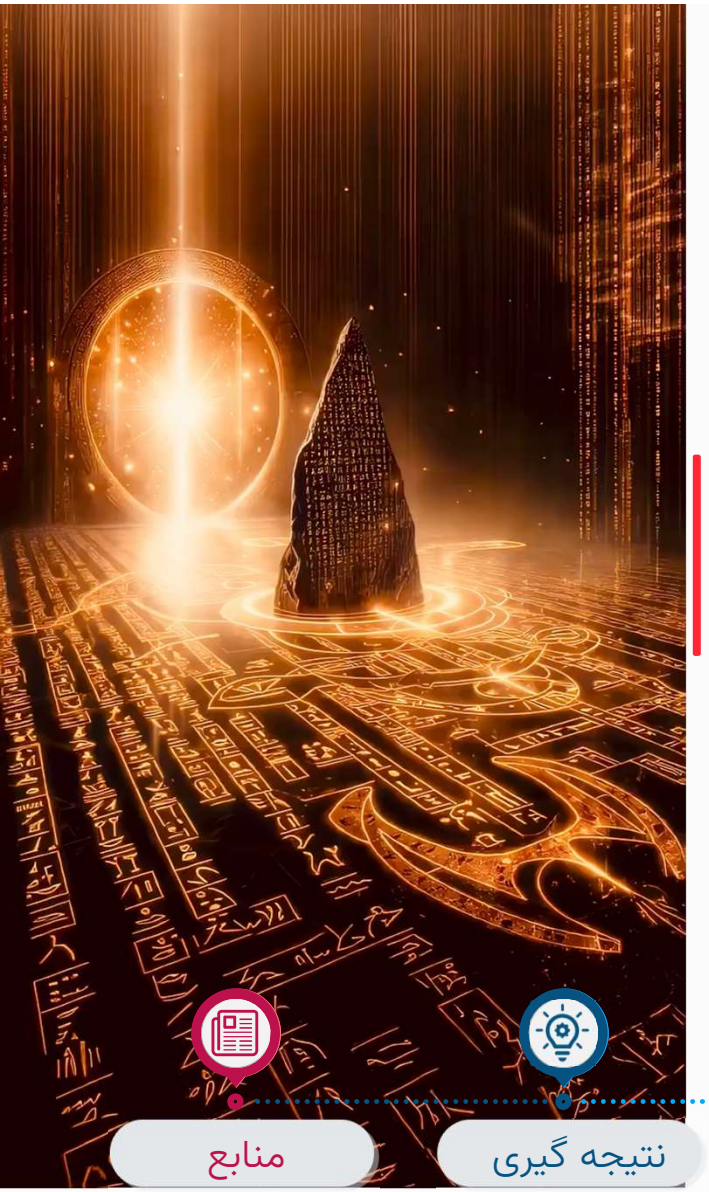
الگوریتم‌های فاکتورگیری



الگوریتم‌های فاکتورگیری

- **فرض پایه:** عدد مورد نظر حاصل‌ضرب دو عدد اول بزرگ ($N = p\ q$) است.
- **ابزار ریاضی:** استفاده از «قضیه باقی‌مانده چینی» برای ساده‌سازی محاسبات از طریق تجزیه به پیمانه‌های کوچکتر CN.
- **روش سنتی(Trial Division)** : بسیار کند و غیربهینه برای اعداد بزرگ (زمان نمایی)🐢 .
- **روش p-1 پولارد:** مناسب برای فاکتورگیری در شرایط خاص) وقتی فاکتورها «اعداد نرم» یا Smooth باشند).🎯
- **روش Pollard's Rho** : یک روش عمومی و سریع‌تر از روش سنتی، اما همچنان در مقیاس‌های بزرگ سنگین است.🌀
- **غربال کوادراتیک (Quadratic Sieve):** الگوریتمی پیشرفته‌تر که سرعت آن فراتر از حالت نمایی (زیرنمایی) است.🕸️
- **قهرمان سرعت(GNFS)** : الگوریتم General Number Field Sieve در حال حاضر سریع‌ترین روش تجزیه اعداد بسیار بزرگ در جهان است.🏆

$$2^{\mathcal{O}((\log N)^{1/3}\cdot(\log\log N)^{2/3})}$$



p-1 پولارد

ایده اصلی: اگه عدد p-1 فقط فاکتورهای اولِ کوچیک داشته باشه، ما می‌تونیم خیلی سریع N رو تجزیه کنیم.
اول یه عدد B پیدا می‌کنیم که مضربی از p−1 باشه (یعنی p−1 مقسومٌ‌علیه Bباشد) ولی مضربی از q−1 نباشد (یعنی q−1 مقسومٌ‌علیه B نباشد).
فرمول محاسبه B: برای ساختن چنین عددی، معمولاً ضرب توان‌های اعداد اول کوچک را تا یک حد kدر نظر می‌گیریم:

$$B = \prod_{i=1}^{k} p_i^{\lfloor n/\log p_i \rfloor}$$

شروع عملیات:
یه عدد تصادفی x از Z*_N انتخاب می‌کنیم و مقدار زیر رو حساب می‌کنیم:
y := x^B − 1 (mod N)
جادوی هم‌ریختی:
طبق قضیه باقی‌مانده‌ی چینی، این تساوی برقرار میشه:
y = x^B − 1 (mod N) ↔ (x_p^B − 1 (mod p) , x_q^B − 1 (mod q))
حذف p:
چون B مضربی از p−1 هست، طبق قضیه کوچک فرما داریم:
x^B (mod p) = 1
پس بخش اول جفت‌مرتب صفر می‌شود:
y ↔ (0 , x_q^B − 1 (mod q))
پیدا کردن فاکتور:
حالا چون y (mod p) = 0ولی با احتمال زیاد y (mod q) ≠ 0، یعنی y بر p بخش‌پذیر است ولی بر q نه؛
پس با یک ب‌م‌م ساده فاکتور رو پیدا می‌کنیم:
p = gcd(y , N)
شانس موفقیت: حدود Ω(1/n) هست که با چند بار تکرار، عملاً قطعی می شود.

p-۱ پولارد

نقطه ضعف الگوریتم:

اگه هر دو عدد $p-1$ و $q-1$ فاکتورهای اولِ خیلی بزرگ داشته باشند، این الگوریتم دیگه کارآمد نیست و زمان اجراش خیلی زیاد می شود.

اعداد اول قوی (Strong Primes)

برای مقابله با این حمله، قبلاً p و q رو طوری انتخاب می‌کردن که $(p-1)/2$خودش یه عدد اول باشد؛ به این اعداد می‌گویند اعداد اول قوی.

نتیجه‌گیری امنیتی

امروزه دیگه حساسیت زیادی روی اعداد اول قوی وجود نداره، چون اگر اعداد اول به‌صورت تصادفی و به اندازه ی کافی بزرگ انتخاب بشن، احتمال نرم بودن $p-1$خیلی کمه و در عین حال الگوریتم‌های فاکتورگیری قوی‌تری هم وجود دارن.

ALGORITHM 10.1
Pollard's $p-1$ algorithm for factoring

Input: Integer N
Output: A nontrivial factor of N

$x \leftarrow \mathbb{Z}_N^*$
$y := [x^B - 1 \bmod N]$
  // B is as in the text
$p := \gcd(y, N)$
if $p \notin \{1, N\}$ **return** p

الگوریتم Pollard's Rho

- در مقابل الگوریتم ۱۰.۱ که فقط برای برخی مدول‌ها مؤثر است، الگوریتم Pollard's rho می‌تواند برای فاکتورگیری یک عدد صحیح دلخواه از شکل $N = p\ q$ به کار رود؛ از این نظر، یک الگوریتم عمومی برای فاکتورگیری محسوب می‌شود.به‌صورت ابتکاری ((heuristic، این الگوریتم با احتمال ثابت، عدد N را در زمانی از مرتبه $\mathcal{O}\left(N^{1/4} \cdot \text{polylog}(N)\right)$ فاکتور می‌کند. هرچند این زمان همچنان نمایی است، اما نسبت به روش تقسیم آزمایشی (trial division) بهبود بسیار چشمگیری دارد.ایده‌ی اصلی این روش، یافتن دو مقدار متمایز x و x' در Z^*_N است که نسبت به پیمانه‌ی p هم‌نهشت باشند؛
- یعنی $x \equiv x' \pmod{p}$ به چنین زوجی، یک «زوج خوب» گفته می‌شود. توجه کنید که اگر x و x' یک زوج خوب باشند، آنگاه داریم $\gcd(x - x', N) = p$ زیرا $x \not\equiv x' \pmod{N}$ است. بنابراین با محاسبه‌ی ب.م.م می‌توان یک عامل غیر بدیهی از N را به دست آورد.
- **مشکل حافظه و زمان اجرا:در صورتی که بخواهیم تمام زوج‌های ممکن از این مقادیر را با یکدیگر مقایسه کنیم، تعداد مقایسه‌ها از مرتبه O(N^(1/2))خواهد بود و در نتیجه، زمان اجرای الگوریتم عملاً به سطح روش‌های سنتی فاکتورگیری بازمی‌گردد.**

ALGORITHM 10.2
Pollard's rho algorithm for factoring

Input: Integer N, a product of two n-bit primes
Output: A nontrivial factor of N

$x \leftarrow \mathbb{Z}_N^*, \quad x' := x$
for $i = 1$ to $2^{n/2}$:
  $x := F(x)$
  $x' := F(F(x'))$
  $p := \gcd(x - x', N)$
  if $p \notin \{1, N\}$ **return** p and stop

الگوریتم Pollard's Rho

راه‌حل پولارد (تابع تکرارشونده):

پولارد برای رفع این مشکل، استفاده از یک تابع تکرارشونده را پیشنهاد کرد؛ برای مثال تابع $F(x) = x^2 + 1 \pmod N$ این تابع دارای یک ویژگی کلیدی است:

اگر $x \equiv x' \pmod p$ برقرار باشد، آنگاه به‌طور خودکار خواهیم داشت $F(x) \equiv F(x') \pmod p$

این خاصیت تضمین می‌کند که پس از وقوع هم‌نهشتی نسبت به p، این هم‌نهشتی در مراحل بعدی تکرار تابع نیز حفظ شود.

ALGORITHM 10.1
Pollard's $p-1$ algorithm for factoring

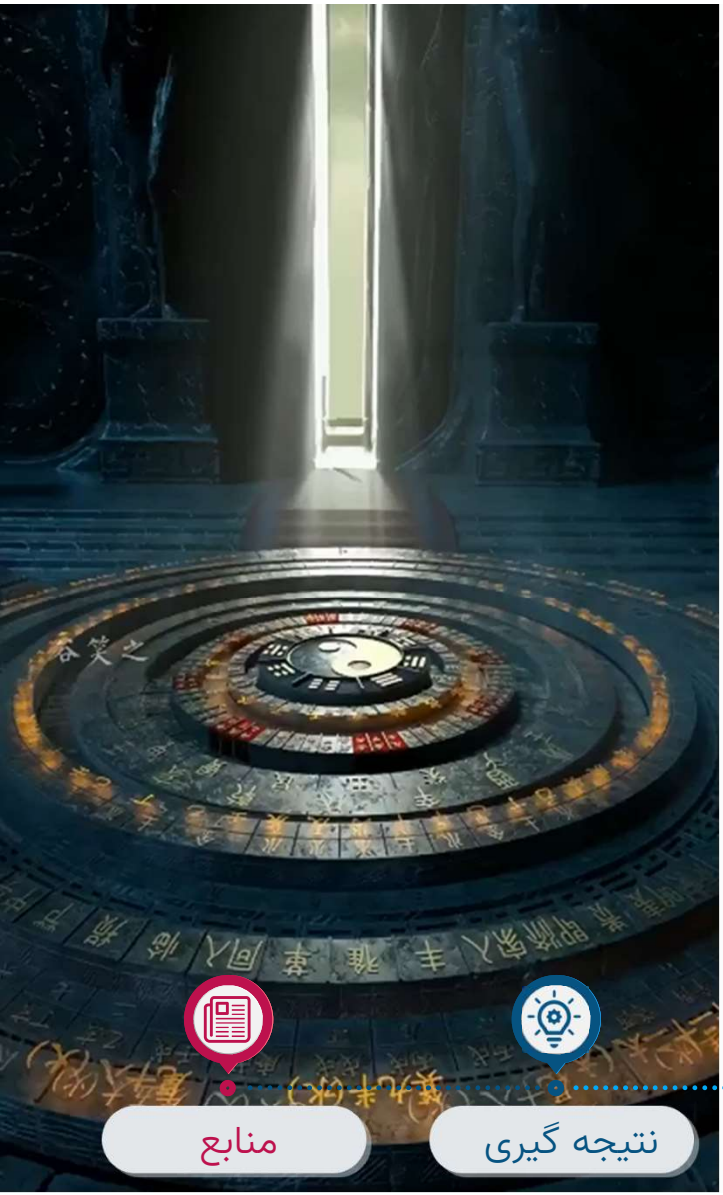
Input: Integer N
Output: A nontrivial factor of N

$x \leftarrow \mathbb{Z}_N^*$
$y := [x^B - 1 \bmod N]$
// B is as in the text
$p := \gcd(y, N)$
if $p \notin \{1, N\}$ return p

الگوریتم تشخیص سیکل (Algorithm 10.2): برای اجتناب از مصرف حافظه زیاد، از دو متغیر استفاده می‌شود که به روش «لاک‌پشت و خرگوش» حرکت می‌کنند.

متغیر اول با سرعت معمولی پیش می‌رود: $x := F(x)$ و متغیر دوم با سرعت دو برابر حرکت می‌کند: $x' := F(F(x'))$

شرط توقف: در هر مرحله مقدار $\gcd(x - x', N)$ محاسبه می‌شود. اگر این مقدار عددی غیر از ۱ و N باشد، یک عامل غیر بدیهی از N به دست آمده و الگوریتم متوقف می‌شود.



الگوریتم Pollard’s Rho

نام Rho از این واقعیت گرفته شده است که اگر مسیر تولید مقادیر را ترسیم کنیم، دنباله ابتدا به صورت یک مسیر خطی پیش می‌رود و سپس وارد یک چرخه می‌شود؛ شکلی که از نظر بصری شبیه حرف یونانی ρ(رو) است

احتمال موفقیت:

این الگوریتم ماهیتی ابتکاری (heuristic) دارد؛ به این معنا که تضمین قطعی در هر اجرا وجود ندارد، اما در عمل با احتمال ثابت و در زمان کوتاه موفق به یافتن فاکتور می‌شود.

الگوریتم غربال کوادراتیک (Quadratic Sieve)

این روش خیلی هوشمندانه‌تر از قبلی‌هاست و تا قبل از دهه ۹۰، سریع‌ترین روش فاکتورگیری دنیا بود. هنوز هم برای اعدادی تا حدود ۳۰۰ بیت (حدود ۹۰ رقم) بهترین گزینه است.

ایده‌ی اصلی

پایه‌ی QS بر این مشاهده‌ی عددی استوار است: اگر بتوان دو عدد صحیح x,y یافت به‌طوری که

$x^2 \equiv y^2 \pmod N$ و $x \not\equiv \pm y \pmod N$ آنگاه مقدار $\gcd(x - y , N)$ با احتمال غیرناچیز یکی از فاکتورهای غیربدیهی N را به دست می‌دهد. دلیل این امر آن است که $N \mid (x - y)(x + y)$ در حالی که $N \nmid (x - y)$ و $N \nmid (x + y)$ و در نتیجه خواهیم داشت $\gcd(x - y , N) \in \{p , q\}$.

چالش و راه‌حل

یافتن مستقیم چنین جفتی (x , y) که در آن $x^2 \bmod N$ خود یک مربع کامل باشد، احتمال بسیار اندکی دارد. به همین دلیل، الگوریتم QS به‌جای آن به دنبال مجموعه‌ای از مقادیر $x_1 , \dots , x_\ell$ می‌گردد که در آن

$q_i := x_i^2 \bmod N$ همگی B صاف (B-smooth) باشند؛ یعنی تمام فاکتورهای اول آن‌ها در مجموعه $P = \{p_1 , \dots , p_k\}$ از اعداد اول کوچکتر یا مساوی B قرار داشته باشند. در این صورت، برای هر i می‌توان نوشت

$$q_1 = [x_1^2 \bmod N] = \prod_{i=1}^{k} p_i^{e_{1,i}}$$

$$\vdots$$

$$q_\ell = [x_\ell^2 \bmod N] = \prod_{i=1}^{k} p_i^{e_{\ell,i}}.$$

الگوریتم غربال کوادراتیک (Quadratic Sieve)

مرحله دوم: یافتن یک زیرمجموعه مربع‌ساز با جبر خطی پس از جمع‌آوری مقادیر B-صاف

$$q_j = x_j^2 \bmod N = \prod_{i=1}^{k} p_i^{e_{j,i}}, \quad j = 1, \dots, \ell,$$

(که در آن $\{p_1, \dots, p_k\}$مجموعه تمام اعداد اول $B \leq$است)، هدف یافتن یک زیرمجموعه غیرتهی $S \subseteq \{1, \dots, \ell\}$ است که حاصل‌ضرب

$$z := \prod_{j \in S} q_j$$

یک مربع کامل در اعداد صحیح باشد.

با جای‌گذاری فرم فاکتوری مقادیر q_j، داریم:

$$z = \prod_{j \in S} q_j = \prod_{i=1}^{k} p_i^{\sum_{j \in S} e_{j,i}}$$

این عدد zیک مربع کامل است اگر و تنها اگر برای هر i=۱,...,k، مقدار

$$\sum_{j \in S} e_{j,i}$$

زوج باشد. این مشاهده نشان می‌دهد که تنها باقی‌مانده توان‌ها modulo 2 اهمیت دارد. بنابراین، برای هر j، بردار توان modulo 2 را به‌صورت زیر تعریف می‌کنیم:

$$\sum_{j \in S} \mathbf{v}_j = \mathbf{0} \in \mathbb{F}_2^k.$$

برای یافتن چنین زیرمجموعه‌ای، ماتریس $\Gamma \in \mathbb{F}_2^{\ell \times k}$را تشکیل می‌دهیم که سطرهای آن همان بردارهای $\mathbf{v}_j$هستند

الگوریتم غربال کوادراتیک (Quadratic Sieve)

$$\begin{pmatrix} \gamma_{1,1} & \gamma_{1,2} & \cdots & \gamma_{1,k} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{\ell,1} & \gamma_{\ell,2} & \cdots & \gamma_{\ell,k} \end{pmatrix} \stackrel{\text{def}}{=} \begin{pmatrix} [e_{1,1} \bmod 2] & [e_{1,2} \bmod 2] & \cdots & [e_{1,k} \bmod 2] \\ \vdots & \vdots & \ddots & \vdots \\ [e_{\ell,1} \bmod 2] & [e_{\ell,2} \bmod 2] & \cdots & [e_{\ell,k} \bmod 2] \end{pmatrix}$$

اگر $\ell = k + 1$، آنگاه ماتریس Γدارای **سطرهای بیشتری نسبت به ستون‌ها** است. از آنجا که فضای پوچ (nullspace) یک ماتریس $\ell \times k$ با $\ell > k$همواره شامل یک بردار غیرصفر در $\mathbb{F}_2^\ell$است، وجود یک زیرمجموعهٔ غیرتهی Sکه

$$\sum_{j \in S} \mathbf{v}_j = \mathbf{0}$$

تضمین می‌شود. چنین زیرمجموعه‌ای را می‌توان با استفاده از الگوریتم‌های استاندارد جبر خطی روی $\mathbb{F}_2$(مانند حذف گاوسی) به‌صورت کارآمد یافت.

فرض کنید چنین زیرمجموعه‌ای یافت شده باشد. آنگاه، برای هر i، مقدار $\sum_{j \in S} e_{j,i}$ زوج است، و می‌توان نوشت:

$$z = \prod_{j \in S} q_j = \prod_{i=1}^{k} p_i^{\sum_{j \in S} e_{j,i}} = \left(\prod_{i=1}^{k} p_i^{\frac{1}{2} \sum_{j \in S} e_{j,i}} \right)^2 .$$

از سوی دیگر، چون $q_j = x_j^2 \bmod N$، داریم:

$$z = \prod_{j \in S} q_j \equiv \prod_{j \in S} x_j^2 = \left(\prod_{j \in S} x_j \right)^2 \pmod{N}.$$

بنابراین، دو ریشه مربع $z$modulo Nبه‌دست آمده‌اند:

$$y := \prod_{i=1}^{k} p_i^{\frac{1}{2} \sum_{j \in S} e_{j,i}} \in \mathbb{Z}, \text{و} x := \prod_{j \in S} x_j \in \mathbb{Z}_N^*,$$

که در آن

$$x^2 \equiv y^2 \pmod{N}.$$

اگر اتفاقی رخ دهد که $x \not\equiv \pm y \pmod{N}$، آنگاه یک فاکتور غیربدیهی از Nرا آشکار می‌سازد. هرچند این اتفاق تضمین‌شده نیست، اما با استدلال‌های هیوریستیک، احتمال آن یک ثابت مثبت (مستقل از N) است.

بخش دوم

الگوریتم‌های محاسبه لگاریتم‌های گسسته

الگوریتم‌های محاسبه لگاریتم‌های گسسته

- فرض کنید G یک گروه چرخه‌ای از مرتبه معلوم q باشد و $g \in G$ یک مولد آن. مسئله لگاریتم گسسته (DLP) عبارت است از یافتن $x \in \mathbb{Z}_q$ به‌گونه‌ای که $g^x = h$
- برای $h \in G$ داده‌شده. در حالت کلی، جستجوی جامع به زمان $O(q)$ نیاز دارد.

الگوریتم‌های عمومی (Generic Algorithms)

این الگوریتم‌ها تنها از عمل گروهی و تساوی عناصر استفاده می‌کنند و به نمایش باینری عناصر گروه وابسته نیستند.

۱- الگوریتم پولیگ–هلمن: اگر $q = \prod_{i=1}^{t} p_i^{e_i}$ فاکتورگیری q باشد، آنگاه می‌توان $x \bmod p_i^{e_i}$ را در هر زیرگروه از مرتبه $p_i^{e_i}$ محاسبه و سپس با قضیه باقی‌مانده چینی، $x \bmod q$ را بازسازی کرد. بنابراین،

$$\text{زمان محاسبه} = O\left(\sum_{i=1}^{t} \sqrt{p_i}\right),$$

و در بدترین حالت، توسط بزرگ‌ترین عامل اول q' تعیین می‌شود. این دلیل اصلی استفاده از گروه‌های مرتبه اول در رمزنگاری است.

۲- روش گام‌های کوچک/گام‌های بزرگ (Shanks):

با پیش‌محاسبه g^j برای $0 \le j < m$ (baby steps) و جستجوی $h \cdot g^{-im}$ برای $0 \le i < \lceil q/m \rceil$ (giant steps)، جواب را در $O(\sqrt{q})$ عمل گروهی و $O(\sqrt{q})$ حافظه پیدا می‌کند.

الگوریتم‌های محاسبه لگاریتم‌های گسسته

۳- الگوریتم رُهوی پولارد:

با شبیه‌سازی یک راه‌تصادفی در گروه و یافتن برخورد (collision) در یک تابع تصادفی-مانند، جواب را در $O(\sqrt{q})$ عمل گروهی و **حافظه ثابت** به‌دست می‌آورد.

ثابت شده است که هر الگوریتم عمومی برای DLP حداقل $\Omega(\sqrt{q})$ عمل گروهی نیاز دارد — بنابراین این دو الگوریتم از نظر پیچیدگی **بهینه** هستند.

➢ **اهمیت نمایش گروه**

از دیدگاه ریاضی، تمام گروه‌های چرخه‌ای از مرتبه q ایزومورف‌اند. با این حال، **نحوهٔ نمایش عناصر** تأثیر مستقیمی بر پیچیدگی محاسباتی دارد.

به‌عنوان مثال، در گروه جمعی $(\mathbb{Z}_q, +)$، مسئله

$$x \cdot g \equiv h \pmod{q}$$

در زمان چندجمله‌ای حل می‌شود، زیرا $x = h \cdot g^{-1} \bmod q$ قابل محاسبه است. این امکان‌پذیری ناشی از وجود ساختار **اضافی** (ضرب و الگوریتم اقلیدسی) روی مجموعه زیربنایی است.

الگوریتم‌های محاسبه لگاریتم‌های گسسته

الگوریتم‌های اختصاصی و پیامدهای رمزنگاری

در گروه‌هایی مانند $\mathbb{Z}_p^*$(با pاول)، الگوریتم **حساب اندیس** و سپس **الک میدان اعداد عمومی** (GNFS) مسئله DLP را در زمان زیرنمایی

$$L_p\left[\frac{1}{3}, c\right] = \exp\left((c + o(1))(\log p)^{1/3}(\log\log p)^{2/3}\right)$$

حل می‌کنند.

در مقابل، **در گروه‌های منحنی بیضوی عمومی هیچ الگوریتم زیرنمایی شناخته نشده است.**

بهترین حملات عمومی همچنان از مرتبه $O(\sqrt{n})$هستند که nمرتبه گروه منحنی است. این تفاوت بنیادی توضیح می‌دهد که چرا در **رمزنگاری منحنی بیضوی** (ECC) می‌توان از کلیدهایی با طول ۲۵۶~ بیت استفاده کرد که امنیتی معادل RSA با کلید ۳۰۷۲ بیتی فراهم می‌کنند و در نتیجه سیستم‌های کارآمدتری طراحی شده‌اند.

۱۰.۲.۱ الگوریتم پولیگ–هلمن

الگوریتم پولیگ–هلمن مسئله لگاریتم گسسته در گروه چرخه‌ای Gاز مرتبه qرا به مسائلی در زیرگروه‌های کوچک‌تر تقلیل می‌دهد، به‌شرط آنکه فاکتورگیری جزئی یا کامل qمعلوم باشد.
فرض کنید $q = \prod_{i=1}^{k} q_i$که در آن برای $i \neq j$. با تعریف

$$g_i = g^{q/q_i}, h_i = h^{q/q_i},$$

و با استفاده از لم ۱۰.۴، $\mathbf{ord}(g_i) = q_i$. بنابراین،

$$g_i^x = h_i \Rightarrow x \equiv x_i \pmod{q_i},$$

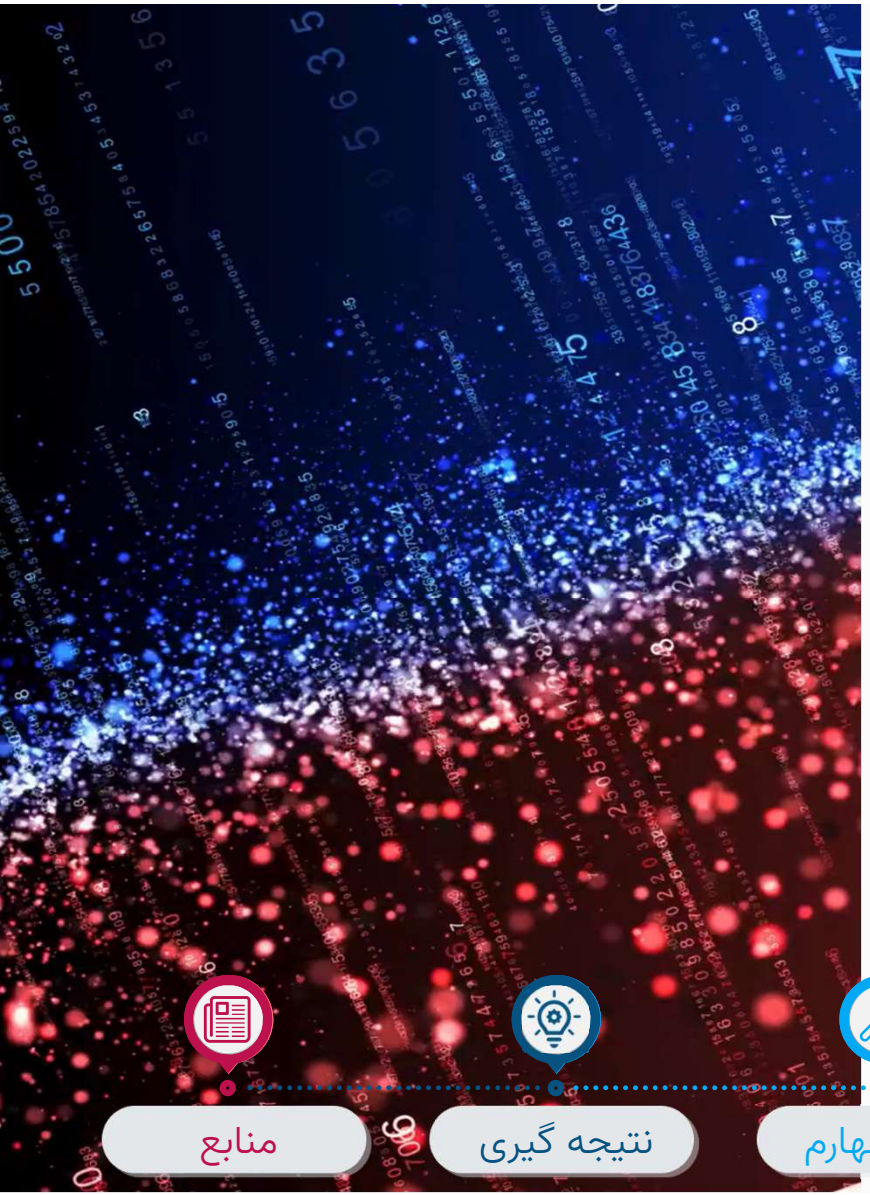
که در آن $x_i = \log_{g_i}(h_i)$در $\mathbb{Z}_{q_i}$محاسبه می‌شود.
با جمع‌آوری تمام مقادیر x_i، سیستم همنهشتی

$$x \equiv x_i \pmod{q_i}, i = 1, \dots, k$$

را حل می‌کنیم. با تعمیم قضیهٔ باقی‌مانده چینی، جواب یکتا $\mathbf{modulo}\ q$وجود دارد و به‌صورت کارآمدی قابل بازیابی است.
در نتیجه، پیچیدگی زمانی الگوریتم برابر است با:

$$T(q) = \sum_{i=1}^{k} T_{DLP}(q_i) + O(k \log q),$$

که در آن $T_{DLP}(q_i)$زمان حل $\mathbf{DLP}$ در گروه مرتبه q_iاست. اگر $q = \prod p_i^{e_i}$ فاکتورگیری اول کامل باشد، آنگاه بدترین زمان توسط بزرگ‌ترین $p_i^{e_i}$ تعیین می‌شود.



۱۰.۲.۱ الگوریتم پولیگ–هلمن

نتیجه‌ی رمزنگاری

برای جلوگیری از کارایی این الگوریتم، معمولاً از گروه‌هایی با مرتبه‌ی اول استفاده می‌شود تا $q' = q$ و هیچ کاهش پیچیدگی رخ ندهد.

۱۰.۲.۲ الگوریتم گام‌های کوچک/گام‌های بزرگ

الگوریتم **Shanks**(شناخته‌شده به‌عنوان Baby-Step/Giant-Step) یک روش قطعی برای حل مسئله لگاریتم گسسته در گروه‌های متناهی است که زمان اجرایی $O(\sqrt{q})$ و حافظه $O(\sqrt{q})$ دارد.

ایده اصلی: با نمایش $x = kt - i$ (که در آن $0 \le i < t$ و $t = \lfloor \sqrt{q} \rfloor$)، معادله $g^x = h$ را به‌صورت

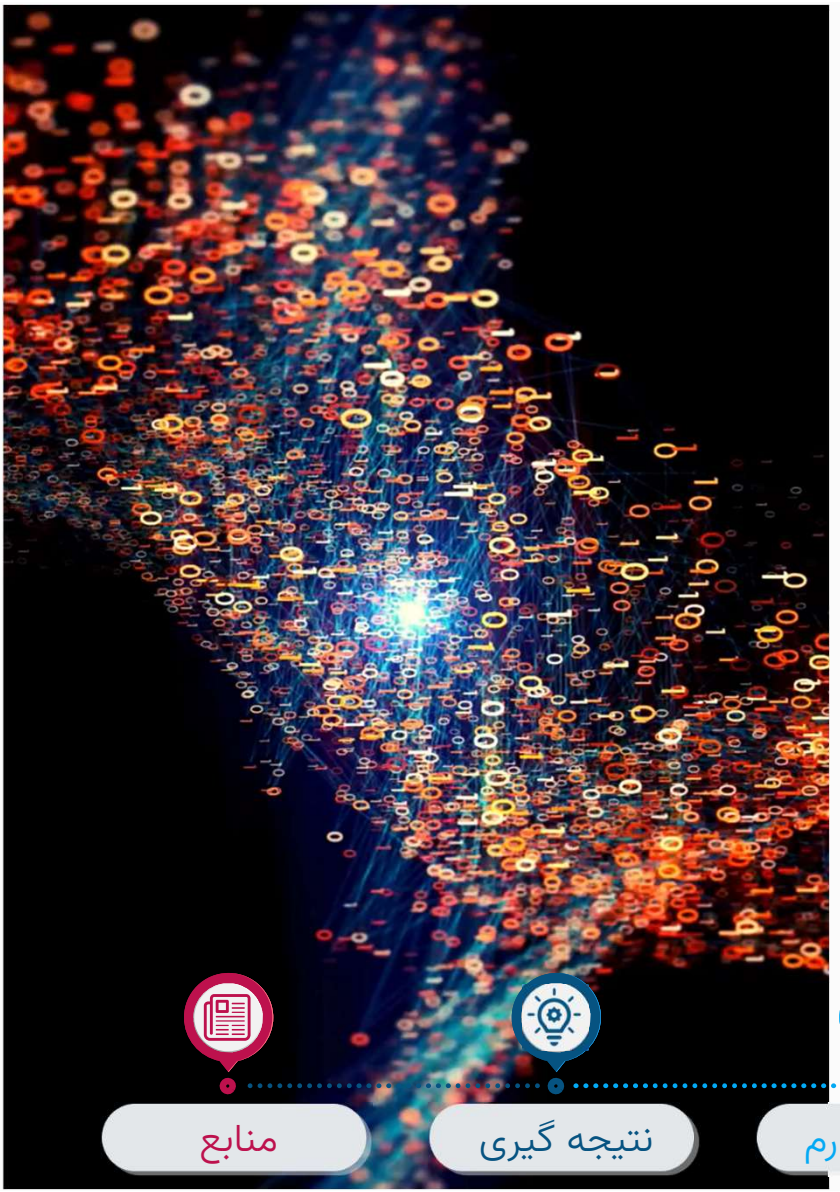
$$g^{kt} = h \cdot g^i$$

بازنویسی می‌کند. سپس:

- **گام‌های بزرگ**: مجموعه $\mathcal{G} = \{g^{kt} \mid 0 \le k \le \lceil q/t \rceil\}$ را محاسبه و مرتب می‌کند.
- **گام‌های کوچک**: برای هر $1 \le i \le t$، مقدار $h \cdot g^i$ را محاسبه و در $\mathcal{G}$ جستجو می‌کند.

اگر تطابقی یافت شود (مثلاً $h \cdot g^i = g^{kt}$)، آنگاه

$$x \equiv kt - i \pmod{q}.$$



۱۰.۲.۲ الگوریتم گام‌های کوچک/گام‌های بزرگ

پیچیدگی:

- تعداد عملیات گروهی: $O(\sqrt{q})$
- حافظه: $O(\sqrt{q})$
- زمان کلی: $O(\sqrt{q} \cdot \log q)$(با در نظر گرفتن مرتب‌سازی و جستجوی دودویی)

این الگوریتم **بهینه** در میان الگوریتم‌های عمومی است، چرا که ثابت شده هر الگوریتم عمومی برای DLP حداقل $\Omega(\sqrt{q})$ عمل گروهی نیاز دارد.

ALGORITHM 10.6
The baby-step/giant-step algorithm

Input: Elements $g, h \in \mathbb{G}$; the order q of $\mathbb{G}$
Output: $\log_g h$

$t := \lfloor \sqrt{q} \rfloor$
for $i = 0$ to $\lfloor q/t \rfloor$:
  compute $g_i := g^{i \cdot t}$
sort the pairs (i, g_i) by their second component
for $i = 1$ to t:
  compute $h_i := h \cdot g^i$
  if $h_i = g_k$ for some k, **return** $[(kt - i) \bmod q]$

۱۰.۲.۳ لگاریتم گسسته از طریق برخوردها

مسیری جایگزین برای حل مسئله لگاریتم گسسته (DLP)، با **حافظه ثابت** و پیچیدگی زمانی $O(\sqrt{q})$، از طریق کاهش به مسئله **یافتن برخورد** در یک تابع هش ساخته‌شده از g و h ممکن است.

تعریف تابع هش:

$$H_{g,h}: \mathbb{Z}_q \times \mathbb{Z}_q \to G, H_{g,h}(x_1, x_2) = g^{x_1} h^{x_2}.$$

اگر برخوردی یافت شود، یعنی

$$g^{x_1} h^{x_2} = g^{x_1'} h^{x_2'} \text{ و } (x_1, x_2) \neq (x_1', x_2'),$$

آنگاه (با فرض $h = g^x$) داریم:

$$g^{x_1 + x x_2} = g^{x_1' + x x_2'} \Rightarrow x(x_2 - x_2') \equiv x_1' - x_1 \pmod{q}$$

اگر $x_2 \not\equiv x_2' \pmod{q}$، آنگاه x که همان $\log_g h$ است.

۱۰.۲.۳ لگاریتم گسسته از طریق برخوردها

پیاده‌سازی با حافظه ثابت:

با ترکیب $H_{g,h}$ با یک تابع هش رمزنگاری $F: G \to \mathbb{Z}_q \times \mathbb{Z}_q$(مانند SHA-2)، تابع ترکیبی

$$H(k) = H_{g,h}\big(F(k)\big)$$

تعریف می‌شود. سپس با استفاده از **الگوریتم فضای کم جستجوی برخورد** (مانند الگوریتم Floydیا Brent)، برخوردی در Hدر زمان مورد انتظار $O\left(\sqrt{q}\right)$و حافظه $O(1)$یافت می‌شود.

نکته فلسفی:

این رویکرد یک دوگانگی بنیادی را آشکار می‌سازد:

ثابت شده که **سختی DLP ⇒ مقاومت به برخورد** $H_{g,h}$.

اما همین ساختار مجازی، **الگوریتم حمله‌ای** بر پایه یافتن برخورد می‌سازد.

این تناقض نیست؛ بلکه نشان می‌دهد که **کاهش‌های امنیتی** دو طرفه هستند: هر حمله به ساختار، مستقیماً به حمله به فرض پایه تبدیل می‌شود.

بخش سوم

حساب شاخص‌ها

حساب شاخص‌ها

یک روش **غیرعمومی** (non-generic) برای محاسبه لگاریتم گسسته در گروه چرخه‌ای $\mathbb{Z}_p^*$(برای p اول) است که در زمان **زیرنمایی** نسبت به $\log p$ اجرا می‌شود. برخلاف الگوریتم‌های عمومی (مانند رُهو یا baby-step/giant-step) که برای هر گروهی کار می‌کنند، این روش به ساختار عددی عناصر $\mathbb{Z}_p^*$(یعنی نمایش آن‌ها به‌صورت اعداد صحیح) وابسته است و از مفهوم **فاکتورگیری صحیح** بهره می‌برد.

الگوریتم حساب اندیس از نظر مفهومی شباهت زیادی به **الگوریتم الک درجه‌دو** (Quadratic Sieve) در فاکتورگیری دارد و نیز شامل دو فاز **پیش‌پردازش** و **حل موردی** است.

حساب شاخص‌ها

- **پیش‌فرض‌ها و نمادها**
- p: عدد اول.
- $g \in \mathbb{Z}_p^*$: مولد گروه (پایه لگاریتم).
- $h \in \mathbb{Z}_p^*$: عنصری که می‌خواهیم $x = \log_g h$ را بیابیم.
- B: یک کران بالا (smoothness bound).
- $\{p_1, p_2, \dots, p_k\}$: مجموعه تمام اعداد اول $\leq B$ (پایه فاکتوری).

مرحله ۱: پیش‌پردازش (Precomputation)

این مرحله **مستقل از** h است و تنها نیاز به p و g دارد. بنابراین می‌توان آن را یک‌بار انجام داد و از نتایج برای حل **چندین نمونه** از مسئله لگاریتم گسسته (با همان p و g) استفاده کرد.

در این مرحله، $\ell \geq k$ مقدار متمایز $x_1, \dots, x_\ell \in \mathbb{Z}_{p-1}$ پیدا می‌شوند به‌طوری که

$$g_i := [g^{x_i} \bmod p]$$

یک عدد B-**صاف** (B-smooth) باشد، یعنی تمام فاکتورهای اول آن در $\{p_1, \dots, p_k\}$ باشند. این مقادیر با انتخاب تصادفی x_i و بررسی صاف‌بودن $g^{x_i} \bmod p$ یافت می‌شوند.

حساب شاخص‌ها

پس از فاکتور کردن هر g_i، دستگاه معادلات زیر به‌دست می‌آید:

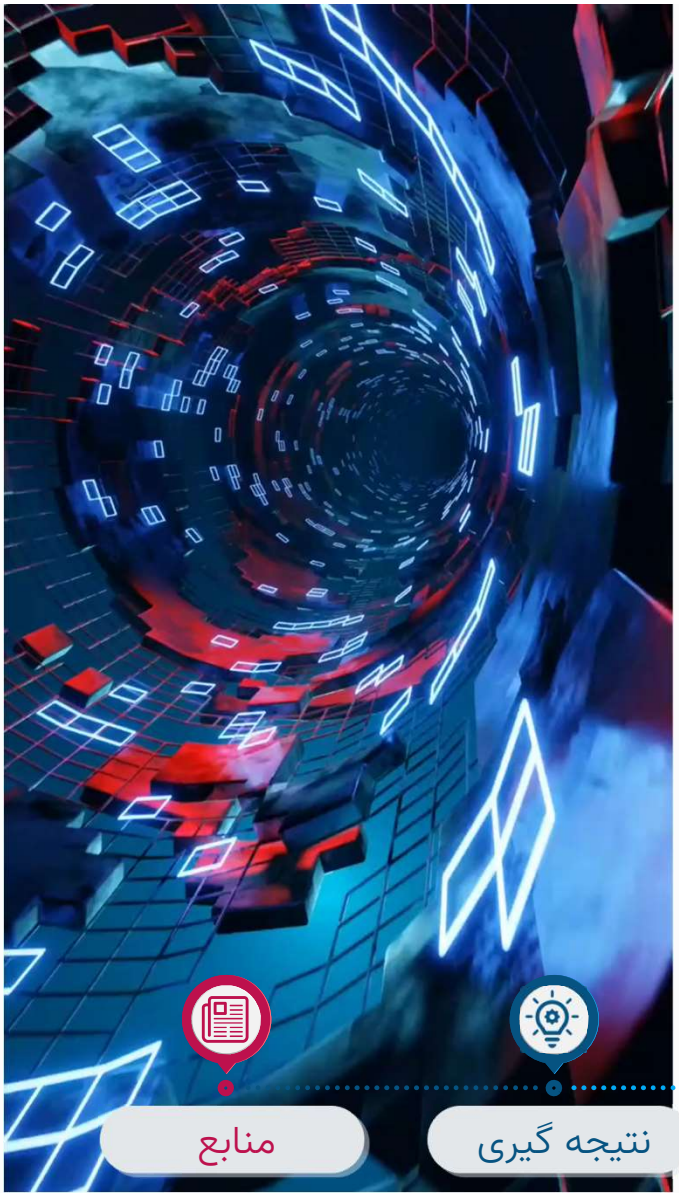
$$\begin{aligned} g^{x_1} &\equiv \prod_{j=1}^{k} p_j^{e_{1,j}} \pmod p \\ g^{x_2} &\equiv \prod_{j=1}^{k} p_j^{e_{2,j}} \pmod p \\ &\vdots \\ g^{x_\ell} &\equiv \prod_{j=1}^{k} p_j^{e_{\ell,j}} \pmod p. \end{aligned} \qquad (۱۰.۳a)$$

با گرفتن لگاریتم گسسته از دو طرف (نسبت به پایه g) و استفاده از خاصیت

$$\log_g(ab) = \log_g a + \log_g b \bmod (p-1)$$

، این معادلات به سیستم خطی زیر تبدیل می‌شوند:

$$\begin{aligned} x_1 &\equiv \sum_{j=1}^{k} e_{1,j} \cdot \log_g p_j \pmod{p-1} \\ x_2 &\equiv \sum_{j=1}^{k} e_{2,j} \cdot \log_g p_j \pmod{p-1} \\ &\vdots \\ x_\ell &\equiv \sum_{j=1}^{k} e_{\ell,j} \cdot \log_g p_j \pmod{p-1}. \end{aligned} \qquad (۱۰.۳b)$$



مرحله ۲: حل موردی (Individual Logarithm)

اکنون برای hداده‌شده، می‌خواهیم $\log_g h$را بیابیم. در این مرحله، یک مقدار x $\in \mathbb{Z}_{p-1}$به‌صورت تصادفی انتخاب می‌شود تا عدد

$$[g^x \cdot h \bmod p]$$

B-صاف باشد. فرض کنید:

$$g^x \cdot h \equiv \prod_{j=1}^{k} p_j^{e_j} \pmod{p}. \qquad (۱۰.۴)$$

با گرفتن لگاریتم گسسته از دو طرف:

$$x + \log_g h \equiv \sum_{j=1}^{k} e_j \cdot \log_g p_j \pmod{p-1}. \qquad (۱۰.۵)$$

در این معادله:xو e_j **معلوم** هستند. p_jها از مرحله ۱ شناخته‌شده‌اند. $\log_g$تنها مجهول $\log_g h$است.بنابراین می‌توان $\log_g h$را مستقیماً محاسبه کرد:

$$\log_g h \equiv \left(\sum_{j=1}^{k} e_j \cdot \log_g p_j - x \right) \bmod (p-1).$$

حساب شاخص‌ها

پیچیدگی و نکات کلیدی

- **پیش‌پردازش قابل اشتراک‌گذاری است**: یک‌بار برای جفت (p, g)انجام می‌شود و برای ه h قابل استفاده است.
- **پیچیدگی کلی**: زمان پیش‌پردازش و حل موردی هر دو از مرتبه زیرنمایی هستند:
- $$L_p\left[\frac{1}{2}, c\right] = \exp\left((c + o(1))\sqrt{\log p \log\log p}\right),$$

که سریع‌تر از الگوریتم‌های عمومی $O(\sqrt{p})$است.

- **وابستگی به ساختار عددی**: این الگوریتم فقط در گروه‌هایی که امکان **فاکتورگیری معنادار** از عناصر وجود دارد (مانند $\mathbb{Z}_p^*$) کاربرد دارد.
- **کاربرد در ECC ندارد**: در گروه‌های منحنی بیضوی، مفهوم «فاکتورگیری عدد صحیح» روی نقاط تعریف نمی‌شود، بنابراین حساب اندیس **کار نمی‌کند** همین دلیل اصلی برتری ECC در کارایی است.
- **نسخه‌های پیشرفته‌تر**: الگوریتم‌هایی مانند **الک میدان اعداد** (Number Field Sieve) برای DLP، همین ایده را با ساختارهای جبری پیچیده‌تر (میدان‌های عددی) بهبود می‌دهند و پیچیدگی را به $L_p[1/3, c]$کاهش می‌دهند.

حساب شاخص‌ها

جمع‌بندی نهایی

حساب اندیس یک **الگوریتم دو مرحله‌ای** است:

- **پیش‌پردازش** یافتن روابط خطی بین لگاریتم اعداد اول کوچک.
- **حل موردی** بیان h(یا ضریبی از آن) بر حسب همان اعداد اول و استفاده از روابط خطی.

✓ بهره‌برداری از **ساختار عددی** $\mathbb{Z}_p^*$(یعنی امکان فاکتورگیری اعداد صحیح) کلید کار است.

✓ **قابلیت اشتراک‌گذاری پیش‌پردازش** یک مزیت حمل‌محور (precomputation advantage) برای مهاجمان ایجاد می‌کند.

✓ **کارایی زیرنمایی** آن، **انگیزه اصلی** برای استفاده از گروه‌هایی مانند منحنی‌های بیضوی است جایی که چنین الگوریتمی وجود ندارد.

✓ این الگوریتم، **همزمان با الک درجه‌دو در فاکتورگیری**، نشان می‌دهد که *سختی فاکتورگیری و DLP از یک دسته نیستند* DLP در اینجا **آسان‌تر** است.

بخش ۴

طول کلیدهای توصیه‌شده

طول کلیدهای توصیه‌شده

انتخاب طول مناسب پارامترهای رمزنگاری مستلزم درک پیچیدگی بهترین الگوریتم‌های شناخته‌شده برای شکستن مسائل پایه (مانند فاکتورگیری یا لگاریتم گسسته) است. بدون این درک، ممکن است سیستمی طراحی شود که ظاهراً امن است، اما در عمل با حملات هوشمندانه‌تر از **brute-force** شکسته می‌شود.

معیار NIST:

- ✓ طول مؤثر **۱۱۲ بیت** تا سال **۲۰۳۰** برای بسیاری از کاربردها قابل قبول است.
- ✓ برای امنیت **فراتر از ۲۰۳۰**، **حداقل ۱۲۸ بیت** توصیه می‌شود.

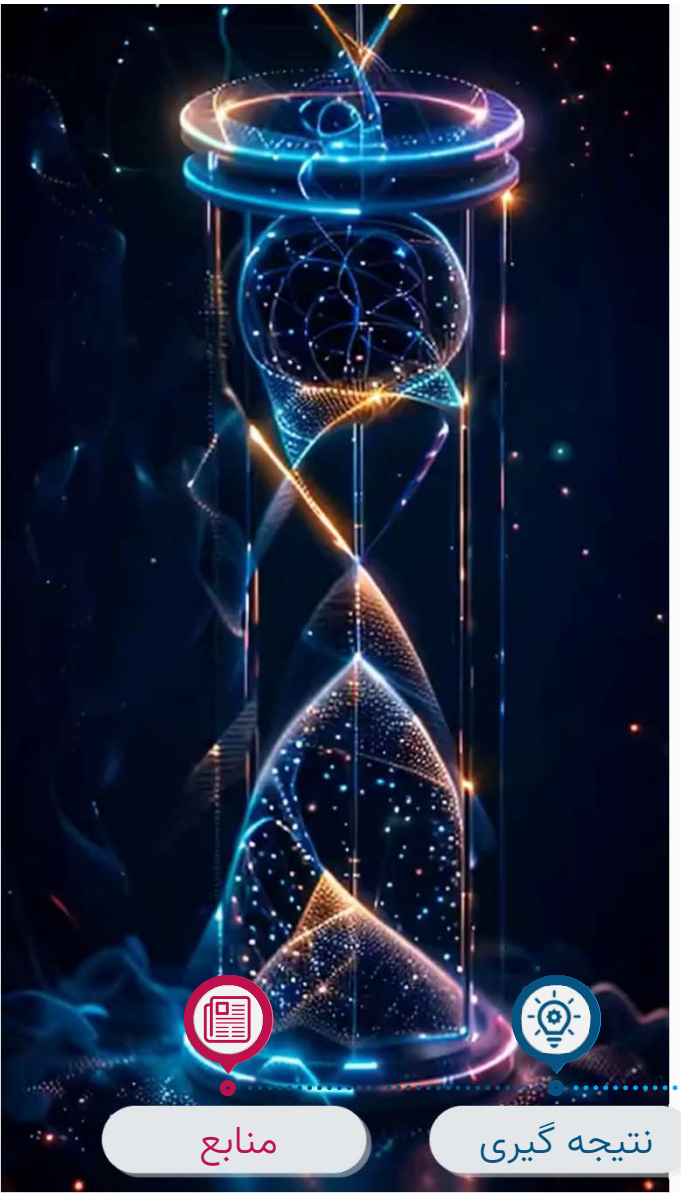
«طول کلید مؤثر» (Effective Key Length):

- مقدار n به‌صورتی که بهترین الگوریتم برای حل یک مسئله رمزنگاری در زمان تقریباً 2^n اجرا شود.
- این معیار، سختی یک مسئله را با سختی جستجوی جامع در یک سیستم کلید متقارن n-بیتی یا یافتن برخورد در یک هش 2n-بیتی (بر اساس حمله تولد) مقایسه می‌کند.

طول کلیدهای توصیه‌شده

جدول زیر طول‌های پارامترهای مختلف را برای دستیابی به سطوح امنیتی معادل n-بیت مؤثر خلاصه می‌کند:

DLPدر منحنی بیضوی (طول q)	DLPدر $\mathbb{Z}_p^*$(طول p, طول q)	RSA(طول N)	امنیت مؤثر (بیت)
۲۲۴	$p = 2048, q = 22$	2048	112
256	$p = 3072, q = 25$	3072	128
384	$p = 7680, q = 384$	7680	192
512	$p = 153 \quad , q = 512$	15360	256



طول کلیدهای توصیه‌شده

تحلیل تفاوت‌های طول پارامترها

۱. منحنی‌های بیضوی (ECC)

- هیچ الگوریتم زیرنمایی برای DLP در گروه‌های منحنی بیضوی مناسب شناخته نشده است.
- بنابراین، بهترین حملات، الگوریتم‌های عمومی مانند رُهوی پولارد هستند که در زمان $O(\sqrt{q})$ اجرا می‌شوند.
- برای دستیابی به امنیت n-بیتی، باید:

$$\sqrt{q} \approx 2^n \Rightarrow q \approx 2^{2n} \Rightarrow \text{طول } q = 2n \text{ بیت}.$$

- ← همین دلیل است که برای امنیت ۱۲۸ بیتی، منحنی‌ای با مرتبهٔ ۲۵۶ بیتی کافی است.

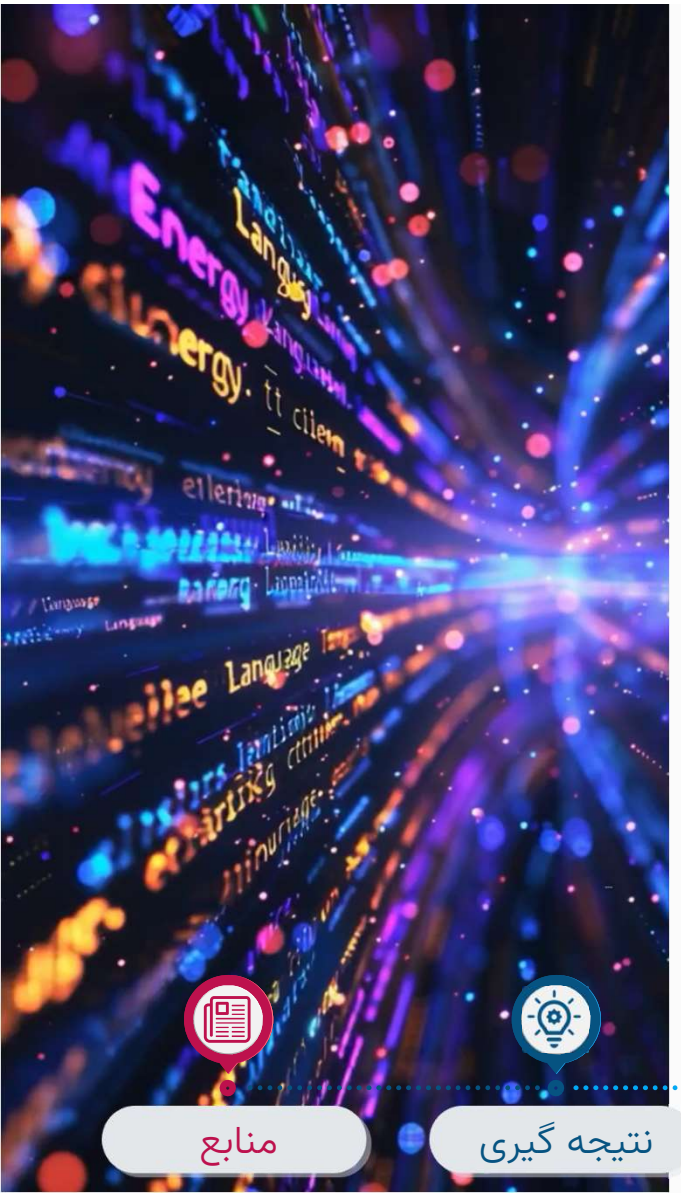
۲. گروه‌های $\mathbb{Z}^*_p$

در اینجا، دو الگوریتم رقابت می‌کنند:

- الگوریتم‌های عمومی (مانند رُهو) → پیچیدگی $O(\sqrt{q})$ → نیاز به $q \approx 2^{2n}$.
- الگوریتم‌های اختصاصی (حساب اندیس، الک میدان اعداد) → پیچیدگی زیرنمایی نسبت به $\log p$.

➢ بنابراین، برای تعادل امنیتی، باید p را آن‌قدر بزرگ انتخاب کرد که زمان حملهٔ زیرنمایی روی p تقریباً برابر با زمان حمله عمومی روی q باشد.

➢ این نیاز، طول p را به‌شدت افزایش می‌دهد (مثلاً برای امنیت ۱۲۸ بیتی، $p = 3072$ بیت در حالی که $q = 25$ بیت است).



طول کلیدهای توصیه‌شده

۳. RSA

- ✓ امنیت RSA مبتنی بر سختی فاکتورگیری $N = pq$ است.
- ✓ سریع‌ترین الگوریتم، الک میدان اعداد عمومی (GNFS) است که در زمان زیرنمایی نسبت به $\log N$ اجرا می‌شود.
- ✓ برای معادل‌سازی با امنیت n-بیتی، باید طول N را آن‌قدر بزرگ انتخاب کرد که زمان GNFS $\approx 2^n$.
- ✓ این منجر به طول‌های بسیار بزرگ N می‌شود (مثلاً ۳۰۷۲ بیت برای امنیت ۱۲۸ بیتی).

نتیجه گیری

نتیجه گیری

۱. سختی مسائل رمزنگاری نسبی است، نه مطلق

۲. الگوریتم‌های عمومی در برابر همه گروه‌ها کار می‌کنند، اما بهینه‌اند

۳. ساختار گروه، امنیت را تعیین می‌کند

۴. منحنی‌های بیضوی بهینه‌ترین تعادل امنیت/کارایی را فراهم می‌کنند

منابع

منابع

[1] J. Katz and Y. Lindell, Introduction to Modern Cryptography, 3rd ed. Boca Raton, FL, USA: CRC Press, 2021.

با تشکر از توجه شما