# ناگفته‌هایی از رمزنگاری که باید گفته می‌شد!

نویسنده: محمدکمال

# مقدمه

علم رمزنگاری یکی از مهمترین علم‌های دنیای کامپیوتر هست. درواقع خیلی ساده بگم که بدون رمزنگاری، عملاً شما امنیتی در دنیای کامپیوتر نخواهید داشت!

همه باید یه آشنایی مختصر با این علم داشته باشن. در حدی که کارشون راه بیوفته و یکم این دنیا رو بشناسن.

- چرا؟

+ چون اگر نشناسین، هیچ‌وقت درک مناسبی از امنیت و حریم شخصیتون نخواهید داشت. پایه و اساس امنیت و حریم شخصی شما، روی رمزنگاری بنا شده.

اما یه مشکل بزرگ!

اکثر منابع موجود در اینترنت یکی از سه حالت کلی هستن:

۱. بسیار ریاضیاتی و سخت.
۲. پر از توصیه‌های اشتباهی که ناشی از دانش کم رخ داده.
۳. خوب ولی ناکامل. (مباحثی که یه نفر باید بدونه رو به صورت کامل و یک‌جا نیووردن. مثلاً صرفاً یه بخش کوچیکی رو توضیح دادن و دوره‌ای کامل براش وجود نداره!
۴. رایگان و آزاد نیستن!

خب یه اصطلاحی هست که میگه «چه مشکلی رو قراره حل کنی؟»[1].

خب من قراره مطالبی رو که افراد نیازه درباره رمزنگاری بدونن رو یک‌جا جمع کنم و با توضیحی ساده - و نه فارغ از نظریه اعداد و آنالیز عددی و احتمالات پیچیده- بیان کنم و در عین حال نسبتاً دقیق باشه!

البته هر از چندگاهی تلاش کردم که مباحث ریاضی‌طور هم بیارم ولی شما می‌تونین اون قسمتا رو رد کنین و فقط قسمتای «*نتیجه*» انتهایی رو بخونین.

یکی از مهم‌ترین چیزایی که سعی کردم رعایت کنم آینه که سعی کردم نظراتم بر اساس منابع معتبر باشه. متأسفانه در حوزه رمزنگاری، اکثر چیزایی که توی اینترنت می‌خونیم، چیزای اشتباهی هستن و ناشی از سلیقه‌های شخصی افراد نامتخصص هست. این خیلی مهمه که متوجه شیم چه چیزایی ممکنه یه رمزنگاری رو به خطر بندازه؟! چرا الگوریتم رمزنگاری باید شفاف باشه و... .

من سعی کردم تقریباً تمام چیزایی که میگم، یه ارجاع به منابع معتبر داشته باشه و برخلاف اکثر آموزشا، سلیقه شخصی رو دخیل ندم که توصیه‌ای اشتباه کنم!

رمزنگاری سخته! یه فرد که حتی دکترا داره هم قرار نیست بتونه توصیه خوبی کنه. پس باید از افراد معتبرتر و باتجربه‌تر و باعلم‌تر کمک گرفت! کاری که من در این آموزش انجام دادم.

تعداد صفحات به چشم زیاده، ولی مطمئن باشین یه روزه هم می‌تونین تقریباً تمام چیزایی که مدنظرم بود که یاد بگیرین رو یاد بگیرین!

---

1 What problem are you solving?

خب آیا تونستم به این اهداف برسم؟!

شما باید نظر بدین! اگر نظری داشتین حتماً بگین تا در نسخه‌های بعدی، بهترش کنم. این کتابچه ۱۲۰ صفحه‌ای، رایگان و آزاده! شما می‌تونین در بهترکردنش کمک کنین. اینطوری یه منبعی خوب و آزاد و رایگان در اختیار همه قرار می‌گیره و شما هم قسمتیشو بهتر کردین!

# دونیت

این مطالب به صورت آزاد و رایگان پخش شده، پس اگر گسترش رایگان علم براتون مهمه و نظرتونه که کارم خوب بوده که به صورت رایگان آموزش میدم و« نباید علم به صورت پولی باشه که بگیم اگر پول داری، بهت آموزش میدم؛ اگر پول نداری، باید از آموزش عقب بمونی»، می‌تونین به من کمک مالی (دونیت) کنین:

https://zarinp.al/mkamal

درواقع گسترش علم رایگانه، ولی تولیدش خیر! بالاخره منم مثل شما زندگی دارم و خب برای گذران زندگی، نیاز به درآمد دارم. پس خوشحال میشم که اگر دوست داشتین، بهم دونیت کنین تا منم بتونم این کارها رو ادامه بدم. (همونطور که یه کتاب **رایگان** ۲۰۰ صفحه‌ای آموزش پایتون رو نوشتم و توی گیتهابم موجوده!)

# حق نشر



۲ البته مشخصاً مطالب یا عکس‌هایی که از جاهای دیگر نقل یا آورده شده‌اند و یا متعلق به من نیستند، باید طبق لایسنس خودشان انتشار یابند.

# کدگذاری

## ASCII (American Standard Code for Information Interexchange):

درواقع برای تبادل اطلاعات، اومدن گفتن خب ما یه کد ۸ بیتی (یعنی شامل ۸ تا خونه که هر خونه میتونه صفر یا یک باشه) اختراع می‌کنیم که به هر کرکتر یه عدد اختصاص بدیم. با ۸ بیت، چند کرکتر رو می‌تونیم نامگذاری کنیم؟ خب اگر unsigned درنظر بگیریم، میشه ۲ به توان ۸ که میشه ۲۵۶ تا. یعنی از کمترین که همش صفر هست یعنی ۰۰۰۰۰۰۰۰ تا بیشترین که همش یک هست یعنی ۱۱۱۱۱۱۱۱ که یعنی از ۰ تا ۲۵۵.

توی برنامه‌نویسی خیلی با این ASCII ها سر و کار داریم. چون ببینین کامپیوتر که نمیفهمه abcd چیه دیگه! ولی ۰ و ۱ رو میفهمه. پس ما این a رو میگیم طبق جدول ASCII که خودتونم میتونین برید توی اینترنت سرچ کنین، ۹۷ هست. پس ۹۷ رو تبدیل می‌کنیم به باینری و ذخیره‌اش می‌کنیم.

- خب حروف انگلیسی که ۲۶ تا حروف کوچیکه و ۲۶ تا بزرگ و اگر اعداد ۰ تا ۹ هم حساب کنیم (که ده تا هستن)، مجموعاً میشه ۲۶+۲۶+۱۰ یعنی ۶۲ تا. چرا پس ۲۵۶ تا اختصاص دادن؟

+ چون بقیه کرکترهای کنترلی هستن. مثل \n که نشون‌دهنده خط جدید یا new line هست.

بعدشم کرکترهای دیگه مثل « ! @ # $ % ^ & * ( ) _ − + / . » و... هم هستن.

برای دیدن جدول ASCII سرچ کنین «ASCII table».

اما ASCII یه مشکل یا بهتره بگم یه محدودیت داره. اون چیه به نظرتون؟ یکم فکر کنین.

+ خب ببینین با ASCII فقط ۲۵۶ تا چیز رو می‌تونستیم نشون بدیم. از اسمش هم معلومه! « American». خب پس زبونای دیگه مثل فارسی چی؟!

مثلاً «ک» زبون فارسی رو با U+06A9 نشون میدن. برای پرینت کردنش:

```
printf("\U000006A9")
```

توجه: نباید برای زبون فارسی از حرف عربیش استفاده شه. (U+0643)

برای چیزای مثل اینکه باید رعایت شه، به سایت زیر که مال Unicode هست برید:

https://cldr.unicode.org/translation/language-specific/persian

تذکر: ASCII و Unicode، یه نوع کدگذاری هستن و رمزنگاری نیستن!

نکته! توی برنامه‌ها مثلاً LibreOffice یا Microsoft Office میتونین با زدن Alt + x کد یونیکد اون کرکتری که انتخاب کردین رو ببینین.[۳]

---

۳ این ترفند رو از وبسایت https://www.computerhope.com/jargon/u/unicode.htm یاد گرفتم. کلاً وبسایت Computer Hope خیلی خوبه و توضیحات خیلی ساده‌ای و روانی داره.

# Cryptography

## چرا باید درباره cryptography دانش داشته باشم؟!

- خب خیلی خلاصه بگم، پایه و اساس امنیت شما توی اینترنت به رمزنگاری بستگی داره. درواقع اگر رمزنگاری نباشه، تقریباً میشه گفت اطلاعات شما رو هواست!
- یه نفر می‌تونه اطلاعات بانکی شما رو در خرید اینترنتی بدزده.
- همه می‌تونن برن تو گوشیتون و برن توی گالریتون.
- آدما می‌تونن جای شما پیام به بقیه بفرستن.
- می‌تونن به جای شما به یکی دیگه زنگ بزنن و پیام بفرستن
- بقیه می‌تونن ببینن وارد چه وبسایتایی شدین و چیکارا کردین و چیا روی صفحه دیدین! عیناً چیزی که شما دیدین رو می‌تونن ببینن.
- اینکه شما می‌تونین بدون اینکه بقیه بفهمن چی رد و بدل می‌کنین، حرفای خصوصی توی چت به بقیه بزنین.
- با رمزنگاری، شما یه فایلی رو دانلود می‌کنی و مطمئن میشی که دستکاری نشده و مخرب نیست.
- با رمزنگاری، رمزارزها رو داریم! چیزایی مثل بیت‌کوین.
- اینکه اسرار نظامی یه کشور چه‌جوری انتقال پیدا کنه که به دست بقیه نیوفته
- اینکه شما یه چیزی رو سرچ کنی و گوگل نفهمه تو کی بودی و چه سرچی کردی ولی باز بتونه بهت جواب بده. بله درست متوجه شدین! نه بفهمه کی بودین و نه بفهمه چی سرچ کردین ولی بازهم بتونه جواب بده بهتون[4] (البته هنوز این پیاده‌سازی نشده)

همه و همه رو مدیون cryptography هستین. (علم مربوط به ارتباط امن)

حالا دلیل این چیزا رو بعداً می‌فهمیم.

4 Privately Outsourcing Computation

# هش (Hash)[5]

فرض کنین من یه وبسایتی دارم که مردم میان توش اکانت میسازن و پسورد وارد میکنن. خب این پسوردایی که وارد میکنن من باید چجور ازشون محافظت کنم؟ پسوردها و یوزرنیم‌ها رو همینطور توی یه فایل اکسل نگه دارم؟ به نظرتون خطرناک نیست؟ اگر یه هکر به سروری شرکت نفوذ کنه، فایل رو برمیداره و عملاً همه پسوردا و نام‌های کاربری رو داره! پس من اگر به صورت زیر داده‌هارو نگه دارم، خیلی بد میشه دیگه:

| Password | Username |
|---|---|
| Alex256[6] | alex |
| James512 | james |
| maria1024 | maria |

طرف راحت به تمام حساب‌ها دسترسی پیدا می‌کنه.

پس باید چیکار کنم؟ اینجا بحث یه چیزی به نام «هش (Hash function)» مطرح میشه.

## هش چیه؟

i) هش یه الگوریتم ریاضی **یکطرفس** که هر مقداری بهش بدی، یه متن عجیب غریب بهت تحویل میده. مثلاً من اگر بهش بدم hi، بهم متن زیر رو میده:

```
8f434346648f6b96df89dda901c5176b10a6d83961dd3c1ac88b59b2dc327aa4
```

خیلی عجیبه نه؟! درواقع میاد یه سری الگوریتم ریاضی روی این «hi» انجام میده و درنهایت به ما اون string و متن طولانی و عجیب رو میده. و از اون متن عجیب هم نمیشه رسید به متن اصلی! چرا؟ چون یکطرفس!

ii) هر متنی، هش مخصوص به خودشو داره و با متنای دیگه هشش فرق می‌کنه. (متن متفاوت بدیم به الگوریتم هش، خروجی هم متفاوت میشه و دو ورودی مختلف، هش یکسانی **ندارن**.) مثلاً[7]:

hello:

```
2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
```

Jadi:

```
b8481276057a02977b6f7ada7c421335228fc8467bdba23d6a6570c7a32bd041
```

---

۵ به طور کلی ما فعلاً بحثمون روی هش‌هایی هست که برای رمزنگاری استفاده میشن (cryptographic hash). پس چیزایی که می‌گیم، برای هش‌های عادی لزومی نیست! بلکه برای هش‌های cryptographic صحیحه.

۶ خب اولاً هیچ‌وقت همچین پسوردایی نگذارین. بعداً به بخش پسورد میرسیم. فقط فعلاً بدونین پسورد بالای ۲۰ کرکتر.

۷ با دستور زیر توی ترمینال لینوکس میشه هش یه متن رو حساب کرد:

```
Printf “TEXT YOU WANTED” | sha256sum
```

iii) با کوچک‌ترین تغییری در ورودی، خروجی هش هم به شدت تغییر میکنه. یعنی اگر من به جای hi، ورودی Hi رو بدم، به هش زیر میخورم که بسیار بسیار متفاوت‌تره:

```
hi => 8f434346648f6b96df89dda901c5176b10a6d83961dd3c1ac88b59b2dc327aa4
Hi => 3639efcd08abb273b1619e82e78c29a7df02c1051b1820e99fc395dcaa3326b8
```

دیدین؟ من فقط یک حرف رو از حرف کوچیک تبدیل به حرف بزرگ کردم؛ چقدر تغییر کرد هش ما! خصوصیت هش هم همینه. باید با کوچکترین تغییری (حتی اضافه‌شدن یک نقطه)، هش به شدت تغییر کنه!

iv) نه تنها نمیشه رسید به متن اولیه، بلکه حتی اینطوریه که من نمی‌تونم حدس بزنم که چیو باید بدم که چی خروجی بده! یعنی نمی‌تونم فکر کنم که بگم اگر اینجای متن رو عوض کنم، خروجی اونجاش فلان چیز میشه! این خیلی مهمه که هش قابل حدس‌زدن نباشه. اگر اینطور بود من می‌فهمیدم اگر فلان جاش تغییر بدم، فلان چیز تولید میشه ولی اینطور چیزی ممکن نیست! درواقع من نمی‌تونم تصویر خروجی رو حدس بزنم![8]

## مرور!

یه دور خصوصیات هش cryptographic رو مرور می‌کنیم. این کنار دستتون باشه که در ادامه و مخصوصاً قسمت حمله‌های مربوط به هش خیلی باهاش کار داریم و ازتون می‌خوام با استدلال به این‌ها بتونین به سؤالات پاسخ بدین!

۱. هش یک طرفس. نمیشه از خروجی فهمید ورودی چی بوده.[9]
۲. هش یه چیز یکسانه. یعنی هزار بار دیگه توی هزار دستگاه مختلف هم حسابش کنم، چون الگوریتم ریاضیش یکیه همون رو بهم میده. (یه تابع ریاضیه دیگه!)[10]
۳. با کوچک‌ترین تغییر توی ورودی، هش کامل عوض میشه.[11]
۴. با هر سایز ورودی، خروجی سایزش ثابته.[12]
۵. نشه فهمید که من با چه نوع ورودیی، خروجی مورد نظرم رو می‌تونم بگیرم.[13]
۶. دو نوع ورودی مختلف، خروجی یکسانی ندن![14]

8 Pre-image resistant
9 one-way
10 Deterministic
۱۱ به طور میانگین ۵۰ درصد هش کامل عوض میشه. مراجعه شود به:
“2.4 One-Way Hash Functions” from “Applied Cryptography” by “Bruce Schneier”.
12 Fixed-size (بعضی هشا)
13 Pre-image resistant
14 Hash colission

خب خب! ویژگی‌های بالا (مخصوصاً ۱ و ۵)، ویژگی‌های خیلی خوبی هستن که میشه ازشون استفاده کرد که بتونیم پسوردها رو به صورت امن‌تری[۱۵] ذخیره کنیم.

فرض کنین پسورد من «hi» بود. حالا من به جای اینکه خود «hi» رو ذخیره کنم، هشش رو ذخیره می‌کنم. اینطوری اگر اطلاعات شرکت هم لو بره، هش لو رفته. و چون یه طرفس، کسی نمی‌تونه از اون متن عجیب غریب برسه به «hi» و پسورد من رو بفهمه!

| username | password[16] |
|---|---|
| alex256 | x20m0pty3cn49mc2qokc9m8243n3cqs w08wqluija43ushkw8 |
| james512 | nx8964umec90wxmc3xq8tn39c74923yr y732n49jwe8mqx09 |
| maria1024 | mc2qejxm3mc334cn04nt234n890fj4m0 89qowxku8r23cnem |

*متوسط:* خب این مکانیزم هش پسورد برای ورود چه‌جوری کار میکنه؟

من یه وبسایت دارم. شما میای تو قسمت ثبت‌نام وبسایت، قسمت پسورد، پسورد رو می‌نویسین. پسورد به سرور شرکت انتقال داده نمیشه. **توی همون صفحه ثبت‌نام**، هش پسورد حساب میشه و هش انتقال داده میشه به سرور و **هش** (و نه پسورد) توی کامپیوتر سرور ذخیره میشه. حالا جدول اینطور میشه:

| username | password |
|---|---|
| alex256 | x20m0pty3cn49mc2qokc9m8243n3cqs w08wqluija43ushkw8 |
| nx8964umec90wxmc3xq8tn39c74923yr y732n49jwe8mqx09 | james512 |
| maria1024 | mc2qejxm3mc334cn04nt234n890fj4m0 89qowxku8r23cnem |

حالا توی صفحه ورود، شما وقتی یوزرنیم و پسورد وارد کنین، دوباره هش حساب میشه و هش به سمت سرور میره. سرور چک می‌کنه ببینه هش حساب‌شده، با هشی که قبلاً ذخیره کرده بود یکسانه یا نه؟ اگر آره، وارد میشین. باحال بود نه؟ (این البته کامل نیست و نباید اینطوری به کار ببرین! چون ناامنه هنوز! مکانیزم رو به مرور زمان کامل‌تر می‌کنیم و یه مثال دنیای واقعی توی قسمت *نحوه لاگین (شامل هش پسورد) و رمزنگاری در* Tutanota بیان میشه.

---

۱۵ نه لزوماً امن امن! چون کلی چیز دیگه هم باید رعایت شه!

۱۶ هش‌های نوشته‌شده، واقعی نیستن! دستی یه چیزی همینطوری نوشتم.

### پیشرفته:

الگوریتم‌های هش چیز خیلی عجیب غریبی نیستن! بیایم یه الگوریتم هش بسازیم! مثلاً من میگم الگوریتم هش من اینطوریه که به تمام حروف الفبا، یه عدد اختصاص میده. یعنی:

| alphabet | a | b | c | d |
|---|---|---|---|---|
| Number associated | 1 | 2 | 3 | 4 |

بعد میگم هش یه متن برابر میشه با جمع این اعداد. یعنی:

abd => 1 + 2 + 4 = 7

حالا کارهای دیگه هم می‌تونم کنم. مثلاً بگم اگر جمع عدیی بزرگ‌تر از اعداد یه رقمی شد، بیا بر مثلاً ۷ باقی‌مانده بگیر و هش رو بنویس:

aadae => 1 + 1 + 4 + 1 + 5 = 13 → get mod → 12 % 7 = 5

جمع شد ۱۲. چون بیشتر از یه رقمی شد، بر ۷ باقی‌مانده گرفتم و شد ۵.

یه ایده‌هایی می‌تونم بگیریم که وقتی می‌گیم هش یه‌طرفس یعنی چی. اصلاً یه‌طرفه بودن چطور ممکنه؟ مثلاً اینجا هش من یه حالتی یه طرفه بودن خیلی خیلی ضعیفی داشت. چون کسی که ۵ رو ببینه، نمی‌فهمه چی بوده.

۱. آیا هش در حالت عادی ۵ بوده؟ اگر آره، ورودی چی بوده؟ «aaaaa» یا «aaab» یا «baaa» یا «ad» یا چی؟! از کجا بفهمم چی بوده؟!

۲. آیا باقی‌مونده گرفته شده که شده ۵؟ اگر آره، چی بوده؟ ۱۲ بوده؟ ۱۹ بوده؟ ۲۶ بوده؟ چی بوده؟!

خلاصه آره! خواستم بگم ساخت یه چیزی که یه طرفه شه، خیلیم کار سختی نیست![۱۷]

اما هش خودساخته من، یه سری مشکل داره! مثلاً:

۱- مگه همه متنا انگلیسی هستن؟ پس فارسی چی؟ اسپانیایی چی؟

۲- اصلاً گیریم که مورد بالا رو بیخیال شیم! این هش اصلاً قابل اطمینان نیست! چون جمع ac با جمع bb برابره! پس با دادن دو مقدار متفاوت، هش یکسان می‌گیریم! این خیلی خیلی بده! مورد ۶ رو یادتونه؟ قرار بود دو ورودی متفاوت، خروجی یکسانی ندن. اگر بدن، بهش میگن hash collision.

## Birthday problem

یه مسأله خیلی معروف هست که به نام مسأله تاریخ تولد (Birthday problem) مشهوره.

این مساله رو با عمد اوردم. به دو دلیل:

۱- فهمیدن یه مساله مهم در دنیای هش.

---

۱۷ مثلاً XOR خیلی می‌تونه کمکمون کنه. درواقع بیشتر با همین XOR و Rotate کردن و... اتفاق میوفته.

۲- بفهمیم چرا رمزنگاری‌های خودساخته یا هش‌های خودساخته امن نیستن و اکثراً درست کار نمیکنن و مشکلاتی دارن و چرا ما می‌گیم هش‌ها و رمزنگاری‌ها باید توسط ریاضی‌دان‌ها و برنامه‌نویس‌هـا و کسـانی که منطق کامپیوتری دارن چک شه؟

خب بریم سراغ خود مساله:

((چند درصد احتمال داره توی یه کلاس ۲۰ نفره، هیچ دونفری تاریخ تولدشون یکسان باشه؟ یعـنی روز تولدشون با هم متفاوت باشه.))

در نگاه اول شاید سریع جواب بدیم ۲۰/۳۶۵.

اما جواب این نیست! بیایم با ریاضی و احتمال بررسیش کنیم:

نفر اول چند حالت داره برای روز تولدش؟ ۳۶۵ روز داریم، پس ۳۶۵ روز.

نفر دوم برای اینکه تولدش با نفر اول توی یه روز نباشه، چند حالت داره؟ ۳۶۴ حالت.

پس تا اینجا اگر بخوایم همه شرط‌ها درست باشن، چقدر احتمال داره؟

$$(\frac{365}{365})\times(\frac{364}{365})=(\frac{365\times364}{(365)^2})\simeq0.997=\%\,99.7$$

پس برای دو نفر، ۹۹.۷ درصد احتمال داره توی یه روز متولد نشده باشه.

برای سه نفر چی؟

نفر سوم ۳۶۳ حالت داره. خب بریم حسابش کنیم:

$$(\frac{365}{365})\times(\frac{364}{365})\times(\frac{363}{365})=\frac{365\times364\times363}{365^3}=\frac{\frac{365!}{362!}}{365^3}=\frac{\frac{365!}{(365-3)!}}{365^3}\simeq0.992=\%99.2$$

توضیح قسمت آبی رنگ و **بزرگ‌شده** (بولدشده):

اومدم ۳۶۵ * ۳۶۴ * ۳۶۳ رو به صورت فاکتوریلی نوشتم.

- چرا؟

+ معمولاً توی احتمال و این‌ها سعی میکنیم عبارت رو به صورت ساده‌تر بنویسیم که بعداً بتونیم از توش فرمول کلی به دست بیاریم. ما نمیتونیم برای همه حالات بیایم از اول بنویسـیم! ولی اگـر راه کلی رسـیدن بهش رو بلد باشیم، فرمول کلی رو بلد باشیم، جواب رو راحت به دست میاریم. یه فرمول کلی بـرای عبـارت بسازین:

$$\frac{\frac{365!}{(365-n)!}}{365^n}=\frac{365!}{(365-n)!\cdot365^n}$$

این احتمال اینکه دو نفر تاریخ تولدشون یکسان نباشه. حالا برای ۲۰ نفر چجور میشه؟

$$\frac{365!}{(365-20)!\cdot365^{20}}\simeq0.589\simeq\%59$$

یعنی ۵۹ درصد احتمال داره توی یه روز نباشن. یعنی درواقع ۴۱ درصد احتمال داره باشن!!! عجیبه نه؟ در نگاه اول شاید به نظرمون میومد که بابا ۲۰ نفر تعداد خیلی کمی هستن! اما میبینیم با ۲۰ نفر، احتمـال ۴۱ درصد وجود داره حداقل دو تاریخ تولد مثل هم باشه.

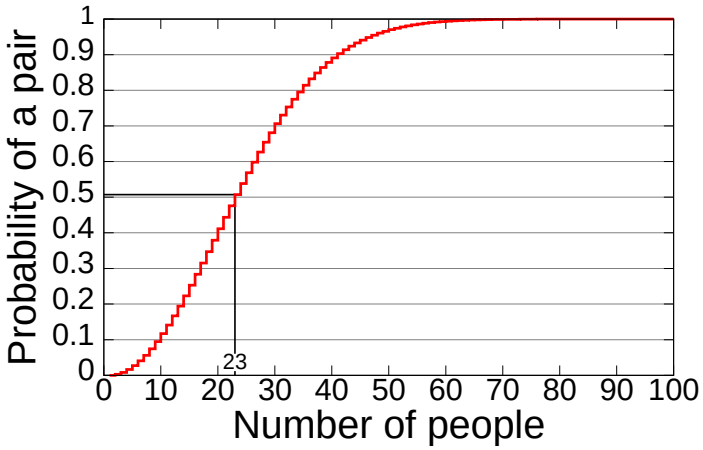
با ۲۳ نفر به نظرتون چند درصد احتمال داره تاریخ تولدشون یکسان باشه؟
۵۰.۷ درصد!!! یه دفعه چقدر درصد زیاد شد! از ۴۱ رسیدیم به ۵۰.
حالا برای ۷۰ نفر چی؟

$$\frac{365!}{(365-70)!\cdot 365^{70}} \simeq 0.00084 \simeq \%0.084$$

۹۹.۹۱۶ درصد!!! خیلی عجیب شد نه؟ با ۷۰ نفر اصلاً فکرشو نمی‌کردیم که ۹۹.۹ درصد احتمال وجـود داشتن دو نفر با تاریخ تولد یکسان باشه.

*نتیجه:* برای همین میگیم رمزنگاری و هش و این‌ها باید توسط ریاضی‌دان‌ها بررسی شه. صرفاً چیزی که ما فکر می‌کنیم درست نیست! بلکه باید بـا ریاضـیات اثبـات بشـه. اونم توسـط کسـانی کـه درک عمیـق از ریاضیات، ساختمان گسسته، نظریه اعداد (و درواقع رمزنگاری) دارن.

اینم نمودارش از ویکی پدیا:




Rajkiran g, Birthday Paradox, CC BY-SA 3.0

Probability of same birthday table[18]:

| *n* | *p*(*n*) |
|---|---|
| 1 | 0.0% |
| 5 | 2.7% |
| 10 | 11.7% |
| 20 | 41.1% |
| 23 | 50.7% |
| 30 | 70.6% |
| 40 | 89.1% |
| 50 | 97.0% |
| 60 | 99.4% |
| 70 | 99.9% |
| 75 | 99.97% |
| 100 | 99.99997% |

18 https://en.wikipedia.org/wiki/Birthday_problem#Calculating_the_probability, Creative Commons Attribution-ShareAlike License 3.0

| | |
|---|---|
| 200 | 99.99999999999999999999999998% |
| 300 | $(100 - 6\times10^{-80})\%$ |
| 350 | $(100 - 3\times10^{-129})\%$ |
| 365 | $(100 - 1.45\times10^{-155})\%$ |
| ≥ 366 | 100% |

درواقع یه فرمول کلی هست که اگر بخوایم بفهمیم احتمال collision چقدره (این فرمول ساده‌شدس و تقریبی جواب میده)[۱۹]:

$$P(n,d)=1-e^{-(\frac{n^2}{2d})}$$

مثلاً برای ۴۰ نفر:

$$P(40,365)=1-e^{-(\frac{30^2}{2\times365})}=0.89$$

یا یه حالت کلی‌تر[۲۰]:

$$p(n,d)=1-(\frac{d-1}{d})^{(\frac{n(n-1)}{2})}$$

نکته: میگن که اگر مثلاً یه هشی ۱۲۸ بیت خروجی میده، شما نیاز دارین که

- $2^{128}$ متن بدین که خروجی مورد نظرتون پیدا شه
- $2^{(128/2)}=2^{64}$ متن بدین که یه collision پیدا شه.

اما این برای من گنگه. شما اگر می‌دونین، دلیلش رو بگین بهم لطفاً که این داک کامل‌تر شـه. آخـه من با محاسباتی که می‌کنم، می‌بینم که در صورت این کار، احتمال کالیژن ۰.۳۹ هست. خب اینکـه از ۵۰ بیشتر نیست که بگیم کالیژن رخ میده! وقتی میگیم کالیژن رخ بده که احتمال بالای ۰.۵ باشه.
کد پایتونش هم زدم و نتیجه[۲۱]:

*Code:*

```
from decimal import Decimal

def f2(n, d):  # second formula
```

19 https://en.wikipedia.org/wiki/Birthday_problem#Approximations, Creative Commons Attribution-ShareAlike License 4.0

20 https://en.wikipedia.org/wiki/Birthday_problem#Probability_of_a_shared_birthday_(collision), Creative Commons Attribution-ShareAlike License 4.0

۲۱ از «Decimal» استفاده کردم که دقت اعداد اعشاری بره بالا.

```
    n = Decimal(n)
    d = Decimal(d)
    base = (d-1)/d
    exp_ = (n*(n-1))/2
    return (1-base**exp_)

print(f2(2**32, 2**64))
```

*Output:*

```
0.39346934008095526304284536 34
```

شما اگر دونستین بگین لطفاً!

*تمرین:* فرض کنین یه سیستمی رو بهتون دادن که باید به هر کاربر به صورت شانسی و رندوم یه شماره (ID) یکتا اختصاص بدین.

فرض کنین که تعداد آدما ۳۰۰,۰۰۰ هزار نفره. خب به نظرتون استفاده از یه عدد `int` که ۳۲ بیتیه (که ۲ به توان ۳۲ حالت عدد[۲۲] رو می‌تونه تولید کنه) خوبه یا نه؟

*پاسخ:*

خیر! خوب نیست! از فرمول و کد بالا بریم، می‌بینیم که به احتمال ۹۹.۹۹۷ درصد به colission بر می‌خورین!

برگردیم به خود هش. از لحاظ تئوری، چون بی‌نهایت ورودی مختلف داریم ولی سایز خروجیمون ثابته، قاعدتاً به collision بر می‌خوریم. منطقی هم هست. شما بی‌نهایت ورودی دارین ولی خروجیتون مثلاً صرفاً می‌تونه ۲۵۶ بیت رو نشون بده. (۲ به توان ۲۵۶ حالت). پس قاعدتاً collision می‌خورین. (۲ به توان ۲۵۶ بعلاوه یک حالت بدین، collision رو دارین! ولی این در عمل بسیار بسیار بسیار سخته. هش‌ها جوری ساخته میشن که این کار رو خیلی خیلی سخت کنن.

ما هش‌های مختلفی داریم. توی هش‌هایی که داریم، از لحاظ پیدانشدن collision، فعلاً هشی با نام SHA-2 و SHA-3 امن هستن. هش «MD5» و «SHA1» امن نیستن.

خود سازنده «MD5» توی سال ۲۰۰۵ اومد گفت که دیگه هشی که ساخته بودم اصلاً collistion-resistant نیست![۲۳]

یا برای SHA-1:

22 0 to 4,294,967,295

23 https://mail.python.org/pipermail/python-dev/2005-December/058850.html

+ On 2012 Jesse Walker: When Will We See Collisions for SHA-1?[24]

طبق پیش‌بینیاش:

"A collision attack is therefore well within the range of what an organized crime syndicate can practically budget by 2018, and a university research project by 2021."

و گوگل در سال ۲۰۱۷، اولین SHA-1 Collision رو پیدا کرد![25]

متوسط:

- خب میگی که هش نباید توسط افراد عادی ساخته شه، پس این هش‌های معروف مثل همین «MD5» و «SHA-2» و اینا چی هستن؟ اینا چجور ساخته و تأیید شدن؟

+ ببینین منظور اینه افراد عادی نباید فکر کنن که هشی رو که ساختن، دیگه بهترینه. شما بهترین رمزنگار دنیا هم که باشی، اگر هشی بسازی، باید اون هش توسط متخصصان مورد بررسی و تست قرار بگیره. مراجعه شود به بخش «*در امنیت، خلاقیت شخصی بدون بررسی توسط متخصصان، امن نیست!*»

اما بریم یه سری از هش‌ها رو بررسی کنیم:

- ## هشی استاندارد برای چک‌کردن یکسان‌بودن و صحیح‌بودن مقادیر

یادتونه گفتیم که هش باید جوری باشه که با کوچک‌ترین تغییری، هش به شدت عوض شه؟ خب به نظرتون از این قابلیت برای چه کاری میشه استفاده کرد؟ یکم فکر کنین خودتون!

+ ببینین مثلاً یه فردی یه فایلی رو ساخته. حالا اون فایل رو توی یه سرور میگذاره که من اگر خواستم دانلودش کنم، از اون سرور بگیرمش. اما شاید صاحب اون سرور آدم بدی باشه و بخواد فایل دستکاری‌شده و فایلی که حاوی بدافزاره رو بهمون بده! خب من از کجا بفهمم که آیا این فایلی که دارم دانلود می‌کنم، دقیقاً همون فایلی هست که سازنده معرفی کرده؟!

اینجا پای یه سری هش‌ها میان وسط! سازنده میاد میگه این فایل هشش فلانه. حالا وقتی دانلود کردی، تو هم هشش رو حساب کن و ببین آیا با هشی که من بهت دادم یکسانه یا نه؟ اگر بود یعنی فایل درست و دستکاری نشده‌ای رو دانلود کردی! چون اگر حتی یه نقطه هم تو فایل عوض میشد، هش هم عوض میشد!

مثلاً نگاه کنیم:

24 https://www.schneier.com/blog/archives/2012/10/when_will_we_se.html
25 https://security.googleblog.com/2017/02/announcing-first-sha1-collision.html

Current Version: 1.5.1

**macOS**

For 10.13 and later

Download (Universal)

**Windows**

For 10 and later

Download (x64 64 bit)

Download (x64 64 bit Portable Zip)

For ARM Devices

Download (ARM 64)

Download (ARM 64 Portable Zip)

**Linux**

Flatpak

Install via flathub.org

or

Download (64bit)

QuickSync Plugin Download (64bit)

**Other**

Command Line Version

Source Code

**Development Builds**

Hosted externally on Github

**Old Releases**

Release Archives

**Download Safety**

Please take note that HandBrake.fr is the only official place where HandBrake can be downloaded from.
There are many unofficial mirrors of HandBrake and while most of them offer legit versions of HandBrake, there are a few that don't.

- Read our guide to Downloading and Installing HandBrake
- Check the integrity of your download with Checksums (mirrored on our GitHub Wiki)
- Check the authenticity of your download with Open PGP (mirrored on our GitHub Wiki)

image

**Validate your Download**

Please see our guide to safely downloading and installing HandBrake.

## Version 1.5.1

| File | Size (MB) | SHA256 |
|---|---|---|
| HandBrake-1.5.1-x86_64-Win_GUI.exe | 19.49 | 01167a96a338cb394ee2e339545379cd156dfe4b1837af64764a08279f8af33e |
| HandBrakeCLI-1.5.1-win-aarch64.zip | 14.64 | dfcc82e756ddbc67ba3d71fcf67377d06f21aadf1c54a8413797e6398294d49d |
| HandBrakeCLI-1.5.1-win-x86_64.zip | 17.45 | 496e91f1341095305e46f331463281e93fabed926381d28b79a2bd3785c49954 |
| HandBrakeCLI-1.5.1-x86_64.flatpak | 9.8 | 61ec9503d8e656bc16d1d5e220497491eb9ae4dc484a51c4d79f365b164ba509 |
| HandBrakeCLI-1.5.1.dmg | 32.06 | 328ad1fbacb855b644b63899450c004cb18e5e819ad519549c4c4bc863a60f90 |
| HandBrake-1.5.1-source.tar.bz2 | 15.39 | 3999fe06d5309c819799a73a968a8ec3840e7840c2b64af8f5cdb7fd8c9430f0 |
| HandBrake-1.5.1-arm64-Win_GUI.zip | 22.8 | dc9e8395945778d6819bd0063ab21bd666dbc9abfd93130d4900652c80c36350 |
| HandBrake-1.5.1-arm64-Win_GUI.exe | 15.27 | b5468cfb3e8d469e72a68a28f57624ad287e5b22c7407582c1aed4193ea70299 |
| HandBrake-1.5.1-x86_64-Win_GUI.zip | 27.44 | 69e499d88df6f77a5ce663c8f5ae3ff2e6210a908152a7c437d10bca7294d0be |
| HandBrake-1.5.1-x86_64.flatpak | 23.26 | 23a459b3dd02c4ccd9c53c4c1585a452e7557d6011be276740afa6634a2ca66f |
| HandBrake-1.5.1.dmg | 35.91 | 767cb16314e3869c42cff78db92bcad7a7faa861c70f97b1326fe3686c62b61f |
| Plugin.HandBrake.IntelMediaSDK-1.5.1-x86_64.flatpak | 58.31 | 61768dce2776df0220d18477e2f5bda8e7512583bff6e994157e691f532efe46 |

## Version 1.5.0

| File | Size (MB) | SHA256 |
|---|---|---|
| HandBrake-1.5.0-source.tar.bz2 | 25.94 | 72d79e8e0c6759f5855407c9b4de4273eb5fb6cc363238bf8d9a992c4b2a3c1a |
| HandBrake-1.5.0-arm64-Win_GUI.exe | 15.25 | 42a8520911a70d46fcd9b4358a79d69cf36f71044a64df483daf01ec66ad2c58 |
| HandBrake-1.5.0-arm64-Win_GUI.zip | 22.8 | a3d67b951b20a378095f1c3cedc3e451d8ce7043fed37672a44789f3ead575c3 |

همونطور که می‌بینیم، اومده ورژن‌های مختلف برای پلتفرم‌های مختلف رو با هش گذاشته. که ما می‌تونیم با محاسبه هش، ببینیم آیا فایل دستکاری شده هست یا نه. (البته Open PGP هم هست که حالا فعلاً موضوع بحثمون نیست.)

*کاربرد:* دقیقاً همین توی لینوکس رخ میده. یعنی وقتی شما یه پکیجی رو دانلود می‌کنین، به صورت خودکار هش چک میشه که آیا مثلاً فایل توی رفت و آمد خراب شده یا نه؟! (یا بهتر بگم، امضای دیجیتالش هم چک میشه. امضای دیجیتال چیه؟ بعداً بهش میرسیم!)

یا مثلاً توی اعتبارسنجی‌های مختلف مثلاً توی TLS و اینها، هش چک میشه که یکسان باشه. (بعداً می‌رسیم بهش)

به نظرتون خصوصیت این نوع هش باید چجور باشه؟
+ مطمئناً نباید به ازای مقادیر متفاوت، خروجی یکسان بده! اینطوری اگر میبود، من میتونستم فایلی تقلبی رو به شما بدم که هشش هم با هش فایل اصلی یکسان میبود! پس این اولین نکتشه! درواقع باید collision resistant باشه!
+ خیلی هم کند نباشه. فرض کنین من یه فایل ۱۰ گیگی رو دانلود کردم. می‌خوام هشش رو حساب کنم ببینم فایل آیا دستکاری نشده؟ خب اگر خیلی کند باشه خب مثلاً من باید ۱۰ دقیقه صبر کنم تا هش حساب شه. خب سخته و زمان‌بر! حالا این یه فایله. شما سرورها رو در نظر بگیرین که در آنِ واحد دارد صدها فایل رو جابه‌جا و چک می‌کنن. اگر قرار بود خیلی کند باشه که بشدت سرعت میومد پایین.

*متوسط:* پروتکل‌هایی که روی این چک‌کردن بنا شدن، بسیار کند میشن. مثلاً شما میری وارد یه صفحه میشی و به جای یک ثانیه، سه ثانیه طول میکشه باز شه! خب سه برابر زمان بیشتری داری میگذاری! این خوب نیست!

هش‌های زیادی داشتیم. قدیما md5 بود که الآن ناامنه. (به راحتی میشه من یه فایلی بسازم که هشش با یه فایل دیگه یکسانه) بعدش SHA1 که توسط NSA[۲۶] ساخته شده بود اومد و توسط سازمان استاندارد تأیید شد. البته SHA-1 امروز ناامنه. بعدش SHA2 اومد که اونم توسط NSA ساخته شده و توسط سازمان استاندارد تأیید شد. فعلاً SHA2 امنه. اما سازمان استاندارد (NIST)[۲۷] اومد گفت اگر یه وقت SHA-2 مثل SHA-1 شکست و ناامن شد چی؟ باید یه برنامه جایگزین داشته باشیم. باید هرچه زودتر خودمونو برای اون زمان آماده کنیم. نگیم وایسیم هر وقت ناامن شد جایگزین بدیم. بلکه جایگزین رو آماده داشته باشیم که اگر یه وقت مشکلی پیش اومد بتونیم ازش استفاده کنیم. اینطور بود که SHA-3 قصه ما به وجود اومد :)

از الآن به بعد اسم سازمان NIST رو زیاد می‌شنوین. این سازمان کارش استانداردکردنه و خلاصه کارهای بزرگ مربوط به رمزنگاری، توسط این سازمان انجام میشه.

---

۲۶ سازمان امنیت ملی آمریکا (National Security Agency)

27 National Institute of Standard and Technology

- چه نیازی به سازمان استاندارد هست؟

+ ببینین همونطور که گفتیم هش‌های خودساخته امن نیستن! مگر اینکه خلافش ثابت شه. یعنی توسط متخصصان بررسی شه و بگن آره امنه. خب یکی باید بشینه رمزنگارها رو دور هم جمع کنه که بررسی کنن دیگه! وگرنه آدما دور هم جمع نمیشن که روی یه چیز واحد نظر بدن.

همچنین شرکتا خیلی دوست دارن از چیزای خودساخته خودشون استفاده کنن. مثلاً شرکت فلان میاد میگه ((که ما یه هش استثنایی ساختیم! تمام دستگاه‌های ما به وسیله این هش عالی خودمون امن میشن.))

و خلاصه بیاد الکی برای خودش تبلیغ کنه. درحالی که اصلاً معلوم نیست اون هش درست بررسی شده یا نه؟!

خلاصه یه زور باید بالا سرشون باشه که بگه که از هش امنی استفاده کنین که امنیت داده‌های کاربران به خطر نیوفته. یه زوری که بتونه هم هش‌ها رو بررسی کنه و هم روند بررسی شفاف باشه و هم با سازمان‌های دیگه همکاری کنه که شرکتا رو مجبور کنن که از چیزای امن استفاده کنن.

پس درواقع NIST وظیفه خیلی مهمی داره.

همینطور که اینجاییم، یه چیز انگیزشی بگم بهتون :)

الهام طبسی، همینقدر بس که بگیم:

"Elham Tabassi is a Senior Research Scientist at the National Institute of Standards and Technology (NIST) and the Associate Director for Emerging Technologies in the Information Technology Laboratory (ITL). She also leads NIST's Trustworthy and Responsible AI program that aims to cultivate trust in the design, development, and use of AI technologies."[28]

و سال ۲۰۲۳، بر اساس مجله «Time»، جزء ۱۰۰ فرد تاثیرگذار در حوزه هوش‌مصنوعی معرفی شد.[۲۹]

هش‌های SHA-1 و SHA-2 توسط NSA ساخته شده بودن.

- چرا NSA؟

+ چون در اون زمان، قدرت رمزنگاری دنیا دست NSA بود. دنیای دانشگاهی و آکادمیک، اصلاً در حد NSA نبودن. NSA دانش بیشتر، پول و بودجه بیشتر، افراد بیشتر و کلی چیزای بیشتر داشت. پس برای همین NSA مسئول ساخت شده بود.

More (How much security agencies are good at cryptography):

28 https://www.nist.gov/people/elham-tabassi
29 https://time.com/collection/time100-ai/6310638/elham-tabassi/

"A circular left shift operation has been added to the specifications in section 7, line b, page 9 of FIPS 180 and its equivalent in section 8, line c, page 10 of FIPS 180.* This revision improves the security provided by this standard."[30]

البته NSA هیچ توضیحی نداد که چرا این کار رو کرده و دلیلش چی بوده. فقط گفت امنیت بیشتری میاره.

+ Impact of Rotations in SHA-1 and Related Hash Function[31]
+ How Advanced Is the NSA's Cryptanalysis—And Can We Resist It?[32]
+ The Secret Story of Nonsecret Encryption[33]
+ More one the NSA's Capabilities[34]

اما برای SHA-3، سازمان NIST یه فکری به سرش زد.[۳۵] گفت که خب از جایی که سازمان‌های امنیتی کاملاً مورد اعتماد نیستن[۳۶] و همیشه کارشون جاسوسیه، ممکنه چیزایی طراحی کنن که ما نفهمیمش و فکر کنیم امنه ولی خودشون بتونن بشکوننش و اینکه حالا هم دنیای آکادمیک پیشرفت بیشتری کرده، پس ایندفعه برای SHA-3 یه مسابقه جهانی برگزار می‌کنیم. از آدمای متخصص سرتاسر دنیا دعوت می‌کنیم که هش‌هاشون رو برامون ارسال کنن و ما در طی چند سال با روند شفاف که همه آدما در سرتاسر دنیا بتونن روش نظر بدن و فکر کنن، یه هش رو برنده اعلام می‌کنیم و مراحل استانداردکردن و تبدیلش به SHA-3 اش رو انجام میدیم.

اومد چندتا کارگاه و کنفرانس گذاشت. صحبت کردن. بحث کردن. الگوریتما رو چیدن وسط. گفتن نه‌تنها ما، بلکه تمام محققان سرتاسر دنیا می‌تونن تستشون کنن و لطفاً نتایج رو به ما بگن و خلاصه مکانیزم خیلی خیلی شفافی شد. رقابت سختی در گرفت. هرکی تلاش می‌کرد راهی پیدا کنه تا الگوریتمای دیگه شکسته شن و خودش مرحله به مرحله بره بالاتر. خلاصه بعد سه راند، بالاخره یه الگوریتم به عنوان الگوریتم برنده انتخاب شد! هوراااا.

- به نظرتون این فرایند چقدر طول کشید؟ یه ماه؟ دوماه؟ شش ماه؟ یه سال؟ چقدر؟

+ خب یادتونه گفتم رمزنگاری مبحث خیلی پیچیده‌ای هست؟ خب باید به عرضتون برسونم که این مراحل ۵ سال طول کشید! بله ۵ سال!!! بله کار یه شب و دو شب نیست که! عملاً نتیجه ۵ سال تحقیق بهترین رمزنگارها و ریاضی‌دان‌های دنیاست.

حالا فهمیدین که چرا میگیم از هش‌های خودساخته شرکتا دوری کنین؟ چون اینهمه زمان نیازه. اینهمه متخصص باید بشینن بررسی کنن. اینهمه مقاله چاپ شه.

---

30 Proposed Revision of Federal Information Processing Standard (FIPS) 180, Secure Hash Standard → https://www.federalregister.gov/documents/1994/07/11/94-16666/proposed-revision-of-federal-information-processing-standard-fips-180-secure-hash-standard, https://www.govinfo.gov/content/pkg/FR-1994-07-11/html/94-16666.htm

31 https://csrc.nist.rip/groups/ST/hash/documents/Rechberger_ImpactOfRotations.pdf

32 https://www.schneier.com/essays/archives/2013/09/how_advanced_is_the.html

33 https://www.schneier.com/essays/archives/1998/04/the_secret_story_of.html

34 https://www.schneier.com/blog/archives/2015/05/more_on_the_nsa_1.html

35 https://csrc.nist.gov/Projects/hash-functions/sha-3-project

36 e.g. ISO Rejects NSA Encryption Algorithms → https://www.schneier.com/blog/archives/2017/09/iso_rejects_nsa.html

مثلاً بیایم SHA3 که توسط NIST استاندارد شد رو بررسی کنیم[۳۷].

- چرا بررسی کنیم؟

+ تا با روند استانداردسازی یه چیز کامپیوتری بیشتر آشنا شیم. ببینیم چه نکات مهمی نیازه رعایت شه.

مثلاً برای SHA3 خود NIST گفته[۳۸]:

## “Security

As was the case for the AES competition, security is the most important factor when evaluating the candidate hash algorithms. However, there remains significant disagreement within the cryptographic community-at-large over what security definitions should be used to evaluate hash algorithms. While initially proposed for use in digital signatures, cryptographic hash algorithms are used in a wide variety of applications, including message authentication codes, pseudorandom number generators, key derivation, and one-way functions for obfuscating password files. All of these applications have different security requirements.”

امنیت مهم‌ترین رکن این بررسی‌هاست. خصوصیت اول هشی که برای SHA بخواد استفاده بشه (چون SHA خیلی وقتا برای چک‌کردن و تفاوت‌نداشتن و اینا استفاده میشه)، امن بودنشه! هشی که امن نباشه و مثلاً collision resistant نباشه، این هش اصلاً به درد SHA نمیخوره!

## “Cost and Performance:

FRN-Nov07 identified cost as the second-most important criterion when evaluating candidate hash algorithms. In this case, cost includes computational efficiency and memory requirements. Computational efficiency essentially refers to the speed of an algorithm. NIST expects SHA-3 to offer improved performance over the SHA-2 family of hash algorithms at a given security strength. Memory requirements refer both to code size and random-access memory (RAM) requirements for software implementations, as well as gate counts for hardware implementations.”

دومین رکن مهم، سرعت خوب، استفاده مناسب از رم و هزینه کم برای محاسبه‌اش هست. ببینین وقتی یه چیزی قراره استاندارد باشه و کل دنیا در همه جا تقریباً باید استفادش کنن، این هش باید به اندازه کافی سریع باشه.

- چرا؟

+ چون قراره همه جا استفاده بشه! همه دستگاه‌ها که قدرت محاسباتیشون بالا نیست که! همه که به اندازه کافی مموری ندارن که! رم لپتاپ شما ممکنه ۱۶ گیگابایت باشه و هیچ دغدغه‌ای نداشته باشی! اما

---

۳۷ همگی در لینک زیر در دسترسن. پیشنهاد می‌کنم سری بهش بزنین. خیلی جالبه!
https://csrc.nist.gov/Projects/hash-functions/sha-3-project

38 Status Report on the First Round of the SHA-3 Cryptographic Hash Algorithm Competition
=> https://csrc.nist.gov/pubs/ir/7620/final → In the PDF

یه سری کارت هوشمند اون مقدار حافظه رو ندارن. یه سری کـارت هوشـمند فقـط ۲۵۶ بیت. (کمـتر از ۱ کیلوبایت) حافظه اصلی دارن! اون مقدار قدرت پردازشی رو ندارن. وقتی یه چیزی قراره استاندارد باشه، باید بشه ازش همه جا استفاده کرد! چه توی تُستِر هوشمند، چه یخچال هوشـمند، چـه کـارت هوشـمند و چـه لپ‌تاپ‌ها و... . پس فاکتور سرعت محاسبه و اون هزینه‌ای و Cost ای که برای محاسبش صرف میشـه هم یکی از عوامل مهمه!

## “2.2.3 Algorithm and Implementation Characteristics

The SHA-3 competition has received many candidate algorithms with new and interesting designs, and with unique features that are not present in the SHA-2 family of hash algorithms. Candidate algorithms with **greater flexibility** may be given preference over other algorithms. This includes algorithms capable of **running efficiently on a wide variety of platforms**, as well as algorithms that use parallelism or instruction set extensions to achieve higher performance. In addition, simple and elegant designs are preferable, in order to encourage understanding, analysis and design confidence.”

همچنین خود الگوریتم هم مهمه. مثلاً باید انعطاف داشته باشه کـه بشـه همـه جـا اسـتفادش کـرد. یـا طراحیش باید ساده باشه که بشه فهمیدش و آنالیز و بررسیش کرد و از درستی طراحیش، مطمئن شد.

- آیا فهمیدن، بررسی، آنالیز و چک‌کردن امنیت یه الگوریتم ساده راحت‌تر، یا یـه الگـوریتمی کـه خیلی سخته و اصلاً قابل فهم نیست و مثلاً کدش ده هزار خطه؟

+ مطمئناً اونی که ساده‌تره. چون وقتی ساده باشه، میشه بهتر فهمیدش و میشه بهتر چکش کرد که آیـا مطمئنه یا نه. همچنین میشه براش حمله‌های بهتری طراحی کرد. چون بهـتر فهمیـدیمش، بهـتر میتـونیم بررسیش کنیم. همچنین پیاده‌سازیش تـوی برنامه‌هـا و جاهـای مختلـف هم سـاده‌تر میشـه! چـون میشـه راحت‌تر برنامه‌ای که محاسبش کنه رو نوشت و بهتر میشه سخت‌افزارها رو جوری طراحی کرد که در مقابل حمله‌های فیزیکی مقاوم باشـه. مثلاً یکی از مهم‌تـرین چـیزا مخصوصـاً در رمزنگـاری، اینـه کـه در مقابـل حمله‌هایی مثل side-channel attack مقاوم باشه. (مراجعه شود به فایل «Security_Roadmap» در گیتهابم) اگر الگوریتم ساده‌تر باشه، درکمون میره بالا و سخت‌افزارهایی متناسب اون می‌سازیم کـه بشـه از این حملات جلوگیری کرد.

*نتیجه مهم:* برای یه چیز استاندارد، اونی که لزوماً امن‌تـره بهـتر نیسـت! بلکـه همـه جـوانب رو در نظـر می‌گیرن. امنیت، سرعت و هزینه محاسبه، نحوه طراحی و... .

بله تانک هم از ماشین استحکامش بیشتره و مثلاً اگر بخوره به دیوار، سرنشین آسیب نمی‌بینـه. ولی آیـا باید برای رفت و آمد از تانک استفاده کنیم؟! معلومه که نه! ممکن نیست!

بله شما می‌تونی الگوریتمی بنویسی که هزار برابر امن‌تر باشـه (از لحـاظ تئـوری)، امـا محاسـبش خیلی خیلی خیلی زمان‌بر باشه. نشه توی سخت‌افزارهایی با قدرت محاسباتی کم تولیدش کرد. نشـه تـوی کـارت هوشمند به کارش برد. خب پس به چه دردی می‌خوره؟!

توی مرحله اول NIST درباره کاندیداها یه سری توضیحات داده که با هم مرورش می‌کنیم[۳۹]:

BLAKE’s performance is quite good. It has modest memory requirements and appears to be suitable for a wide range of platforms.

مثلاً اینجا گفته که performance و سرعت (البته ترجمه سرعت اینجا زیاد جالب نیست) خـوبی داره و میشه در جاهای زیادی استفاده‌اش کرد.

The most serious cryptanalytic results against BMW are from impractical pseudo-collision attacks, and practical near-collision attacks [30]. These results raise questions about the security of the design.

درباره هر کاندیدا، هم درباره performance و هم security شون نوشته.

“Skein has good performance on high-end platforms, particularly in 64-bit mode, and is also expected to perform well in constrained platforms and in dedicated hardware implementations. It has modest memory requirements and benefits from the pipelining used in modern processors.

The most significant cryptanalytic results on Skein are distinguishing attacks against reduced-round versions of Threefish; these do not appear to pose a threat to the full hash algorithm at this time.”

درواقع خیلی جالبه که NIST میاد رقابت میگذاره و بررسی میکنه و بـرای عمـوم منتشـر میکنـه. افـراد دیگه هم میتونن به صورت مستقل بررسیش کنن. خلاصه یه تلاش دسته‌جمعی بـرای سـاخت اسـتانداردی خوب.

- مرحله دوم[۴۰] (زیر صفحه میتونین نسخه کاملش رو از سایت NIST بخونین)

## “4.2.3 Discussion

Although BMW has very good software performance, and good potentials for pipelining and area-efficient high-throughput values, a disadvantage of the algorithm is its irregular and not well-understood design. Since the compression function of BMW does not have a conventional iterated-round structure, there appears to be no obviously simple way to adjust the security margin, or to trade performance for security. Moreover, the attacks on the algorithm, even after an extensive tweak, did not provide confidence in the security of the algorithm. For these reasons, BMW was not selected as a finalist.”

39 Status Report on the First Round of the SHA-3 Cryptographic Hash Algorithm Competition: https://csrc.nist.gov/pubs/ir/7620/final

40 NISTIR 7764, *Status Report on the Second Round of the SHA-3 Cryptographic Hash Algorithm Competition* => https://nvlpubs.nist.gov/nistpubs/Legacy/IR/nistir7764.pdf

مثلاً اینجا گفته که درسته خیلی performance خوبی داره و این‌ها ولی خب چون الگوریتمش رو نمیشه درست فهمید و با توجه به ساختار نامنظمش، نمیشه security margin شو درست اندازه گرفت.
پس ببینین. برای انتخاب یه استاندارد، باید همه جوانب رو در نظر گرفت. امنیت بدون سرعت، سرعت بدون امنیت، هیچکدوم خوب نیست! بلکه باید همه چی در حد خوبی رعایت شده باشه!

مرحله سوم رو اگر دوست داشتین ببینین[41]. خیلی توضیحات فنی‌تر و بهتری داره!

More Reading:

+ NIST Cryptographic Standards and Guidelines Development Process[42]

## SHA-256 چجور کار می‌کنه؟

خلاصه اینکه فعلاً اسم SHA-2 و SHA-3 رو برای چک‌کردن یکسان‌بودن فایل و این‌ها زیاد می‌شنوین. نحوه کار SHA2-256 رو می‌تونین توی وبسایت‌های مختلف بخونین. همچنین وبسایت زیر، کاملاً قشنگ شیوه کارش رو تصویری نمایش داده و برای یادگیری خیلی عالیه:

https://blog.boot.dev/cryptography/how-sha-2-works-step-by-step-sha-256/
https://sha256algorithm.com/

- ## هش‌هایی برای پسورد

توجه کنید که هش‌هایی که برای پسورد هستن، با هش‌های دیگه یکم باید متفاوت باشن. یه سری ویژگی خاص رو باید داشته باشن! قبل اینکه اصلاً وارد داستان ویژگی‌های هش مناسب پسورد بشیم، با یه سری حمله‌ها به هش آشنا میشیم که بدونیم هش مناسب پسورد باید چه ویژگی‌هایی رو داشته باشه و در برابر چه حالت‌هایی باید مقاوم باشه!

### ✔ Traditional Brute Force

قبلش بیایم یه مثال دنیای واقعی بزنیم. فرض کنین میخوایم یه گاوصندوق باز کنیم اگر رمزش رو ندونیم، چیکار می‌کنیم؟ میایم دونه دونه حالتای مختلف رو تست می‌کنیم. بالاخره یه جایی بعد مثلاً دویستا تلاش باز میشه. برای پسورد هم همینه. من میام تعداد زیادی پسورد رو برای یه اکانت تست می‌کنم. مثلاً هی میام مقادیر مختلف میدم ببینم چی میشه.

Password guesses: Admin1234 – admin 1234 – amir123 – james56 – angela89 …

همینطور میام مقادیر زیادی رو تست می‌کنم. صدها هزار مقادیر تست می‌کنم. بالاخره شانسی پسورد پیدا میشه. این میگن بروت فورس.

اما خب کپچاها[43] میتونن مانع brute-force بشن!

41 https://nvlpubs.nist.gov/nistpubs/ir/2012/NIST.IR.7896.pdf
42 https://www.schneier.com/blog/archives/2014/03/how_nist_develo.html
۴۳ دیدین وقتی وارد یه سایت میشین میگه اثبات کن ربات نیستی؟ اونا همون کپچا هستن برای جلوگیری از اینکه یه ربات نتونه نفوذ کنه و یا بروت‌فورس کنه. درواقع کپچاها راه‌حلی برای شناسایی انسان از ربات هستن.

برای اینکه بتونیم brute-force رو برای حمله‌کننده سخت‌تر کنیم، از یه سری هش خاص استفاده می‌کنیم. به نظرتون این هشا چه ویژگیی باید داشته باشن؟ یکم فکر کنین!

+ هش مناسب پسورد، باید به دست اوردنش سخت و هزینه‌بردار باشه. یعنی حمله‌کننده باید زمان و هزینه زیادی خرج کنه. مثلاً هر ثانیه فقط بشه مثلاً چند صدتا هش حساب کرد.

یعنی من نتونم با یه کامپیوتر معمولی، تعداد زیادی هش رو محاسبه کنم! چون اگر بتونم، میتونم به سادگی brute-force کنم. پس هش باید کُند باشه، حجم زیادی از رم رو اختصاص بده به خودش، در مقابل brute-force کردن با CPU ها و GPU های قدرتمند مقاوم باشه.

مثلاً بیایم تفاوتشو در عمل ببینیم[۴۴]:

```
time printf "password" | argon2 somesalt -id -t 4 -m 21 -p 1
time printf "password" | sha256sum
```

دستورات بالا رو در ترمینالتون وارد کنین[۴۵] که ببینین چقدر تفاوت زمانی در محاسبه کردنشون هست و چرا نباید از SHA برای پسورد استفاده کنیم. چون SHA سریع حساب میشه و Brute-force کردنش راحته.

یعنی درواقع من اگر یه پسوردی رو امتحان کردم، چون هش‌کردن اون پسورد طول می‌کشه، من نمی‌تونم سریع یکی دیگه رو تست کنم. زمان‌بره. باید یکم صبر کنم تا مقدار بعدی رو تست کنم. اینطوری عملاً سرعت امتحان‌کردن پسورد منو کند کرده.

## ✔ Hash Table Attack

فرض کنیم دیتابریچ رخ میده. داده‌ها لو میرن. دیتابیس شرکت لو میره. یعنی مثلاً جدول زیر لو میره:

| Username | Leaked passwords |
|---|---|
| Alice | 6f403cce6bb38bd0a424f416cc7250372dd3977d6f4740cd1db4ab569400a8ac |
| Bob | **fc8252c8dc55839967c58b9ad755a59b61b67c13227ddae4bd3f78a38bf394f7** |
| Angela | ddaf16f831fb442fb83ae6dcf6f998d66c50fb6ff90908269aba64de0655770d |

خب شاید بگین هش لو رفته و خب از هش نمی‌تونیم برسیم به پسورد. پس کاری نمیشه کرد. خب در اشتباهین! میشه! یکم خودتون فکر کنین که چطور ممکنه.

درسته ما میدونیم از هش نمی‌تونیم برسیم به پسورد. ولی میتونیم هش پسوردای پرتکرار یا حروف انگلیسی رو حساب کنیم یه دیتابیس و جدول از هش‌های پرکاربرد برای خودمون بسازیم. مثل زیر:

---

۴۴ حواستون باشه که دستور اول ۲ گیگ رم مصرف می‌کنه! اگر حجم رمتون کمه، به جای عدد ۲۱، عدد ۱۶ رو بگذارین.

۴۵ یه نکته لینوکسی! حواستون باشه که از دستور «printf» استفاده کنین و از دستور «echo» استفاده نکنین. چون «echo» به صورت پیشفرض یه «\n» هم می‌زنه و عملاً هشی که می‌خواین، هش اون متن نیست! (هش متن + کرکتر «\n» هست!)
هش Argon2 یکی از هش‌های معروف برای پسورده.

| Password | Hash |
|---|---|
| p@ssword | 28e4707c0fadfddd5f586d117edab14899b23116d5c1aec44d33921be6cdcea2 |
| admin | **fc8252c8dc55839967c58b9ad755a59b61b67c13227ddae4bd3f78a38bf394f7** |
| 123456 | e150a1ec81e8e93e1eae2c3a77e66ec6dbd6a3b460f89c1d08aecf422ee401a0 |
| 1234 | a883dafc480d466ee04e0d6da986bd78eb1fdd2178d04693723da3a8f95d42f4 |
| Alex123 | 8b233917ea77178701a87739f40dccc8f04078392f8b9688e1028d260933dc06 |
| House | 1d538e1e6def82b438d60adf4491a4c3afdb064583badc4cf239b81fbb05de9f |

خب حالا مثل دفترچه تلفن مطابقت بدیم. ببینیم آیا اون هش‌های لو رفته، توی دیتابیسی که خودم از هش رمزهای پرتکرار و رایج ساختم هس یا نه؟ عه دومی مشترکه. پس پسورد Bob ما، admin بوده... .
خب به این صورت من میتونم بفهمم پسوردش چی بوده. به این میگن Hash table attack. برای همین هم هست که میگن پسوردتون کلمات انگلیسی و بامعنی نباشه. چون برای همه کلمات انگلیسی و پسوردهای ساده، دیتابیسی از پسوردهایی که باید امتحان بشه هست و هش‌ها رو هم از قبل حساب کردن و نگه‌داشتن و راحت میتونن Hash table attack بزنن.

- شاید سؤال پیش بیاد خب از کجا میتونن بفهمن که کدوم الگوریتم هش استفاده شده که برای اون از Hash table اون هش استفاده کنن؟ (چون هش‌های متفاوتی داریم)

+ خب کار زیاد سختی نیست. یکم فکر کنین ببینین میشه چه کارهایی کرد؟
طول هش، خصوصیات هش، مبنایی که تولید شده، میتونه یه سری راهنمایی‌هایی کنه.

علاوه بر این یه سری ابزار هم هستن که هش رو بهشون میدیم و بهمون میگن این چه هشی میتونه باشه! مثلاً hashid:

```
pip install hashid
pip install --upgrade hashid
pip uninstall hashid
```

مثلاً بیایم تستش کنیم:

```
htpasswd -bnBC 10 "" password | tr -d ':\n'
hashid '$2y$10$1s6PYXSPwM/NmoYzrRF3VO47y3Sf68TWlVj17yTIOw.d7ZVB7lURK'

output:
[+] Blowfish(OpenBSD)
[+] Woltlab Burning Board 4.x
[+] bcrypt
```

راه بعدی؟ ما می‌دونیم همیشه بخشی از پسوردهای لورفته، پسوردهای ناامنی هستن که آدما انتخابش کردن. پسوردایی مثل «Alexa1234» یا «user1234» یا «password1234» و... .

خب پس من می‌تونم دیتابیسی از الگوریتم هش‌های مختلف این پسوردای رایج بسازم. بعد توی دیتابیس بگردم ببینم آیا چیزی رو می‌بینم که شبیه اینا باشه؟! اگر آره، خب عملاً می‌فهمم که الگوریتم هش چی بوده.

leaked hash:
b9c950640e1b3740e98acb93e669c65766f6670dd1609ba91ff41052ba48c6f3

| Hash function | Computed hash |
|---|---|
| Md5 (absolutely insecure) | bdc87b9c894da5168059e00ebffb9077 |
| SHA3-256 (insecure for password) | 2f7d3e77d0786c5d305c0afadd4c1a2a6869a3210956c963ad2420c52e797022 |
| SHA2-256 (insecure for password) | **b9c950640e1b3740e98acb93e669c65766f6670dd1609ba91ff41052ba48c6f3** |
| BCrypt | $2y$10$1s6PYXSPwM/NmoYzrRF3VO47y3Sf68TWlVj17yTIOw.d7ZVB7lURK |

با بررسی هش‌های پرتکرار، دیدیم یکی از هش‌هایی که توی دیتابیس لو رفته بود، هش «password1234» که با الگوریتم SHA2-256 هش شده بود هست.

پس فهمیدیم که هش مورد استفاده توسط سایت، SHA2-256 هست.

نمونه‌ای خیلی ساده از Hash table که با دیکشنری توی پایتون پیاده‌سازی شده:

```
sha3_256_dict = {
'fb001dfcffd1c899f3297871406242f097aecf1a5342ccf3ebcd116146188e4b': 'admin',
'9fc68b83499793838b29c32aa0661f4ef2cc03f49446de9499bb5d17994c850a': 'admin1234 ',
'b61aeb71c55c58496eb1a089ee9ccaac1e0375566fbd9e3d3d25741107242bce': 'P@$$w0rd',
'769b81efe68ed7dfec00c1901cbd5e84a6303bb1a14f2455c617ccff58ebe968': 'qwerty1234',
'e9b4deac3abe51686ea31012e2b7dee74ddf5b55fca7bd29c6780c969ea9f67b': 'Harper1984'
}
```

روی کلیدها حرکت می‌کنیم. هش که یافت شد، value که پسورد هست رو نگاه می‌کنیم.

## تفاوت Hash table و Rainbow table:

ببینین وقتی میخوان پسورد کرک کنن، لیست خیلی خیلی بلندی از پسوردای معروف و هشاشون دارن. میلیون‌ها پسورد. فرض کنیم. یک میلیارد پسورد داریم. فرض هم کنیم هش به کار رفته Bcrypt هست که ۲۳ بایت هشه (که به صورت ۳۱ ASCII کرکتر نمایش داده میشه).

خب فرض کنیم یک میلیارد پسورد داریم، میخوایم هششون رو ذخیره کنیم.

1,000,000,000 * 23 = 23,000,000,000

فقط هش‌ها ۲۳ میلیارد بایت میشن. یعنی ۲۳ گیگابایت فقط هش! خیلی زیاد شد که!
خب چیکار کنیم؟ میدونیم اگر یه قسمت کوچیک ورودی هم عوض شه، عملاً بخش بزرگی از هش عوض میشه! مثلاً برای SHA3-256 ببینیم:

```
hello this is Hannah:
64968d0d4b8c3cdb7effea421d314b33f39d282b977bc009876dfb0d184ea08d

Hello this is Hannah:
6de185ea103945a93ac1fd303d9b6e3bbda0b75870ba8189a23c5aa1365e2e9a
```

دیدین؟ فقط با عوض‌کردن بزرگی و کوچیکی فقط یکدونه حرف، هش کاملاً عوض شد! تشابهشون، ۲۰ درصد هست فقط. یعنی ۸۰ درصدش عوض شده!
خب یکم فکر کنین این به چه دردی میخوره که ما حجم رو کاهش بدیم.

*یه راهنمایی:* برای کاهش دادن حجم، باید مقدار کمتری string ذخیره کنیم.

خب چون هش عوض میشه و حالتی یونیکه (چون Hash collision) نداریم، فقط اگر یه چندتا از ابتدا و یه چندتا از انتها ذخیره کنیم، عملاً می‌تونیم با دیتابیس لورفته مطابقت بدیم مثل قبل. اما ایندفعه فقط چندتای اول و چندتای آخر رو چک می‌کنیم!
مثلاً اینطوری ذخیره می‌کنیم:

| Hash table | Rainbow table |
|---|---|
| **64968d0**d4b8c3cdb7effea421d314b33f39d282b977bc009876dfb0d1**84ea08d** | 64968d0 - 84ea08d |
| **6de185e**a103945a93ac1fd303d9b6e3bbda0b75870ba8189a23c5aa13**65e2e9a** | 6de185e - 65e2e9a |
| **920516d**6e207da345ea4025452e20adaf2672473f709d9a67e88601c7**79f7be3** | 920516d - 79f7be3 |

توی hash table attack، ما میایم کل هش رو مقایسه می‌کنیم، اما توی Rainbow table attack میایم چندتای اول و آخر رو با دیتابیس چک می‌کنیم! اینطوری باعث میشه چیز کمتری ذخیره کنیم و حجم کمتری برای ذخیره‌سازی صرف کنیم!

- خب درسته میگی اول و آخر چک می‌کنیم و اگر یکسان بود به احتمال خیلی زیاد، اون هش، مثلاً admin هست که اول و آخرشو ذخیره کردیم. اما خب ممکنه یه پسورد دیگه هم اول و آخرش شبیه همین شه. مثلاً اینطوری باشه:

admin hash: **6de185e**a103945a93ac1fd303d9b6e3bbda0b75870ba8189a23c5aa13**65e2e9a**
another hash:
**6de185e**a10394507da345ea402bea5452e20adaa0b75870baa23cbe21**65e2e9a**

درسته قبول دارم که ممکنه یه هش دیگه‌ای پیدا شه که اول و آخرش یکسان شه ولی وسطش متفاوت. اما این احتمال خیلی خیلی پایینه. به چند دلیل:

۱- ما داریم پسوردهای معروف رو ذخیره می‌کنیم. پس عملاً احتمال اینکه یه پسورد معروف پیداشده باشه تا یه پسورد عجیب غریب که صرفاً هشش با این یکسان باشه خیلی بیشتره!

۲- این هش‌ها، هش‌های cryptographic هستن و عملاً با کوچک‌ترین تغییر، خروجی کلاً عوض میشه و یافتن دو متفاوت که چندکرکتر اول و آخرشون یکسان باشه، بسیار احتمالش کمه. یعنی بر این اساس هم به احتمال زیاد، همون هشی هست که مد نظر ماست.

۳- فرض کنیم خیلی خیلی خیلی زیاد بدشانس باشیم و حرف شما درست جور در بیاد. خب ایرادی نداره! موقعی که من دارم به اکانت‌های موجود در دیتابیس نفوذ می‌کنم، از مثلاً یه میلیون اکانت، ۱۰۰۰ تاش هم نفوذ کنم و اطلاعاتشون رو بفروشم برام کافیه. پولمو گرفتم بسمه دیگه! پس اینجا هم زیاد اهمیتی برام نداره حالا با همون احتمال خیلی خیلی کم، نتونم به یه اکانت وارد شم! فدا سرم! مگر اینکه حتماً بخوام به یه اکانت خاص حتماً وارد شم. اما خب بازم اون طبق مورد «۱» و «۲» احتمالش بسیار کمه!

به نظرتون نحوه مقابله با این نوع حملات چیه؟ استفاده از salt.

## Salting[46]

بیایم فکر کنیم و ماهی‌گیری رو یاد بگیریم. Rainbow table attack از این به وجود میومد که هش admin رو همه میدونن. پس اگر دیتابیس لو بره، میتونن برسن بهش. خب چیکار کنیم؟ باید کاری کنیم که هش admin، متفاوت باشه. اما نمیشه که! یادمونه یکی از قابلیت‌های هش این بود که هزار بار هم هش یه چیز خاص رو حساب کنیم، جواب همونه! پس چیکار کنیم؟

اگر به ابتدای پسورد، یه string رندوم اضافه کنیم، هش متفاوت میشه. مثلاً به جای اینکه هش admin رو حساب کنیم، هش abcdeadmin رو حساب می‌کنیم! مردم هش admin رو میدونستن. اما هش abcdeadmin رو که حساب نکردن! پس از rainbow table attack و hash table جلوگیری می‌کنیم. مثلاً:

```
hash("avjFSsftAn"+"admin")
hash("saFqGxAgHw"+"1234")
```

اینطوری هش ایجاد شده، با هش مثلاً admin متفاوته و نمیشه اتک زد بهش. اما کلید مؤثر بودن هش، در طول اون (هرچی بیشتر بهتر)[47] و رندوم‌بودنش برای هر پسورد هست.

خب از کجا مشخص میشه که salting استفاده نکرده؟!

---

۴۶ قسمتی از دانشم رو از وبسایت
https://auth0.com/blog/adding-salt-to-hashing-a-better-way-to-store-passwords
یادگرفتم

۴۷ چون هرچی طولانی‌تر باشه، یافتن salt سخت‌تره و brute-force کردن خود salt کُلی زمان می‌بره.

فرض کنیم دیتابیس زیر دست هکرای بد افتاده:

| Hash | Username |
|---|---|
| 8f4343466ae348ff6b96df89dda901c | Alex |
| Jsdvs9fdsaffas1jf823fj81rnowour8r91 | Amanda |
| 8f4343466ae348ff6b96df89dda901c | Angelina |
| Sdi834003jfsau83fmai8rkfo2fk082rko | Hannah |
| dsfajf298rumow8er0d019uejdmklacm | Harper |
| sjfd09r03329rjowierfjslaur20kf2903ri | Hazel |

خب چه چیزی نظرمون رو توی دیتابیس جمع میکنه؟ درسته! هش Alex و Angelina یکسان شده! یعنی من میفهمم که salt استفاده نکرده! اگر هم کرده، رندوم و متفاوت برای همه نبوده! (یعنی اگر سالت رو پیدا کنم. برای همه همین یکیه و نیاز نیست برای هرکدوم سالت جدید پیدا کنم.)

وقتی هش salt رندوم داشته باشه، یه bottleneck ای هم برای اون اتکری که به دیتابیس نفوذ کرده ایجاد می‌کنیم. که مجبوره هی هش رو دوباره حساب کنه و هی اینپوت و... بگیره. یعنی نمی‌تونه از چیزای از پیش حساب شده استفاده کنه. باید هی بشینه دونه دونه دوباره حساب کنه.

آیا راهی هست بشه سالتینگ رو دور زد؟ نفوذ به سرورا! چون توی سرور هش حساب می‌کنین، سالتینگ سرور هم اعمال میشه. (سرور سالت‌هاشو توی خودش نگه میداره) پس با نفوذ به سرورا، عملاً میشه تا حدی سالتینگ رو دور زد و شروع به بروت فورس کرد ولی باز باید هش رو حساب کنین. (سالت رو دارین. ولی محاسبه و brute-force کردن هش برای یافتن پسورد اولیه هنوز پابرجاست.)

+ سالت هم باید مقدار مناسب باشه. رندوم و با فلان قدر بیت که توصیه شده مثلاً برای PBKDF2، فکر کنم باید salt، استرینگی به طول ۱۲۸ بیت باشه. از سایت زیر میتونین استفاده کنین که طول salt رو بدونین:

https://cheatsheetseries.owasp.org/cheatsheets Password_Storage_Cheat_Sheet.html#salting

هش‌هایی که برای تولید secret استفاده میشن رو یه نگاه بندازیم:

- Argon2

+ Winner of Password Hashing Competition[48]

48 https://www.password-hashing.net/

+ Used in: KeepassXC[49] - Bitwarden[50] - Tutanota[51] [52]

+ RFC 9106[53] [54]

- ## Scrypt

+ RFC 7914[55] by IETF

- ## Bcrypt

+ Used in: Proton Pass[56] - Formally at Tutanota[57]

- ## PBKDF2

+ Recommened by NIST[58] *(Outdated!)*

NIST گفته ما از آدما شنیدیم که دنیا تغییر کرده و کارت‌های گرافیک جدیدتری اومدن که می‌تونن brute-force هش‌ها رو ساده‌تر کنن (مثل RTX4090). حالا ما از شما نظرات شما رو شنیدیم و تصمیم گرفتیم توصیه‌نامتون رو بروزرسانی کنیم:

"In response to the public comments received, NIST proposes to revise SP 800-132,

- to approve an additional memory-hard password-based key derivation function and password hashing scheme, and
- to provide additional guidelines and clarifications on the use of PBKDF2."[59]

منتظر آپدیت باشین[۶۰]

این خیلی خوبه که سازمان NIST تک‌رو نیست و به توصیه‌های آدمای مختلف و توصیه‌های آکادمیک گوش میده.

---

49 https://keepassxc.org/docs/KeePassXC_UserGuide#_database_settings

50 https://bitwarden.com/help/what-encryption-is-used/#argon2id

51 https://tutanota.com/blog/best-encryption-with-kdf

۵۲ یه تذکر! یه جای این سایت گفته که اگر یه وقت به warning برخوردین، یکی از راهکاراتون:
"Launch another browser, using the running Tor instance as a proxy."
می‌تونه باشه. این خیلی توصیه‌ی خطرناکی هست! از Tutanota بعیده! احتمالاً یه فردی این مطلبو نوشته که دانش نداشته. استفاده از مرورگر دیگه‌ای با پراکسی تور، بسیار بسیار اقدام خطرناکی برای ناشناسی هست. به راحتی ناشناسیتون به خطر میوفته.

53 https://datatracker.ietf.org/doc/html/rfc9106

۵۴ یه سری نوشته‌های فنی و راهنمایی تحت عنوان RFC منتشر میشن.

55 https://www.rfc-editor.org/rfc/rfc7914

56 https://proton.me/blog/proton-pass-security-model

57 https://web.archive.org/web/20230203231101/https://tutanota.com/blog/images/encryption/registration.webp

58 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-63b.pdf

59 https://csrc.nist.gov/news/2023/proposal-to-revise-nist-sp-800-132-pbkdf

60 https://csrc.nist.gov/news/2023/decision-to-revise-nist-sp-800-132

یه سوال؟ آیا PBKDF2 برای تولید کلید رمزنگاری ساخته شده و توصیه NIST هم برای تولید کلید رمزنگاریه یا برای ذخیره پسورد (نه تولید کلید رمزنگاری) هم هست؟ (اگر می‌دونین، بگین بهم!)

+ Used in: Lastpass[61] - Bitwarden[62] - 1Password[63] - Apple (e.g. Secure notes)[64]

نکته‌های دیگر:

"The iteration count shall be selected as large as possible, as long as the time required to generate the key using the entered password is acceptable for the users. Since the capability of user machines varies (e.g., from a smart card to high- end workstations or servers), reasonable iteration counts vary accordingly."

"For especially critical keys, or for very powerful systems or systems where user-perceived performance is not critical, an iteration count of 10,000,000 may be appropriate."[65]

برای همینم اپل از ۱۰ میلیون iteration استفاده می‌کنه:

"The keybag—protected with the password set—is run through 10 million iterations of the key derivation function PBKDF2. Despite this large iteration count, there's no tie to a specific device, and therefore a brute-force attack parallelized across many computers could theoretically be attempted on the backup keybag. This threat can be mitigated with a sufficiently strong password."[66]

با پیشرفت GPU ها، تعداد iteration الگوریتم PBKDF2 رو بیشتر کردن (که کار حمله‌کننده سخت‌تر شه) ولی برای اکانت‌های قدیمی‌تر اعمال نشده و باید برین دستی تغییرش بدین!

خلاصه کلام! اگر پسورد منیجر یا چیزی استفاده می‌کردین که از PBKDF2 استفاده می‌کنه، احتمالاً باید برین دستی سختی الگوریتم (تعداد iteration ها رو) بیشتر کنین!

"The default minimum number of password iterations is 600,000 rounds (for new accounts and those who update their existing iteration count)."[67] -Lastpass

61 https://support.lastpass.com/s/document-item?bundleId=lastpass&topicId=LastPass%2Fabout-password-iterations.html

62 https://bitwarden.com/help/what-encryption-is-used/#pbkdf2

63 https://support.1password.com/1password-security/, https://1passwordstatic.com/files/security/1password-white-paper.pdf

64 https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf

65 NIST Special Publicatoin 800-132: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf, RFC 8018 quoted this as well: https://datatracker.ietf.org/doc/html/rfc8018#section-4.2

66 Apple Platform Security, May 2022, page 81: https://help.apple.com/pdf/security/en_US/apple-platform-security-guide.pdf

67 https://support.lastpass.com/s/document-item?bundleId=lastpass&topicId=LastPass%2Fabout-password-iterations.html

"The number of default iterations used by Bitwarden was increased in February, 2023. Accounts created after that time will use 600,001, however if you created your account prior to then you should increase the iteration count. Instructions for doing so can be found in the following section."[68] -Bitwarden

- ## Balloon

**Recommended by NIST** *(Outdated! Wait for its update.)*

### نکته‌ها:

+ به طور کلی در الگوریتم‌های هش secret سعی میشه که مثلاً اگر میگیم بـه جـای یـک عملیـات، ۲ عملیات انجام شه، برای حمله‌کننده، ۲ برابر سخت‌تر نشه. بلکه حتی بیشتر سخت‌تر بشـه. یعـنی مثلاً n^3 برابر سخت‌تر شه. یعنی:

1 operation → difficulty: 1

3 operation → difficulty: 27 (n^3)

اصلاحا میگیم سختیش به صورت توان‌دار و یا حتی exponentially (مثلاً $2^n$ ) زیادتر شـه بهـتره. اینطوری منِ کاربر یکم کندی بیشتری تحمل می‌کنم، ولی حمله‌کننده خیلی کندی و سختی بیشتری باید تحمل کنه.[۶۹]

+ صرف استفاده از password hashing algorithm (PHA) ها، باعث نمیشه اکانت در برابر brute-force و اینا مقاوم باشه. درواقع اگر پسورد بد انتخاب شده باشه، PHA هـا نمی‌تـونن دفـاع خـوبی داشـته باشن. چون من به عنوان حمله‌کننده اول پسوردای ساده و پرتکرار رو چـک می‌کنم. پس نتیجـه می‌گـیریم که اگر سیستمی می‌سازیم، حتماً حتماً پسوردهای ثبت‌نامی رو چک کنیم که ناامن نباشه.

وقتی می‌گم درست، یعنی واقعاً درست. نه اینکه صرفاً چهارتا شرط بذارین که اگر طول بیشتر از ۱۲ بود و یک حرف بزرگ و یک کرکتر و سمبل هم داشت اوکیه. نه اینطوری نباشـه! بلکـه بایـد بـه صـورت امن و درستی چک شه. درباره این بعداً صحبت می‌کنیم.[۷۰]

+ این هش‌ها یه سری پارامتر دارن. مقادیر و پارامترهارو سعی کنین از منابع معتـبرتر بگـیرین. منـابعی مثل RFC ها یا توصیه‌های سازمان استاندارد یا community هایی معروف و معتبرتر:

+ Password Storage Cheat Sheet[71] *(OWASP)*

---

68 https://bitwarden.com/help/what-encryption-is-used/#pbkdf2

69 More at: https://blog.1password.com/bcrypt-is-great-but-is-password-cracking-infeasible/

۷۰ چیزایی مثل چک پسورد که داخل پسوردای لو نرفته نباشه، توی بلک‌لیست نباشه و... .

71 https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

این خیلی جالبه که ما اینقدر ساده داریم از اینترنت استفاده می‌کنیم، اما اون پشت و در جزئیات، یه عده تحقیقات بزرگ و پیچیده‌ای برای تولید استاندارد و تولید نوشته‌های فنی و راهنمایی‌ها (RFC ها) صرف می‌کنن.

+ هش‌کردن بهتره هم سمت client صورت بگیره و هم سمت server. اینطوری از یه سری حملات جلوگیری میشه.

## More:

+ Hashing in Action: Understanding bcrypt[72]

## کاربرد:

- ### Bitcoin (SHA2-256)

توضیح خیلی ساده و **غیردقیق** درباره بیت‌کوین و بلاکچین (توضیح **دقیق نیست و صرفاً برای دادن درک هست!** و اینجا بحث بحث دقیق‌بودن مسائل فنی نیست. صرفاً اینه که بدونیم هش کاربرد داره):

قبلش باید درباره فلسفه پول صحبت کنیم. قبلاً مردم مبادله کالا به کالا میکردن. یعنی چی؟ یکی می‌گفت بیا من دوتا مثلاً سیب میدم بهت تو دوتا پرتقال بده بهم. اما خب این مشکل پیدا کرد. مثلاً من میخواستم برم یه شهر دیگه. اونجا چیکار کنم؟ نمیشد با خودم سیب و پرتقال حمل کنم. اینجا افراد می‌رفتن پیش یه سری افراد معتبر و درست‌کار که امین مردم بودن. مثلاً ده تا سیب میداد به این فرد معتمد و فرد معتمد یه کاغذی چیزی میداد دست اون فرد و روش نوشته بود که آقا فلانی به ارزش فلان مقدار پیش من امانت داره. اونجا که اومد، به اندازه این مقدار بهش جنس یا هرچی میخواد بدین، من بعداً بهتون پس میدم. پس کم کم این رایج شد و شد پایه و اساس پول امروزی.

درواقع پول امروزی هم همینه. این اسکناسی که دست شماست، ارزشی نداره. بلکه نشون دهنده اینه که آره من به ازای ارزش پنج هزار تومن اسکناسی که دارم، ۵ هزارتومن توی بانک مرکزی، طلا دارم. درواقع برای همین هست که میگن پشتوانه پول چیه. پشتوانه پول هر کشور، طلاهایی هست که به ازای اسکناس‌هایی که چاپ میکنه در بانک مرکزیش داره. حالا نه لزوماً طلا میشه چیزای دیگه که برای پول ارزش میارن. طلا مثال بارزشه.

برای همین هم هست که این سؤال رد میشه که «چرا همینطور پول چاپ نمیکنن و همه پول‌دار شن؟»

+ چون این پول به خودی خود ارزشی نداره و ارزشش به واسطه وجود پشتوانه پشتش هست. برای همین میگن الکی اسکناس چاپ نکنین، چون ارزش پول میاد پایین.

درواقع فرض کنین ما ۱ میلیون تومن اسکناس داریم و ۱ میلیون تومن هم طلا. حالا اگر ۱ میلیون دیگه هم اسکناس چاپ کنیم و یعنی به ازای ۱ میلیون طلایی که داریم، ۲ میلیون اسکناس داشته باشیم،

72 https://auth0.com/blog/hashing-in-action-understanding-bcrypt/

ارزش اسکناس ما نصف میشه. چون به ازای ۲ میلیون تومنی که اسکناس داریم، ۱ میلیـون طلا داریم. پس انگار هر ۲ اسکناس ما، ۱ واحد طلا هست. پس هرچی اسکناس بدون پشتوانه تولید شه، به ضرر ارزش پوله.

خب پس فهمیدیم پول چه‌جوری به وجود اومد. حالا یه سوال.

یه سؤال طلا برای چی ارزش داره؟

+ یکی از دلایلش اینه که خاصیت‌هایی داره که در جاهای متفاوت ازش استفاده میشه. اما فارغ از دلایل کاربرد طلا در صنعت و... یه دلیل دیگه هم داره. اونم اینکه چون یافتنش سخته. چون به سبب کمیـابیش و تقاضاش ارزش داره. اگر فردا یه معدن بزرگ طلا یافت شه، ارزش طلا کم میشه. اگـر هم یـه روز همـه از خواب بیدار شدن بگن دیگه طلا رو قبول نداریم، تقاضا کم میشه و وقتی تقاضا کم شه، کسی ارزشـی بـرای طلا قائل نیست و ارزش طلا بازم کمتر میشه (حالا فارغ از خاصیت‌های فیزیکیش)

ببینین بیت‌کوین اومد گفت آقا هرکسی که یه کامپیوتر عادی هم داره بتونه منـو داشـته باشـه و مـاین کنه. خیلی هم خوب! هرکسی باید بتونه سهمی داشته باشه. اما چجـور این بیت‌کـوین رو پخش کـنیم بین افراد؟ به معنای واقعی کلمه، افراد باید زحمت بکشن. مثل همون یـافتن طلا کـه زحمت داشـت، اینجـا هم افراد باید اثبات کنن که زحمت کشیدن. یا درواقع Proof Of Work. یعنی اثبات انجام کار. حالا یکی از اثباتای انجام کار چیه؟ حدس بزنین؟ آره محاسبه هش! اما بیت‌کـوین از هش SHA-256 اسـتفاده می‌کنـه که به راحتی نشون دادیم چقدر زود محاسبه میشه. برای همین ساتوشی ناکاموتو اومد گفت آدمـا هی بایـد هش حساب کنن تا توی هششون مثلاً فلان قدر صفر پیدا شه. یعنی هی باید بروت‌فورس کنن تـا اون مثلاً صفرهای خاص پیدا شه. تا اینجا همه چی عادیه. اما این اوضاع ناعادلانه شد. درواقـع دسـتگاه‌هایی سـاخته شد مخصوص فقط ماین بیت‌کـوین. می‌دونیم کامپیوترهـای مـورد اسـتفاده مـا می‌تـونن بـازی اجـرا کنن، وبگردی کنن و ویدیو ببینین و هزار تا کار دیگه. و همین چندمنظوره بـودنش یعـنی می‌تونـه همـه کـار رو انجام بده ولی نه لزوماً با سرعت خیلی بالا. ولی دستگاه‌هایی به وجود اومد که فقط و فقـط بلـد بـودن هش حساب کنن ولی اون رو با سرعت خیلی خیلی بالاتری حساب می‌کردن. به اینا می‌گن ماینر. یا ماین‌کننـده. اینجا اوضاع ناعادلانه شد. چون هرکسی پول بیشتری داشت می‌تونست دستگاه‌های بیشـتری بخـره. یعـنی پول‌دار پول‌دارتر میشد و فقیر فقیرتر. چـون گـوی رقـابت محاسـبه هش رو اون ثروتمنـده کـه می‌تونسـت دستگاه بخره می‌برد. برای همین کوین‌های دیگه‌ای اومدن که از هش‌های مقاوم‌تری در برابر این دسـتگاه‌ها استفاده کردن که به سادگی نمیشه براش دستگاه‌های مخصوص ساخت. هزینه ساخت اون دستگاه‌ها هم به شدت میره بالا و عدالت یکم بهتر حفظ میشه. ولی بـرای بیت‌کـوین بـا یـه هزینـه کم بیشـتر، پـول خیلی بیشتری نسیب شما می‌شد.

- به نظرتون این هش‌ها چه نوع هشایی می‌تونن باشن؟ هشای پسورد!

+ میشه از هشای مربوط به پسورد استفاده کرد. کاری که کوین‌های زیر مشابهشو می‌کنن.

- Litecoin (Scrypt)
- Zcash (Equihash)

## • Cloud-base applications

موقعی که شما یه فایل می‌خواین برای دوستتون توی چت عادی تلگرم بفرستین، گاهی دیدین تا دکمه ارسال رو می‌زنین، اون فایل سریع دایرش پر میشه و ارسال میشه. این چجوریه؟! چه‌جوری بـه این سـرعت ارسال شد و تیک خورد؟!

به این دلیله که تلگرم نگاه می‌کنه میگه عه! هش این فایلی که داری می‌فرستی بـا هش یکی از فـایلای توی سرور من یکیه. خب چرا بخوای الکی پول اینترنت بدی؟! نیاز نیست بفرستیش! من خـودم فـایلی کـه هشش عین همینه (یعنی عیناً همین فایل هست)، رو از توی سرورم برای فرد مقابل می‌فرستم!

### یافتن فایل مشخص در دیتابیس

بعضی اوقات نیازه یه فایل خاصی رو پیدا کنیم. میگیم هی هش حساب کن ببین فایل پیدا میشه یا نـه. مثلاً می‌خوایم یه عکسی که کاپی رایت داره رو پیدا کنیم. خب کل گوشی طرف رو اسـکن می‌کـنیم و اگـر هش پیدا شد، یافتیمش :)

دقیقاً اینطوری دیتابیس دارن که شامل هش‌های فایل‌های کاپی رایت‌دار هست که بـا فیلمـای جاهـای مختلف، عکسای جاهای مختلف تطبیق بدن.

- راه دورزدنش چیه؟

+ یه تغییر خیلی کوچیک توی متن پشت عکس یا فیلم یا... انجام بدیم که هش عوض شه[۷۳] :)

## More:

+ How Hash-Based Safe Browsing Works in Google Chrome[74]

۷۳ البته با اومدن هوش مصنوعی، دیگه میان خود عکس رو آنالیز می‌کنن و نه لزوماً هش تنها.

74 https://security.googleblog.com/2022/08/how-hash-based-safe-browsing-works-in.html

# رمزنگاری

بریم یه خرده وارد رمزنگاری شیم که بتونیم بهتر درکش کنیم که اصلاً رمزنگاری یعنی چی؟
فرض کنین من می‌خوام یه نامه برای یکی بنویسم. خب باید با پست ارسال شه. من از کجا بدونم کارمند پست، نامم رو باز نمی‌کنه که بخونتش؟ هیچ تضمینی نیست که کسی وسط راه نخونتش!
اینجا بحث رمزنگاری پیش میاد. یعنی من بیام یه متن رمزی بنویسم که فقط خودم و فرد مقابلم بفهمه یعنی چی. مثلاً به نظرتون متن زیر یعنی چی؟

– هنوخ مدیسر نم مالس

عجیبه نه؟ یکم فکر کنین ببینین میتونین بفهمین چیه؟
خب سعی کنین از آخر به اول بخونینش. چی میشه؟
میشه:

+ سلام من رسیدم خونه.

دیدین چی شد؟ عملا پیام بالا قابل خوندن نیست و تنها کسی که بدونه چه‌جوری باید بخونتش، میفهمه من چی نوشته بودم. به این تکنیکا که عملاً پیام اصلی مشخص نباشه که چی هست و عملاً من پیام اصلی رو پنهان کردم، رمزنگاری و پنهان‌نگاری و این‌ها میگن. بعداً با هم تفاوتاشونو می‌فهمیم.
بیایم یه نمونه دیگه رو ببینیم. جمله زیر یعنی چی؟

– شمبن نو زشاذن دهوی.

+ یکم سعی کنیم با متن بازی کنیم. از آخر به اول بخونیم. چیزی دستگیرمون نشد. بیایم حرف اول هر کلمه رو کنار هم قرار بدیم. میشه «شنزد»! بازم چیزی دستگیرمون نشد. پس چیه؟
بازم همون «سلام من رسیدم خونه» هست! ایندفعه من اومدم جای هر حرف، حرف بعدیش رو توی حروف الفبا گذاشتم! جای «س»، حرف «ش». جای «ل»، حرف بعدیش که «م» هست و الی آخر.
جالب بود نه؟
درواقع من تلاش کردم که پیامی رو به صورت رمزی بنویسم که کسی نفهمه.

*اصل اول در رمزنگاری:*

مهم نیست که آیا داره پیامی ارسال میشه یا نه. اما نباید معلوم بشه اصل منظور ما چیه. یعنی ایرادی نداره یه متن رمزشده منتقل شه. فقط باید اون متن، رمزشده باشه که معلوم نشه منظور و پیام اصلی ما چی بوده.

یه رمزنگار قرن ۱۹، یه سری قانون‌طور برای رمزنگاری نظامی ارایه کرد. یکی از اون‌ها الآن پایه و اساس رمزنگاری‌های مدرنه که خوبه باهاش آشنا شیم:

## Kerckhoffs's six design rules for military cipher

1. The system must be practically, if not mathematically, indecipherable;

2. It should not require secrecy, and it should not be a problem if it falls into enemy hands;
3. It must be possible to communicate and remember the key without using written notes, and correspondents must be able to change or modify it at will;
4. It must be applicable to telegraph communications;
5. It must be portable, and should not require several persons to handle or operate;
6. Lastly, given the circumstances in which it is to be used, the system must be easy to use and should not be stressful to use or require its users to know and comply with a long list of rules.[75]

چیزی که امروزه برای ما اهمیت داره، دومیشه. یعنی امن‌بودن اون سیستم، به این وابسته نباشه که مخفی باشه. بلکه درواقع حتی اگر همه روندش هم پابلیک باشه، کسی نباید بتونه بشکونتش! این به «Kerckhoffs's principle» معروفه. (درواقع این رمزای ساده‌ای که با هم ساختیم، اصلاً امن نبودن!)

مثلاً بعضی شرکتا برای اینکه زمانی که شما رمزتون رو یادتون بره، بازم بتونین به کامپیوتر دسترسی داشته باشین، یه راه برای ریکاوری رمز قرار میدادن. اینطور بود که تو مثلاً یه سری اطلاعات از سیستمت میدادی، اونا یه شاه‌کلید بهت میدادن که بتونی وارد شی.[۷۶]

خب حدس بزنین مشکل چیه؟

مشکل اینه که اگر اون روش به دست آوردن شاه کلیدمون لو بره، عملاً امنیت بی‌معنا میشه.

درواقع امنیت یه چیز نباید به پنهون‌بودنش بستگی داشته باشه.

فرض کنین امنیت یه چیز به پنهون‌بودنش بستگی داشته باشه. حالا اگر اون راز لو بره، خب دیگه سیستم امن نیست! باید از اول یه سیستم جدید طراحی کنی و این خیلی بده! نمیشه از اول بشینی یه سیستم جدید طراحی کنی. تمام چیزای قبلی عملاً بی‌فایده هستن.

یا حتی نمیشه اون رو مورد تست و بررسی قرار داد. صرفاً باید اعتماد کنی که درست کار می‌کنه. نمی‌تونی تستش کنی. چون مخفیه.

درواقع توی رمزنگاری ما می‌گیم که فرایند باید شفاف و کاملاً پابلیک باشه ولی از لحاظ ریاضیات و رمزنگاری نشه بشکونیمش! یعنی حتی اگر به دست دشمن برسه، نتونن بشکوننش.

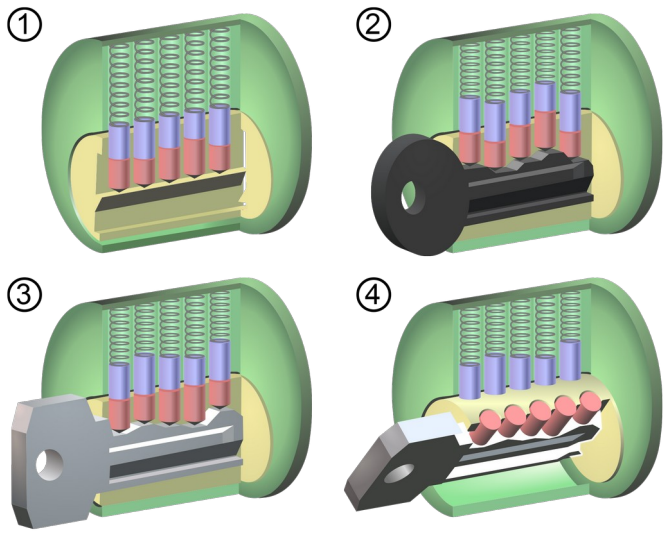
پس چیکار کنیم؟ چطور ممکنه سیستمی طراحی کنیم که مشخص باشه چطور کار می‌کنه ولی نشه شکستش؟! خیلی عجیبه! خب اگر بدونیم چطور کار می‌کنه، خب برعکسش رو می‌ریم تا بشکنه.

بذارین یه مثال بزنم. هش رو یادتونه؟ موقعی که صحبت از باقی‌مونده‌گرفتن توی هش خودساخته‌ام کردم رو یادتونه؟ حالا فرض کنین مثلاً من یه الگوریتمی بسازم که مثلاً میگم ۲ به توان ۱۲۸ بار بیاد باقی‌مونده بگیره. خب حالا از خروجی بخوایم ورودی رو بفهمیم که چی بود، خیلی خیلی خیلی سخت و زمان‌بره. همینطور باید حالات مختلف رو تست کنیم. اونقدر حالاتی که باید تست کرد زیاد میشه که با

75 https://en.wikipedia.org/wiki/Kerckhoffs%27s_principle#Origins
76 BIOS Password Backdoors in Laptops: http://dogber1.blogspot.com/2009/05/table-of-reverse-engineered-bios.html (But it's a little bit old)

قدرت کامپیوترهای الآن و حتی سوپرکامپیوترهای آینده هم نمیشه دونستش! پس چیز خیلی عجیبی نیست![۷۷] (یا XOR های متوالی)

درواقع ما تلاش نمی‌کنیم سیستم رو مخفی کنیم. سیستم شفافه **جز** یه چیز. کلید! همه می‌دونیم قفل چطور کار می‌کنه ولی تا وقتی کلیدش نباشه باز نمیشه.
درواقع قفل در خونه اینطوریه:




image[78]

درواقع یه سری میله چسبیده به فنر هست. کلید که داخل میره، میله‌ها میرن بالا و پایین که قسمت برش‌خوردشون، روبروی قسمت چرخان قرار بگیره. حالا با چرخوندن (چون میله‌ها درست قرار گرفتن که بشه چرخید)، قفل باز میشه. فقط زمانی که کلید هست، میشه بازش کرد.
درواقع امنیت رمزنگاری به کلیدیه که داره. اگر کلید لو رفت، زودی مثل عوض‌کردن پسورد، کلید رو عوض می‌کنیم (دیتا رو با یه قفل دیگه رمز می‌کنیم). دوباره امن میشه. یعنی با یه کار ساده سیستم امن میشه و نیاز به طراحی مجدد یه سیستم دیگه نیست. پس امنیت به اون کلید ماست که راحت هم میشه عوضش کرد.

یعنی میشه تستش کرد بررسیش کرد. مثلاً سازمان استاندارد بیاد تستش کنه و ببینه واقعاً امنه یا نه. ولی اگر ندونه چطور کار می‌کنه و امنیتش صرفاً به پنهون‌بودنش بستگی داشته باشه، هیچ‌وقت نمیشه استانداردش کرد و یا مورد تستش قرار داد. (بعداً توی قسمت «*در امنیت، فلاقیت شخصی بدون بررسی توسط متخصصان، امن نیست!*» بیشتر با هم صحبت می‌کنیم)

---

۷۷ فکر نکنین رمزنگاری صرفاً باقی‌مانده هستا! این صرفاً یه مثال بود!

78 PNG: User:Wapcaplet; SVG: User:Pbroks13; this combined image: User:Crisco 1492., https://commons.wikimedia.org/wiki/File:Tumbler_key_lock_explained_in_four_steps.png, Creative Commons Attribution-Share Alike 3.0 Unported, File:Pin tumbler with key.svg, File:Pin tumbler unlocked.svg, File:Pin tumbler no key.svg, File:Pin tumbler bad key.svg,

حالا بیایم یه مورد دیگه رو ببینیم:

jdosf@hdfsj987&ijksd-isj!mjlsfj9)bkjsd+akdlfj42fs%cjsdlfjoi1}kpak

به نظرتون این چه چی هست؟ یکم دقت کنین!

خب من این رمز رو خودم ساختم و با خودم قرارداد کردم که دقیقاً حرف بعد هر سمبل خاص (مثل {}- =()@$+ و...)، جمله من رو میسازه. یعنی:

jdosf@hdfsj987&ijksd-isj!mjlsfj9)bkjsd+akdlfj42fs%cjsdlfjoi1}kpak

Hi. I'm back

درواقع یه پیام رو توی دل یه پیام دیگه پنهون کردم. به این کار میگن پنهون‌نگاری. (با رمزنگاری متفاوته)

*مثال‌هایی بیشتر از پنهان‌نگاری:*

پنهان‌نگاری موجود در این جدول رو بشکنین:

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

پاسخ:

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

We need help!

مشابهش توی ویکی‌پدیا هست[۷۹]:

```
W . . . E . . . C . . . R . . . L . . . T . . . E
. E . R . D . S . O . E . E . F . E . A . O . C .
. . A . . . I . . . V . . . D . . . E . . . N . .
```

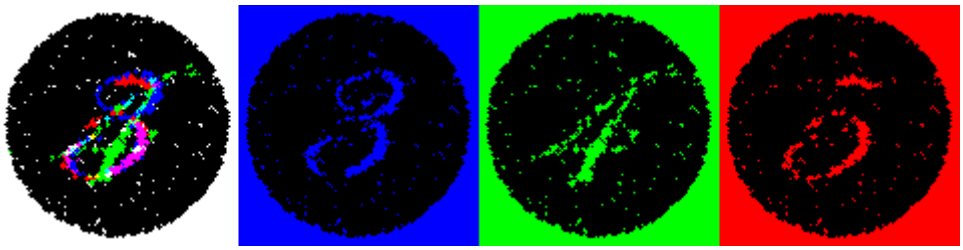
**Message:** "WE ARE DISCOVERED FLEE AT ONSteganographyCE"

*اصل اول در پنهان‌نگاری* (Steganography):

پنهون‌کردن پیام در دِل یه چیز دیگه. باید اصلاً انتقال پیام پنهان باشه. یعنی اصلاً کسی نفهمه که ما قصدمون ارسال پیام هست. یعنی هدف اصلی، نفهمیدن اینکه اصلاً پیامی فرستادم هست.

مثلاً قدیما میومدن سَرِ پیک نامه‌رسان رو کچل می‌کردن. پیغام رو روی سرش تتو می‌کردن و بعد وایمیستادن موهاش بلند شه و تتو پنهون شه. بعد می‌گفتن حالا که پنهون شد، برو فلان جا پیام برسون. برو اونجا باز کچلت کنن پیامو بخونن :))) *(بدبخت پیامرسانا!)*

یا بیاین مثلاً این عکس رو ببینین:



image[80]

اگر این عکس سمت چپ که سفید هست رو با رنگ آبی ببینیم عدد ۸. اگر با رنگ سبز ببینیم عدد ۵ و اگر با رنگ قرمز ببینیم، عدد ۵ مشخص میشه! این عملاً steganography هست. چون پنهونه و معلوم نیست که اصلاً می‌خواستیم مثلاً عدد ۵ رو ارسال کنیم!

## مخفی‌کاری در بیت‌های کم‌ارزش عکس[۸۱]

79 https://en.wikipedia.org/wiki/Transposition_cipher, Creative Commons Attribution-ShareAlike License 4.0

80 https://commons.wikimedia.org/wiki/File:Steganography.png, Creative Commons CC0 1.0 Universal Public Domain Dedication,

۸۱ بعد خوندن ایستگاه اعداد، برگردین به این:
فرض کنین من بخوام برای جاسوسم توی یه کشور دیگه یه پیغام بفرستم. اگر توی یه جای امن بفرستم ایمیل رمزشده، دولت اون کشور می‌بینه که هر روز ساعت فلان، فلانی از یه ایمیل رمزشده پیغام دریافت می‌کنه. خب پس شک می‌کنه که شاید داره اطلاعات مهمی رو منتقل می‌کنه. به جاش یه عکس خیلی عادی رو مثلاً توی eBay می‌ذارم و کسی هم نمی‌دونه این عکس حاوی اطلاعات مهمیه. اون فرد همیشه میاد eBay و عکس منو دانلود می‌کنه و پیغامو می‌خونه :)
یا یه ویدیو. دولت اون کشور شک نمی‌کنه. چون فکر می‌کنه خب هر روز داره میره ویدیو نگاه می‌کنه دیگه. از ایمیل رمزشده که چیزی نگرفته!

مثلاً یکی از تکنیک‌های مخفی‌نگاری اینه که بیایم بیت‌هـای کم‌ارزش یـه عکس رو بـرداریم و عوضـش کنیم و توش اطلاعات مخفی کنیم. به دلیل اینکه بیت‌های کم‌ارزش تأثیر کمی توی تصـویر دارن[۸۲]، میشـه عوضش کرد بدون اینکه تصویر عوض شه. میشه توی بیت‌های کم‌ارزش اطلاعات مخفی کرد.

– من می‌تونم یه متنی رو رمز کنم و بعد با steganogaphy مخفیش کنم؟ اینطوری هم پیام مخفیه و هم اگر پیدا هم شه، نمی‌تونن بخوننشون.

+ آره میشه. البته حواستون باشه که به کار بردن steganogaphy بـرای ناشناسـی ممکنـه گـاهی بـد باشه. اینطوری حساس میشن روتون که لابد چیزی داری برای پنهون‌کردن کـه steganogaphy بـه کـار بردی![۸۳]

More about Steganography:

+ Steganography[84] (Wikipedia)

+ "Exploring Steganography: Seeing the Unseen"[85] (About how to steganogaphy an image)[86]

+ Hide secret messages into a spammic message[87]

## تعریف تابع:

ببینید تابع مثل یک دستگاهه که شما یک مقدار ورودی بهش میدی، این دستگاه یکسری کار روی این داده انجام میده و بعد بهتون یک مقداری رو پس میده. مثلا یک دستگاهی داریم که هر عدد رو بهش بدی، دو برابر میکنه و بهتون پس میده. یعنی ۱ بدی، ۲ بهت میده. دو بهش بدی ۴ میده. ۴ بدی ۸ میـــده. ۱۰۰ بدی، ۲۰۰ بهت میده. مثلاً این دستگاه یه نوع تابع هست. درواقع شما به تابع یک مقداری رو بهش میدی و یکسری کار و عملیات روش انجام میده و بعد یک چیزی رو به صورت خروجی بهتون تحویل میـده. بـه این کاری که انجام میده میگن ضابطه تابع یا عملکرد تابع. مثلاً برای تابع دوبرابر کردن، داریم:

$$y = 2x$$

حرف «y» یعنی خروجی. حرف «x» یعنی ورودی. یعنی خروجی «y» ما از دوبرابرکـردن ورودی‌هـا بـه دست میاد. خروجی برابر است با ورودی ضربدر ۲!

82 e.g. https://www.jjtc.com/stegdoc/sec313.html

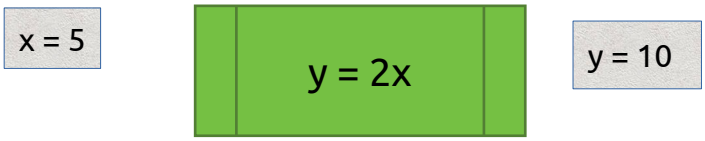
۸۳ درباره روش‌های یافتن steganogaphy با deep learning تحقیق کنین.

84 https://en.wikipedia.org/wiki/Steganography

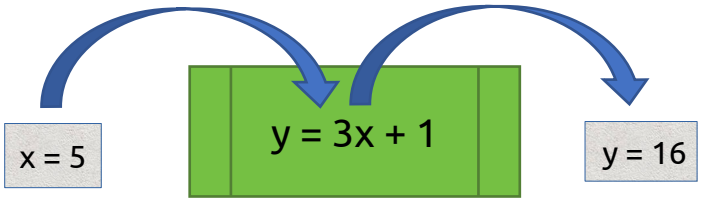
85 https://www.jjtc.com/pub/r2026a.htm

۸۶ یکم قدیمیه ولی دید خوبی میده که اوایل دوران اینترنت از چه تکنیکایی استفاده میشه که توی عکس داده مخفی کنن

87 https://www.spammimic.com/




حالا مثلا یک تابع دیگه «y = 3x + 1» یعنی هرچی بهش بدی، سه برابر میکنه، بعـدش یکی بهش اضافه یکنه و بهتون تحویل میده. یعنی ۱ بدی، ۴ بهت میده. ۲بدی، ۷ میده. ۵ بدی، ۱۶ میده.




- خب این برای رمزنگاری به چه درد میخوره؟

+ درواقع یادتونه گفتیم که رمزنگـاری یـه الگـوریتم ریاضـیه؟ درواقـع بیـایم بهـتر بگیم. رمزنگـاری یـه تابع‌طوره.

مثلاً بیایم بیایم دستی یه چیزیو رمز کنیم.

می‌تونین رمزهاتونو قبل ذخیره، دستی رمزنگاری کنین که کار خواننده سخت‌تر (ولی نه نـاممکن!) بشه. مثلا به جای نوشتن یه عدد روی کاغذ، از تابعی استفاده کنین که عدد رو رمز کنین و یک عدد دیگـه رو روی کاغذ بنویسین:

Y = 2x / y = 2x + 1

7x - 5 / 3x + 4

مثلا به جای هر عدد، دو برابرش کنین و منهای یک کنین و بنویسین که اگر یکی به دست نوشته دست یافت، نتونه اکانتتون رو بـاز کنـه (این دیگـه بـر میگـرده بـه دانش ریاضـی شـما مثلا یکی میگـه تـوان دو می‌رسونم منهای یک می‌کنم. دو برابر می‌کنم، سه تا هم اضاف می‌کنم. دیگه بستگی به خودتون داره) یا به جای هر حرف انگلیسی، یک حرف قبلش رو بنویسین رو کاغذ.

مثلاً رمزتون اینه:

kAmZ)!rf**3**V+qzQ**2**iVQfPVcXaE**4**

با استفاده از تابع میشه اینطوری نوشتش:

kAmZ)!rf**5**V+qzQ**2**iVQfPVcXaE**8**

چه تابعی استفاده کردم؟ تابع 3x – 4. اگر مجبور شدم رمزم رو روی کاغذ بنویسم یا برای کسی بفرستم، برای اینکه اگر کاغذ دست یه فرد بد افتاد، واردشدن به اکانت براش سخت‌تر شه، عددا رو دستی رمز کردم. یعنی دومی رو روی کاغذ می‌نویسم.

حالا خودم می‌دونم که از 3x - 4 استفاده کرده بودم، پس:

$5 = 3x - 4 \rightarrow 3x = 9 \rightarrow x = 3$

راه‌های مختلفی هست که یکم متن رمز و پنهان یا عوض کنی که کسی متوجه نشه. البته این روشا درسته کار آدمِ بد رو سخت‌تر میکنه ولی نباید بگیم که غیرممکن میکنه! (چون ممکنه اونم بتونه حدس بزنه چه تابعی به کار بردیم!)

یه راه دیگه. مثلا رمزتون اینه:

aWf8$dG}H5b#jfRgT>st3ch:tDa{7gS

شما میتونی اونو اینطور بنویسی:

aWf8$dG}H5b#jfR2gT>st3ch:tD@a{7gS

و میدونی که هیچوقت تو رمزهات، عدد ۲ و علامت @ رو استفاده نمی‌کنی. پس موقع وارد کردن، اونارو داخل رمزت نمیزنی ولی کسی که به رمز دسترسی پیدا کنه، نمیدونه که باید اینارو حذف میکرد و این حروف رو الکی نوشتیم و جزء رمز نیست. پس نمی‌تونه وارد شه. یا مثلا:

aWf8$dG}H5b#jfRgT>st3ch:tDa{7gS

رو اینطور بنویسین:

sv#1af8w(s2Ca:seaWf8$dG}H5b#jfRgT>st3ch:tDa{7gSE%dTr^axj7p-fZf

و خودتون از قبل حواستون هست رمزتون بین e و E هست و کسی که متن رو بخونه خب مطمئنا نمیفهمه رمزتون چی بوده.

همه این تکنیکا با هم جمع شن، باعث میشه اگر هم رمزتون به دست کسی افتاد، نفهمه (یا حداقل کارشو برای فهمیدن سخت کنیم).

تمرین! من پسوردمو با چند ترکیب مختلف رمز کردم. خب حالا پسوردمو از دل متن زیر پیدا کنین:

s$FPc+4cvOavqVatW$9}sN-!aZvgWd?x3+e&fT{SF%a:sXBdW:q#cfG5vXR

## پاسخ:

من با خودم قرارداد کردم که پسوردم بین q و X هست. یعنی فعلاً پسوردم خلاصه شده به قسمت بنفش رنگ:

s$FPc+4cvOavqVatW$9}sN-!aZvgWd?x3+e&fT{SF%a:sXBdW:q#cfG5vXR

همچنین من توی قسمت بنفش‌رنگ، تمام a هارو الکی گذاشتم و با خودم قرارداد کردم که این a ها الکین و توی پسورد به کارش نبردم.

s$FPc+4cvOavqVatW$9}sN-!aZvgWd?x3+e&fT{SF%a:sXBdW:q#cfG5vXR

پس درواقع رمزم فعلاً اینه:

VtW$9}sN-!ZvgWd?x3e&fT{SF%:s

اما باز من به همینم راضی نشدم. بلکه گفتم اعداد هم با یه تابع به دستش اوردم. درواقع اعدادی که وجود داره توی پسورد رمزشده، از تابع y = 2x + 3 به وجود اومده. یعنی چی بوده که دوبرابر شده و بعد بعلاوه ۳ شده و ۹ رو ساخته؟

y = 2x + 3 => 9 = 2x + 3 => 6 = 2x => x = 3

توضیح قسمت سبز: من دوطرف تساوی رو از ۳ کم کردم. (وقتی یه کاری رو با دوطرف تساوی انجام بدیم، تغییری رخ نمیده. پس 2x = 6 به دست اومد.

توضیح قسمت نارنجی: حالا دوطرف رو تقسیم بر ۲ کردم که x ما، ۳ به دست اومد.

عدد اولی ۳ بوده. برای عدد دومی چی؟ حسابش کنیم:

3 = 2x + 3 => 0 = 2x => x = 0

پس درواقع رمزاصلی من این بوده:

VtW$**3**}sN-!ZvgWd?x**0**e&fT{SF%:s

دیدین؟ خیلی بهتر شد! یعنی اگر یه روز مجبور بودم پسوردمو روی یه کاغذی بنویسم، حداقل رمزشدشو می‌نویسم که اگر هم کاغذ به دست فرد بدی افتاد، نتونه وارد اکانتم شه.

چند نکته هم نیازه اشاره کنم:

رمز و ایمیل و یوزرنیم و این‌ها رو کنار هم نگذارین. مطمئناً برای یه آدم بد خیلی سادس که شما ایمیل و پسورد و سایت ثبت‌نامی رو توی یه صفحه نوشته باشین! خب لقمه آمادس. مثلاً اینطور ننویسین:

StackOverflow:

random_name@gmail.com

VtW$**3**}sN-!ZvgWd?x**0**e&fT{SF%:s

- نه آخه پسورد رو رمز کردم! چرا باز ایراد داره که پسورد رو کنار ایمیل نباید بنویسم؟!

+ خب رمز کرده باشی! قرار نیست اون پسورد رمزشده، شکست‌ناپذیر باشه که! با یکم آزمون و خطا میشه اون رو احتمالاً شکست. ترجیحاً سعی کنین کنار هم نباشن!

*نکته:* برای نکات نگه‌داری رمز روی کاغذ به مطلب «*انتخاب و نگه‌داری درست رمز عبور*» مراجعه کنین.

اما همونطور که شاید فکر کردین، میشه راحت رمز داخل این پیامارو با یکم فکر کردن و تحقیق و بررسی روشون پیدا کرد. بالاخره هی فکر میکنی چه‌جوری بوده و بالاخره رمزی که توش بوده رو پیدا میکنی. تازه نیاز به فکر شما هم نیست. کامپیوترها کار رو براتون ساده می‌کنن. اونا تست رو انجام میدن. اما من صرفاً خواستم شما رو با اینکه رمزنگاری و پنهان‌نگاری چیه، آشنا کنم. درواقع کاریه که جز کسایی که نحوه رمزگشایی پیام رو بلد هستن، کسی نتونه بفهمه شما مقصودتون چی بوده.

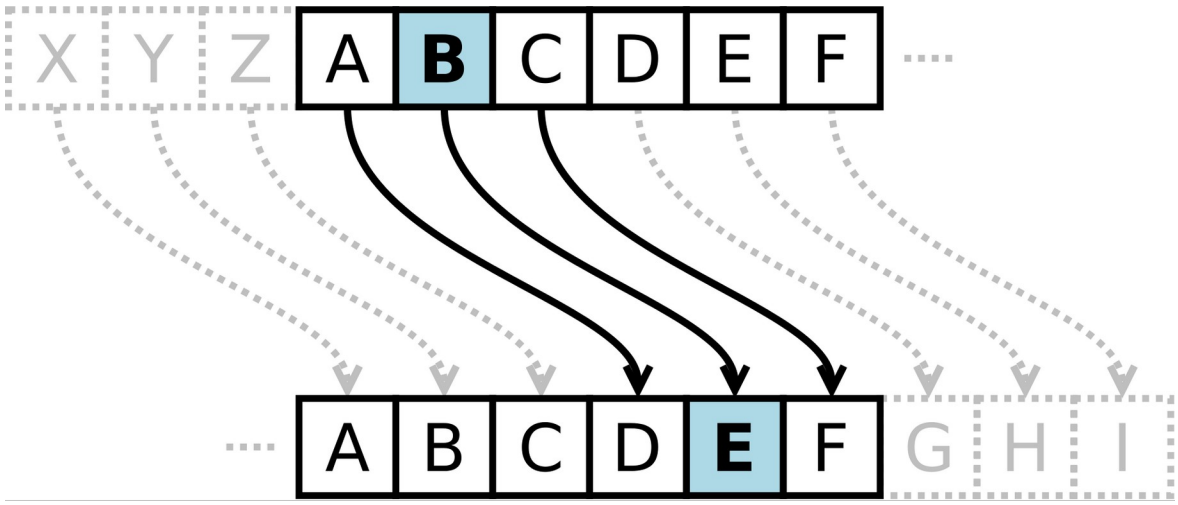
درواقع پایه اصلی رمزنگاری اینه که من یه رمزیو به صورت ساده و صَرف زمان کم تبدیل به رمز کنم ولی برگردوندنش به حالت قبل، اگر ندونم چجور باید برش گردونم، خیلی خیلی زمان بیشتری ببره. یعنی:

انجام، سریع؛ برگشت در صورت عدم دانش، سخت! (در صورت داشتن دانش، ساده)

رمزنگاری از خیلی قبلنا استفاده میشده. مثلاً یه فرمانده می‌خواسته تو جنگ یه پیامی رو به یه فرمانده دیگه بفرسته. اون موقع پیام رو رمزی مینوشته که اگر یکوقت اون پیک و سربازی که میخواست پیام رو ببره اسیر و یا کشته شد، کسی نتونه با خوندن نامه بفهمه اون فرمانده چی نوشته.
مثلاً در قدیم و یونان باستان، یه نوع رمزنگاری داشتیم که الآن به نام «رمزنگاری ژولیوس سزار» معروفه.

## رمزنگاری ژولیوس سزار

یادتونه اوایل من اومدم هر حرف فارسی رو با حرف بعدیش جایگزین کردم و یه متنی رو رمز کردم؟
این روش در یونان باستان و در زمان ژولیوس سزار استفاده میشد.
درواقع هر حرف با سه حرف بعدیش جایگزین میشه.
بیایم با عکس بفهمیمش:




درواقع تبدیل به رمزش سادست. ولی کسی که ندونه چجور باید رمزگشاییش کنه، باید حالت‌های مختلف رو شانسی امتحان کنه. یه بار از آخر به اول بخونه. یه بار هر حرف رو با حرف بعدی توی حروف الفبا جایگزین کنه، یه بار فلان کار کنه و...
درواقع هی باید حالت‌های مختلف رو تست کنه تا بالاخره یکیش جواب بده. پس یعنی *انجام ساده، ولی رمزگشایی در صورت ندانستن روش، بسیار سخت. (ولی نه ناممکن!)*

## ایستگاه اعداد

یکم بریم مثالای جذاب‌تر رو با هم ببینیم. فایل صوتی زیر رو گوش بدین:

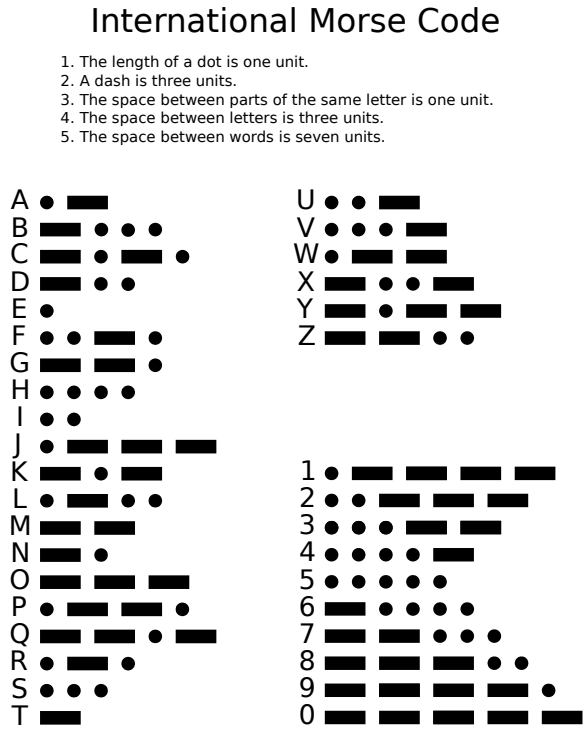
https://www.numbers-stations.com/english/e01-ready-ready/

گوش کردین؟ عجیب بود نه؟ این عددا از یه سری ایستگاه رادیویی پخش میشده. بریم یکی دیگه رو هم گوش بدیم:

https://www.numbers-stations.com/morse/m01/

این عجیب‌تر بود نه؟

این رو به وسیله کد مورس منتقل کرده بودن. کد مورس چیه؟ درواقع یه کدیه که هر حرفی، به یه سری نقطه یا خط متصل میشه:




یعنی اگر می‌خوایم بنویسیم «Hello» باید بگیم:

Hello = “.... . ._.. ._.. _ _ _”

اگر صدا باشه مثل چیزی که شنیدین، صدای کوتاه نشون‌دهنده «نقطه» و صدای بلند نشون‌دهنده «خط فاصله» هست.

یا مثلاً فرض کنیم بخوایم با نور مورس کد انتقال بدیم. نوری که به صورت فلش و یکلحظه‌ای هست نقطه و اونی که به صورت نور طولانی هست، خط فاصله.

مثلاً توی جنگ ویتنام زمانی که ویتنام اسیر گرفته بود، اسیرا رو شکنجه میکرد. ویتنام شمالی اومد یه مصاحبه ترتیب داد که یه خبرنگار ژاپنی بیاد مصاحبه کنه از اسیر که بگه نه همه چی خوبه و ما با زندانیا خوب برخورد می‌کنیم و مشکلی نیست و همه چیز خوبه و ما خیلی خوبیم. اما خب مطمئناً این اسیر

نمی‌تونست جلوی دوربین بگه دارن مارو شکنجه می‌کنن. اومد یه تکنیک زد. اومد بـا پلک‌زدن کـد مـورس منتقل کرد. پلک طولانی نشانه «خط فاصله» و پلک سریع نشان دهنده «نقطه». این فرد اومد پیام «T-o-t-u-r-e» به معنای شکنجه‌کردن رو منتشر کرد به دنیا.[۸۸]

برگردیم به اون مسأله اصلی. این عددا و حروف چی هستن که از یه ایستگاه رادیویی پخش میشده؟

مثلاً این پخش شده:

A – F – H – H - P – X

34

اینا هم نوعی رمزنگاری هست. مثلاً یه کشوری تـوی یـه کشـور دیگـه جاسـوس داره. مطمئنـاً نمیتونـه پیامش رو جوری بفرسته که کسی نفهمه. مثلاً اگر به صورت رمزنگـاری بفرسـته، درسـته دولت اون کشـور نمیفهمه این رمز چیه، اما وقتی ببینه مثلاً هر روز ساعت ۹ صبح، به فلانی از فلان طریق یـه داده رمزشـده ارسال میشه. خب مطمئناً وزارت اطلاعات اون کشور، شک میکنـه کـه چـرا فلانی هی داره در فلان سـاعت خاص، پیام رمزشده دریافت میکنه؟ برای همین میره طرف رو دستگیر میکنه. بـرای همین یـه ایـده‌ای بـه وجود اومد به نام ایستگاه اعداد. مثلاً کشور جاسوس، میاد به جاسوسش یه دفترچه میده و میگه رمزی کـه میگیم رو یادداشت کن و مطابق دفترچه پیداش کن و دستورت رو بگیر. مثلاً اعداد، صفحه رو نشون میـدن که یعنی مثلاً ۴ ۳ یعنی صفحه ۳۴ رو بگرد و مثلاً اون حروف رندوم هم معنی داره.

و اون حروف رو با جدول تطابق میده و می‌بینه که چه پیامی می‌خواستن بهش بدن.

مثلاً اینطوریه:

| Character | Meaning |
|---|---|
| A | You should |
| P | We need you to go to |
| H | Now |
| F | Run |
| X | Boss room |

یه یه جور دیگه ممکنه باشه!

مثلاً نگاه کنین:[۸۹]

| Character | Meaning[۹۰] |
|---|---|

۸۸ ویدیوش:
https://www.military.com/video/operations-and-strategy/vietnam-war/pow-blinks-torture-in-morse-code/1381254901001

89 Substitution cipher

۹۰ درواقع به اینا میگن کلید. یعنی کلیدی که بتونه اون رمز ما رو برامون نمایان کنه.

| | |
|---|---|
| R | A |
| O | P |
| N | H |
| U | F |
| W | X |

حالا پیام رو تطبیق بدیم:

A – F – H – H - P – X = Run, now

فرار کن، همین حالا!

## More:

+ رادیوگیک شماره ۲۴[۹۱]
+ رادیوگیک شماره ۳۹ از جادی[۹۲]
+ معمای ویدیوهای اسرارآمیز در یوتیوب[۹۳]

## چند نکته:

+ به هیچ‌وجه نباید یه پد، دوبار استفاده شه!
+ نباید حروف، زبون خاصی باشن. اینطوری مثلاً اگر زبان فارسی پخش شه، معلوم میشه یه ایرانی جاسوسه یا حداقل طرف یه آشنایی با زبون فارسی داره.
– خب بفهمن ایرانی بوده مگه چی میشه؟
+ دایره شناسایی جاسوس و پخش‌کننده رادیو کمتر و کمتر میشه. نباید تمایز ایجاد شه. اینطوری سیستم امنیتی روی افراد ایرانی یا اونایی که زبونشون فارسیه، تمرکز بیشتری پیدا می‌کنه.

یا مثلاً فرض کنیم طرف ایرانی نیست ولی فارسی داره بهش مخابره میشه، طرف مثلاً یه حرف فارسی رو یادش میره که چه‌جوری هست، میره سرچ می‌کنه حروف الفبای فارسی. حالا با بررسی لاگ سرچ افرادی که درباره حروف الفبا سرچ کردن، میشه دایره شناسایی رو تنگ‌تر کرد. (می‌بینن هر وقت رادیو پخش میشه، یکم بعدش فلانی داره حروف الفبای فارسی رو سرچ می‌کنه. خب مشخص میشه!)

+ فرض کنیم من می‌خوام ایستگاه رادیویی راه بندازم؛ مطمئناً من نمی‌خوام با صدای خودم اون رو پخش کنم. پس چیکار می‌کنم؟ میرم توی اینترنت سرچ می‌کنم: «تبدیل متن به صوت»

خب بعداً می‌تونن برن لاگ افرادی که اینو سرچ کردن پیدا کنن و من متمایزتر شم یا بهتر بگیم، مثلاً می‌فهمن که این صداها رو یکی از سایت فلان گرفته، حالا میرن بالای سر اون وبسایت ازش می‌خوان آیپی

91 https://jadi.net/2013/04/radiogeek-24-numbers-station/
92 https://jadi.net/2014/05/radio-geek-39-narenji-bleed/
93 https://www.bbc.com/persian/science/2014/05/140502_an_youtube_mystery_webdriver_torso

و مشخصات دستگاه کسایی که اومدن توی وبسایتت یا کسی که این متن خاص رو خواسته تبدیل کنی به صوت رو بهمون بده. اینطوری عملاً لو میرم.

گروهی از افرادی که می‌خواستن ناشناس بمونن ولی شناسایی شدن، در یه سری اتفاقاًت خاص، حواسشون نبوده، درباره اون موضوع با آیپی اصلیشون سرچ کردن و یا مطلب نوشتن. این دقیقاً مقدمه شناساییشون بوده.

حالا با این اشتباهات توی قسمت «Dangerous behaviors lead to de-anonymizing» بیشتر آشنا میشیم ولی حتماً یادتون باشه که بعداً دوباره به اینجا برگردین و یه بار دیگه بخونینش که دقیق متوجه شین چرا باعث شناسایی من میشه!

خب در ظاهر تنها کسی که میتونه رمز رو پیدا کنه، کسیه که دفترچه رمزگشا رو پیدا کنه. چون فقط اون فرده که میدونه هر حرف به چه چیزی تصویر شده. اما خب نه! میشه شکوندش!

بیایم یکم با ریاضیات و جایگشت سال دهمتون بازی کنیم.

کاری که میکردیم در این رمزنگاری این بود که هر حرف رو به یه حرف دیگه متصل میکردیم و به جاش حرفی که بهش وصل کرده بودیم رو قرار میدادیم.

میدونیم تعداد حروف انگلیسی ۲۶ تاست.

فرض کنین من از حرف A شروع می‌کنم و می‌خوام اونو به یه حرف دیگه وصل کنم. چندتا انتخاب دارم برای وصل کردن A به یه حرف دیگه؟ ۲۶ حالت. (با فلش و پاور پوینت نشون بده). مثلاً A رو به H وصل میکنم. حالا حرف B، برای این چند حالت دارم؟ یه حرف برای وصل کردن حرف A هدر رفت. یعنی الآن من میتونم B رو به هر حرف دیگه جز H که مصرف شد وصل کنم. یعنی ۲۶ منهای یکی که H بود. پس برای وصل کردن B چندتا انتخاب دارم؟ ۲۵ تا. مثلاً B رو به D وصل میکنم. حالا حرف C. احتمالاً حدس زدین که حرف C رو میشه به تمام حرفا جز H و حرف D وصل کنم. یعنی ۲۴ تا انتخاب دارم. همینطور ادامه بدم.

$$26 \times 25 \times 24 \times \ldots \times 3 \times 2 \times 1 = 26! = 2 \wedge 88$$

در رمزنگاری، یه چیز که ۲ به توان ۹۰ حالت داشته باشه عملاً شکستنش مقداری سخته! (طبق اعلام سازمان استاندارد، برای درامان بودن از دست کامپیوترها تا سال ۲۰۳۰، حداقل باید ۲ به توان ۱۱۲ حالت باشه)[۹۴] پس چجور میگیم این شکسته میشه؟ یکم فکر کنین ببینین چجور متوجه شیم این A معنای H میده مثلا؟

ببینین اگر تکرار حروف الفبای انگلیسی در کلمات یکسان بود، درسته قابل شکستن نبود. اما ما می‌دونیم که مثلاً حرف z خیلی کمتر استفاده میشه نسبت به حرف e.

جدول رو ببینین. حرف تعداد تکرار حرف e تقریباً ۱۲ درصده. بعدش t بعدش A. و در آخر z.

94 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf

| حرف | تکرار |
|---|---|
| E | % 12 |
| T | % 9 |
| A | % 8 |
| O | % 7.6 |
| I | % 7.3 |
| . | . |
| . | . |
| X | % 0.17 |
| Q | % 0.11 |
| J | % 0.10 |
| Z | % 0.07 |

پس میشه توی متن بلندی که رمز شده گشت و اون کلمه‌ای که بیشتر از همه تکرار شده رو فرض کنیم e هست. کمترین تعداد z. همینطور آزمون خطا پیش بریم تا کم‌کم رمز رو پیدا کنیم. تازه همینم نیست! مثلاً میدونیم که تکرار دو حرف در انگلیسی چجوریه.

| دو حرف پشت هم | تکرار |
|---|---|
| th | 0.033 |
| he | 0.030 |
| an | 0.018 |
| in | 0.018 |
| er | 0.017 |
| ... | ... |
| ... | ... |
| ew | 0.005 |
| ra | 0.005 |
| ri | 0.005 |
| sh | 0.005 |

حالا میشه توی متن این هم دخیل داد و عملاً با در کنار هم قرار گرفتن اینا، میشه رمز رو پیدا کرد!

- آیا راه دیگه‌ای هم هست؟

+ مثلاً اگر دو کشور با هم جنگن. مثلاً انگلیسی‌ها شایعه پخش می‌کنن که کمبود آب داریم و بعد مثلاً آلمانی‌ها فکر می‌کنن واقعاً کمبود آبه و پیام کمبود آب رو رمز می‌کردن و می‌فرستادن و از اونور انگلیسی‌ها میومدن این رو مینوشتن و هی شایعه پخش می‌کردن تا هی رمز رو با شایعه تطابق بدن و کامل متوجه شن هر حرف معادل کدوم حرفه و عملاً دیگه هر پیامی رو آلمانی‌ها بفرستن انگلیسی‌ها میفهمن. وقتی شما بدونین هر حرف معادل کدومه، مطمئناً می‌فهمین پیام رمز شده چی بوده. عملاً انگار دفترچه رو دارین! از این تکنیکا هست که تلاش میشه رمز رو بشکنن.[۹۵]

خلاصه کلی راه هست که بشه با کمک همدیگه رمز رو شکوند.

جالب بود نه؟

عملاً رمزنگاری سراسر پُر از ریاضیات و احتمالات و جایگشت و اینهاست. برای همین میگن کسی رمزی که خودش ساخته رو فکر نکنه این دیگه چون خودم ساختمش و خودم میدونم چیکار کردم و بقیه نمیدونن، قابل شکستن نیست. درسته فقط خودت میدونی. اما نشون دادیم که رمز جایگشتی یعنی رمزی که به جای هر حرف، حرف دیگه‌ای بگذاری، عملاً با ریاضیات قابل شکستنه. پس یه اصل رو توی رمزنگاری متوجه می‌شیم:

الگوریتم مورد استفاده در رمزنگاری باید مشخص باشه، اما بر اساس ریاضیات و علم ریاضی و برنامه‌نویسی، قابل شکستن نباشه. تقریباً تمام الگوریتم‌های استاندارد رمزنگاری حال حاضر که در کامپیوتر استفاده میشه، حاصل سالها تلاش ریاضی‌دان‌ها بوده و بر اساس ریاضیات، قابل شکستن نیست!

# Rotor machines

- ## Hebern

یکی از مثال‌های substituting cipher، قرن ۱۹، rotor machine ها مثل Hebern بودن:

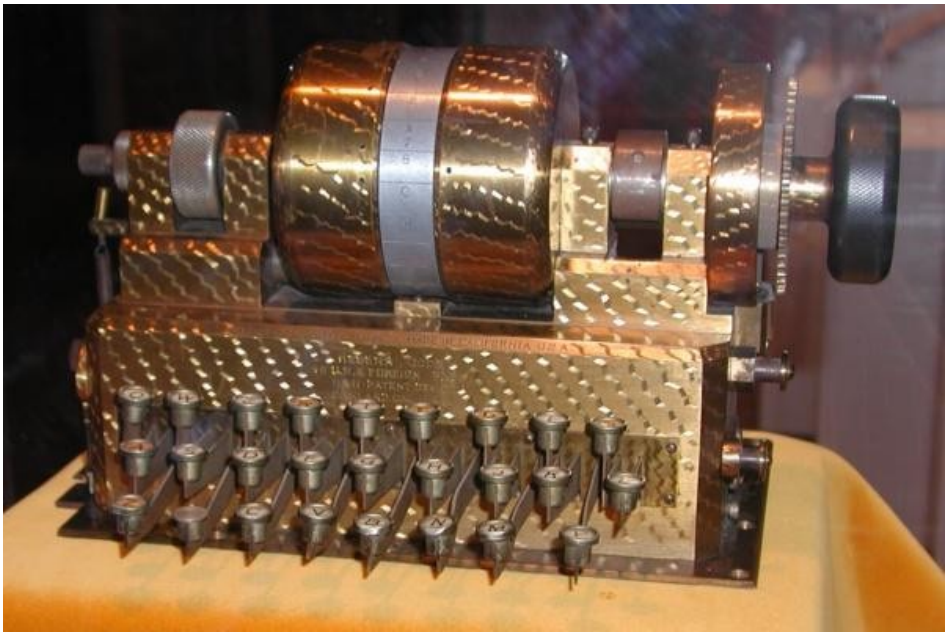
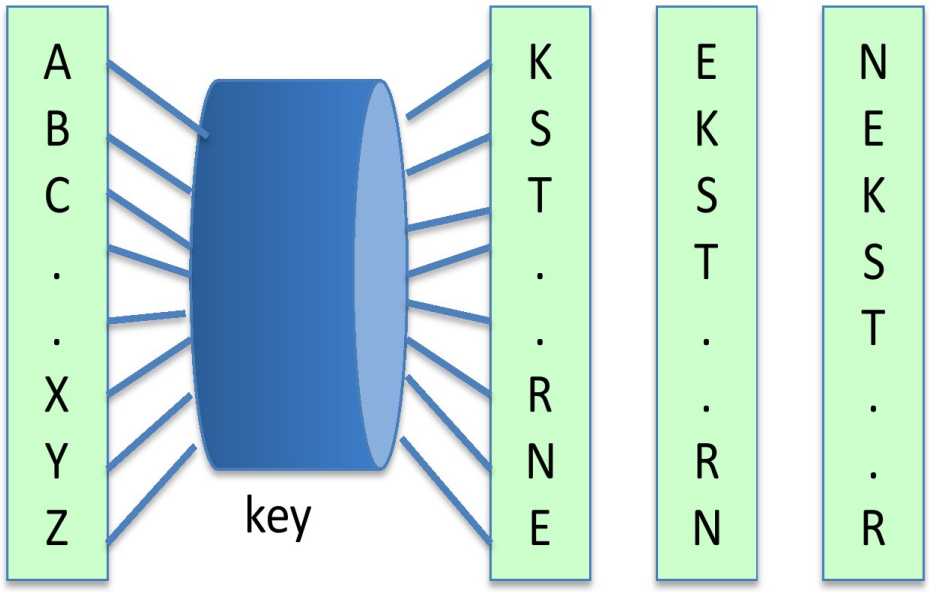


image: en:User:Matt Crypto, Hebern1, CC BY-SA 3.0

۹۵ این ایده رو از جادی یاد گرفتم.




این عکس از اسلایدهای دوره رمزنگاری دانشگاه استنفورد آورده شده.[۹۶] (fair use)

درواقع بعد زدن هر دکمه کیبرد، آخری میرفت اول. اینطوری substituting table جدیدی به وجود میومد. مثلاً اگر C رو سه بار فشار بدیم، به ترتیب تبدیل به:

1) C => T
2) C => S
3) C => K

درواقع اون initial value ها secret key بودن که بعد هر کلمه یه بار شیفت می‌خوردن.

- Enigma

https://hackaday.com/2017/08/22/the-enigma-enigma-how-the-enigma-machine-worked/

- آیا رمزنگاری فقط برای مخفی کردن یه پیام کاربرد داره؟ نه!

96 https://online.stanford.edu/courses/soe-y0001-cryptography-i

+ ببینین پشت هر چیز توی کامپیوتر یه سری متن وجود داره.[۹۷] کامپیوتر اون متن‌های خاص رو می‌بینه و با استفاده از اون، فایل رو میفهمه و براتون نمایش میده.

گنگ بود یکم؟ ایرادی نداره! بریم ببینیم اینکه پشت هر چیز یه سری متن هست یعنی چی؟

من یه عکس دارم. گفتم پشت هر چیز یه سری متن هست درسته؟ خب چجور اون متن رو ببینم؟ فکر کنین یکم!

+ عکس رو به جای اینکه با حالت معمول باز کنم، با یه چیزی که بتونه متن رو بخونه باز میکنم! که ببینم پشت این عکس چه متنی هست! خب چه چیزی متن رو میخوند؟ نرم‌افزارهای Text editor مثل Notepad! خب پس به جای double click روی یه عکس، روش کلیک راست می‌کنیم که گزینه‌های مختلف رو ببینیم. بعدش روی گزینه open with کلیک می‌کنیم. بعد میریم توی لیست برنامه‌ها و یکم میایم پایین و می‌زنیم show more و برنامه notepad رو انتخاب می‌کنیم. حالا عکس ما با نوت‌پد باز شد.

یه سری متن عجیب غریب نمایش داده شده؛ درسته؟!

خب کامپیوتر این متن‌ها رو میبینه و میفهمه باید یه تصویر رو فلان‌طور نشون بده. خب چه ربطی به رمزنگاری داره؟ فکر کنین یکم!

+ خب اگر من بیام این متن رو عوض کنم، عملاً دیگه کامپیوتر نمیتونه بفهمه متن اصلی پشت این تصویر چی بوده! چون متن عوض شده! وقتی هم نفهمه، نمی‌تونه بازش کنه. عملاً می‌گیم فایل قفل شده و باز نمیشه!

مثلاً وقتی شما روی یه فایل word یا documentای رمز می‌گذارین که فقط خودتون با زدن پسورد، بتونین اون رو باز کنین، درواقع دارین متن پشتش رو رمز می‌کنین!

اگر قرار بود فایل رمز نشه، من به جای اینکه با آفیس بازش کنم، میرفتم با فلان‌چیز دیگه بازش میکردم و متن پشتش رو میخوندم! برای همین اینجا میان این متن پشتش رو رمز میکنن که نشه با هیچ نرم‌افزاری خوندش![۹۸]

خب به نظرتون از رمزکردن فایل چه جاهای دیگه‌ای میشه استفاده کرد؟

+ معمولاً هکرا برای اینکه آنتی ویروس، بدافزارشون رو تشخیص نده، اونو داخل فایل zip میگذارن و رمز میکنن! اینطوری دیگه معلوم نیست متن پشت اون zip دقیقاً چیه (یا درواقع چه فایلی توشه!) پس آنتی ویروسا فایلای zip رمزدار رو اسکن نمی‌کنن! (چون فایده‌ای نداره! متن عوض شده!)

---

۹۷ دقیق‌تر اگر بخوایم بگیم، پشت هر چیز توی کامپیوتر، یه سری بیت وجود داره. شما اگر با نوت‌پد بازش کنی، کامپیوتر بنا بر استانداردی که براش تعریف شده، میگه خب بهم گفتی متنی بیتا رو بخونم؟ خب باید سعی کنم ازش متن استخراج کنم.
یا اگر به صورت عکسی بازش کنی، میگه خب باید اینقدر اینقدر بیت کنار هم جدا کنم و پیکسلا رو نشون بدم.
خلاصه هر جور بازش کنی، عینک اونجوری رو به چشمم می‌زنه و میگه خب بر اساس استانداردی که می‌خوای، سعی می‌کنم بیتا رو بخونم و بهت نمایشش بدم.

۹۸ پیشرفته: البته اینکه میگم با هیچ نرم‌افزار، منظورم اینه که اگر اون رمزنگاری درست باشه. ممکنه صرفاً یه پسورد باشه فقط هم اون پسورد برای وقتی باشه که بخوایم با اون نرم‌افزار بازش کنیم! یعنی عملاً متن پشتش رمزنشده باشه! که این البته اشکال اون برنامه رمزکننده هست که عملاً انگار هیچ کاری نکرده. زحمت کشیدی که فقط وقتی با نرم‌افزار خودت میخوایم باز کنیم رمز نیاز داری و متن پشتش رو رمز نکردی!
اکثر اوقات این رمز AES (Advanced Encryption Standard) هست که اگر درست پیاده‌سازی شده باشه، شکسته نمیشه! یا بهتره بگم چندین برابر عمر جهان هستی نیازه که یه سوپر کامپیوتر بتونه اونو بشکنه!
- اگر درست پیاده‌سازی شه یعنی چی؟
+بعدا دربارش صحبت می‌کنیم.

خب آیا رمزکردن فایل، میشه ازش استفاده مخرب هم کرد؟ یکم فکر کنین؟

این دقیقاً تکنیکی هست که باج‌افزارا میزنن. باج‌افزار درواقع یه بدافزار هست که میاد توی کامپیوتر و متن پشت فایل‌هارو عوض میکنه. درواقع اونهارو رمز میکنه و درواقع دیگه فایل‌های شما باز نمیشن! حالا بهتون میگه ۳۰۰ دلار بهم بده تا رمز بازکردن فایل‌هاتو بهت بدم!

باج‌افزارا واقعاً یکی از بدترین و بی‌انسانیت‌ترین کاراست. چرا؟ مثلاً شما فرض کنین که دارین روی پروژه دانشگاهتون کار می‌کنین. بعد یه روز از خواب بیدار میشین و میبین کل فایلاتون رمز شده و باز نمیشه. عملاً ممکنه اون درس رو بیوفتین! یا مثلاً توی قضیه باج‌افزارا پیش اومده که مثلاً یه نفر تو بیماستان بوده و کامپیوترهای بیمارستان به باج‌افزار آلوده شدن و پزشکا نتونستن عملش کنن و فوت شده.

شاید اسم باج‌افزار Wannacry که خیلی تو دنیا سر و صدا کرد رو یادتون باشه. حدود ۳۰۰ هزار کامپیوتر در ۱۵۰ کشور آلوده شدن.[۹۹] خلاصه بله! بیمارستان‌ها، بانک‌ها، شرکتا، خیلی جاها از کار افتادن. به خاطر چی؟ به خاطر اینکه تمام فایلاشون رمز شده بود!

## بیشتر:

+ یه پازل جالب و نظرات جالب زیرش برای تلاش و نحوه فکر برای پیداکردن رمز یه نوشته عجیب.[۱۰۰]

خب خب. تا اینجا کلی درباره چیزای مختلف صحبت کردیم. کم‌کم بریم سمت عمق رمزنگاری.

99 https://tribune.com.pk/story/1423609/shadow-brokers-threaten-release-windows-10-hacking-tools/

100 https://www.schneier.com/blog/archives/2006/01/handwritten_rea.html

## بریم یکم سمت رمزنگاری‌های مدرن‌تر:

ما به طور کلی دو نوع رمزنگاری در کامپیوتر داریم:

۱- رمزنگاری با کلید متقارن (symmetric-key)

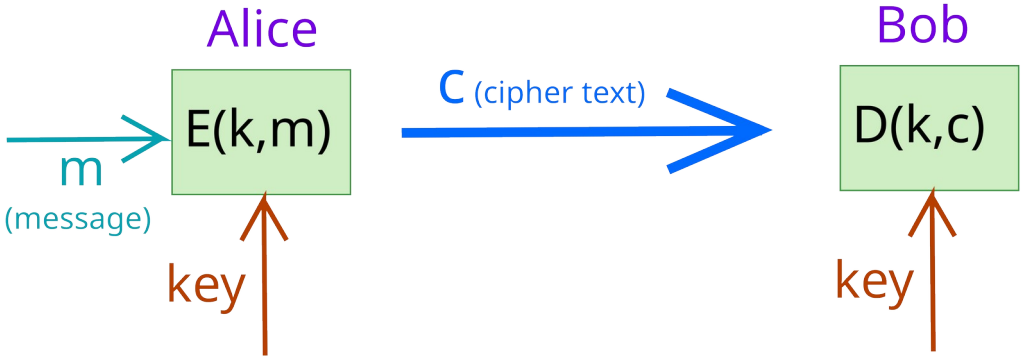
۲- رمزنگاری با کلید نامتقارن (asymmetric-key)

## رمزنگاری symmetric:

درواقع فرض کنین یه قفل دارین. خب قفل با همون کلیدی که قفل بشه، بـا همـونم بـاز میشـه دیگـه. درسته؟

توی رمزنگاری symmetric هم همینه. یه کلید هم می‌تونه باز کنه هم می‌تونه پیام رو رمز کنه.

درواقع مثلاً میگیم:




m: مخفف message به معنای پیام

k: مخفف key به معنای کلید رمز

c: متن رمزشده (cipher-text)

E: تابع رمزکردن (Encryption)

D: تابع رمزگشایی‌کردن (Decryption)

درواقع آلیس یه تابع داره که یه کلید و یه پیام رو می‌گیره. پیام رو با تابع E رمز می‌کنه. بعد رمز رو می‌فرسته سمت باب. باب میاد با تابع D که یه متن و یه کلید رو می‌گیره، پیام رو رمزگشایی می‌کنه.

انجام رمز و یا رمزگشایی با کلید، بسیار بسیار سریع. ولی وقتی کلید رو نداریم، باید کلید رو حدس بزنیم. حدس‌زدن کلید خیلی خیلی سخته. مثلاً برای یه کلید ۲۵۶ بیتی، باید ۲ به توان ۲۵۶ حالت رو حدس بزنیم.[۱۰۱]

هر بیت کلید هم زیاد شه، یه کوچولو رمزکردن بیشتر طول می‌کشه ولی شکستنش (از لحاظ تئوری) ۲ برابر سخت‌تره.

سؤال: به نظرتون اگر کلید رمزنگاریمون رو از ۱۲۸ بیت ببریم به ۲۵۶ بیت، از لحاظ تئوری شکستنش چقدر سخت‌تر میشه؟

+ پاسخ:

هر بیت که زیادشه، یه جایگاه اضافه میشه که خودش دو حالت ۰ یا ۱ داره. پی به ازای هر افزایش بیت، شکستن بدون داشتن رمز (با امتحان تمام حالت‌ها)، ۲ برابر سخت‌تر میشه:

$$2^{(256-128)}=2^{128}=340,282,366,920,938,463,463,374,607,431,768,211,456$$

اینهمه سخت‌تر میشه!

حالا بریم یکی از معروف‌ترینا‌شو بررسی کنیم تا متوجه شیم:

## Vigenère cipher[102] (16th century)

ما یه پیام داریم و یه کلید. کلید رو میایم اونقدر تکرار می‌کنیم که هم‌طول پیام بشه. (مثلاً کلید ما اینجا، KEY هست. بعدش میایم پیام رو با کلید جمع می‌کنیم و متن رمزشده رو می‌سازیم. پیام رو با کلید جمع می‌کنیم یعنی چی؟ بیایم روی جدول توضیحش بدیم:

| Text | H | E | L | L | O | H | O | W | A | R | E | Y | O | U | A | R | E | Y | O | U | O | K |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Key | K | E | Y | K | E | Y | K | E | Y | K | E | Y | K | E | Y | K | E | Y | K | E | Y | K |
| Encrypted (Text + Key) | R | I | J | V | S | F | Y | A | Y | B | I | W | Y | Y | Y | B | I | W | Y | Y | M | U |

۱۰۱ هر بیت ۲ حالت داره. ۰ یا ۱. پس مثلاً اگر ۲ بیت داشتیم: دو جایگاه داریم:

- -

برای هر جایگاه ۲ حالت. پس ۲ * ۲ حالت. حالا سه بیت داشتیم، ۲ * ۲ * ۲. یا ۲ به توان ۳. برای ۲۵۶ بیت هم ۲۵۶ جایگاه داریم که هرکدوم ۲ حالت. پس میشه ۲ به توان ۲۵۶.

۱۰۲ به روندی که دیتای ما به دیتای رمزشده تبدیل میشه، رمزنگاری یا «encryption» گفته میشه.
به الگوریتمی که این کار رو انجام میده، میگن «cipher»

یعنی مثلاً H که ۷ حرف جلوتر از A هست (چون هشتیمن حرف الفباست) رو با K جمع می‌کنیم. یعنی از K، تعداد ۷ تا بار میریم که می‌رسیم به R. خب رمز ما ساخته شد.

اگرم از Z جلو زدیم، میریم اول حروف الفبا؛ یعنی یه دونه از Z جلو زدیم، حرف A قرار میگیره. دوتا B سه تا C و همینطور تکرار میشه.

درواقع کلید رو داشته باشیم، هم می‌تونیم رمز رو بسازیم و هم بشکونیم.

## پیاده‌سازی در پایتون

خب کار شما اینه که اول یه متن از کاربر می‌گیرین. بعد یه کلید می‌گیرین. بعدش رمز رو چاپ می‌کنین.

توجه کنین که متن می‌تونه شامل حروف غیر کرکتری هم باشه! اما حروف کرکتریش، همش کرکتر کوچیکن.

حروف غیرکرکتری مثل «؟ !» نیاز به رمزشدن نداره.

برای سادگی کار، در ابتدا فرض کنین که طول کلید هم اندازه طول متن هست. و نیاز به گسترش توسط شما نیست! و حالا بعداً حالتی که کلید کوچکتر از متن هست رو در نظر بگیرین!

*راهنمایی:*

خب باید دونه‌دونه رو کرکترا پیش بریم و کرکتر کلیدی که هست رو باهاش جمع کنیم (مقدار عددیشونو جمع کنیم.)

باید حواسمون باشه که از z نزنه جلو. اگر زد، باید برش گردونیم به عقب. عدد اسکی «a» مقدارش ۹۷ هست. خب کار با عدد اسکی که از ۹۷ (a) شروع میشه تا ۱۲۲ (z) میره ساده‌تره یا اینکه من بیام بگم a مقدارش ۰ هست. b مقدارش a و z مقدارش ۲۵. حالا اگر از ۲۵ جلو زد، بیام از صفر پیش برم؟ قاعدتاً حالت دوم ساده‌تره برای نوشتن. اینطوری خیلی بهتر میشه نوشت. پس سعی می‌کنم اول مقدار عددیشون رو حساب کنم و بعد بیام منهای ۹۷ کنم که از صفر شروع شن:

```
a = 97 → 97 - 97 = 0
b = 98 → 98 - 97 = 1
c = 99 → 99 - 97 = 2
…
z = 122 → 122 - 97 = 25
```

پس حواسم هست بهش.

*تست کیس:*

*input:*

*text:* a

*key:* a

*output:*
a
---
*input:*
*text:* b
*key:* b

*output:*
c
---
*input:*
*text:* hello
*key:* key

*output:*
rijvs
---
*input:*
*text:* z
*key:* z

*output:*
y

تست کیس بیشتر می‌خواین؟

من یه سری تست کیس حساس مثل a و z و z و a رو بهتون دادم که مطمئن شین کدتون این نقاط حساس رو هم پوشش میده. اگر بازم می‌خواین، خودتون با وبسایت زیر تولید کنین:

https://vigenerecipher.com/

پاسخ:

خب تابعی می‌سازم که ازم یه متن می‌گیره و یه کلید. فعلاً هم برای سادگی فرض می‌کنم که طول کلید هم اندازه طول متنه. بعدش باید روی تک‌تک کرکترا حرکت کنم و با کلید جمع کنم. یعنی ایندکس‌های متناظر رو با هم جمع می‌کنم. اگر هم کرکتری نبود، باید هومنطوری ولش کنم و کاریش نکنم.

من میام رمز رو توی یه متغیر به نام encrypted_text می‌گذارم:

```
def vigenere_encrypt(plain_text, key):
    encrypted_text = ''
    for i in range(len(plain_text)):
        if is_alpha(plain_text[i]):

        else:
            encrypted_char = plain_text[i]

        encrypted_text += encrypted_char
```

می‌گم اگر اون کرکتر من (یعنی همون plain_text[i]) به صورت کرکتری بود، یه کار انجام بده. اگر نبود، همون توی رمز قرار می‌گیره. بلاک else میگه که خود کرکتر رو بریز توی encrypted_char. و در نهایت encrypted_char مرحله به مرحله به آخر encrypted_text اضافه میشه.

خب دیدین؟ لزوماً قرار نیست همون اول if رو کامل کنم. فعلاً else رو نوشتم. قسمت بلاک if رو بعداً کامل می‌کنم.

تازه یه تابعی رو نوشتم که هنوز تعریفش نکردم. یعنی دیدم عه نیازه به یه چیز برای فهمیدن اینکه کرکتر حروفی هست یا نه دارم. خب فعلاً اسمشو می‌گذارم is_alpha و if ام رو تکمیل می‌کنم و بعد تکمیل if، تابع is_alpha رو می‌نویسم.

این مواقع می‌تونین از کلمه pass استفاده کنین که به خاطر عدم تکمیل کد، کدایی پایینی به ارور نخورن. کلمه pass هیچ کاری نمی‌کنه. صرفاً می‌گه عبور کن از این خط:

```
def vigenere_encrypt(plain_text, key):
    encrypted_text = ''
    for i in range(len(plain_text)):
        if is_alpha(plain_text[i]):
            pass
        else:
            encrypted_char = plain_text[i]

        encrypted_text += encrypted_char
```

خب میایم قسمت بلاک if رو بنویسیم:

```
def vigenere_encrypt(plain_text, key):
    encrypted_text = ''
    for i in range(len(plain_text)):
        if is_alpha(plain_text[i]):
                    encrypted_char = (ord(plain_text[i]) - ord('a')) +
(ord(key[i]) - ord('a'))
```

```
        else:
            encrypted_char = plain_text[i]

        encrypted_text += encrypted_char
```

قسمت بلاک if خیلی طولانی شد. این تمیز نیست! اصطلاحاً کدتون نباید به صورت افقی scroll بشـــه. یا horizontal scrolling نباید داشته باشه. اینطوری تمیزتره که هی نخـوایم افقی اسـکرول کـنیم! پس اینطوری می‌نویسمش:

```
def vigenere_encrypt(plain_text, key):
    encrypted_text = ''
    for i in range(len(plain_text)):
        if is_alpha(plain_text[i]):
            encrypted_char = (
                    ord(plain_text[i]) - ord('a')) + (ord(key[i]) -
ord('a'))
        else:
            encrypted_char = plain_text[i]

        encrypted_text += encrypted_char
```

درواقع میگیم که بیا ord برای کرکتر plain_text ما حساب کن. منهای ord کرکتر a کن کـه انگـار کرکتر از ۰ شماره‌گذاری شده. بعدش بعلاوه ایندکس متناظرش در key کن. اونم منهـای a کن کـه اونم از صفر انگار نامگذاری شه. پرانتزا رو هم براتون رنگی کردم که بهتر درکش کنین ฅ^•ﻌ•^ฅ

در آخر هم chr رو حساب کردم که یعنی تبدیل به کرکترش کنه.

خب حالا چی نیازه؟

اینکه چک کنیم آیا از ۲۶ زده بیرون یا نه؟ (شماره‌گذاریمون از ۰ تا ۲۵ بود) اگـر زده بیـایم از اول. این کار رو می‌تونیم با منهای ۲۶ انجام بدیم. می‌تونیم هم از باقی‌مونده «٪» کمک بگیریم. یعنی باقی‌موندش به ۲۶ بگیریم. بعدش هم باید مقدار عددی که از ۰ تا ۲۵ هست رو بعلاوه مقـدار عـددی a کـنیم کـه بعـدش بتونیم با chr کرکترش رو بسازیم:

```
def vigenere_encrypt(plain_text, key):
    encrypted_text = ''
    for i in range(len(plain_text)):
        if is_alpha(plain_text[i]):
            encrypted_char = (
                    ord(plain_text[i]) - ord('a')) + (ord(key[i]) -
ord('a'))
            if encrypted_char >= 26:
                encrypted_char = encrypted_char % 26
```

```
            encrypted_char = chr(encrypted_char + ord('a'))
        else:
            encrypted_char = plain_text[i]

        encrypted_text += encrypted_char

    return encrypted_text
```

در آخر هم مقدار encrypt شده رو return کردیم.

خب خیلی هم عالی! حالا که اینو ساختیم، بیایم حالتی رو بسازیم که کلید کوچکتر از متنه.
یعنی فرض کنیم طول کلید ۳ بود. پس:

```
abc     defg   hi
qwe     qwe    qw
```

یعنی درواقع ما باید یه ارتباطی پیدا کنیم که هر ایندکس، کدوم ایندکس کلید رو می‌خواد؟

| Text index | Key index |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 0 |
| 4 | 1 |
| 5 | 2 |
| 6 | 0 |

یعنی درواقع ارتباط اینه:

```
plain_text[i] → key[i % len]
```

یعنی ایندکس i ام، باید با ایندکس i % len تناظر پیدا کنه. اینطوری i همیشه بین ۰ تا ۲ (یعنی همون رنج ایندکس‌های key قرار می‌گیره. پس کد رو درست می‌کنیم:

```
def vigenere_encrypt(plain_text, key):
    encrypted_text = ''
    for i in range(len(plain_text)):
        if is_alpha(plain_text[i]):
            encrypted_char = (
                ord(plain_text[i]) - ord('a')) + (ord(key[i % len(key)])
- ord('a'))
            if encrypted_char >= 26:
                encrypted_char = encrypted_char % 26
            encrypted_char = chr(encrypted_char + ord('a'))
```

```
        else:
            encrypted_char = plain_text[i]

        encrypted_text += encrypted_char

    return encrypted_text
```

خب حالا کد is_alpha رو بنویسیم:

حواستون باشه که تابع is_alpha باید قبل از تابع vigegenere_encrypt باشه. چون توی اینجا داره استفاده میشه. پس قبلش باید تعریف شده باشه.

```
def is_alpha(char):
    if 'a' <= char <= 'z':
        return True
    return False
```

اگر کرکتر بین a تا z بود (حروف کوچیک) ریترن کن True رو.

البته اینطورم می‌تونستیم بنویسیمش:

```
def is_alpha(char):
    return 'a' <= char <= 'z'
```

البته بهتر بود اسم تابع رو می‌گذاشتیم is_lower به معنای اینکه «آیا حروف کوچیکه؟» چون اینطوری بهتر بود. این اسم is_alpha یکم غلط اندازه که آدم فکر می‌کنه حروف بزرگ هم بدیم جواب درست میده. ولی خب چون سوالمون صرفاً حروف کوچیک داشت، اوکیه.

خب حالا کد رو تکمیل کنیم:

```
def is_alpha(char):
    return 'a' <= char <= 'z'


def vigenere_encrypt(plain_text, key):
    encrypted_text = ''
    for i in range(len(plain_text)):
        if is_alpha(plain_text[i]):
            encrypted_char = (
                ord(plain_text[i]) - ord('a')) + (ord(key[i % len(key)])
- ord('a'))
            if encrypted_char >= 26:
                encrypted_char = encrypted_char % 26
```

```
            encrypted_char = chr(encrypted_char + ord('a'))
        else:
            encrypted_char = plain_text[i]

        encrypted_text += encrypted_char

    return encrypted_text


plain_text = input("Enter the text: ")
key = input("Enter the key: ")
print(vigenere_encrypt(plain_text, key))
```

حالا به من بگین که برنامه‌نویس خوب حواسش به چی هست؟

+ حواسش به این هست که طول کلید رو بزرگ‌تر از طول متن ندن! یه if ساده می‌تونه شرط رو چک کنه. از همین الآن سعی کنین ذهنتونو درست پرورش بدین. می‌دونم سخته هی بخواین به حالتی که کاربر چیز غلط میده رو فکر کنین و یا هی فکر کنین که چه چیزایی ممکنه اشتباه شه ولی اینو بگم که ذهن شما الآن مثل یه خمیره. اگر درست شکل بگیره، بعداً هم برنامه‌نویس خوبی میشین ولی اگر بد شکل بگیره و از همین الآن که کدا ساده هستن نتونین فکر کنین که حالتای حساس و اینا چه‌جوری ممکنه رخ بدن، ذهنتون بد شکل می‌گیره و بعداً مشکل خواهید داشت. از همین الآن این مهارت رو تمرین کنین!

## *خب چرا این رمز امن نیست؟ چه روشایی برای کمک به شکستنش هست؟*

خب بیایم با هم فکر کنیم که یاد بگیریم. فرض کنین من متن زیر رو بدم:

“Hi, How are you? I am OK. How about you?”

خب به نظرتون چه کلمه/حرف‌هایی میتونه دردسرساز شه؟

+ ببینین حرف i ممکنه دردسرساز شه.

- چرا؟

+ چون ما چندتا حرف با معنی یک حرفی توی انگلیسی داریم؟ معلومه که اولین چیزی که به ذهنمون میرسه، ضمیر i به معنای «من» هست! خب پس من با نگاه به متن رمزشده، مطمئناً کلید این قسمت رو متوجه میشم که چی بوده که با یه تک حرف که i باشه جمع شده و فلان چیز رو به وجود اورده. پس این قسمت کلید لو رفت. با همین چیز کوچیک یه قسمت لو رفت.

بعد اگر من حالا با یه تکنیکی طول کلید رو متوجه بشم، میتونم بگم خب باید جدول رو به دسته‌های سه‌تایی تقسیم کنم. پس میدونم کدوم حروفا با حرف اول (K) رمز شدن، کدوما با حرف دوم (E) و کدوما با حرف سوم (E) رمز شدن.

حالا خیلی راحت با تکرارهای حروف انگلیسی که باهاش قبلاً آشنا شدیم، میتونم رمز رو بشکنم. هرچی طول پیام بیشتر، کار ما راحت‌تر.

تکرار برای این چیه؟

توی جدول زیر ببینین:

توی سبزا Y سه بار تکرار شده. توی صورتیا، i سه بار تکرار شده و توی نارنجیا، W دوبار تکرار شده. خب من میتونم بگم این i که سه بار تکرار شده و تکرارش هم خیلی زیاد بوده، احتمالاً به خاطر اینه که کلید ما حرف E بوده. چرا؟ چون حرف E توی انگلیسی بیشترین تکرار رو داره و می‌بینیم که احتمالم هم درسته! حالا میگم چی بوده که با E جمع شده و مثلاً i رو ساخته؟ آها فلان چیز بوده! عملاً دارم تک تک اون حروف رمزنشده رو پیدا میکنم و میرسم بهشون. کم کم با این تکنیکا میتونم بشکونمش.

- خب توی سبزا هم Y سه بار تکرار شده بود اما کلیدش K بود!

+ اینجا به خاطر کوتاه بودن متنی هست که من دارم. اما وقتی شما یه نامه یا یه متن بلند داشته باشین، دقت کارتون میره بالاتر و بهتر میتونین احتمال بدین که اینی که تکرار شده احتمالاً یه حرف پرتکرار انگلیسی بوده.

| Text | H | E | L | L | O | H | O | W | A | R | E | Y | O | U | A | R | E | Y | O | U | O | K |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Key | K | E | Y | K | E | Y | K | E | Y | K | E | Y | K | E | Y | K | E | Y | K | E | Y | K |
| Encrypted text | R | I | J | V | S | F | Y | A | Y | B | I | W | Y | Y | Y | B | I | W | Y | Y | M | U |

چطور این رمزنگاری رو قوی کنیم؟ یکم فکر کنین ببینین چجوری؟ ما به خاطر تکرارا شروع به رمزگشایی میکردیم. خب با توجه به این، چیکار میتونیم بکنیم که شکستنش سخت شه؟

- چطوره بیایم در کلید حروف پرتکرار انگلیسی رو استفاده نکنیم. مثل E و T و A و اینها.

+ اوکی شاید یه خورده کار رو سخت‌تر کنه ولی بیایم یکم بهتر فکر کنیم:

- هرچی طول بیشتر باشه، مقدار تکرار کلید خیلی کمتر میشه. یعنی به جای اینکه سه حرفی به کار ببریم، که مجبوریم مثلاً ۱۰۰ بار تکرارش کنیم، اگر ده حرفی به کار ببریم، نیازه فقط ۳۰ بار تکرارش کنیم. پس تکرار کلید کمتر و بررسی تکرارای تولیدی در متن رمزشده سخت‌تر.

- آره به نظرم این روشی که گفته شد هم خوبه. اما بازم تکرار داریم. چطوره کلیدمون رندوم باشه و اندازه طول پیام! اینطوری دیگه تکرار هم نداریم و شکستن خیلی سخت میشه!

+ کاملاً درسته. کلید اگر **رندوم** باشه و **به اندازه طول پیام** باشه که **هیچ‌وقت تکرارش هم نکنیم** و بعد استفاده بریزیمش دور و **هیچ‌وقت دوبار ازش استفاده نکنیم** (**حتی قسمتی ازش** رو) و **message authentication** انجام بدیم، مشکلات قبلی رو نداریم. به این میگن one-time pad. تنها روش رمزنگاری که مطمئنیم امنه.[۱۰۳] (قسمتای بولدشده یا همون خصوصیاتی که گفتم، مهمن! در صورت عدم رعایت حتی یکیشون، رمز امن نخواهد بود!)

۱۰۳ بر طبق ریاضیات و دانشی که ما داریم، فعلاً کسی نتونسته بشکونتشون. یعنی فعلاً حمله‌ای پیدا نشده که منجر به شکستنشون بشه. بله ممکنه فردا یه حمله پیدا شه و بشکنن. ولی احتمالش کمه

## • One-time pad[104]

– عه پس رمزنگاری‌های جدید که توی مثلاً https استفاده میشه چی؟ اونا امن نیستن؟

+ اونا بر طبق دانش و ریاضی که بلدیم، امنن! یعنی اونقدر طول می‌کشه باز شن که زمانش از عمر جهان هستی هم بیشتره.

اما ممکنه ده سال بعد یه حمله پیدا شه که بشکونتش. اما فعلاً بر طبق دانش ریاضیاتی که داریم امنه.

اما one-time pad تنها روشی هست که مطمئنیم امنه.

درواقع فرض کنین ما رمز sjmv رو داریم.

در رمزنگاری استانداردهای الآن، صرفاً یه کلید هست که می‌تونه اون متن مورد قبول رو بسازه. یعنی بقیه کلیدها چیزای عجیب غریب میسازن و متن انگلیسی درست نمی‌سازن.

اما در رمزنگاری one-time pad، هر کلیدی می‌تونه درست باشه! یعنی نمی‌فهمیم کدوم کلید درسته! ممکنه با یه کلید blue در بیاد. ممکنه با یه کلید stop در بیاد. ممکنه با یه کلید hell در بیاد و...

درواقع چون اندازه خود متن، کلید برای رمزگشایی داریم، حالت‌های مختلف بامعنا در میان و هیچ‌وقت نمی‌تونیم بفهمیم کدوم حالت درسته؟ نمی‌تونیم بدونیم متن stop بوده یا blue؟![105]

اما یه سوال! به نظرتون چرا خیلی وقتا این کار رو نمی‌کنیم؟

+ خب من فکر کردم دیدم باید دقیقاً به اندازه متن، کلید تولید کنیم یکم دیدم رو بازتر کردم و گفتم خب رمزنگاری باید بشه ازش جاهای مختلف استفاده کرد. مثلاً من بتونم پیام‌هامو رمز کنم، نمیدونم فایل حاوی متنم رو رمز کنم. خب مثلاً من بخوام یه فایلی شامل ۲۰۰,۰۰۰ کرکتر رو رمز کنم، باید ۲۰۰ هزار کرکتر برای کلید رندوم تولید کنم. ۲۰۰ هزار کرکتر رندوم برای کلید خیلی خیلی سخت و زمان‌بره. چون باید صبر کنم بی‌نظمی و entropy زیاد شه. یادمه که ساخت اعداد و کرکتر رندوم خیلی سخته. عملاً به شدت عملیات کند میشه. یا رندوم بودنش کم میشه و علاوه بر اون، حجم فایل دوبرابر میشه. رمزنگاری علاوه بر امن بودن، باید قابل انجام در دنیای واقعی باشه.

بعدشم این ۲۰۰ هزار کرکتر رندوم رو باید بدم به کسی که میخواد رمز رو باز کنه! حالا جابه‌جایی این ۲۰۰ هزار کرکتر چقدر زمان‌بر و سخته!

فرض کنین بخواین یه SSD یک ترابایتی رو رمز کنین، باید یه SSD جداگانه براش بگذارین تا بتونین رمز کنین! (یا یه هارد. تازه بعد رمزکردن باید هارد رو به روش امنی و نه هر روشی! پاک کرد!)

بعدشم هربار باید کلید جداگانه‌ای ایجاد کنیم و برای هر بار رمز شدن باید این رو تکرار کنیم که هیچ‌وقت یه کلید رو دو بار استفاده نکنیم! هیچوقت!

---

۱۰۴ ایده از رادیوگیک جادی و Bruce Schneier:
https://www.schneier.com/crypto-gram/archives/2002/1015.html#7

105 https://www.schneier.com/crypto-gram/archives/2002/1015.html#7

ارتباط اینترنتی چه‌جوری با one-time pad انجام شه؟ ما در لحظه میلیون‌ها بایت داده ارسال می‌کنیم. پس میلیون‌ها بایت در ثانیه باید کلید بسازیم. این خیلی خیلی سخته! کلیدها بی‌نظمی کافی ندارن و رندوم نمیشن و نزدیک دو برابر اینترنت بیشتری مصرف میشه! و کلی چیز دیگه!

one-time pad در دنیای واقعی و نرم‌افزار معمولاً استفاده‌ای نداره. بیشتر برای چیزهای نظامی و روی کاغذ نوشتن کاربرد داره. مثلاً جاسوس‌ها روی کاغذهایی که به شدت قابل اشتعال‌بودن رمز رو می‌نوشتن و با یکم مالششون به هم، سریع آتش می‌گرفت که از بین برن.

توجه! ۹۹ درصد نرم‌افزارهایی که ادعا می‌کنن رمزنگاری one-time pad دارن، الکین! بیشترشون کلید رو تکرار می‌کنن، کلید رو دوبار استفاده می‌کنن و یا تولیدکننده کلیدرمزنگاریشون، رندوم نیست! بلکه با یه الگوریتم ساخته میشه که راحت میشه با بررسیش، کلید رو پیدا کرد! یا نحوه پیاده‌سازیشون غلطه و ممکنه در مقابل bit-flipping attack مقاوم نباشن! یعنی پیام رو اعتبارسنجی نمی‌کنن که تغییر نکرده باشه یا حتی در مقابل replay attack ها مقاوم نیستن و...

- bit-flipping attack چیه؟

+ یکم دیگه بهش می‌رسیم.

تذکر! نگیم طول کلید ۱۰ گیگابایته و همه فکر می‌کنن رندومه و دوباره استفادش کنم کسی نمی‌فهمه که بخواد حمله کنه. خیر! هیچ کلیدی حتی اگر طولش ۱۰ گیگابایت باشه نباید دوباره استفاده کرد! شاید شما بتونین از دید چشم انسان پنهونش کنین، اما از دید ماشین، نه! وقتی میگیم one-time یعنی one-time!!!

## نتیجه‌ها:

۱- کسی نباید بتونه با استفاده از طول کلید، روش خاصی رو برای رمزگشایی پیدا کنه! دیدین؟ من قرار نیست بیام بگم رمزگشایی باید مستقل از طول کلید باشه. بلکه میام با هم نمونه می‌بینیم که چراییشو بفهمیم. یادتونه روز اول گفتم سعی میکنیم با هم درک کنیم و یاد بگیریم؟ فرق هست بین درک‌کردن و یا حفظی یه چیزی بگن و بگی چشم که طول کلید باید مستقل باشه! باید چرایی رو بفهمین!

- این به چی بر می‌گشت؟

+ به Kerckhoffs's principle.

درواقع در طراحی و دیزاین الگوریتم رمزنگاری، الگوریتم نباید جوری باشه که به صورت خاصی بهش بهش حمله کرد. تنها حمله باید Brute-force باشه. یعنی اگر ۲۵۶ بیت هست، باید ۲ به توان ۲۵۶ بیت امتحان شه.

برای همینم هست که وقتی رمزنگارها یه روش خاصی رو برای رمزگشایی (خارج از Brute-force ساده) پیدا می‌کنن، میگن یه حمله جدید (یه حالت خاص جدید) پیدا کردیم. مثلاً تونستیم با تکنیک

خاصی، صرفاً با امتحان ۲ به توان ۲۰۰ حالت بشکونیمش. خب قاعـدتاً خیلی خیلی بهـتر از brute-force هست که ۲ به توان ۲۵۶ حالت بود.

پس نتیجه گرفتیم که دیزاین و طراحی الگوریتممون نباید جوری باشه که بـه صـورت خاصـی (مثلاً بـا دونستن طول کلید)، بشه بهش حمله کرد.

۲- دونستن رمزنگاری‌های قدیمی‌تری که شکسته شدن، میتونن به درک ما از پایه‌های رمزنگاری کمک کنه و بدونیم چرا شکسته شدن و چه چیزیشون مشکل داشته. پس اینطوری نیست که بگیم بابا چـرا داری تاریخچه رمزنگاری رو میگی! تاریخچـه بـه چـه دردی میخـوره آخـه؟ خب شکسـته شـدن دیگـه! بـه درد نمیخورن! دیدین؟ به درد میخوره! به درد یادگیری و درک بیشتر از رمزنگاری میخوره.

۳- باز هم تکرار می‌کنیم که رمزنگاریی که خودتون اختراع کنین و مورد تست ریاضی‌دان‌ها و رمزنگارها و برنامه‌نویس‌ها و افراد متخصص قرار نگرفته باشه، امن نیست! درسـته از هیچی بهـتره و ممکنـه هم حـتی کسی نفهمه و رمزتون هم جواب بده، اما لزوماً دلیل بر شکست‌ناپذیریش نیست!

۴- بهینگی و سریع بودن رمزنگاری و قابلیت استفاده در دنیای واقعی یکی از چـیزایی هسـت کـه بایـد درنظر بگیریم. (**اگر** one-time pad درست پیاده شه، امنه ولی کاربردش صرفاً بـرای یـه سـری جاهـای خاص هست!)

## Two-time pad real-world examples

- این دقیقاً اشتباهی بود که شوروی انجام داد و آمریکا تونست رمزاشون رو بشکنه.[۱۰۶]
- پروتکل PPTP توی Windows NT که مایکروسافت برای ارتباط با سرور ساخته بـود هم همین مشکل رو داشت.

یعنی:

اول یه seed بین سرور و کلاینت به اشتراک گذاشته میشه. seed که هر دو از اون اسـتفاده می‌کنن و عملاً یه کلید تولید می‌کنن.

کلاینت: تمام پیام‌ها با هم concatenate میشن و به stream cipher داده میشن. دونه دونه بایت‌ها میرن XOR میشن با کلید.

سرور: تمام پیام‌هـا بـا هم concatenate میشـن و بـه stream cipher داده میشـن و خب مشـکل اینجاست! دقیقاً با همون کلید رمز انجام میشه. یعنی هم سرور و هم کلاینت یـه کلیـد دارن و خب دوبـاره مشکل دوبار استفاده‌کردن کلید رو داریم!

- پروتکل WEP مودم

ارتباط بین شما و مودم باید رمز بشه. خب در پروتکل WEP هم همین اتفـاق میوفتـاد. اومـده بـودن از رمزنگاری stream cipher استفاده می‌کردن اما می‌دونستن که seed ای که برای ساخت کلید اسـتفاده میشه، نباید هر دفعه یکسان باشه. یعنی کلید باید هی عوض شـه و two-time pad نباشـه. امـا غافـل از اینکه درست این رو درک نکرده بودن!

106https://en.wikipedia.org/wiki/Venona_project

یعنی seed همیشه عدد بود که concatenate شده بود با iv (initial vector). این iv ما درواقع شماره موجود در پکت بود. که ۲۴ بیتیه و عملاً از ۰ تا $2^{24}-1$ ادامه پیدا می‌کنه و بعد ریست میشه.
یعنی درواقع من هر دفعه میام seed رو با **عددی که ا** و **iv** می‌سازم (concatenate شده دوتاشون). بعدش seed رو میدم به یه pudeo random generator و کلید ساخت رمز رو میده. بعد پیام رو رمز می‌کنم و می‌فرستم سمت مودم. مودم iv رو از توی پکت می‌خونه. کلید رو هم که از پیش تعیین شده و بلده. میاد concatenate می‌کنه و seed رو به دست میاره. با این seed میاد کلید رمزنگاری رو می‌سازم همینجا دقیقاً مشکل به وجود میاد. بعد $2^{24}-1$ بار رمزشدن، عملاً iv میشه ۰ و دوباره داریم یه seed یکسان که قبلاً استفاده شده بود رو استفاده می‌کنیم.

## More:

+ Hacking Secret Ciphers with Python[107]

# Simple XOR Encryption:

اول با رمزنگاری XOR آشنا شیم:

*Plain_text* XOR *Key* = *Cipher_text*
*Cipher_text* XOR *Key* = *Plain_text*

اما یه مشکل!

*Plain_text* *XOR* *Cipher_text* = *Key*

یعنی صرفاً برای یکی از چیزا هم متن رمزشده و هم خود پیام رو داشته باشیم،[۱۰۸] عملاً کلید پیدا میشه و عملاً با داشتن کلید، تمام پیام‌های رمزی دیگه هم که با همین کلید رمز شدن، باز میشه.
رمز با XOR خیلی خیلی معروفه ولی خب امنیتش خوب نیست!

خب حالا بریم سر یه موضوع مهم:

## از یه کلید دوبار استفاده نکنید!

- چرا؟

+ فرض کنین من با XOR پیام m1 و m2 رو رمز می‌کنم:

C1 = m1 ⊕ *key*
C2 = m2 ⊕ *key*

خب چون هم مسیج ۱ و هم مسیج ۲ با یه کلید رمز شدن، من اگر XOR دوتا C1 وC2 رو به دست بیارم، عملاً XOR مسیجا به دست میاد!

107 https://inventwithpython.com/hacking/

۱۰۸ به این حالت می‌گن «known plain-text attack».

$C1 \oplus C2 = m1 \oplus m2 \oplus key \oplus key = m1 \oplus m2$

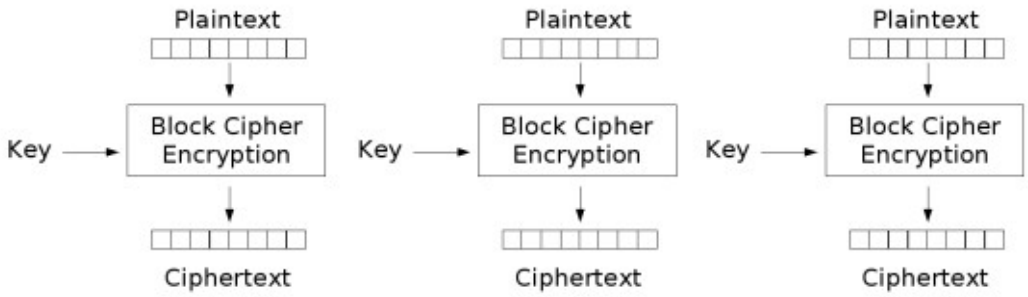
توضیح: وقتی *key* با *key* میاد XOR میشه، حاصل بیتا همه صفر میشه. صفر هم که تو XOR به تنهایی تأثیر نداره. پس از کل عبارت، فقط $m1 \oplus m2$ می‌مونه.

وقتی XOR دو عبارت m1 و m2 رو داشته باشیم، رسیدن به خود m1 و m2 کار خیلی خیلی سختی نیست. با تکرار حروف انگلیسی، میشه به m1 و m2 رسید. عملاً پیام رو می‌تونم پیدا کنم.

خب حالا فهمیدیم که وقتی میگیم با یه کلید رمز بشه و با یه کلید باز شه، یعنی اینکه هرکلیدی که برای رمزشدن استفاده بشه، با همون کلید هم می‌تونیم بازش کنیم. (Symmetric Encryption) از این نوع رمزنگاری توی کامپیوتر به طور معمول دو نوع داره:

Stream Cipher: بیت‌ها دونه‌دونه بیان و رمز بشن. (یا کرکترها دونه دونه بیان و رمز بشن)

Block Cipher: بیت‌هایی که می‌خوایم رمز کنیم رو به بلاک‌بلاک تقسیم می‌کنیم. (مثلاً ۶۴ تا ۶۴ تا جدا می‌کنیم) بعد هر بار یه بلاک (مجموعه‌ای از بیت‌ها) رو بگیریم و رمز کنیم:




Electronic Codebook (ECB) mode encryption

image[109]

+ More about block and stream ciphers[110] (A little old but infromative)

# Bit-flipping attack to AES-CBC

رمزنگاری AES (اسم رمزنگاری استانداردی که استفاده می‌کنیم)، روش‌های مختلفی برای انجام داره. بهش می‌گن «mode» هاش. مثلاً «CTR» و «CBC» و «GCM». اما همشون مثل هم نیستن. یه سری چیزا کم و زیاد دارن. مثلاً AES-CBC چک نمی‌کنه که آیا یه وقت کسی اومده روی cipher-text خرابکاری کرده یا نه؟! آیا سالمه و تغییر نیافتس یا تغییر یافته؟ خلاصه اصطلاحاً message

109 https://commons.wikimedia.org/wiki/File:ECB_encryption_less_blocks.svg → Public domain
110 https://www.schneier.com/crypto-gram/archives/2000/0115.html#BlockandStreamCiphers

authentication انجام نمیده. پس برای همین میشه یه سری حمله بهش زد.[111] (اگر خودمون هم دستی authentication رو لنجام ندیم...)

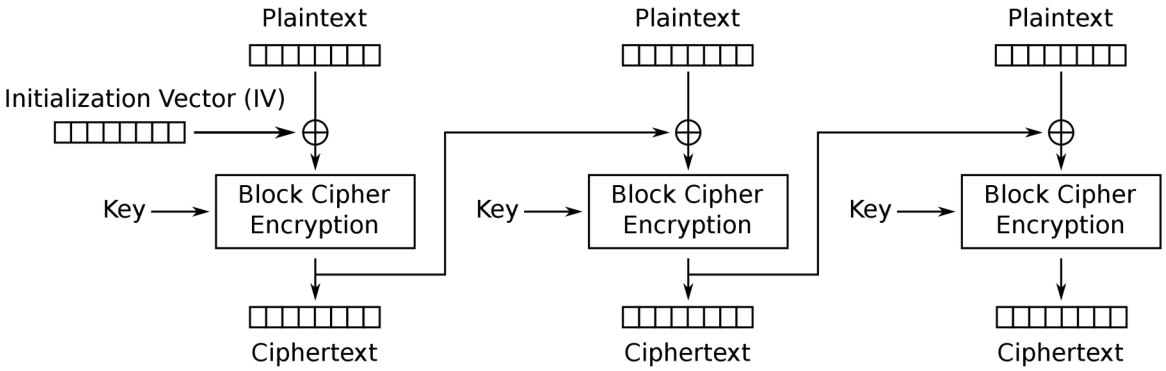
من این قسمت رو از وبسایت زیر یاد گرفتم:

https://alicegg.tech/2019/06/23/aes-cbc.html

برای بلاک‌های متفاوت، باید کلید عوض بشه.

- چه‌جوری مثلاً؟

+ مثلاً AES-CBC میاد میگه من یه initiation vector (iv) دارم. قبل دادن plain-text به اون رمزنگاری، میام XOR اش می‌کنم با iv. بعدش میدم که رمز شه. اینطوری یه cipher-text می‌سازم. حالا برای دفعه‌های بعد، میام متنم رو با cipher-text قبلی XOR می‌کنم و بعد رمز می‌کنم. همینطور تا آخر:

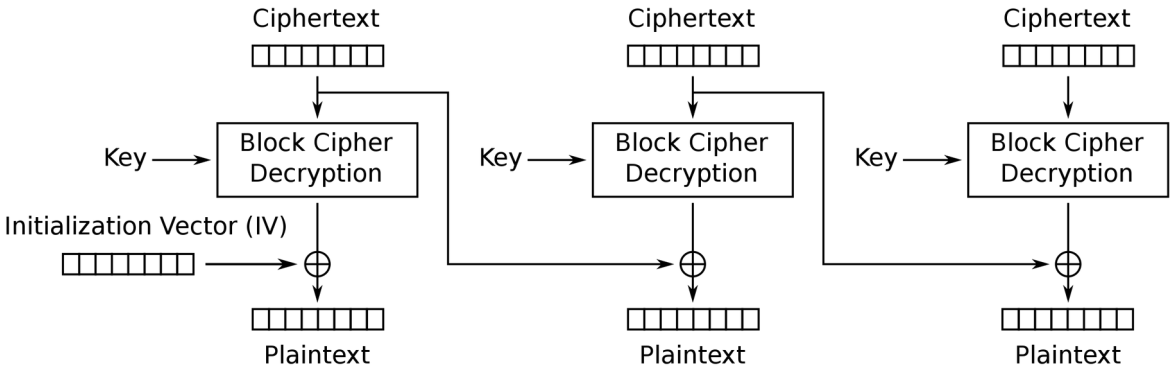



Cipher Block Chaining (CBC) mode encryption

خب رمزگشاییش چطوره؟

توی رمزنگاری اول XOR کردیم و بعد دادیم به block cipher. حالا رمزگشایی باید برعکس پیش بریم. یعنی اول بدیم به block cipher (یادتونه که یه توی رمزنگاری symmetric یه کلید رمز می‌کنه و همون کلید باز می‌کنه؟!) و بعدش XOR می‌کنیم:

111 https://learn.microsoft.com/en-us/dotnet/standard/security/vulnerabilities-cbc-mode




Cipher Block Chaining (CBC) mode decryption

حالا اوکی تا اینجا همه چیز عادیه و میگیم خب امنه! اما از دیدگاه ما امنه! از دیدگاه یـه رمزنگـار و یـه محقق امنیتی نه!

- چجوری؟

بیایم با نحوه کارکرد XOR آشنا شیم:

| plain-text | iv | Output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

وقتی iv ما عوض میشه، output ما هم عوض میشه. یعنی output اگر ۱ بـود میشـه ۰ و اگـر ۰ بـود میشه ۱!

خب این کجا کاربرد داره؟

فرض کنیم Alice وارد حساب بانکیش شده و می‌خواد به Bob مقـدار ۱۰۰ دلار بفرسـته. میـاد دکمـه تأیید میزنه. پیام اینطوری میشه:

```
“Send $100 to Bob”
```

این پیام رمز میشه و توی مسیر بـا http ارسـال میشـه. وسـط ارتبـاط، Eve میـاد داده‌هـا رو می‌دزده. درسته داده‌ها اینطورین مثلاً:

```
iv: 9bc423909ac569b5016525cb4b2660b5
ciphertext: c6d55918176051c5a603d62cdf23fa8a
```

اما Eve مثلاً از گیرنده فرستنده و یا اینکه این دو نفر دوست صمیمی هم هستن و... می‌تونه حدس بزنه احتمالاً پیام اینطور چیزیه (خیلی وقتا میشه حدس زد فرمت کلی پیام چجوریه. مثلاً هدر خیلی از فایلا مشخصه که کجاش چیه. از طریق فرمت کلی پیام می‌تونم حدس بزنم متنا چی می‌تونه باشه.):

“Send $xxx to Bob”

اما خب یادتونه با تغییر iv میشد output رو تغییر داد؟

خب اینجا باید میگه چجور باید iv رو تغییر بدم که توی output به جای Bob، اسم خودم باشه! یعنی اینطوری:

“Send $xxx to Eve”

خب میاد میگه باید اسم رو تغییر بدم. پس بیام دو پیام زیر رو XOR کنم که ببینم تفاوتشون چیه:

“Send $xxx to Bob”

“Send $xxx to Eve”

Output: 00000000000000000000000000071907

خب. پس من باید یه جوری اون iv رو تغییر بدم که این برام ساخته شه. قسمتای صفر یعنی نباید تغییر پیدا کنن. بیایم صرفاً برای عدد آخری انجام بدیم متوجه شیم:

قبول دارین در output جدید ما ۷ تولید شده؟

بیایم ۷ رو به باینری بنویسیم:

0111

درواقع ۷ تفاوته! یعنی بیت چهارم تغییر نکرده. ولی بیت اول تا سوم تغییر کردن.

خب یادتونه گفتیم خاصیت XOR اینه که اگر چیزی که باهاش XOR میشه بیتاش عوض شه، بیتای خروجی هم عوض میشه؟ حالا باید بگیم بیتای اول تا سوم خروجی عوض شده. پس بیت‌های اول تا سوم iv جدید ما دقیقاً باید NOT اون iv قدیمی باشن که خروجی عوض شده باشه. ولی بیت چهارم نه. بیت چهارم چون صفر مونده یعنی بیت چهارم output قدیمی با جدید تفاوتی نداره (XOR دو چیز یکسان، صفر میشه) و پس نیازی نیست بیت چهارم iv ما تغییر کنه!

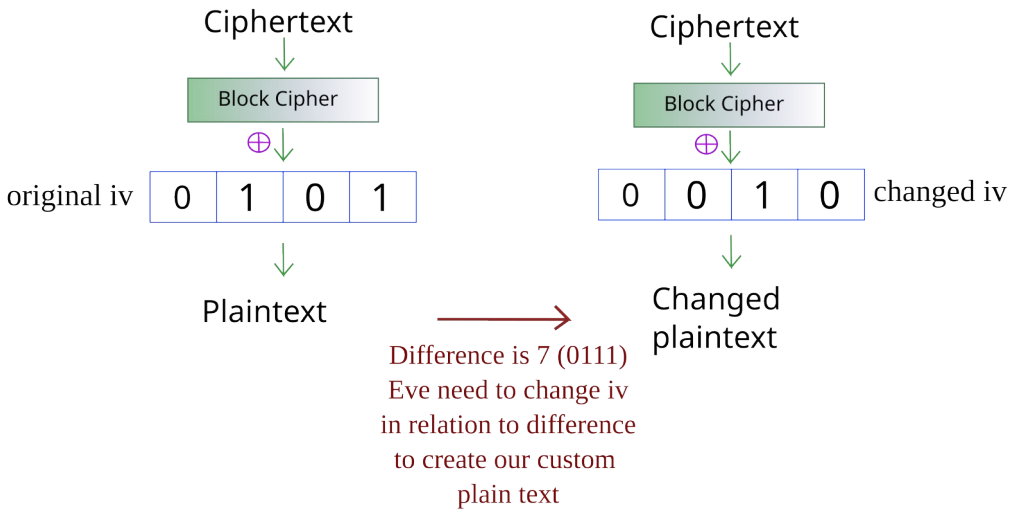
خب بایت اول iv قبلی ما ۵ بود. یعنی

0101

خب iv جدید ما چجور باید باشه که بیت اول تا سوم رو تغییر بده؟ باید جاهایی که خروجی عوض میشه، بیت‌ها دقیقاً متضاد iv قبلی باشه.

یعنی:

0010




*بیت چهارم:* چون توی تفاوت output بیت چهارم صفر بود و تغییر نکرده بود، iv هم نباید تغییر کنه.

*بیت سوم:* چون توی تفاوت output بیت سوم یک بود و تغییر کرده بود، بنا بـر خاصـیت XOR، بیت سوم iv جدید باید دقیقاً NOT بیت سوم iv قبلی شه. بیت سوم iv قبلی ۱ بود. پس الآن باید بشه ۰.

*بیت دوم:* چون توی تفاوت output بیت دوم یک بود و تغییر کرده بـود، بنـا بـر خاصـیت XOR، بیت دوم iv جدید باید دقیقاً NOT بیت دوم iv قبلی شه. بیت دوم iv قبلی ۰ بود. پس الآن باید بشه ۱.

*بیت اول:* چون توی تفاوت output بیت اول یک بود و تغییر کرده بـود، بنـا بـر خاصـیت XOR، بیت اول iv جدید باید دقیقاً NOT بیت اول iv قبلی شه. بیت اول iv قبلی ۱ بود. پس الآن باید بشه ۰.

این کار رو ما صرفاً برای بایت اول انجام دادیم. اگر برای همش انجام بدیم، iv جدید ما تولید میشه و بـا iv قبلی جایگزین می‌کنیم (چون اتصال http بود و امن نبود) و می‌فرستیم:

```
iv: 9bc423909ac569b5016525cb4b2179b2
ciphertext: c6d55918176051c5a603d62cdf23fa8a
```

حالا پیام عوض شده و به جای:

```
“Send $xxx to Bob”
```

تغییرش داده و گفته:

```
“Send $xxx to Eve”
```

خب تبریک! شما تونستین یه هک ساده انجام بدین!

**نتیجه:** message authentication هم اندازه Encryption لازمه (چک کنیم که پیام تغییر نکرده باشه و از طرف همون فرد مورد نظر اومده باشه) و رمزنگاری بدون authentication چیز درستی نیست!

پس از رمزنگاری‌ها یا mode هایی استفاده کنیم که authentication هم انجام میدن. (مثل AES-GCM) یا اگر حتماً می‌خوایم از mode ای استفاده کنیم که authentication انجام نمیده، حتماً خودمون حواسمون باشه که انجامش بدیم:

"RFC 4346 [24] describes timing attacks on CBC cipher suites, as well as mitigation techniques. TLS implementations shall use the bad_record_mac error to indicate a padding error when communications are secured using a CBC cipher suite. Implementations shall compute the MAC regardless of whether padding errors exist."[112]

"Prefer GCM or CCM modes over CBC mode. The use of an authenticated encryption mode prevents several attacks"[113]

## در امنیت، خلاقیت شخصی بدون بررسی توسط متخصصان، امن نیست!

ابتدا برید مطلب «فلسفه علم و ابطال‌پذیری» رو مطالعه کنین و بعد برگردین اینجا.

خب برگشتین؟ درواقع اینجا هم همینه. زمانی که شما بخوای اثبات کنی که برنامت امنیت داره یا رمزنگاری که ساختی امنیت داره، همیشه یه راه پیدا می‌کنی که بخوای گفته‌ات رو تأیید کنی. بگی برنامم بهترینه. نگاه کن کلی چیز براش قرار دادم که بقیه نصف این قابلیت‌ها رو هم ندارن. اما این درست نیست! بلکه باید مورد تست و بررسی قرار بگیره. باید تلاش شه رد بشه.

فرض کنین به شما بگن این درس رو بخون و لوله‌کشی رو یاد بگیر. حالا که یاد گرفتی، کدوم یک از شرایط زیر سخت‌تره؟

۱) یه لوله‌کشی ساده

۲) بررسی اینکه با چه فشار آبی، به چه حالت خاصی که یه شیر باز باشه، یه شیر بسته، یه شیر نصفه باز و... لوله‌ها نشتی پیدا می‌کنن؟

معلومه مورد ۲. چون هیچ راه مشخصی نیست. باید هی خلاقیت به خرج بدی و تست و بررسی کنی.

درواقع علم و مخصوصاً امنیت برای همین سخته. چون باید تلاش کنی که راهی پیدا کنی که اون سیستم درست کار نکنه و این با خلاقیت و داشتن دانش عمیق از اون موضوع و اینکه چجور شرایط خاص رو به کار ببرم که اون حالت اصلیش کار نکنه میسره.

---

112 NIST: Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations (2019) → Section 3.3.2 → https://csrc.nist.gov/pubs/sp/800/52/r2/final

113 NIST: Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations (2019)→ Section 3.3.1.1 https://csrc.nist.gov/pubs/sp/800/52/r2/final

اما شبه علم نیاز بـه این نـداره. چهـار نفـر داروت رو بخـورن و خـوب شـن می‌تـونی ادعـا کـنی داروت بهترینه![۱۱۴]

شما نمی‌تونی ادعا کنی که چیزی که ساختی بهترینه و مشکلی نداره و پس نیاز نیست بقیـه بررسـیش کنن. به قولی:

"Anyone, from the most clueless amateur to the best cryptographer, can create an algorithm that he himself can't break."[115] -Bruce Schneier

حتی اگر شما بهترین رمزنگار دنیا هم که باشی، باید چیزی که ساختی مورد تست و بررسی قرار بگیره.

بارها اینو تکرار کردم ولی بازم نیازه دوباره تکرارش کنم! از چیزای استاندارد استفاده کنین!

چه توی رمزنگاری چه ساخت برنامه‌هاتون، چیزایی به کار نبرین که خودساخته باشـن. اگـر هم بـه کـار می‌برین، بدونین که امکان شکستنش هست! حتی اگر از نظرتـون اون الگـوریتم مبنـای ریاضـیاتی داره؛ تـا بارها توسط افراد متخصص بررسی نشه، امن نیست!

– من یه رمزنگاری رو توی اینترنت پیدا کردم که توسط یه ریاضی‌دان ساخته شده. امنه پس درسته؟
+ نه! رمزنگاری باید توسط تعداد **زیادی** از متخصصان بررسی شه.

مثلاً رمزنگاری McGuffin (یا MacGuffin؟) که توسط Bruce Schneier یکی از رمزنگارهای خیلی معروف ساخته شده، توسط Rijment (یکی از سازندگان رمزنگـاری AES) بررسـی شـد و دیـدن امـنیتش برای Differential attack چندان بهتر از DES نیست و

"Modifying a scheme with only existing attacks in mind however is not a good design principle."[116] -Cryptoanalysis of McGuffin by Vincent Rijmen and Bart Preneel

یعنی قرار نیست هر رمزنگاری که یه متخصص اختراع کرده هم بی‌نقص باشه. بلکه باید بررسی شه!

یا حتی مثلاً Bruce Schneier توی بلاگش میگه:

"Key schedules are very hard, and I didn't understand them very well in 1993 when I designed Blowfish. I'm much prouder of the key schedule in Twofish and Threefish."[117] - Bruce Schneier

یعنی حتی متخصص هم باشین، در حوزه رمزنگاری که حوزه بسـیار بسـیار پیچیـده‌ای هسـت، هـر روز چیزای جدیدی یاد می‌گیرین.[۱۱۸]

---

۱۱۴ شاید اصلاً به خاطر چیز دیگه باشه. یا حتی خیلی از بیماری‌های میکروبی (شامل ویروس) بعد یه مدت توسط خود سیستم ایمنی بدن از بین میرن. اصلاً هم ربطی به اون دارویی که دادی نداره! بعد طرف میاد ادعا می‌کنه که دیدی داروی دست‌ساز فلان تونست کرونا رو شکست بده؟ درحالی که خود سیستم ایمنی بدن خودکار دفع کرده و ربطی هم به اون چیزی که دادی نداشته!

115 https://www.schneier.com/crypto-gram/archives/1998/1015.html#cipherdesign

116 https://web.archive.org/web/20180724081030/http://link.springer.com/content/pdf/10.1007%2F3-540-60590-8_27.pdf

117 https://www.schneier.com/blog/archives/2009/09/the_doghouse_cr.html

۱۱۸ نگین وای چقدر سخت. پس حتی اگر علاقه دارم نرم سمت رمزنگاری. اینطور نیست. حتی بهترین شرکت‌های هواپیمایی و مسافری دنیا با بهترین مهندسان دنیا هم گاهی اشتباه می‌کنن یا همه چیزو نمی‌دونن. اگر می‌دونستن که هیچ‌وقت هیچ حادثه سقوط هواپیما و... نداشتیم!

– متأسفانه کارفرما از اینکه بهش بگی یه الگوریتم خفن طراحی کردم که داده‌ها رو رمز می‌کنه، بیشتر خوشش میاد تا اینکه بگی از رمزنگاری استاندارد استفاده کردم.

کارفرما فکر می‌کنه اگر خودت انجام بدی خیلی خفن‌تره و باحال‌تره! درحالی که بذارین مثال سامسونگ رو براتون بزنم:

سامسونگ از الگوریتم درستی برای رمزنگاری استفاده کرده بود (AES-GCM) اما اشتباه پیاده‌سازیش کرده بود و ۱۰۰ میلیون دستگاه رو در معرض خطر قرار داد.[۱۱۹]

باز هم می‌رسیم به قضیه اینکه الگوریتم باید پابلیک باشه. نه در خفا! مهم نیست سامسونگین، کوالکامین، اپلید، چی هستین. الگوریتم رمزنگاری **باید** بررسی بشه! (نه فقط توسط خودتون! بلکه توسط استادان و دنیای آکادمیک):

"Vendors including Samsung and Qualcomm maintain secrecy around their implementation and design of TZOSs and Tas."

"The design and implementation details should be well audited and reviewed by independent researchers and should not rely on the difficulty of reverse engineering proprietary systems."[120]

خلاقیت به خرج ندین! نگین خب من دانای اعظمم پس مراحل استاندارد رو پیش نمیرم و مراحلی که از نظرم صحیحه رو پیش میرم! یا از فلان سایت نامعتبر یا توصیه‌های افراد عادی کمک می‌گیرم.

این کار رو نکنید! شما متخصص نیستین! افراد متخصص یه چیزایی می‌دونستن که گفتن فلان رمزنگاری، فلان نوعش. شما ولی با یه دانش سطحی و آگاه‌نبودن به مسائل، ممکنه کاری کنین که از نظر خودتون امنیت رو افزایش میده ولی درواقع ممکنه در ده جای دیگه امنیت رو کاهش بده! دیگه از سامسونگ و Qualcomm که گنده‌تر نیستین که!

همچنین ریاضیات رمزنگاری معمولاً قوی‌ترین حلقه امنیت هست. امنیت از ضعیف‌ترین حلقه زنجیر می‌شکنه. یعنی چیزایی مثل درست نگه نداشتن کلید و implementation اشتباه و... .[۱۲۱]

حتی ممکنه الگوریتمش خوب باشه ولی random number generator اش بد باشه.

---

119 **Trust Dies in Darkness:** Shedding Light on Samsung's TrustZone Keymaster Design → https://eprint.iacr.org/2022/208

120 Conclusions in PDF:

```
@misc{cryptoeprint:2022/208,
    author = {Alon Shakevsky and Eyal Ronen and Avishai Wool},
    title = {Trust Dies in Darkness:  Shedding Light on Samsung's TrustZone Keymaster Design},
    howpublished = {Cryptology ePrint Archive, Paper 2022/208},
    year = {2022},
    note = {\url{https://eprint.iacr.org/2022/208}},
    url = {https://eprint.iacr.org/2022/208}
}
```

۱۲۱ به طور معمول وقتی implementation درست نباشه، حملات side-channel رخ میده. حملاتی مثل timing attack. می‌تونین به «sucrity_roadmap» توی گیتهابم مراجعه کنین و timing attack رو بشناسین.

– random generator چیه؟

+ یکم صبر کنین می‌رسیم بهش.

## More reading

+ Snake Oil[122] *(Highly recommended)*

+ An example of snake oil[123] *(Highly recommended)*

+ Snake Oil Warning Signs: Encryption Software to Avoid[124]

+ “Desktop Google Finds Holes”[125](Cache of encrypted files, must be inaccessible!)

122 https://www.schneier.com/crypto-gram/archives/1999/0215.html#snakeoil
123 https://www.schneier.com/crypto-gram/archives/2003/0215.html#4
124 https://web.archive.org/web/20030207174457/https://www.interhack.net/people/cmcurtin/snake-oil-faq.html

125 https://www.schneier.com/blog/archives/2004/11/desktop_google.html

### مشکل ناتوانی در استفاده از symmetric در بعضی شرایط:

خب گفتیم در رمزنگاری symmetric، با هر کلیدی ک رمزکنیم، با همون کلید باز میشه. پس هم کسی که می‌خواد پیام رو رمز کنه باید کلید رو داشته باشه و هم کسی که می‌خواد رمزگشایی کنه باید کلید رو داشته باشه!

خب فرض کنین من می‌خوام وارد یه وبسایت شم، من که نمی‌تونم کلید رمزنگاریم رو به وسیله اینترنت برای سایت بفرستم و بگم این کلید هست و با این کلید باز کن.

- چرا؟

+ چون کلید رو باید به صورت plain-text بفرستم و عملاً هرکسی که نگاه به ارتباط اینترنتی من کنه (مثل افرادی که با من به یک Wi-Fi وصلیم، شرکت ارائه‌دهنده اینترنت (به دلیل اینکه داده‌هام از اونجا عبور می‌کنه) و هر کسی که داخل اون شبکه باشه یا توی مسیر ارتباطی باشه)، می‌تونه کلید رو به صورت plain-text ببینه! (و خب می‌تونه پیاما رو باز کنه بعدا!) پس اینجا صحبت از یه نوع رمزنگاری جدید پیش میاد. رمزنگاری با دو کلید! درواقع یه قفل و یه کلید!

# Public-Key Encryption[126]

### توضیح ساده رمزنگاری کلید عمومی و کلید خصوصی:

بذارین روش رو اینطور توضیح بدم که شما یه جعبه و یه قفل بعلاوه یه جفت کلید دارین.

قفلمون اینطور کار می‌کنه که با هرکدوم از کلیدا قفلش کنیم، با کلید دیگری باز میشه.

یکی از کلیدا همیشه پیش خودم می‌مونه. اسمشو می‌ذارم کلید خصوصی.

یکی از کلیدا دست هر کسی می‌تونم بدم. بهش می‌گن کلید عمومی.

حالا اگر شما بخوای به من پیام بدی، من قفل و کلید عمومی رو براتون می‌فرستم. شما نامتو می‌نویسی، می‌ذاری توی صندوق. قفل رو می‌زنی سَر صندوق و با کلید عمومی من قفلش می‌کنی. حالا بسته رو می‌فرستی برای من. چون گفتیم با هرکدوم از کلیدا قفل شه، با اون یکی باز میشه و کسی کلید مخصوص و خصوصی که صرفا دست خودمه رو نداره، وسط راه کسی نمی‌تونه بازش کنه.

من اگر بخوام به شما پیام بدم، می‌گم قفلت و کلید عمومیتو بفرست. من نامم می‌نویسم می‌ذارم داخل یه صندوقی، قفلتو می‌زنم سَرِش و با کلید عمومیت قفلش می‌کنم. چون کلید خصوصی صرفاً دست توعه، فقط خودت می‌تونی بازش کنی!

این دقیقاً یکی از مکانیزمایی هست که توی HTTPS به کار میره.

حالا اینجا یه سؤال ممکنه پیش بیاد. به نظرتون نقطه ضعف این نوع رمزنگاری تو کدوم حلقشه؟ یکم فکر کنین!

---

۱۲۶ حاصل کارهای افرادی مثل «Whitfield Diffie» , «Martin Hellman»، «Ralph Merkle».

*راهنمایی میفواین؟* شما فرض کنین تمام قفل‌ها خوب و امنن و کاری به این حلقه نداشته باشین. به این فکر کنین که این رمزنگاری برای ارتباط از راه دوره و من مستقیم پیش طرف نیستم.

*پاسخ:* خب وقتی از راه دور هست، از کجا بدونم که این قفلی که برام ارسال شده، واقعاً قفل اون فردی هست که من میخوام؟! شاید یه نفر توی اداره پست نشسته بین ما و وقتی من میگم قفلتو بفرست، مامور پست قفلشو به سمت من میفرسته و وقتی اون فرد مقابلم میگه قفلتو بفرست، مامور پست قفل خودشو برای اون میفرسته. یعنی درواقع برای من نقش فرد مقابلم و برای فرد مقابلم نقش من رو بازی میکنه و وسط نشسته و به راحتی پیام‌های ما رو میخونه. من که نمیدونم این قفل مال کیه!

درواقع سناریو اینطور رخ میده:

+ آلیس: هی باب. من آلیس هستم. قفلتو بفرست برای من.

* مامور پست: درود آلیس. بیا این قفل من!

همچنین باز مامور پست: درود باب. من آلیس هستم! قفلتو بفرست برای من.

- باب: باشه. بیا این قفل من.

مکانیزم رمز و رمزگشایی:

+ آلیس: خب این پیام قفل شده رو بفرستم برای باب. (اما نمیدونه داره اشتباهی با قفل مامور پست پیامشو قفل میکنه!)

* مامور پست پیام رو میگیره و با کلیدش بازش میکنه! میخونتش و بعد با قفل باب اونو قفل میکنه. حالا میفرسته برای باب.

- باب فکر میکنه که پیام از طرف آلیس اومده. اما نمیدونه این پیام وسط راه باز شده. اینجا باب میاد پیام رو باز میکنه و میخونه. حالا باب هم بخواد پیام بفرسته، با قفل مامور پست قفل میکنه. وسط راه مامور پست بازش میکنه. بعد قفل آلیس رو میزنه سرش و میفرسته برای آلیس. آلیس هم اونور با کلید خودش باز میکنه و میخونه.

به این میگن «man-in-the-middle attack» (حمله فرد میانی). (یکم گنگه ولی دوباره سعی کنین بخونین و فکر کنین بهش تا بفهمین. وقتی فهمیدین دیگه ساده میشه!)

## متوسط:

خب یادتونه درباره قفل و کلید صحبت کردیم؟ بهتره یکم دقیق‌تر و درست‌تر صحبت کنیم.

ببینین ما در رمزنگاری Public-key دو تا کلید داریم. یکی کلید عمومی (همون قفلی که دربارش صحبت کردیم) و یکی کلید خصوصی (همون کلیدی که دربارش صحبت کردیم).

من صرفاً برای این اصطلاح قفل رو به کار بردم که درک شما راحت‌تر بشه. اما زیاد اصطلاح درستی نیست.

بیایم یه بار روند رو با کلید عمومی و خصوصی دوباره ببینیم:

من به هرکی که بخواد بهم پیام بده، کلید عمومیم رو میفرستم. بهش میگم با این پیام رو رمز کن. اون با کلید عمومی من که دست همه هم میتونه باشه رمز می‌کنه. پیام که اومد پیش من، من با کلید

خصوصیم رمزگشاییش می‌کنم. من هم بخوام به کسی پیام بدم، کلید عمومیشو می‌گیرم. با اون رمز می‌کنم و براش می‌فرستم. اون با کلید خصوصی بازش میکنه. این روندیه که رخ میده.

باز هم حالا باید یه جوری تأیید بشه که کلید عمومی که اومده پیش من، متعلق به اون فرد مورد نظر بوده و کسی بین راه خودشو جای اون جا نزده باشه. اگر این تأیید نشه، man-in-the-middle attack رخ میده.

- حالا اگر با کلید خصوصی رمز کنیم چی؟

+ با کلید عمومی باز میشه.

- خب اینکه به درد نمیخوره که! کلید عمومی که دست همه هست! اون کلید خصوصی هست که فقط دست یه نفره. خب اینطوری که همه پیام رو میتونن باز کنن. این به چه دردی میخوره دیگه؟

## • Digital signature

خب اینجا پای کاربرد امضای دیجیتال یا Digital signature وسط میاد. من اگر بخوام یه پیامی رو تأیید کنم که من فرستادمش و کسی دیگه اونو نفرستاده، میام با کلید خصوصی رمز میکنم، و خب وقت با کلید خصوصی رمز شده، فقط با کلید عمومی من و نه هیچ‌کس دیگه باز میشه. پس من اگر بخوام تأیید کنم پیام از طرف من هست، با کلید خصوصی رمز میکنم و پیام رو ارسال میکنم. مردم سعی میکنن پیام من رو با کلید عمومی که توی اینترنت گذاشتم باز کنن، اگر باز شه، تأیید میشه که فرستندش منم. چون قرار بود فقط با کلید عمومی من باز شه. این روشی هست که به کار میره برای اینکه من تأیید کنم فلان پیام یا فلان فایل از طرف من هست و کَس دیگه‌ای اونو ننوشته!

### پس چی شد؟

برای رمزنگاری، با کلید عمومی طرف مقابل رمز می‌کنم که فقط طرف مقابل که کلید خصوصی رو داره بتونه بازش کنه.

برای امضای دیجیتال و تأیید خودم، با کلید خصوصی خودم رمز می‌کنم تا همه بتونن با کلید عمومیم باز کنن و تأیید کنن که من فرستادم.

حالا این چطور با ریاضیات ممکنه؟ اصلاً چطوریه؟

### پیشرفته:

مثلاً یه نمونه از Public-key Encryption که تقریباً خیلی جاها استفاده میشه رو با هم بررسی کنیم:

## RSA (Ron Rivest, Adi Shamir, and Leonard Adleman)

من اولین بار، الگوریتم RSA رو از کانال یوتوب «Eddie Woo» با دو ویدیوی زیر یاد گرفتم:

+ The RSA Encryption Algorithm (1 of 2: Computing an Example)[127]

+ The RSA Encryption Algorithm (2 of 2: Generating the Keys)[128]

127 https://youtu.be/4zahvcJ9glg
128 https://youtu.be/oOcTVTpUsPQ

توضـیحش خیلی خـوب بـود پس ایمیـل زدم و اجـازه‌دادن کـه در شـرایط «Free and publicly available» و با «ذکر منبع» استفاده کنم. خیلی توضیح خوبی برای RSA داد. به نظرم یه سر به کانـالش بزنین چیزای خیلی خوبی رو توضیح میده. بیشتر ریاضی‌طور هست.

خب من یه کلید عمومی دارم که به بقیه میدم. اون ۵ و ۱۴ هست.

- چرا ۵ و ۱۴؟

+ فعلاً میخوایم عملیشو ببینیم. بعداً میریم سراغ توضیحات.

یه متنی رو هم میخوام رمز کنم. خب متن رو اول تبدیل می‌کنم به یه عدد. فرض کنیم که

"Hello, world" => 2

حالا میایم اینو به توان عدد اولی می‌رسونیم و بعد ازش نسبت به دومی mod یا باقی‌مونده می‌گیریم:

Public Key (Encryption Key): **(5, 14)**

$$2^5 \% 14 = 32 \% 14 = 4$$

علامت درصد «%» معمولاً به معنای باقی‌مونده گرفتن هست.

خب پس حالا ما پیام رمز شده رو ساختیم. که عدد ۴ پیام رمزشده ماست.

Private Key (Decryption Key): **(11, 14)**

برای باز کردنش هم نیاز به یه کلید خصوصی داریم. که عـدد اولش ۱۱ هسـت و عـدد دوم هم بـا قبلی مشترکه و ۱۴ هست.

- این ۱۱ و ۱۴ از کجا اومد؟

+ بعداً میگم.

خب باز روند همونه. پیام رمزشده رو بـه تـوان عـدد اول و بعـد نسـبت بـه دومی یعـنی ۱۴ باقی‌مونـده میگیریم:

$$4^{11} \% 14 = 4194304 \% 14 = 2$$

عه رسیدیم به عدد ۲ یا همون متن ابتدایی یعنی Hello, world! دیدین چجور انجام شد؟

$$Encrypted\ message = Message^{FirstNum} \% SecondNum$$

$$Decrypted\ message = EncryptedMessage^{FirstNum} \% SecondNum$$

خب دیدین عملی چجور انجام شد؟ بریم سراغ توضیحات فنی:

۱- ما دوتا عدد اول **رندوم** باید انتخاب کنیم. مثلاً ۲ و ۷. اسمشون رو میگذاریم p و q. یعنی:

$$p = 2 \quad , \quad q = 7$$

۲- اون دوتارو در هم ضرب می‌کنیم. ضربشون میشه ۲*۷ = ۱۴. این میشه عدد دومی که مشترک هم بود بین کلیدا. اسمشو میگذاریم N.

۳- اعداد ۱ تا ۱۴ رو می‌نویسیم:

1 2 3 4 5 6 7 8 9 10 11 12 13 14

حالا میایم اون عددهایی که با ۱۴ مقسوم‌علیه مشترک دارن رو خط می‌زنیم. خب اولین مقسوم‌علیه ۱۴ چیه؟ (۱ رو درنظر نمی‌گیریم چون مقسوم‌علیه همه اعداده) عدد ۲ اولین مقسوم‌علیه هست. پس تمام اعداد زوج چون مقسوم‌علیه ۲ دارن خط می‌خورن:

1 ~~2~~ 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~

حالا میریم سراغ دومین مقسوم‌علیه که ۷ هست:

1 ~~2~~ 3 ~~4~~ 5 ~~6~~ ~~7~~ ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~

مقسوم‌علیه بعدی چیه؟ ۱۴. پس اونایی که مقسوم‌علیشون ۱۴ هست باید خط بخورن. خب فقط ۱۴ هست که قبلاً خط خورده.

ما الآن چندتا عدد برامون موند؟ ۶ تا.۱ و ۳ و ۵ و ۹ و ۱۱ و ۱۳. به اینا میگن co-prime.

اما خب محاسبه اینطوری یکم سخته. بیایم از یه فرمول استفاده کنیم:

$$\varphi(N) = (p-1)(q-1)$$

$$\varphi(14) = (2-1)(7-1) = 6$$

عدد ۶ رو اینطوری پیدا میکنن.

به این تابعی که تعداد ۶ رو به دست آورد میگیم $\varphi(N)$

۴- حالا عدد اول public key چجور به وجود میاد؟ اسمشو میگذاریم e.

$$e \begin{cases} 1 < e < \varphi(N) \\ \text{co-prime with N, } \varphi(N) \end{cases}$$

خب $\varphi(N)$ که ۶ هست. پس عدد e باید ۲ یا ۳ یا ۴ یا ۵ باشه.

خب e با N = 14 و $\varphi(N) = 6$ باید co-prime باشه. یعنی مقسوم‌علیه مشترک نداشته باشن. یعنی یه e پیدا کن که با ۱۴ و ۶ مقسوم‌علیه مشترک نداشته باشه. خب اول از ۱۴ شروع میکنیم. به خاطر ۱۴، ۲ خط و ۴ خط میخوره. چون ۲ و ۴ با ۱۴ مقسوم‌علیه مشترک دارن.

~~2~~ 3 ~~4~~ 5

حالا باید اونایی که با ۶ مقسوم‌علیه دارن رو خط بزنیم:

~~2~~ ~~3~~ ~~4~~ 5

فقط عدد e = 5 باقی‌موند! حالا کلید رمزنگاری ما کامل شد: ۵ , ۱۴
حالا بریم سراغ کلید خصوصی. ۱۴ که یکسانه و باید دنبال عدد اول بگردیم:

d => (d*e) % $\varphi$(N) = 1
(d * 5) % *14 = 1*

+ d چی میشه؟ ۵ – ۱۱ – ۱۷ ....

ما اولی رو در نظر نمی‌گیریم و میریم سراغ ۱۱. خب d = 11 هم به دست اومد:

Public key = (5, 14)
Private key = (11, 14)

- چرا اولی رو درنظر نمی‌گیریم؟

+ نمیدونم. (شما می‌دونین؟ بگین تا اضافش کنم!)

- یه سوال! من ۱۴ رو میدم به عنوان public key! خب مشخصه پشتش ۲ و ۷ بوده! چون عدد ما همیشه از ضرب دو عدد اول به وجود میاد! خب طرف راحت میتونه این دوتا عدد اول رو پیدا کنه و بعدش مسیر رو طی کنه و برسه به کلید خصوصی. مثل کاری که در ساخت کلید خصوصی استفاده کردیم. اینکه راحت میشکنه! فقط نیازه بفهمه که دو عدد اول پشت ۱۴ چی هستن! چطور میگن امنه پس؟!

+ سؤال کاملاً بجایی بود. آره میشه فهمید پشت اون ۱۴ چه عددای اولی هست و میتونه هم مسیر رو طی کنه و برسه به کلید خصوصی و عملاً بشکونتش. کاملاً درسته! اما اما! یه مسأله‌ای هست. در RSA، اعداد بسیار بسیار بزرگن! مثلاً وقتی از RSA-2048 حرف میزنیم، یعنی عدد ما ۶۱۷ رقمه! حالا پیداکردن فاکتورهای اولش، به شدت سخته! دیگه عدد ما ۱۴ نیست که راحت بگیم خب ۲ و ۷ هست دیگه! بلکه عدد ما ۶۱۷ رقمه! به شدت بزرگه! حتی سوپرکامپیوترها هم نمی‌تونن همچین عدد بزرگی رو فاکتور بگیرن!

درواقع امنیت RSA، برپایه اینه که ضرب‌کردن دو عدد اولی که فقط خودم میدونم سادس ولی اون عدد حاصل که ضرب دو عدد اول هست و میره دست بقیه، فاکتورگیری ازش به شدت سخت و زمان‌بره.

درواقع وقتی میگیم به شدت سخته، یعنی **فعلاً با دانش امروزمون**، هیچ الگوریتمی پیدا نشده که بتونه در زمان معقولی،[۱۲۹] اون دو عدد اول پشت یه عدد رو فاکتور بگیره. (اگر یه روز پیدا شه، بله امنیت به خطر میوفته! ولی فعلاً نشده! درواقع اثبات نشده که الگوریتمی وجود نخواهد داشت. بلکه فعلاً چیزی که ریاضی‌دان‌ها می‌دونن اینه که فعلاً پیدا نشده.)

۱۲۹ مثلاً time complexity چندجمله‌ای.

درواقع هدف ما توی رمزنگاری کلید عمومی و خصوصی اینه که یه فرمولی، تابعی، الگوریتمی داشته باشیم که انجامش ساده باشه ولی برگردوندنش خیلی سخت.

یعنی اگر بخوایم متن عادی (plain-text) رو به متن رمزشده (cipher-text) تبدیل کنیم، ساده باشه ولی رسیدن از متن رمزشده به متن عادی به شدت سخت باشه.[۱۳۰] (نشه متن رمزشده رو برعکس کرد و رسید به رمزنشده) (به این الگوریتم‌ها میگن Trapdoor Function)

از خیلی خیلی سخت، منظورمون اینه که حتی با سوپر کامپیوترها، میلیاردها سال طول بکشه که بخوایم برعکس عملیات رو انجام بدیم! برای همین عملاً میگیم اینا نمی‌شکنن! همچنین خیلی هزینه‌بردار باشه که به‌صرفه نباشه! یعنی مثلاً صدها میلیون دلار هزینه و انرژی و برق و... ببره تا بشه عملیات رو انجام داد! که عملاً به‌صرفه نباشه!

هم هزینه‌بردار بودن و هم اینکه زمان زیاد ببره برای شکستنش خیلی مهمن. چون هزینه به تنهایی، مفید نیست! چون سازمان‌های دولتی مثل NSA در مواقع نیاز، ممکنه پول خیلی زیادی صرف کنن برای شکستنش! پس علاوه بر هزینه‌بردار‌بودن، باید خیلی زمان زیادی ببره تا شکسته شه!

+ شاید شنیده باشین که کامپیوترهای کوانتومی می‌تونن RSA رو بکشنن.

دلیلش اینه که یه الگوریتم برای اونا هست که برای کامپیوترهای عادی نیست. و اون الگوریتم، یافتن فاکتورهای اول رو به شدت ساده می‌کنه.[۱۳۱] پس وقتی کامپیوترهای کوانتومی قوی‌تر شن و به قدرت مورد نیازمون برسن، درواقع امنیت RSA (اینکه شناسایی اون فاکتورهای اول سخت هست)، شکسته میشه!

نکته: کلاً توی کامپیوتر اعمال جمع و منها خیلی سادس. ضرب هم تا حدودی اوکیه. اما چیزایی مثل رادیکال‌گیری و فاکتورگیری، هزینه‌بر و زمان‌برن.

درواقع شما از کجا میتونین این روند رو برعکس و reverse کنین؟ نمیشه! یا حداقل بسیار بسیار دشواره.

شما اگر تابعی مثل 3x+2 رو داشتین، راحت میتونستین برعکسش کنین. مثلاً x = 2 رو توی تابع بگذاریم، بهمون مقدار زیر رو میده:

3(2)+2 = 8

اگر هم بخوایم برعکسش کنیم، راحت منهای دو و بعد تقسیم بر ۳ می‌کنیم!

اما برای برعکس‌کردن باقی‌مانده، کار خیلی سخت میشه! خیلی خیلی سخت. مثلاً دونستیم که توی روند رمزکردن، پیام رمز ما، ۴ بود:

---

۱۳۰ شاید یهو گیج شده باشین که این مگه همون هش نبود؟ که انجامش ساده ولی برگردوندنش سخت؟
ولی دقت کنین که در هش‌هایی که بررسی کردیم (نه همه هشا)، چیزی به نام ۲ کلید نداشتیم! اینجا ولی دو کلید داریم. یکی قفل کنه، با دیگری باز میشه. ولی اگر با یکی قفل شد و کلید دیگر رو نداشتیم، رمزگشایی به شدت سخت میشه. (ناممکن نیست! ولی اونقدر سخته که میگیم اوکی با قدرت کامپیوترهای الآن و حتی آینده، میلیون‌ها سال هم‌زمان بذاریم، غیرقابل شکستنه)

۱۳۱ درواقع توی time complexity چندجمله‌ای انجامش میده:
Shor's algorithm: https://en.wikipedia.org/wiki/Shor%27s_algorithm

$2^5 \% 14 = 32 \% 14 = 4$

حالا بیایم برعکسش کنیم. ۴ باقی‌مونده تقسیم یه چیزی بر ۱۴ بوده. خب اون چیز چی بوده؟ خب میتونه ۴ – ۱۸ – ۳۲ – ۴۶ – ۶۰ ... باشه!

از کجا بدونیم که ۳۲ بوده؟ دیدین؟! برای همین برعکس‌کردنش سخت هست! بعدشم از کجا بدونیم اون پیام چیه؟ باید بگیم یه چیزی به توان ۵ رسیده که شده فلان. حالا اون چیز چی بوده؟ درواقع زمانی که اعداد به شدت بزرگ باشن این چیزا در کنار هم کار رو به شدت سخت و هزینه‌بردار می‌کنه.

## Low-entropy message attack

فرض کنین یه پیام کوتاه با RSA رمز شده و ارسال شده. حالا فکر کنین که چه‌جوری بدون اینکه حتی من خودِ RSA رو بشکونم، میشه بفهمم اون پیام کوتاه چی بوده؟

+ میام پیام‌های کوتاه مختلف مثل: hi, OK, yes, no, hello, ... رو با کلید عمومی طرف هی رمز می‌کنم تا ببینم چه زمانی عیناً رمز ما تولید میشه. مثلا:

Real encrypted message: `b7a0c9`

حالا من میام پیام‌های کوتاه مختلف رو با brute-force رمز می‌کنم تا ببینم کی به پیام واقعی رد و بدل شده می‌رسم:

hi: `be1e8f`
hello: `ab2ff0`
yes: `b7a0c9`
no: `bd3a0d`

دیدین؟ پیدا شد! پس می‌فهمم پیام رد و بدل‌شده، «yes» هست. من RSA رو نشکستم ولی تونستم بفهمم پیام چی بوده!

جای این پیام کوتاه، هر پیام دارای entropy[۱۳۲] کم و قابل‌حدس مثل اطلاعات Json[۱۳۳] یه فرد، قابل brute-force هست. خیلی وبستایا به این نوع حملات میگن «short message attack». اما من خودم اسم «low-entropy attack» رو بیشتر دوست دارم.

چون که هرچی متن پیچیده‌تر باشه، brute-force کردنش سخت‌تره. مثلاً یه تیکه از یه تیکه کلام معروف (حالا هرچقدرم طولانی باشه!)، خیلی راحت بروت‌فورس میشه! درواقع شبیه پسورده! که گفتیم هرچی پسورد ساده‌تر و قابل‌حدس‌زدن‌تر، بروت‌فورس و دیکشنری‌اتک و این‌هاش هم ساده‌تر. اینجا هم هرچی متن بامعنی‌تر و ساده‌تر و کوتاه‌تر، brute force اش ساده‌تر!

---

۱۳۲ انتروپی یعنی بی‌نظمی، آشفتگی، تصادفی‌بودن. وقتی میگیم entropy یه پیام پایینه، یعنی بامعناس. یعنی تصادفی و رندوم نیست کرکتراش. مثلاً:

low entropy: Hello
high entropy: lsuq

۱۳۳ یه فرمت برای تبادل اطلاعات در وب.

### راه مقابله

به نظرتون راه مقابله چیه که نتونه بروت فورس کنه؟

*راهنمایی*: توی هش مشابهشو داشتیم!

راه‌حل اینه که یه چیزی مثل salt داشته باشیم. (اینجا بهش می‌گیم padding). یعنی یه استرینگی بهش اضافه کنیم که نتونن بروت‌فورس کنن![۱۳۴]

134https://en.wikipedia.org/wiki/Padding_(cryptography)

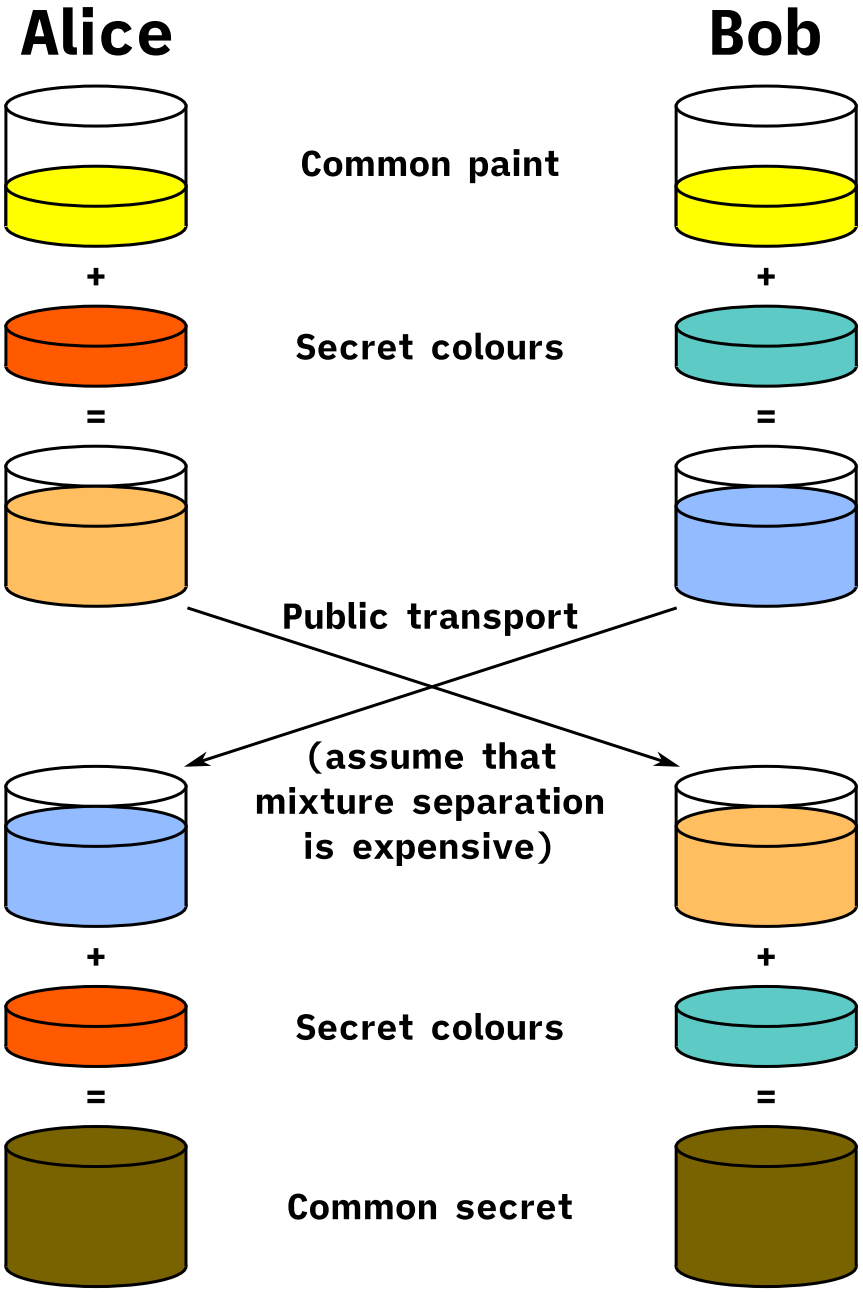
# Diffie-Hellman-Merkle key exchange

*ما نیاز داریم بدون اینکه بخوایم کلید رمزنگاری* symmetric *رو توی اینترنت ارسال کنیم، کاری کنیم که کلید در دو طرف ارتباط ساخته شه. یعنی یه الگوریتم بیابیم که کلید رو در دو طرف بسازه و عیناً کلیدا یکسان شن ولی خود کلید هیچ‌وقت جابه‌جا نشه! جابه‌جا کردن خود کلید چیز جالبی نیست!*

من این مطلب رو از وبسایت زیر یاد گرفتم:

https://skerritt.blog/diffie-hellman-merkle/

وبسایت خیلی خوبیه و پیشنهاد می‌کنم حتماً یه سری بهش بزنین. مطالب آموزشی و با زبان ساده برای چیزای کامپیوتری می‌گذاره.



Image[135]

اول با این پیش میریم که ببینیم کلاً نحوه‌کار مفهومش چجوریه.

ما یه رنگ مشترک داریم که زرده و هم Bob و هم Alice هردوشون تـوی ظرفشـون این رو دارن. بعـد هرکدومشون میان یه رنگ خصوصی و مخصوص خودشون رو اضافه میکنن. Alice میـاد یـه رنـگ نـارنجی اضافه میکنه و Bob میاد رنگ فیروزه‌ای رو اضافه میکنه. حالا رنگا عوض شد درسته؟ حالا میان ظرفاشـونو با هم عوض میکنن. یعنی Alice ظرفش رو میفرسته برای Bob و Bob هم ظرفشو برای Alice می‌فرسته. حالا که ظرفا جابه‌جا شد، آلیس رنگ خصوصی و مخصوص خودش رو به ظرفی کـه دسـته اضـافه میکنـه. رنگ قهوه‌ای به دست میاد. از اونور هم Bob به ظرفی که دستشه، رنگ خودشو اضـافه میکنـه، بـازم رنـگ قهوه‌ای به دست اومد! عملاً تونستیم سر یه کلید به توافق برسیم. دلیل هم اینه که هـردو ظـرف، از رنگـای زرد، نارنجی و فیروزه‌ای تشکیل شدن! این روشی هست که بـدون اینکـه بگیم رنـگ مشـترک مـا قهـوه‌ای هست، رنگ قهوه‌ای رو بسازیم!

خب ببینین Bob میام یه دو عدد انتخاب می‌کنه به نام‌های p = 23 و g = 5. میاد اینو میفرسته برای Alice. بعد تو یه Alice یه عدد رندوم انتخاب می‌کنه که اسـمش رو میگـذاره a. مثلاً a = 4. حـالا میـاد عملیات زیر رو انجام میده:

$$A = g^a \% p \rightarrow 5^4 \% 23 = 4$$

اسم این چیزی که به دست میاد چون Alice حسابش کـرده، میگـذاریم A. حـالا ۴ رو میفرسـته بـرای Bob.

Bob هم دقیقاً کار Alice رو انجام میده. یعنی یه a مخصوص به خودش انتخاب می‌کنـه و عملیـات رو انجام میده. مثلاً a = 3:

$$B = g^a \% p \rightarrow 5^3 \% 23 = 10$$

اسم این چیزی که به دست میاد چون Bob حسابش کـرده، میگـذاریم B. حـالا Bob هم عـدد ۱۰ رو میفرسته برای Alice.

حالا Alice میاد با چیزی که Bob فرستاده براش، عملیات رو انجام میده. Bob هم با چیزی که Alice براش فرستاده عملیات رو انجام میده. طبق چه فرمولی؟ طبق فرمول زیر:

$$calculatedSelfNum^{otherNum} \% p$$

توضیح: عددی که خودش قبلاً حساب کرده بود با فرمول، به توان عددی که براش فرستادن، حـالا اینـو باقی‌مونده بگیره بر p.

کاری که Alice میکنه:

$$A^B \% p \rightarrow 4^3 \% 23 = 18$$

کاری که Bob میکنه:

$$B^A \% p \rightarrow 10^4 \% 23 = 18$$

135https://en.wikipedia.org/wiki/File:Diffie-Hellman_Key_Exchange.svg

هردو به ۱۸ رسیدن! دیدین؟ پس مشکل حل شد! بدون اینکه بخوایم کلید رمزنگاری رو جابه‌جا کنیم، میتونیم کلید رو بسازیم بدون اینکه کسی متوجه شه!

البته این باز نسبت به MITM آسیب‌پذیره!

## چرا در بعضی شرایط، RSA نه؟

چون RSA بر این قدرت استوار بود که فاکتورگیری و رسیدن به دو عدد اول سخته. اما از اونور الگوریتم‌های جدید و بهتری دارن میان[۱۳۶] که این کار رو ساده کنن.

حتی گاهی ریاضی‌دان‌ها هم به چیزی بر می‌خورن که انتظار نداشتن. مثلاً سال ۱۹۷۷، Ronald Rivest (یکی از خالقین خود RSA) طبق الگوریتمای اون زمان گفته بود که فاکتورگرفتن یه عدد ۱۲۵ رقمی، حدود ۴۰ quadrillion سال (۴۰ ضربدر ۱۰ به توان ۱۵ سال) طول می‌کشه.[۱۳۷]

سال ۱۹۹۴ با پیشرفت ریاضیات، یه عدد ۱۲۹ رقمی فاکتور گرفته شد![۱۳۸]

حتی این الگوریتم‌های جدید، وقتی عدد بزرگ‌تر میشن، بهینه‌تر عمل می‌کنن! بله! عدد بزرگ‌تر شه، الگوریتم بهینه‌تر عمل می‌کنه![۱۳۹]

- یعنی سایز کلید رو باید کوچکتر کنیم؟ چون میگی کلید بزرگ‌تر شه، سریع‌تر عمل می‌کنن!

+ خیر! گفتم بهینه‌تر! یعنی efficient تر (نه سریع‌تر)! یعنی اگر سایز کلید رو زیاد کنیم، درسته شکستنش سخت‌تر میشه ولی اونقدری که انتظار داریم سخت نمیشه! یعنی مثلاً سایز رو دوبرابر می‌کنیم ولی فقط چند درصد شکستنش سخت‌تر میشه. یعنی درواقع الگوریتم بهینه‌تر عمل کرده.

پس اگر می‌خواین چیزی رو برای مدت طولانی ذخیره کنین، سعی کنین کلید بزرگ‌تر از مقداری باشه که پیش‌بینی شده. مثلاً اگر گفتن RSA-3072 کافیه، شما RSA-4096 به کار ببرین که یه وقت اگر پیشرفتی در ریاضیات رخ داد، کمتر آسیب ببینین![۱۴۰]

- خب باشه! اصلاً ما میایم سایز کلید رو خیلی خیلی زیادتر می‌کنیم که درسته الگوریتم بهینه عمل می‌کنه ولی خب سخت‌تریش بیشتر شه. مثلاً سایز کلید رو بیست برابر می‌کنیم که مثلاً چهار برابر سختی بیشتر شه.

+ وایسید ببینم! مگه همینطوریه؟ خب اگر همینطوری قرار بود سایز کلید رو زیاد و زیادتر کنیم، خب یه دفعه یه کلید به طول مثلاً یک میلیون بیت می‌ساختیم! (یا حتی one-time pad)! اما شدنی نیست! به چه دلیل؟ به دلیل اینکه دستگاه‌های کوچک، محاسبه اون عدد بسیار بزرگ براشون سخته. دستگاه کند

---

136 Like “Quadratic Sieve and General Number Field Seive
و خب خیلی سریع‌تر از حد انتظارمون داره الگوریتم‌های بهتری پیدا میشه.

137 Gardner, Martin (1977). "Mathematical Games, August 1977" (PDF). Scientific American. **237** (2): 120–124. doi:10.1038/scientificamerican0877-120.

138 Hayes, Brian (July 1994). "The Magic Words are Squeamish Ossifrage" (PDF). Advances in Cryptology – ASIACRYPT'94. Retrieved 28 September 2015.
More info: https://en.wikipedia.org/wiki/The_Magic_Words_are_Squeamish_Ossifrage

139 https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/

140 Based on “Applied Cryptography” by “Bruce Schneier” → 7.6 Caveat Emptor

میشه، یا حتی ممکنه دستگاهه اصلاً اونقدر قدرت محاسباتی نداشته باشه! وقتی می‌خوایم یه استاندارد رمزنگاری معرفی کنیم، باید حواسمون به این‌ها هم باشه!

اینجاست که ما می‌تونیم از الگوریتم‌های دیگه‌ای که **امنن** استفاده کنیم. و اینجاست که ECC وارد می‌شود!

## ECC (Elliptic-Curve Cryptography) $(y^2=x^3+ax+b)$ [141]

مقایسه امنیتی که هر سایز کلیدی ایجاد می‌کنه:

ECC-256 = RSA-3072

ECC-384 = RSA-7680

ECC=512 = RSA-15360[142]

می‌بینین؟ ۵۱۲ بیت ECC برابر ۱۵۳۶۰ بیت RSA هست. چقدر تفاوت زیاده!

While a 2048-bit RSA public key is about 256 bytes long, an ECDSA P-384 public key is only about 48 bytes. Similarly, the RSA signature will be another 256 bytes, while the ECDSA signature will only be 96 bytes. Factoring in some additional overhead, that's a savings of nearly 400 bytes per certificate. Multiply that by how many certificates are in your chain, and how many connections you get in a day, and the bandwidth savings add up fast.[143]

- چندتا نکته درباره نمودار:

۱- نسبت به محور X ها متقارنه. (یه نقطه انتخاب کنیم، در طرف دیگه عیناً قرینه‌اش هست)

۲- هر خط غیرعمودی، حداکثر در سه جا نمودار رو قطع می‌کنه.

توجه! توضیحات این قسمت کامل نیست! لطفاً کمک کنین کاملش کنیم!

نحوه کارش

- `A dot B = -C`
- Reflect across the X-axis from -C to C
- `A dot C = -D`
- Reflect across the X-axis from -D to D
- `A dot D = -E`
- Reflect across the X-axis from -E to E

– نقطه دوم چجور به دست میاد؟

+ نقطه اول رو دات می‌کنیم با خودش. این -B رو به ما میده. حالا نسبت به محور X قرینه‌اش می‌کنیم. حالا B یا نقطه دوم به ما داده میشه!

---

۱۴۱ منابع یادگیریم:

https://hackernoon.com/elliptic-curve-cryptography-a-basic-introduction
https://blog.cloudflare.com/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/

142 https://arxiv.org/abs/quant-ph/0301141

143 https://letsencrypt.org/2020/09/17/new-root-and-intermediates.html

اما باید این رنج رو فیکس کنیم که تا یه حدی بره اینور اونور. برای همین یه X تعیین میشه کـه اگـر از اینجا بیرون زد بره اونطرف.[۱۴۴]

– خب آیا دونه دونه دات‌کردن برای عددای بزرگ سخت نیست؟ چون Private-key مـا خیلی بزرگـه. مثلاً ECC-256 یعنی ۲۵۶ بیت کلید. یعنی یه عدد ۲۵۶ بیتی! یعنی مثلاً عدد زیر:

`115792089237316195423570985008687907853269984665640564039457584007913129639935`

یعنی این تعداد بار باید دات بشه. خب خیلی سخته. فکر نمی‌کنم کامپیوترها بتونن اینقدر بار دات کنن.

+ دقیقاً! درست می‌گی! تقریباً ناممکنه با قدرت حتی سوپرکامپیوترها اینقدر بار دات کنیم.

– خب مگه نگفتی دونه دونه دات می‌کنیم تا برسیم به نقطه نهایی؟ خب مگه راه دیگه‌ای هم داره؟ نکنه راه میان‌بری هست!

+ آره راه میان‌بری هست! اینطوری:

P dot P = 2P

2P dot 2P = 4P

4P dot 4P = 8P

حالا فرض کنین بخوایم 9P رو به دست بیاریم. به جای رفتن حالت زیر که ۸ مرحلس:

1) P dot P = 2P

2) P dot 2P = 3p

3) P dot 3P = 4p

4) P dot 4P = 5p

5) P dot 5P = 6p

6) P dot 6P = 7p

7) P dot 7P = 8p

8) P dot 8P = 9p

میگیم:

1) P dot P = 2P

2) 2P dot 2P = 4P

3) 4P dot 4P = 8P

144https://blog.cloudflare.com/content/images/image01.gif

4) 8P dot P = 9P

تمام! صرفاً ۴ مرحله شد!

حالا فرض کنین 35P بخوایم حساب کنیم. در حالت عادی ۳۴ مرحله نیازه ولی برای حالت میان‌بر باید 35P = 32P dot 3P رو حساب کنیم:

1) P dot P = 2P

2) 2P dot 2P = 4P

3) 4P dot 4P = 8P

4) 8P dot 8P = 16P

5) 16P dot 16P = 32P

6) 2P dot P = 3P

7) 32P dot 3p = 35P

تمام! هرچی ارقام زیادتر میشه این راه میان‌بر **بهینه‌تر** و بهتر میشه! چون توان و ضربی داریم میریم بالا!

*پس بگین ببینم قدرت این رمزنگاری از چی میاد؟*

\+ از اینکه من اگر نقطه شروع و تعداد انجام عملیات یا همون n که private-key ما هست رو داشته باشم، کارم خیلی سادس و راحت با روش ضربی میرم جلو و نقطه پایانی رو پیدا می‌کنم. اما اگر n رو نداشته باشم و صرفاً نقطه ابتدایی و انتهایی رو داشته باشم، مجبورم دونه دونه هی حساب کنم و چک کنم ببینم درسته یا نه و توی کدوم n، نقطه نهایی به دست میاد؟ این برای اعداد ۲۵۶ بیتی که خیلی بزرگن، به شدت سخته و عملاً با کامپیوترهای عادی و حتی سوپرکامپیوترهای کلاسیک نشدنیه! (البته فعلاً!)

\- بگین ببینم با روش ضربی-توانی، برای یه عدد که بین $2^{128}$ تا $2^{256}-1$ هست، چند بار دات‌کردن نیازه؟

\+ بیایم قبلش مسائله رو برای همون ۳۵ حل کنیم. خب باید اول ۳۲ رو حساب کنیم. ۳۲ یعنی ۲ به توان ۵. دیدیم ۵ مرحله هم شد تا برسیم به ۳۲.

حالا برای این مسأله باید حداقل ۱۲۸ بار دات کنیم تا برسیم به $2^{128}$ بعدش دیگه حالا بستگی به عدد دات‌های کوچک اضافه رو انجام می‌دیم.

چرا ECC نه؟

اگر میخوایم یه چیزی رو برای آینده رمز کنیم، با کامپیوترهای کوانتومی، احتمالاً ECC زودتر از RSA میشکنه! (البته خب RSA نهایت یه مدت کوتاهی بیشتر مقاومت می‌کنه. همین! RSA هم می‌شکنه به زودیش)

+ Can Elliptic Curve Cryptography be Trusted? A Brief Analysis of the Security of a Popular Cryptosystem[145] (نخوندمش ولی به نظر جالب میاد)

https://www.schneier.com/blog/archives/2013/09/good_summary_of.html

## خنی تر:

یه مثال از P, 2P, 3P, 4P, …, 9P

https://web.archive.org/web/20160310005719/https://www.certicom.com/index.php/52-the-elliptic-curve-discrete-logarithm-problem

https://www.purplealienplanet.com/node/27

چیزای دانشگاهی سختن

https://math.uchicago.edu/~may/REU2020/REUPapers/Shevchuk.pdf

چرا ECC؟ به خاطر اینکه

جاهایی که استفاده میشه:

- TLS[146]
- “Bitcoin uses a custom elliptic curve called "*secp256k1*" with the ECDSA algorithm to produce signatures. The equation for this curve is $y^2=x^3+7$”[147]

## نقطه ضعف:

145 https://www.isaca.org/resources/isaca-journal/issues/2016/volume-3/can-elliptic-curve-cryptography-be-trusted-a-brief-analysis-of-the-security-of-a-popular-cryptosyste

146Staying on top of TLS attacks => https://blog.cloudflare.com/staying-on-top-of-tls-attacks/

147https://en.wikipedia.org/wiki/Bitcoin#Notes

نسبت به RSA در مقابل کامپیوترهای کوانتومی زودتر شکسته میشه!

# Attacks to Diffie-Hellman-Merkle

https://crypto.stackexchange.com/questions/89870/is-there-a-complete-summarized-list-of-attacks-on-diffie-hellman/

==========

## جلوگیری از MITM:

- خب پس چجور بفهمیم آیا MITM (حمله فرد میانی) رخ داده یا نه؟!

+ خب من باید مطمئن شم که اون public-key که به دستم رسیده، وسط راه عوض نشده که یکی بیاد public-key خودشو جایگزین کنه. پس من باید از یه کانال دیگه‌ای که می‌دونیم امنه، از شما بپرسم که آیا این public-key واقعاً مال خودته؟!

مثلاً به دوستمون بگیم بره بپرسه ازش. برخلاف symmetric، نیاز به دادن رمز به دوستمون نیستیم. صرفاً کلید عمومی رو بهش میدیم که مقایسه کنه. درواقع صرفاً نیازه یه نفر راست‌گو بین آدما بشینه و تأیید کنه که آیا کلیدا یکسانن یا نه.

اما این یه مشکل داره! اونم اینکه معمولاً public-key ها به شدت بزرگ و طولانی هستن. پس چیکار کنیم که باز بتونم تأیید کنم ولی نخوام عدد به اون گندگی رو بخونم؟!

+ از هش استفاده می‌کنیم! به جای یه پیام خیلی طولانی، هش رو می‌خونم و اگر هش پابلیک‌کی تو و با هشی که خودت از پابلیک‌کی حساب کردی یکسان شد، یعنی ارتباط امنه!

این دقیقاً توی سیگنال[148] و سیکرت‌چت تلگرم رخ میده. اگر شما روی پروفایل کاربر کلیک کنین، گزینه:

View encryption key

View safety number

رو احتمالاً میبینین. توی اونجا، هش پابلیک‌کی نوشته شده. حالا شما مثلاً زنگ می‌زنی به طرف مقابل میگی ببین هش رو می‌خونم ببین آیا یکسانه یا نه؟! یا اگر می‌خواین ارتباط توی سیگنال رو چک کنین که امن هست یا نه، هش رو از یه کانال امنی که قبلاً امنیتش چک کردین، برای طرف می‌فرستین و میگین چک کن ببین این عددا و حروفا یکسانن؟!

درواقع دقیقاً همین مکانیزم برای HTTPS انجام میشه. امنیت HTTPS به رمزنگاری public-key بستگی داره. پس باید تأیید شه که کلید عمومی که ارسال شده، آیا متعلق به خود سایت google.com هست یا نه؟

148https://www.signal.org/blog/safety-number-updates/

برای این کار از همون دوست راست‌گو کمک می‌گیریم. گوگل میره با امضای دیجیتال، خودشو به دوست راست‌گو معرفی می‌کنه. پس دوست راست‌گو می‌تونه مطمئن شه که این public-key متعلق به فلانی هست و یه لیست از چیزایی که دوست راست‌گو امضا کرده توی مرورگر هست.

## بیشتر درباره سرتیفیکیت و MITM؟ شما کمک کنین کامل شه!

یا اگر مرورگر ملی بدن بهتون، بازم ممکنه MITM رخ بده! چون اون مرورگر سرتیفیکیت‌های خودساخته دولت رو اوکی می‌دونه.

یه بار قزاقستان درخواست داده بود که سرتیفیکیت سطح بالای منو اضافه کنین که من بتونم هرچی سرتیفیکیت خواستم دستی امضا کنم. (می‌خواست ارتباط HTTPS شهروندانش رو رمزگشایی و شنود کنه و خب وقتی امضا شده باشه و سرتیفیکیت تأیید شده باشه، مرورگر نمی‌فهمه MITM رخ داده.)

موزیلا و گوگل و مایکروسافت و اپل هم بهش گفتن که امر دیگه‌ای ندارین؟! چایی چی؟ چایی بریزم واستون؟ می‌خوای MITM بزنی؟ برو برو ما امنیت کاربران برامون مهمه.[۱۴۹] [۱۵۰]

*تذکر!* اگر الگوریتم یکسانی هم برای امضای دیجیتال و هم برای رمزنگاری استفاده شه، موجب لو رفتن پیام میشه.[۱۵۱]

همونطور که NIST هم میگه:

"... the key used for key agreement shall be different from the ECDSA key used for digital signatures"[152]

رمزنگاری پیچیدس! به توصیه‌های کوتاه اعتماد نکنین! اگر می‌خواین سیستم راه بندازین، از افرادی که واقعاً متخصص هستن (نه اونایی که ادعای تخصص می‌کنن)، کمک بگیرین.

خب الآن که با asymmetric-key آشنا شدیم، لابد پیش خودتون می‌گین که وقتی این میشه ازش از راه دور استفاده کرد، (چجوری؟ دوست راستگو!) چه نیازی به symmetric-key ها هست؟

خب بیایم RSA رو مثال بزنیم. RSA نسبت به حمله low-entropy message attack[۱۵۳] مقاوم نیست!

همچنین الگوریتم‌های asymmetric خیلی کندن. خیلی! اگر قرار بود همه چیز با asymmetric رمز شه سایتا خیلی کند می‌شدن. برای همین اینجا میان از ترکیب هردو استفاده می‌کنن. جابه‌جایی secret ها و کلید رمزنگاری symmetric و تأیید هویت با asymmetric منتقل میشه. اما وقتی منتقل شد، چون کلید رمزنگاری symmetric دست دو طرف هست، بقیش با symmetric رمز میشه.

149 https://blog.mozilla.org/netpolicy/2020/12/18/kazakhstan-root-2020/
150 https://www.zdnet.com/article/apple-google-microsoft-and-mozilla-ban-kazakhstans-mitm-https-certificate/
151 "On the security of public key protocols" by "D. Dolev" and "A. Yao".
152 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf → 5.6.2 Using Algorithm Suites and the Effective Security Strength → page 59.
۱۵۳ درواقع chosen plain-text attack ها.

# تولید اعداد رندوم

اصلاً رندوم یعنی چی؟ رندوم درواقع یعنی تصادفی بودن. یعنی الگوریتمی یا چیزی پشتش نباشه. یعنی کاملاً شانسی و بدون اینکه از پیش تعیین شده باشه.[۱۵۴] همچنین مستقل از بقیه اعداد باشه. یعنی ربطی به اعداد قبلی و بعدیش نداشته باشه.[۱۵۵]

یعنی اینطور نباشه که بگیم مثلاً:

$$x_{n+1} = (x_n)^2 \ mod \ M$$

توضیح: x جدید ما (مرحله n + 1)، از x مرحله قبلیش (مرحله n) به توان دو، mod یه M که خود M حاصل‌ضرب دو عدد اول هست، به دست میاد.

+ درواقع این از یه الگوریتمی به دست اومده.[۱۵۶]

+ همچنین مستقل از اعداد قبلی و بعدیش نیس.

- چرا الگوریتمی نباشه؟

+ چون اگر بشه از طریق الگوریتم پیش رفت، میشه اعداد بعدی رو پیدا کرد. شده گاهی بشنوین که فلان ریاضی‌دان الگوریتم لاتاری رو پیدا کرد و برنده شد؟ بارها شده که ریاضی‌دان‌ها نحوه تولید اعداد رندوم لاتاری و مسابقات قمار و... رو پیدا کردن و برنده شدن!

+ چون اگر باشه، من کلید رمزنگاریتو می‌تونم پیدا کنم و تمام ارتباطاتت رو رمزگشایی کنم و بخونمشون![۱۵۷]

همچنین همه اعداد شانسشون برای پیداشدن، یکسان باشه. یعنی اینطور نباشه که مثلاً وقتی میگیم یه عدد رندوم از ۱ تا ۱۰۰، شانس ۲ بیشتر از ۷ باشه. بلکه همه اعداد شانس یکسانی برای انتخاب‌شدن داشته باشن. یعنی اگر بین n تا عدد می‌خوایم یه عدد پیدا کنیم، شانس همشون $\frac{1}{n}$ باشه.[۱۵۸]

یه مفهوم جالبی هم تو ریاضی داریم به اسم «قانون اعداد بزرگ» یا همون «law of large numbers». درواقع میگه اگر یه چیز احتمالاتی رو تعداد بار خیلی زیاد انجام بدیم، نتیجه همون چیزی که تئوری میگه میشه. یعنی مثلاً اگر یه سکه رو ۱ میلیارد بار بندازین، حدوداً ۵۰۰ میلیون بار «رو» و ۵۰۰ میلیون بار «پشت» میاد.

154 non-deterministic
155 statistically independent.
156 “Blum Blum Shub” pseudo-random number generator.
157 e.g. https://en.wikipedia.org/wiki/Dual_EC_DRBG

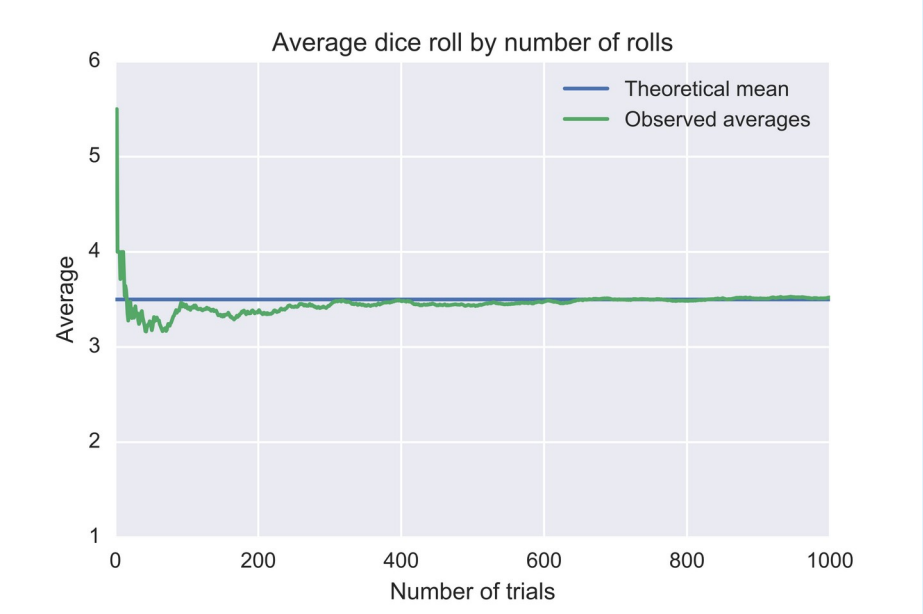
۱۵۸ بهش می‌گن «Uniform Distribution».

مثلاً اگر یه تاس رو n بار (فـرض کـنیم n عـدد خیلی بزرگیـه) بنـدازیم، انتظـار داریم میـانگین شماره‌هایی که افتاده، برابر ۳.۵ شه:

$$\frac{1+2+3+4+5+6}{6}=\frac{1}{6}\sum_{1}^{6}x=3.5$$

درواقع فرض کنین ۶ با انداختین. یه بار ۱، یه بار ۲، یه بار ۳ و... اومده. خب جمعشون تقسـیم بر تعداد حالات.

بله در تعداد کم ممکنه سه بار هم پشت هم ۶ بیاد، ولی در تعداد زیاد، در عمل، بـه سـمت اونی که انتظار داریم (احتمال تئوری) مایل میشه:




image[159]

پس درواقع اعداد رندوم باید سه ویژگی داشته باشن:

۱. از الگوریتم خاصی به دست نیان. (کاملاً شانسی باشن)
۲. مستقل از هم باشن.
۳. توزیعشون یکسان باشه.

## اعداد رندوم مگه به چه کاری میان؟

+ مثلاً شما میخواین یه قرعه‌کشی رو راه بندازین؛ باید اعـدادی کـه تولیـد می‌کـنین رنـدوم و تصـادفی باشن.

+ توی رمزنگاری بسیار بسیار مهمه که پارامترها تصادفی و رندوم باشن کـه نشـه حدسشـون زد و نشـه رمزگشاییش کرد.

159 https://en.wikipedia.org/wiki/File:Lawoflargenumbers.svg, Creative Commons CC0 1.0 Universal Public Domain Dedication

اما یه مشکل! یکی از مهم‌ترین چالش‌های رمزنگاری، تولید اعداد رندوم هست. کامپیوتر بلد نیست اعداد رندوم تولید کنه. درواقع اصلاً بلد نیست چیزی رو شانسی انجام بده! کامپیوتر یه رباته که هرچی بهش بگی انجام میده. باید بهش دستور بدی. خارج دستور هیچی نمی‌فهمه. شانسی کاری نمی‌کنه. دستور باید واضح و مشخص باشه. یعنی مثلاً بگی ۲ رو با ۴ جمع کن. ولی نمی‌تونی بگی یه عدد رندوم بده. بلد نیست! برای همین یکی از مهم‌ترین چالش‌ها اینه که چجور اعداد رندوم تولید کنیم؟ اصلاً ممکنه؟

خب اینجا بحث این مطرح میشه که ما از یه سری چیزای فیزیکی و رندوم (یا حداقل از نظرمون رندوم)، استفاده کنیم که عدد رندوم بسازیم. مثلاً از نویزهای اتمسفر استفاده کنیم که اعداد رندوم بسازیم. مطمئناً نویزهای اتمسفر خیلی منبع بزرگیه و نمیشه به آسونی توش دست برد که بخوایم اعداد رندوم رو دستکاری و deterministic کنیم.

درواقع ما از یه سری منبع طبیعی استفاده می‌کنیم که اعداد رندوم استخراج کنیم. هرچی اون منبع طبیعی رندوم‌تر و غیرقابل دسترسی‌تر (برای خرابکاری و گرفتن نتیجه دلخواه)، بهتر! درواقع به اعدادی که از این طریق به دست میان، True random number (TRN)گفته میشه.

اما یادتونه که اعداد رندوم رو از طریق فرمول‌هایی میشد به دست اورد؟ فرمولی مثل این[۱۶۰]:

$$x_{n+1}=(ax_n+c)\,mod\,m$$

درواقع این فرمول‌های ریاضی، یه سری اعدادی تولید می‌کنن که نسبتاً رندومه. یعنی از دید چشم انسان رندوم به نظر میان. برای همین بهشون میگن pseudo-random number (PRN). این اعداد کاملاً رندوم **نیستن**. همچنین deterministic هستن. همچنین periodic هستن. یعنی بعد یه مدت، دوره تناوب دارن و تکرار میشن.

مثلاً:

9 – 3 – 5 – 2 – 9 – 7 – 8 – 0 – 6 – 1 – 4 – 9 – 3 – 5 – 2 – 9 – 7 – 8 – 0 – 6 – 1 – 4

هرچی این مدته طولانی‌تر باشه بهتره. یعنی اگر یکی بعد $2^{64}$ عدد تکرار شه، خب خیلی بهتر از اونی هست که بعد ۲ میلیارد عدد تکرار میشه.

- خب این PRN ها که رندوم واقعی نیستن که. اصلاً چرا وجود دارن؟

+ تولید اعداد رندوم واقعی خیلی سخته! خیلی! بهینه نیست. زمان‌بره. هزینه‌بره.

اما تولید PRN ها خیلی بهینه‌تر و سریع‌تره.

درواقع هرکدوم برای یه سری کاربرد استفاده میشه. TRN ها برای کارهایی مثل رمزنگاری و اینا. PRN ها برای مدل‌سازی و اینا.

160 Linear congruential generator

مثلاً توی لینوکس، میاد یه سری بی‌نظمی (entropy)، رو از یه سری منابع جمع می‌کنه. از یه سری منابع رندوم. مثلاً حرکت موس، زمان کلیک‌کردن کیبورد، نمونه‌برداری از مقادیر هارددیسک و... . همه این‌ها با هم جمع میشن و یه سری بی‌نظمی تولید می‌کنن. از روی اون میشه کلید رندوم ساخت.

یا مثلاً یه سایت هست به نام random.org که میاد از نویزهای اتمسفر که یه چیز رندوم هستن و کنترل‌کردنش سخته و عملاً نمیشه، عدد رندوم میسازه.

یا مثلاً میشه از ترکیب دما و نویز و نوری که از وبکم میاد و پکت‌های شبکه که رد و بدل میشه، بی‌نظمی به دست آورد که چیزای رندوم تولید شه. (ترکیبشون! نه هرکدوم به تنهایی. چون مثلاً فن لپ‌تاپت، یه چیز periodic هست. برای همین به تنهایی اگر به کارش ببریم، باعث میشه که اعدادمون periodic و deterministic بشه)

خلاصه سعی میشه با ترکیب یه سری منابع، اعداد رندوم تولید شه.

این الگوریتم‌های تولید اعداد رندوم، به فروش میرسن. مثلاً کازینوها، لاتاری‌ها، رمزنگاری‌ها از این چیزا استفاده می‌کنن.

با سخت‌افزار هم میشه اعداد رندوم تولید کرد که اما باز به خاطر ذات پیچیده‌بودن سیلیکون و نتونستن بررسی و audit ساده اون، بهش انتقاد وارده که از این‌ها به طور مستقیم استفاده نشه چون باید مطمئن شد که اعداد رندوم درستی تولید می‌کنن و بکدوری ندارن!

برای همین لینوکس به طور مستقیم ازش استفاده نمی‌کرد. بلکه با بی‌نظمی‌های خودش XOR اش می‌کرد.[۱۶۱]

### پیشرفته:

کد زیر یه کد جالبیه که میاد آخرین رقم فرکانس CPU به KHz رو میگیره و برامون عدد شانسی (نه کاملاً رندوم) میده![۱۶۲]

```
    #include <stdio.h>

    int len(char s[]);     // length of an array.
    char randomNum0To9(); // using the last digit of the cpu clock speed (in KHz) to
generate pseudo-random number!

    char randomNum0To9()
    {
        int length;
        length = 0;
        // ------------ open the file.
        FILE *fp = fopen("/sys/devices/system/cpu/cpu1/cpufreq/scaling_cur_freq", "r");
        if (fp == NULL)
```

161 Code: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/drivers/char/random.c

162 Code is available on my github page

```
    {
        printf("Error opening file!\n");
        return 1;
    }
    char line1[8]; // first line of the file. (thread2's frequency)
    fgets(line1, 8, fp);
    fclose(fp);
    // ------------ file closed.
    length = len(line1);
    return line1[length - 1]; // return the last character of the line.
(thread1_frequency % 10)
}

int len(char s[])
{
    int length = 0;
    while (s[length])
        ++length;
    --length; // because the last character is '\0' or EOF.
    return length;
}
```

## بیشتر:

+ ویدیو جادی ← ((کامپیوترها چطوری عدد تصادفی می‌سازن))[۱۶۳]

+ اگر دنبال اینین که بیشتر درباره اعداد رندوم، چک اینکه یه سری اعداد رندوم تولید شدن یا نــه و...، یه سری به وبسایت «random.org»[۱۶۴] بزنین. خیلی چیزای خوبی داره.

+ وبسایت «lotterycodex»[۱۶۵] که پر از آنالیزهای جالب هست.

+ Random number generator attack[166]

+ Ensuring Randomness with Linux's Random Number Generator[167]

+ Could RDRAND (Intel) compromise entropy?[168]

+ PCG, A Family of Better Random Number Generators[169]

163 https://www.youtube.com/watch?v=cwHZSu-zAcI
164 https://www.random.org/
165 https://lotterycodex.com/
166 https://en.wikipedia.org/wiki/Random_number_generator_attack
167 https://blog.cloudflare.com/ensuring-randomness-with-linuxs-random-number-generator/
168 https://crypto.stackexchange.com/a/10285
169 https://www.pcg-random.org/

البته نخوندمش ولی به نظر چیز جالبی میومد. حداقل یه نگاه انداختن بهش جالبه.

# Post-Quantum Cryptography

*درباره این اطلاعات چندانی ندارم. پس صرفاً به چند چیز بدیهی بسنده می‌کنم.*

یادتونه گفتیم که مثلاً RSA، چون **فعلاً** الگوریتمی پیدا نشده که بتونه در زمان معقول فاکتور اول رو به دست بیاره امنه؟ خب خبر بد! برای کامپیوترهای کوانتومی الگوریتمش پیدا شده. (روی کامپیوترهای عادی کار نمی‌کنه.)

همچنین برای Elliptic-curve cryptography هم پیدا شده. تازه ECC زودتر از RSA شکسته میشه!

– آیا نیازه نگران باشیم؟!

+ شما نه! نگران هم باشین تفاوتی نداره! متخصصان باید تلاش کنن که الگوریتم‌های مقاوم دربرابر کامپیوترهای کوانتومی رو بسازن. در این راستا NIST این کار رو انجام داده.[۱۷۰]

توی این حوزه، صرفاً به افراد متخصص یا سازمان‌های معتبر مراجعه کنین:

NIST anticipates that the evaluation process for these post-quantum cryptosystems may be significantly more complex than the evaluation of the SHA-3 and AES candidates. One reason is that the requirements for public-key encryption and digital signatures are more complicated. Another reason is that the current scientific understanding of the power of quantum computers is far from comprehensive. Finally, some of the candidate post-quantum cryptosystems may have completely different design attributes and mathematical foundations, so that a direct comparison of candidates would be difficult or impossible.[171]

به این دلیلم بود که اولش گفتم که دانش زیادی ندارم دربارش. ساخت الگوریتم‌های Quantum-resistant خیلی سخت و طولانیه.[۱۷۲] هیچکسی جز افراد واقعاً متخصص، افرادی که واقعاً می‌دونن کامپیوترهای کوانتومی چطور کار می‌کنن و ریاضی‌دانن...، دانش کافی نداره. کمک نگیرین ازشون! پس صرفاً از افراد متخصص کمک بخواین نه یه آدم عادی یا یه دانشجوی ساده.

اما برای symmetric-key ها، تعداد security-bit هاش به نصف میرسه. یعنی AES-256 عملاً اندازه AES-128 امنه.[۱۷۳]

170 https://csrc.nist.gov/projects/post-quantum-cryptography
171 https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization/Call-for-Proposals
172 https://www.schneier.com/blog/archives/2023/08/you-cant-rush-post-quantum-computing-standards.html
173 Quantum Computing and Cryptography: https://www.schneier.com/blog/archives/2018/09/quantum_computi_2.html

اما AES-128، میشه AES-64[۱۷۴] که امن نیست! پس اگر می‌خواین یه سری داده رو برای مدتی خیلی طولانی نگهداری کنین، بهتره از AES-256 استفاده کنین که از کامپیوترهای کوانتومی هم در امان باشن.

توجه! لطفاً با جمله اینکه ما رمزنگاری post-quantum آوردیم هیجان‌زده نشین و فکر کنین اون سرویس خیلی خوبه. نه! رمزنگاری‌های post-quantum هنوز اونقدر که باید بررسی نشدن. هنوز دانش ریاضیاتی ما از کامپیوترهای کوانتومی بسیار بسیار پایینه! احتمال قابل‌توجهی وجود داره که رمزنگاری‌های post-quantum بشکنن! چون دانش ما در این زمینه زیاد نیست. هرچی جلوتر میریم حملات بهتری کشف میشه. مثلاً یکی از الگوریتمایی که برای روند استانداردکردن الگوریتم post-quantum به NIST ارسال شده بود، شکست. اونم با کامپیوترهای معمولی![۱۷۵]

خلاصه که آره فکر نکنین post-quantum بهترینه! سرویسا اگر می‌خوان الگوریتم‌های post-quantum به کار ببرن، باید اون رو به عنوان لایه‌ای اضافی بر رمزنگاری‌های عادی به کار ببرن.[۱۷۶] یعنی صرفاً یه چیز اضافه. نه اینکه رمزنگاری‌های عادی رو کلاً بردارن و جاش اینا رو بذارن.

+ Quantum Computing from Schneier.com[177]

## Lattice and Lattice-based problems:[178]

+ A Gentle Introduction to Lattices and Lattice-Based Key Exchange: Part 1[179]
+ A Gentle Introduction to Lattices and Lattice-based Key Exchanges: Part 2[180]
+ Lattice-based cryptography[181]

==========

## چجور متوجه شیم که چه رمزنگاری‌ای/هشی درون یه اپلیکیشن اعمال شده؟

نیازه یکی که دانش داره، کاملش کنه!

https://aluigi.altervista.org/mytoolz.htm#signsrch

---

۱۷۴ کاری به اینکه اصلاً AES ورژن ۶۴ نداره نداریم! صرفاً خواستم نشون بدم که اونقدر میشه.

175 https://csrc.nist.gov/csrc/media/Projects/post-quantum-cryptography/documents/round-4/submissions/sike-team-note-insecure.pdf

176 Quantum Resistance and the Signal Protocol → https://signal.org/blog/pqxdh/

177 Post-Quantum Cryptography: https://www.schneier.com/tag/quantum-computing/

۱۷۸ به دلیل اینکه حل بعضی مسائلش حتی با کامپیوترهای کوانتومی سخته، در رمزنگاری‌های کوانتومی استفاده میشه.

179 https://writing.chelseakomlo.com/gentle-introduction-lattice-crypto/

180 https://writing.chelseakomlo.com/a-gentle-introduction-to-lattices-and-lattice-based-key-exchanges-part-2/

181 https://en.wikipedia.org/wiki/Lattice-based_cryptography

اولین چیز: طول string. من اگر طول هش رو بدونم، میفهمم این حداقل فلان هش و فلان هش نیست و احتمالاً فلان نوع‌هاست.

کرکترهای به کار رفته. آیا مثلاً هگزادسیماله؟ آیا alphabet هست؟ آیا مثلاً کرکتر خاصی وجـود داره؟ مطمئناً هش زیر، از خانواده SHA نیست:

`$2y$10$TfrZ87FWy/sgLft8/DXOX.A06Sv/kG2.8XJ86PwtTLdYKZz9uUZAy`

اگر من پسوردمو عوض کنم و دوباره پسورد قبلی رو بگذارم، آیا هش همونه؟ یا عوض شده؟ اگـر عـوض نشده، احتمالاً رمزنگاری به کار برده یا هشش salt نداشته! اگر عوض شه، احتمالاً یعنی salt رندوم به کـار برده.

من با چندتا اکانت پسورد رو چیزی بذارم که هشش رو میدونم. بعداً که دیتابیس پسـوردا لـو رفت، اگـر سالت استفاده نکرده باشه، هم هش استفاده شده رو میفهمم و هم میتونم اتک بـزنم بهش! پس لزومـاً قـرار نیست هکرای بد، یه مرحله بعد لو رفتن دیتـابیس کـار رو شـروع کنن. این کـارا رو هم میشـه کـرد بـرای سازمان‌های مهم!

یا حتی ممکنه با عمد، دونفر پسورد یکسان به کار ببرن که بعد لو رفتن دیتابیس، بشه فهمید کـه salt استفاده میشه و آیا رندومه یا با یه الگویی هست؟ مثلاً salt با یه الگویی تولید میشه:

```
3639efcd08abb273b1619e1
3639efcd08abb273b1619e2
3639efcd08abb273b1619e3
3639efcd08abb273b1619e4
...
```

اگر یه حدی برای پسورد هست، من احتمالاً میتونم با دانشی که از الگوریتم‌های هش دارم، متوجه شـم که اولاً این احتمالاً هش هست و نه رمزنگاری (چون حداکثر کرکتر محدوده) دوما این احتمـالاً هش فلانـه. چون برای رندوم بودن و بهترینش، مثلاً

هممم. شاید بشه با نگاه کردن به کدش فهمید.

- خب شاید کد نباشه.

با ریورس انجیرینگ میشه؟

# zero-knowledge proof (اثبات دانایی صفر):

یکی از موضوعات جالب در کریپتوگرفی، مبحث zero knowledge proof هست. بیایم با یه مثل بررسیش کنیم:

فرض کنین یه فرد کوررنگ داریم و یه فرد بینا که شما باشین. شما یه گلوله سبز و یه گلوله قرمز به طرف میدین. میگین رنگ این دو متفاوته. اما اون که نمیتونه بفهمه. شما میخواین بدون اینکه به طرف مقابل بگین که مثلا سمت راستی رنگش سبزه و سمت چپی قرمز، بهش اثبات کنین که متفاوتن. چجوری؟

خب من به طرف میگم این دو گلوله رو بگیر ببر پشت سرت، اگر میخوای جابه‌جاش کن. بعد بیار جلو و من بهت میگم جابه‌جا شدن یا نه.

خب فرض کنیم یه مرحله این انجام شد و من چون رنگا رو می‌بینم و سمت چپ و راست رو می‌بینم، بهش گفتم جابه‌جا شدن یا نه. اما خب امکان داره شانسی هم گفته باشم و الکی بوده حرفم. پس ۵۰ درصد شانس داشتم که درست بگم. حالا بهش میگم دوباره برو این رو یا عوض کن یا نکن. من بهت میگم عوض شده یا نه. دوباره بهش میگم. اما ممکنه این بار هم شانسی گفته باشم. یعنی باز ۵۰ درصد شانس دارم که شانسی بگم. که بیایم حساب کنیم:

50/100 * 50/100 => %25

یعنی ۵۰ درصد قبل و ۵۰ درصد الآن. خب اگر هردو اتفاق بیوفته، ضرب رخ میده. تا الآن ۲۵ درصد امکان داره من شانسی گفته باشم. همینطور این عمل رو تکرار میکنم. اگر بار بعد هم ادعا کرد که بابا تو شانسی گفتی و این دو تفاوتی ندارن، شانسی بودن حدس من:

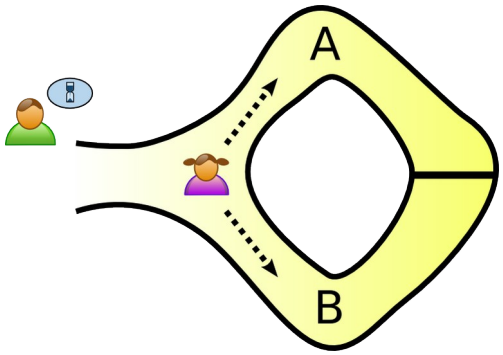
50/100 * 50/100 * 50/100 = %12.5

یعنی حالا من به احتمال ۸۷.۵ درصد درست میگم. اگر ۲۰ بار این رو تکرار کنیم، احتمال اینکه من شانسی گفته باشم، ۰.۰۰۰۰۰۱ هست. یعنی به احتمال ۹۹.۹۹۹۹۹۹ درصد درست دارم میگم و شانسی هم نیست! عملا تعداد بار زیادی عمل رو تکرار کنم، عملا دارم بدون اینکه شانسی باشه جواب میدم. این بهش میگن zero knowledge proof!

بیایم حالا یه چندتا مثال دیگه حل کنیم تا فکرمون رو با این تقویت کنیم.

### سؤال دوم:

فرض کنین یه Bob داریم و یه Alice. یه غاری هم داریم که به دو مسیر ختم میشه. مسیر A و مسیر B. اما در انتهای غار، یه در جادویی هست که فقط با گفتن یه کلمه رمز محرمانه باز میشه. فقط هم Alice این کلمه رو میدونه. حالا Alice میخواد بدون اینکه به Bob کلمه رو بگه، بهش اثبات کنه که کلمه رو میدونه. چطور میتونه این کار رو انجام بده. یکم فکر کنین و بعدش جوابو بخونین:

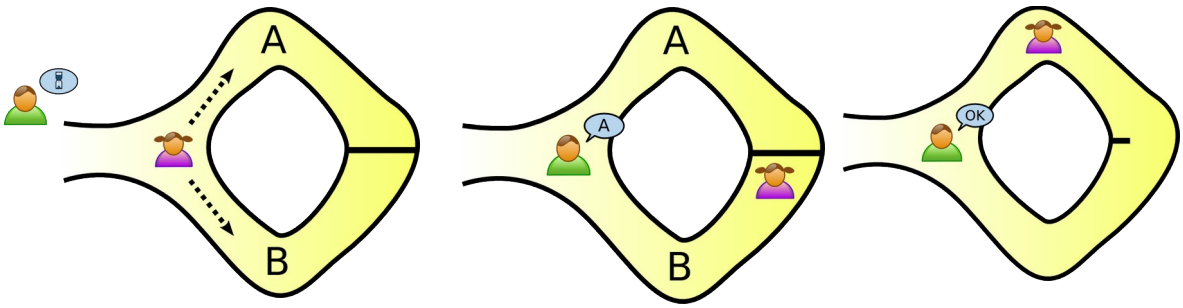



Dake, Zkip alibaba1, CC BY 2.5

پاسخ:

خب ببینین مثل قبل باید با احتمال و این‌ها سعی کنیم پیش بریم. درواقع مثلاً یه کاری انجام بدیم که اوکی ممکنه هی طرف بگه شانسیه، اما بعد چندبار انجام‌دادنش، دیگه احتمال شانس خیلی خیلی کم میشه.

خب Alice به Bob میگه تو بیرون غار بمون. من میرم تو. شانسی یکی از مسیرهای A یا B رو انتخاب می‌کنم. بعدش صدا می‌زنم تو بیا توی غار و جلو مسیرها وایسا. (شکل ۲) حالا به مَنی که ته غار وایسادم، بگو از یکی از مسیرا بیا بیرون. من چه از مسیر A و چه از مسیر B رفته باشم، چون کلمه رمز رو می‌دونم، میتونم دَر رو باز کنم و از اون مسیری که تو میخوای بیام. مثلاً توی تصویرا، Alice از مسیر B رفته. اما Bob که اومده توی غار، به Alice میگه از مسیر A بیا. (شکل ۲) حالا Alice چون رمز رو میدونسته، در رو باز می‌کنه و از مسیر A میاد.




Dake, Zkip alibaba1, CC BY 2.5 | Dake, Zkip alibaba2, CC BY 2.5 | Dake, Zkip alibaba3, CC BY 2.5

اما خب ممکنه Bob بیاد بگه که Alice تو شانس اُوُردی! تو موقعی کـه وارد شـدی، رفتـه بـودی تـوی مسیر A. من ندیدم و بیرون غار بودم. ولی به نظرم شانسی بوده و از اول رفته بودی توی A. تـو ۵۰ درصـد احتمال داشته که از اول A رو انتخاب کرده باشی. پس ۵۰ درصد شانس داشتی که درست بشه و شده!

حالا Alice میاد میگه باشه! یه بار دیگه انجامش میدیم. بار بعد هم Alice از مسیری که Bob میخـواد میاد بیرون. اما Bob باز میگه که ایندفعـه هم شانسـی بـوده. خب اینجـا Alice میگـه من ۵۰ درصـد قبلاً شانس داشتم و حالا برای این بار هم ۵۰ درصد. حالا اگر هردوبـار شانسـی باشـه، ۵۰ درصـد * ۵۰ درصـد شانسی بوده:

50/100 * 50/100 => %25

حالا باز مثل سؤال اولی که گلوله بود. اینم تعداد بار زیاد تکرار کـنیم، احتمـال شـانس خیلی خیلی کم میشه و عملاً نشون میدیم که شانسی نبوده!

### سؤال سوم:

فرض کنین شما یه جدول سودوکو میدین به یه نفر، اون فرد میگه این حل نمیشه. امـا شـما میخـواین بدون اینکه جواب مسأله رو بهش بدین و بگین چه‌جوری حل میشه، بهش نشون بدین که حل‌شدنی هست. حالا چجور این کار رو انجام میدین؟

*Answer*: Interactive Sudoku Zero-knowledge Proof[182]

# Shamir’s secret sharing

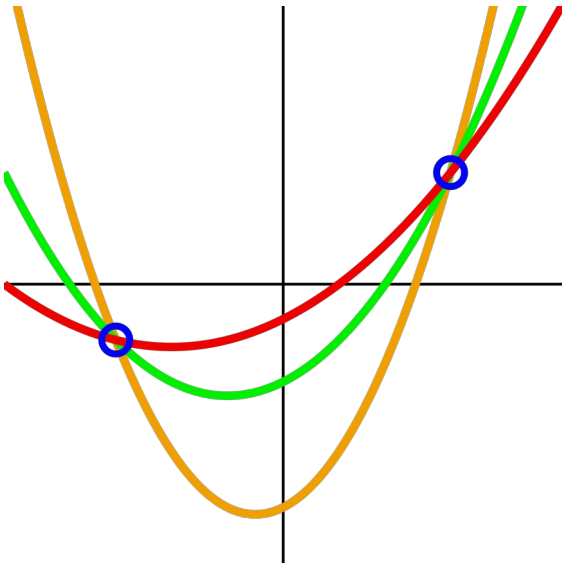
پرسش: می‌خوایم رمز جوری باشه که اگر دست مثلا ۴ نفر باشه، بشه سیستمو بازش کـرد. هـرکس بـه تنهایی نتونه. ممکنه اصلا؟

- خب سادس! کلید رو نصف می‌کنیم و بین دو نفر تقسیم کنیم که هرکسی تنها نتونه باز کنه.

+ اوه! وایسین ببینم! فرض کنین ۱۲۸ بیت باشه. ۶۴ بیت دست توعـه. ۶۴ بیت دسـت یکی دیگـه. پس نیازه صرفاً ۶۴ بیت دیگـه رو بـروت فـورس کـنی و بروت‌فـورس‌کردن ۶۴ بیت، چنـد دقیقـه بیشـتر طـول نمی‌کشـه :) تبریـک! سیسـتم رو نـاامن کـردی! نـه اینطـوری نمیشـه! بـریم سـراغ «Shamir's Secret Sharing».

خب بحث سر توابع چندجمله‌ای هست. مثلا فرض کن دو نقطه داشته باشی. با داشتن ۲ نقطـه، میشـه بی‌نهایت تابع درجه ۲ پیدا کرد که از اون ۲ نقطه عبور می‌کنن:

182 https://manishearth.github.io/blog/2016/08/10/interactive-sudoku-zero-knowledge-proof/



image[183]

اما اگر ۳ نقطه داشته باشی، یه تابع یونیک و یکتا به دست میاد.

درواقع اگر یه معادله به توان n داشته باشیم، n + 1 تا نقطه می‌خوایم که بتونیم معادله رو پیدا کنیم. (با لاگرانژ اینترپولیشن)

حالا من میخوام اگر ۴ نفر اطلاعات داشته باشن، بتونن وارد سیستم شن.

من پسوردمو یه تابع ریاضی درجه ۳ تبدیل می‌کنم. اگر ۴ نقطه کنار هم قرار بگیره (۴ نفر)، معادله درجه ۳ به صورت یکتا پیدا میشه.

خب مثلا ۶ نقطه از تابع رو بین افراد پخش می‌کنم. هرکس با داشتن یه نقطه تنها، تابع رو نمیتونه پیدا کنه. اما چون معادله درجه ۳ هست، اگر ۴ نفر باشن و نقاطو به بدن به سیستم، تابع پیدا میشه (عملا انگار پسورد برای سیستم کار می‌کنه)

هرکس به تنهایی با نقطه‌اش نمی‌تونه تابع بیابه. ولی اگر نقاط به اجماع برسن، تابع پیدا میشه.

البته این روش کاملش نیست. روش کامل رو در وبسایت زیر بخونین:

https://evervault.com/blog/shamir-secret-sharing

ساختاری که مثلاً تا وقتی ۳ نفر از ۵ نفر با هم نباشن، نتونن باز کنن.

183 Vlsergey, 3 polynomials of degree 2 through 2 points, CC BY 3.0

# توصیه‌هایی درباره رمزنگاری

همه جا رو حواستون هست؟

من می‌تونم به یه کارمندتون ۱ میلیون دلار بدم (حقوق بیش از ۱۰ سال!) و بگم کلید رو بده. اونم میده! چرا برم دستگاه و شبکه کامپیوتری بسازم برای بروت‌فورس؟

حواستون هست که کارمنداتون کار بد نکنن؟! بالاخره اونا هم آدمن و حرص و طمع پول دارن. یا ممکنه مورد تهدید قرار بگیرن. اون موقع چیکار می‌کنن؟ شرکت رقیب نمی‌تونه بخرتشون؟

حواستون هست کسی رمز رو نبره خونه یا روی میز نذاره؟

درباره این موارد بیشتر در قسمت «...» صحبت می‌کنیم. بریم صرفاً در عمق رمزنگاری:

رمزنگاری باید سالها تست و بررسی شه. هم در عمل و هم روی کاغذ. اینجاست که میگن تا وقتی دانش کافی ندارین، از همون رمزنگاری‌های استاندارد استفاده کنین. چون این‌ها سالهاست که دارن تست و بررسی میشن.

بله درسته در فینال رقابت AES، الگوریتم‌های دیگه‌ای مثل Serpent و Twofish هم حضور داشتن و مشکل امنیتی هم نداشتن. حتی در بعضی جاها عنوان میشه که روی کاغذ، Serpent مقدار security margin بیشتری داره. اما آیا باید از Serpent استفاده کرد؟

برای کسی که دانش عمیق نداره و نمی‌دونه داره چیکار می‌کنه، خیر! گفتیم رمزنگاری باید تست خودشو پس بده. باید سالها مورد بررسی قرار بگیره.

+ رمزنگاری AES (که استانداردشده Rijndael هست)، سالهاست که وجود داره و به دلیل استانداردبودن، سالها توسط رمزنگارها و ریاضی‌دان‌ها مورد بررسی قرارگرفته و بهبودیافته و عملاً تست خودشو پس داده.

+ همچنین Instruction Set (اگر نمی‌دونین چیه، اینه که به جای اینکه دستورات software ای بدیم که تبدیل بشه به کد اسمبلی و کد ماشین عادی که ماشین می‌فهمه، عملاً یه سری دستورات برای پردازنده تعریف می‌کنیم که بتونه AES رو سخت‌افزاری implement کنن.) هاش در اکثر پردازنده‌های مدرن وجود داره و به دلیل وجودش، سرعت انجامش میره بالا. همچنین به همین دلیل، یه سری حمله‌های سخت‌افزاری و side-channel attack ها رو رو کاهش میده:

"The AES instructions provide important security benefits. By running in data-independent time and not using tables, they help in eliminating the major timing and cache-based attacks that threaten table-based software implementations of AES. In addition, they make AES simple to implement, with reduced code size, which helps reducing the risk of inadvertent introduction of security flaws, such as difficult-to-detect side channel leaks."[184]

184 https://www.intel.com/content/www/us/en/developer/articles/tool/intel-advanced-encryption-standard-aes-instructions-set.html

+ همچنین لایبری‌های انجام AES، بروز و مدرن‌تر و تست‌شده‌تر هستن. ولی لایبری‌های مورد استفاده برای رمزنگاری‌های دیگه، معمولاً قدیمی هستن که تست زیادی هم روشون انجام نشده و مستعد وجود باگ هستن! چه بسا برای اون زبونی که دارین استفاده می‌کنین، کدی وجود نداشته باشه برای رمزنگاری.

- من ایملیمنت Serpent رو توی C دیدم. می‌تونم همونو توی جاوا پیاده‌سازی کنم؟
+ خیر! ممکنه فکر کنین عیناً دارین شبیه اون چیز اصلی پیش میرین ولی ممکنه یه جایی یه اشتباه ریزی کنین و یا تفاوت زمانی باعث بازشدن یه سری دَر برای حملات جدید شه.

حواستون باشه که حتماً داک لایبرری که استفاده می‌کنین رو بخونین! این خیلی مهمه! مثلاً توی داکیومنت میگن که چه‌جور باید فانکشن‌ها استفاده بشن که امن باشه یا فلان فانکشن وجود داره ولی امن نیست. نمونه:

"Per bcrypt implementation, only the first 72 bytes of a string are used. Any extra bytes are ignored when matching passwords. Note that this is not the first 72 *characters*. It is possible for a string to contain less than 72 characters, while taking up more than 72 bytes (e.g. a UTF-8 encoded string containing emojis)."[185]
مثلاً اینجا گفته که اگر چیزی که می‌خواین هش کنین بزرگ‌تر از ۷۲ بایته، صرفاً ۷۲ بایت اول استفاده میشه.[۱۸۶] [۱۸۷]

یا حتی عنوان کرده که فلان فانکشن در برابر timing attack مقاوم نیست:

"... the comparison function is ***not*** time safe."[188]

یا مثلاً توی داکیومنت «GNUPG» نوشته شده:

"Blowfish should not be used to encrypt files larger than 4Gb in size."[189] **(same as 3DES)**

همونطور که اشاره کردیم، «*ساخت سرویس و حل ارور مربوط به کامپیوتر رو تجربی انجام ندین!*» وگرنه به مشکلات امنیتی بر می‌خورین! روی داک پیش برین.

---

185 https://www.npmjs.com/package/bcrypt

۱۸۶ یه راه تشخیص اینکه چه هشی استفاده شده، همینه که مثلاً من رفتم تو دیسکورد و خواستم ثبت نام کنم. بعد گفتم که یه پسورد ۱۰۰ کرکتری وارد کردم. دیدم نوشت که تا ۷۲ کرکتر ساپورت میشه و بیشتر نمیتونی. خب پس احتمالاً داره از Bcrypt استفاده میکنه :)

۱۸۷ یه نکته برای افرادی که مشکلات امنیتی رو پیدا می‌کنن. شاید سیستمی که دارین بررسی می‌کنین، از Bcrypt استفاده می‌کنه و اگر بیش از ۷۲ بایت بهش بدیم، کاتش می‌کنه و بقیشو می‌ریزه بیرون و صرفاً ۷۲ بایت اول رو رمز می‌کنه. این یه ضعف امنیتیه!

188 https://www.npmjs.com/package/bcrypt#a-note-on-timing-attacks

189 https://gnupg.org/faq/gnupg-faq.html#define_fish

یه نکته دیگه اینکه لزوماً رمزنگاری و هش جدید، دلیل بر بهتربودن نیست. بلکه باید تست خودشو پس بده. یعنی اتفاقاً در این حوزه میگن که رمزنگاری و هش باید یه مدت زمانی ازش گذشته باشه و خیلی جدید نباشه که تست‌های خودشو پس داده باشه.

همچنین توصیه‌های دیگه‌ای رو در قسمت «*یک نکته مهم در مورد نرم‌افزار آزاد یا اپن‌سورس که ممکنه اشتباه درموردشون فکر می‌کردین*» بهتون گفتم.

توصیه‌های بیشتر رو از آدمای معتبر بگیرین.
- خب اینقدر گفتی که از منابع معتبر کمک بگیریم. خب بگو چیا خوبن؟
+ سازمان استاندارد NIST و سازمان BSI آلمان و افراد معروف و دانشگاه‌های معتبر. افراد زیر، افرادی هستن که خیلی دانش دارن و هرکدوم خالق بزرگترین رمزنگاری‌ها و پروتکل‌های دنیا هستن! به حرفای این افراد می‌تونین اعتماد خیلی بیشتری نسبت به بقیه داشته باشین:

Bruce Schneier - Ralph Merkle – Martin Hellman - Whitfield Diffie – Ronlad L. Rivest – Adi Shamir – Leonard Adleman - Vincent Rijmen - Joan Daemen - Taher Elgamal – Ross Anderson

## از چه نوع رمزنگاری و چه طول کلیدی استفاده کنیم؟

مهمترین چیز اینه که بدونیم این اطلاعات رو داریم **برای چی** و **از دید و فهمیدن چه کسانی و** برای **چه مدت** رمز می‌کنیم؟

باید بدونیم Threat model مون چیه؟

بله اگر صرفاً برای اینکه از دید یه فرد عادی و بدون هیچ دانشی داریم یه چیزی رو مخفی می‌کنیم، رمزنگاری‌های ساده هم اوکین! حتی تا گاهی رمزنگاری‌های خودساخته![۱۹۰] اما اگر داده‌هامون حساسن یا طرف حسابمون جاهای دیگس، از رمزنگاری استاندارد استفاده کنین! رمزنگاری‌هایی که جلوی فهمیدن مطلب توسط سازمان امنیتی رو هم می‌گیرن!

نکته بعدی زمانه. یه سری رمزنگاری‌ها (یا طول کلیدشون) شاید الآن خوب باشن، اما برای نگهداری داده برای مدت خیلی طولانی (مثلاً ۱۰ سال) خوب نیستن!

خب به صورت کلی (در پایین صفحه آموزش‌های خوبی که از هرکدوم پیدا کردم گذاشتم):

- Symmetric-key

۱۹۰ البته اگر اون فرد از کسی یا فرد متخصصی یا حتی هوش مصنوعی کمک نگیره که بخواد اون رمز رو بشکونه.

- Block cipher: AES[191] – Twofish[192] – Serpent – RC6 – Camellia – Skipjack – DES – 3DES – Blowfish
- Stream cipher: ChaCha20 – RC4 – Salsa20

- Asymmetric-key
  - Factoring Problem: RSA
  - Discrete Logarithm: ECC

برای Symmetric، رمزنگاری «AES» کـه اسـتانداردە و امنـه. امـا رمزنگاری‌هـای «ChaCha20» و «Twofish» و «Serpent» هم اوکین.

برای asymmetric هم RSA و ECC هردو اگر درست استفاده شن، امنن.

رمزنگاری‌های امن Symmetric:

اگر Hardware Acceleration وجود داشـته باشـه، انتخـاب اولتـون AES. ترجیحـاً مـد GCM کـه authentication هم انجام میده.

اگر Hardware Acceleration وجود نداشته باشه، توصیه‌ای ندارم. چرا؟ چون متخصص نیستم.

بعضـی دسـتگاه‌ها قـدیمین و از سـخت‌افزارهای قـدیمی پشـتیبانی می‌کنن کـه از AES بـه صـورت سخت‌افزاری پشتیبانی نمی‌کنن.

وقتی Hardware Acceleration وجود نداشته باشه، AES به یه سری از side-channel attack ها حساس میشه. همچنین سرعت هم میاد پایین. چون instruction هاش به صورت سخت‌افزاری تعبیـه نشدن.

اینجـا بـود کـه سـر و کلـه الگـوریتمی جدیـد پیـدا شـد. ChaCha20 (بـه همـراه Poly1305 بـرای authentication)

چون صرفاً از addition و rotation و XOR اسـتفاده می‌کنـه و همشـون constant-time هسـتن، نسبت به timing attack (حداقل خیلیاش) مقاومه (صرفاً تایمینگ! نه لزوماً بقیه side-channel ها!):

“ChaCha20 is immune to padding-oracle attacks, such as the Lucky13, which affect CBC mode as used in TLS. By design, ChaCha20 is also immune to timing attacks.”[193]

---

191 How does AES work? → https://web.archive.org/web/20090923200756/http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

192 Book: “The Twofish Encryption Algorithm” By “Bruce Schneier”

193 https://security.googleblog.com/2014/04/speeding-up-and-strengthening-https.html

حواستون باشه که صرفاً در برابر timing attack مقاومه. نه همه side-channel attack ها:

Don’t Fall into a Trap: Physical Side-Channel Analysis of ChaCha20-Poly1305 → https://past.date-conference.com/proceedings-archive/2017/pdf/7031.pdf

# • Security bits

- چطور یه رمزنگاری symmetric می‌شکنه؟

+ ساده‌ترین روش برای شکوندن یه رمزنگاری symmetric امتحان تمام حالات (brute-force) کلید هست. یعنی اگر کلید مثلاً ۱۰ بیته، من نیازه ۲ به توان ۱۰ حالت رو تست کنم تا مطمئن شم رمزو به دست میارم.

پس درواقع وقتی رمزنگاری ما ۲۵۶ بیته، یعنی از لحاظ تئوری، ۲ به توان ۲۵۶ حالت باید چک کنیم تا رمز به دست بیاد.

وقتی برای brute-force کردن یه الگوریتمی، $2^{256}$ حالت نیازه. ما می‌گیم تعداد «Security Bits» هاش، ۲۵۶ تاست. خب قاعدتاً از لحاظ تئوری هرچی security bit ها بیشتر باشه، بهتره.

- یکی از اشتباهات رایج اینه که افراد فکر می‌کنن که اگر از RSA-2048 برن به RSA-4096، سختی رمزگشایی دوبرابر میشه. خیر! رمزنگاری symmetric با asymmetric فرق داره! درواقع ساده‌ترین راه برای سنجش کلی میزان امنیت الگوریتم‌های امن دربرابر brute-force،[۱۹۴] سنجش سکیوریتی بیت‌هاست.

مقدار security bit هر الگوریتم رو می‌تونین از طریق NIST ببینین.[۱۹۵]

```
RSA 1024 => 80 security bits
RSA 2048 => 112 security bits
RSA 3072 => 128 security bits
RSA 4096 => 140 security bits
```

با تبدیل RSA 1024 به RSA 2048 که ۴۰ درصد security bit ها زیاد شدن، شکستنش چقدر سخت‌تر میشه؟

```
2^(122)/2^80 = 2^(122-80) = 4398046511104
```

با تبدیل RSA 2048 به RSA 4096:

```
2^(140-112) = 268435456
```

دیدین؟ افزایش سختی کمتر شد. درسته باز سخت‌تر میشه، اما افزایش سختیش، کمتره و زیاد اونقدر که انتظار داشتیم سخن نشد!

چرا؟ چون درواقع تعداد بیتای RSA (مثلاً ۴۰۹۶ و ۳۰۷۲)، سکیوریتی بیت نیستن! بلکه نمایشگر اون عدد توی مراحل RSA هستن. درواقع امنیت RSA به یافتن اعداد اول پشت عددمون بستگی داره. وقتی بیت‌ها بیشتر میشن، درسته عدد بزرگ‌تر میشه، اما یه سری از اون عددا اصلاً اول نیستن!

بعدشم امنیت RSA به امتحان تمام حالات کلیدا نیست! به فاکتورگیری هست!

---

۱۹۴ بحث حملات دیگه مطرح نیست!

195 NIST Special Publication 800-57 Part 1, Revision 5 → Recommendation for Key Management → https://doi.org/10.6028/NIST.SP.800-57pt1r5 (Page 54 → "Table 2: Comparable security strengths of symmetric block cipher and asymmetric-key algorithms")

از ۲۰۴۸ به بعد، اون افزایشه کمتر میشه. یعنی تا یه حدی این افزایش امنیت خیلی زیاد و خوب رخ میداد، اما از ۲۰۴۸ به بعد، ما داریم طول رو خیلی زیاد می‌کنیم، اما اون افزایش زیاد رخ نمیده!

برای همین برای بهینه‌تر شدن و سریع‌تر شدن برنامه‌ها، معمولاً از ۴۰۹۶ استفاده نمیکنن و به همون ۳۰۷۲ بسنده میکنن. چون میخوان برنامه سریع باشه و شما معطل نشی. چون اگر معطل شی، میگی این برنامه چقدر کنده حوصلمون رو سر برد. بریم یه برنامه دیگه![۱۹۶] در ضمن که RSA 3072 امنه و فعلاًها شکسته نمیشه. طبق گفته NIST، حداقل تا سال ۲۰۳۰ امنه. پس برای همین میگن درسته که RSA 4096 امن‌تره، اما خب RSA 3072 هم در کوتاه مدت شکسته نمیشه. پس وقتی نمی‌خوایم داده‌ای رو برای مدت خیلی زیادی نگه‌داریم، برای چی بریم سراغ ۴۰۹۶ که کندتر و اینا‌س؟ استفاده و نگهداری طولانی مدت هم نداریم که. همین برای یه استفاده کوتاه هست. پس همین ۳۰۷۲ کفایت میکنه!

## اندازه طول کلید چقدر باشه؟

طبق اعلام NIST،[۱۹۷] تا سال ۲۰۳۰، حداقل ۱۱۲ سکیوریتی بیت باید باشه (یعنی RSA-2048) و برای نگه‌داری برای بعد سال ۲۰۳۰، حداقل ۱۲۸ بیت.

طبق اعلام BSI،[۱۹۸] از سال ۲۰۲۴، باید ۱۲۰ بیت سکیوریتی داره باشه. یعنی AES-128 به بعد و RSA-3072 به بعد.

بقیه recommendation ها رو می‌تونین از وبسایت «Keylength»[۱۹۹] بگیرین.[۲۰۰] (لطفاً اگر دارین یه سیستمی رو پیاده‌سازی می‌کنین، صرفاً به این تعداد بیتا بسنده نکنین! برید کامل داکیومنت NIST و BSI رو بخونین!)[۲۰۱]

## نکات، سؤالات، تفکرات و اشتباهات رایج:

- آیا brute-force کردن ۲۵۶ بیت security bit ممکنه؟

+ بر اساس فیزیک و ترمودینامیک خیر! درواقع ما وقتی می‌گیم این رمزنگاری‌ها نمی‌شکنن، منظورمون اینه اونقدر اونقدر زمان و انرژی نیازه که حتی سوپرکامپیوترها هم در طی حتی میلیون‌ها سال هم نمی‌تونن

---

۱۹۶ البته تفاوت سرعتی۴۰۹۶ به جای ۳۰۷۲ اونقدر چندان خیلی زیاد نیست که بگیم واو! ولی خب توی دستگاه‌های قدیمی و کم‌توان خودشو بهتر شاید نشون بده.

197 NIST Special Publication 800-57 Part 1, Revision 5 → Recommendation for Key Management → https://doi.org/10.6028/NIST.SP.800-57pt1r5 (Page 59 → "Table 4: Seucrity Strength time frames")

198 BSI – Technical Guideline of "Cryptographic Mechanisms: Recommendations and Key Lengths", Version: 2023-01: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-1.pdf?__blob=publicationFile

199 https://www.keylength.com/en

۲۰۰ البته بعضی جاهاش انگار بروز نیست. مثلاً داکیومنت BSI ورژن ۲۰۲۳ هم جدید اومده ولی سایت آپدیت نکرده خودشو. ولی سایت خوبیه که خودتون بعدش برین سرچ کنین.

۲۰۱ گاهی بعضی شرکتا مجبورن که از توصیه‌های استاندارد تبعیت کنن.

- چرا؟

+ چون اگر نکنن، بسته به قوانین اون کشور، ممکنه در اثر لورفتن داده‌ها جریمه شن. چون بهشون می‌گن که ما گفتیم اینو رعایت کن. حالا که داده لو رفته باید جواب پس بدی. چون رعایت نکردی اینطور شده! (گاهی حتی اگر هم داده لو نره، بهتون ایراد میگیرن که چرا از استاندارد تبعیت نکردی! پس برای این به قانون کشورتون مراجعه کنین.)

بشکونن! اونهمه زمان نیازه! درواقع به این دلیل می‌گیم نمی‌شکنن. برای درک بهتر وبلاگ Bruce Schneier رو بخونین که قشنگ توضیح داده که بدونین تست ۲ به توان ۲۵۶ حالت، عملاً غیر ممکنه.[۲۰۲]

درواقع brute-force راه حل نیست! برای همین رمزنگارها می‌گردن دنبال حملات پیچیده‌ای که زودتر از زمان brute-force بشکونتشون.

- وای یه حمله به AES پیدا شد که ۳۲ برابر سریع‌تر کار رو انجام میده!

+ نترسین! هدف در حمله‌ها، پیداکردن یه چیزی سریع‌تر از brute-force هست. خیلی وقتا که می‌گن حمله به فلان الگوریتم یا «فلان چیز is broken»، منظورشون حمله‌ای سریع‌تر از brute-force هست که لزوماً هم باعث ناامن‌شدن نیست![۲۰۳] ممکنه کاملاً هم غیرعملی باشه در واقعیت. مثلاً از ۲۵۶ رسونده باشه به ۲۵۰. (خب ۳۲ برابر سریع‌تر شده!) خب هنوزم کاملاً غیرعملی هست، اما حمله حساب میشه.

- خب اگر خیلی از این حمله‌ها تأثیر عملی نمی‌ذارن، چرا پس اینقدر بولد میشن؟

یه جمله خیلی معروفی هست که میگه:

"Attacks always get better; They never get worse"

درواقع همه این حمله‌های غیرعملی، پایه و شالوده‌ای برای حمله‌های عملی خواهند بود.

درواقع درسته شاید الآن اثر نذاره ولی محققا هی تلاش می‌کنن حمله رو بهینه‌تر و بهتر کنن که اثر بذاره. درواقع اینا پایه و شالوده حمله‌های دیگه هستن. خیلی وقتا وقتی یه محقق یه روش ناکاملی رو کشف می‌کنه، بقیه میان بررسیش می‌کنن و کامل‌ترش می‌کنن و یه روش کامل و عملی رو منتشر می‌کنن. برای همین رمزنگارا، نتایج و تحقیقای بقیه رو می‌خونن.

- الگوریتمت ۱۰۰۰ برابر ضعیف‌تر شده. چقدر بد!

+ نه لزوماً! فرض کنین اینطوری شده:

$2^{256} \rightarrow 2^{246}$

بله ۱۰۲۴ برابر ضعیف‌تر شده ولی هنوز اونقدر تایم کامپلیکسیتیش زیاده که غیرقابل شکستنه!

- هر بیت به کلید اضافه شه، الگوریتم ۱۰۰ درصد قوی‌تر میشه.

+ آره این یکی درسته! چیز عجیبی نیست! هر بیت اضافه شه، انگار یه ضربدر ۲ (۱۰۰ درصد افزایش) رو برای brute-force داریم.

- اگر به کلید یه بیت اضافه شه، سرعت نصف میشه!

+ نه!

202 https://www.schneier.com/blog/archives/2009/09/the_doghouse_cr.html
203 According to https://www.schneier.com/crypto-gram/archives/2002/0915.html#1

"Threefish, the block cipher inside Skein, encrypts data at 7.6 clock cycles/byte with a 256-bit key, 6.1 clock cycles/byte with a 512-bit key, and 6.5 clock cycles/byte with a 1024-bit key."[204] -Bruce Schneier

+ برای اشتباهاتی که در ساخت کلید رمزنگاری ایجاد میشه، ممکنه به شدت قدرت رمزنگاری رو کاهش بده. (مراجعه شـود بـه «8.1Generating keys» از کتـاب «Applied Cryptography» از «Bruce Schneier» و همچنین داکیومنت‌های بروزتر استاندارد.

یه سناریو: فرض کنین کلید یه سری از فایلای شرکت دست فقط یه نفره. خب فرض کنین فردا اون فرد مرد. خب تبریک میگم! فایلاتون برای همیشه از بین رفت![۲۰۵]

+ کلیدایی که برای امضای دیجتال امضا میشن، نیازه بعد یه مدت منقضی شن. آیا رعایتش می‌کنین؟

+ کلید رو جای اشتباهی که ذخیره نمی‌کنین؟!

+ آیا حواستون هست که اگر کلید رو توی یه فایلی ذخیره کنین، اون فایل رو اگر پاک کنین، بازم اثرش روی HDD هست؟ آیا با روش‌های پاک‌کردن امن (که غیرقابل بازیابی باشن)، آشنایی دارین؟

+ آیا از shadow file ها مطلع هستین؟ اطلاعات حساس ممکنه توی هارد باقی بمونن!

204https://www.schneier.com/blog/archives/2009/09/the_doghouse_cr.html

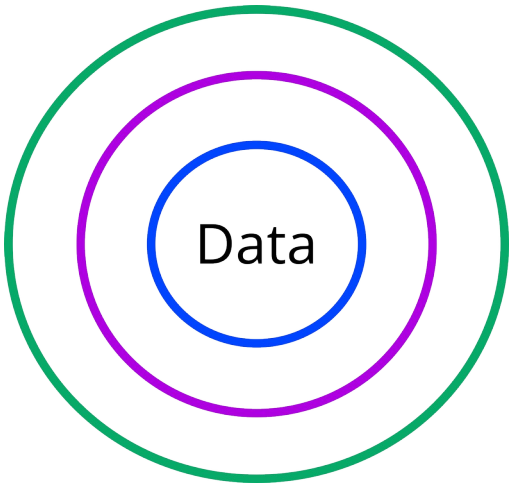
۲۰۵ مگر اون نرم‌افزار رمزنگاری اونقدر بد باشه که بشه رمز رو شکوند. خب در این حالتم خب باید بگم سازمانتون ناامنه! چون از نرم‌افزار بدی استفاده کردین.

## Multiple encryption / cascade encryption / cascade ciphering / superencryption / superencipherment

ببینین تا الآن دونستیم که اگر از رمزنگاری‌های استاندارد و امن **به طور درستی**[۲۰۶] استفاده کنیم، امنیم. اما اما! یادتونه که توی رمزنگاری RSA گفتیم که **فعلاً** الگوریتمی پیدا نشده که بتونه در زمان معقول بشکونتش؟!

اثبات نشده هیچ‌وقت پیدا نمیشه. بلکه فعلاً پیدا نشده! درواقع تمام رمزنگاری‌هایی که گفتیم امنن، بر اساس دانش ریاضیاتی امروزمون امنن. ممکنه بعداً با پیشرفت دانش ریاضیات، این رمزنگاری‌ها بشکنن. (ولی این چیز به همین سادگی رخ نمیده. باید پیشرفتی در دانش ریاضیات و اینا ایجاد شه یا افرادی به نکات ریز پی ببرن و بتونن بشکوننش.)

اینجاست که ما اگر بخوایم یه چیزی رو مدت خیلی طولانی (مثلاً برای ۲۰ سال) نگه‌داریم، می‌تونیم از چند لایه رمزنگاری استفاده کنیم. یعنی اگر یه وقت یه لایه شکست، لایه بعدی باقی بمونه:




توضیح: من اول با لایه آبی رمز می‌کنم، بعد بنفش، بعد سبز.[۲۰۷]

اگر یه وقت کسی خارجی‌ترین لایه رو بشکونه، بازم دو لایه دیگه رمزنگاری هست. دوباره باید بشینه اونا رو دونه دونه بشکونه تا بتونه به دیتای اصلی ما دسترسی داشته باشه. مثلاً:

```
AES(Twofish(Serpent(plain_text)))
```

یعنی بعد شکستن AES به Twofish می‌رسیم و بعدش باید Twofish رو بشکنیم. وقتی شکوندیمش، در نهایت به Serpent می‌رسیم. اینم باید بشکونیم تا به متن اصلی برسیم! البته تمام این‌ها امنن. (البته فعلاً!) هر سه تاشون فینالیست‌های انتخاب استاندارد رمزنگاری پیشرفته یا Advanced Encryption Standard بودن.

این اتفاق برای داده‌های خیلی حساس و محرمانه شرکتا هم انجام میشه.

---

۲۰۶ مثل implementation درست، درست‌نگه‌داشتن کلید، اعدادمون رندوم باشه و... .
۲۰۷ سبز خارجی‌ترین لایه هست. (برای کسایی که کوررنگی دارن)

حتی شاید NSA و بقیه سازمان‌ها الآن نتونن یه رمزنگاری خاص رو بشکنن، اما این داده‌های رمزشده رو در یه سری محل ذخیره بزرگ نگه می‌دارن که بعداً که این رمزنگاری‌ها شکسته شدن، بازشون کنن. البته خب این بیشتر برای جاها و افرادای خاص هست. مثلاً پی بردن به اسناد دولتی کشورای دیگه.[۲۰۸] [۲۰۹]

*مثال دیگه:* من میخوام یه پیام یا یه فایلی برای یکی طبق یه پروتکی بفرستم که امن نیست. مثلاً رمزنگاریش DES هست که امن نیست. خب من برای اینکه این فایل و یا پیامم دست کسی دیگه‌ای نیفته، میام خودم قبل فرستادن با اون پروتکل ناامن، رمز میکنم. اینطور:

```
DES(AES(Plain_Text))
```

اینطوری اگر کسی بتونه DES رو باز کنه (که می‌تونه)، میخوره به AES که باز نمیشه! یا حداقل فعلاً میدونیم اگر درست پیاده‌سازی شه، نمیشکنه.

*مثال دیگه:* فرض کنین شما مجبورین که یه فایلی رو از طریق Gmail یا تلگرم برای کسی بفرستین. در حالت عادی تضمین نمیشه که صرفاً شما و طرف مقابل به فایل و پیاما دسترسی داشته باشین. درواقع خود جیمیل به ایمیلاتون دسترسی داره! برای همین می‌تونین اون فایل خاص رو رمز کنین و در طرف مقابل دوستتون رمزگشایی کنه. اینطوری جیمیل هم نمی‌تونه بفهمه چی بود.
[۲۱۰]

*مثال دیگه:* استفاده از VPN برای وصل‌شدن به وبسایتای HTTPS هم نمونه‌ای از multiple-layer encryption هست. (بعداً به نحوه کار VPN می‌رسیم.)

*تذکر!* همیشه سعی کنین لایه بیرون، رمزنگاری استاندارد باشه. یعنی AES (نه Twofish). چون امتحان خودشو بیشتر پس داده.

*تذکر!* جوگیر نشین! بعضیا صرفاً فکر می‌کنن لایه بیشتر یعنی بهتربودن و اصلاً توجهی به تذکراتی که دادیم (implementation درست، ذخیره درست و نگه‌داری و استفاده درست از کلید رمزنگاری، استفاده از اعداد واقعاً رندوم، iv درست، padding درست و...) رو رعایت نمی‌کنن. ولی به جاش میرن ده لایه رمز می‌کنن! نه! شما اول یه implementaion رو درست انجام بده، بعد به لایه‌های بیشتر فکر کن. وگرنه، اصلاً به لایه‌های بیشتر فکر نکن! اول implementation رو درست کن!

*تذکر!* از الگوریتم‌های امن استفاده کنین. چیزایی مثل Serpent – Twofish – AES. لطفاً از الگوریتم‌های الکی استفاده نکنین. بعضیاشون آسیب‌پذیرن و بیشتر ضرر می‌زنن و همچنین صرفاً دارین

---

۲۰۸ اینجاست که اهمیت multiple encryption مشخص میشه...

209 https://en.wikipedia.org/wiki/Tempora
https://en.wikipedia.org/wiki/Utah_Data_Center

۲۱۰ البته تبادل رمز در این کانال‌ها، یه خرده برای پرایوسی و ناشناسی خوب نیست. چون میگن لابد یه چیزی برای پنهون‌کردن داره که کل پیاماش رمزیه!

implementaion و نگهداری کلید و اینـاتون رو سـخت می‌کـنین. (یـا حـتی جـا بـرای حملات side-channel بیشتری باز می‌کنین.)

*تذکر!* از یه کلید برای تمام الگوریتم‌ها استفاده نکنین!

*تذکر!* نمیدونم آیا حمله‌ای هست که به multiple-layer encryption زده بشه یا نه؟! پس لطفاً قبل از استفاده ازش، از افرادی که واقعاً رمزنگار هستن کمک بخواین. (سرچ کنین)

مثلاً نرم‌افزار TrueCrypt که الآن فورک‌هایی ازش مثل VeraCrypt[۲۱۱] وجود دارن، از سـه رمزنگـاری برای رمزکردن درایو ذخیره‌سازی مثل هارد یا HDD و یا درایوای ذخیره‌سـازی دیگـه مثـل SSD اسـتفاده میکرد.[۲۱۲]

- اگر دوبار با AES-256 رمز کنم، چقدر بهتر میشه؟

خب برای هر AES-256، نیاز به چک ۲ به توان ۲۵۶ حالت هست (brute-force). پس درواقع داریم:

$$2^{256}+2^{256}=2\times2^{256}=2^{257}$$

درواقع اونقدر که انتظار داریم سـخت‌تر نمیشـه. درواقـع اسـتفاده از multiple-layer encryption، زمانی معنا میده که از الگوریتم‌های متفاوتی استفاده شه که اگر در آینده یه اشکالی در AES به وجود اومد، بقیه الگوریتم‌ها پابرجا بمونن.[۲۱۳] وگرنه که brute-force کردن ۲ به توان ۲۵۶ حالت ممکن نیسـت! (قبلاً از لحاظ فیزیک نشون دادیم که نمیشه!)

## More:

+ “Chapter 15: Combining Block Ciphers” from “Applied Cryptography” by “Bruce Schneier”

+ “On the security of multiple encryption” by “Martin Hellman” and “Ralph Merkle”

+ Multiple Encryption[214] by “Matthew Green”

+ Notes on Multiple Encryption and Content Filtering[215] (بخون)

---

211 https://tails.boum.org/doc/encryption_and_privacy/veracrypt/index.en.html

۲۱۲ توجه کنین که SSD هارد نیست! SSD خودش یه نوع دیگه از درایوهای ذخیره‌سازی هست. هارد یعنی HDD. پس لطفاً نگین هارد SSD! کلمه هارد SSD، کلمه‌ای غلط هست.

۲۱۳ هرچند گاهی وقتی یه **روش** برای حمله کشف میشه، به دلیل ساختار مشابهشون، این امکان وجود داره که بقیه الگوریتم‌ها هم تحت تأثیر قرار بگیرن.

214 https://blog.cryptographyengineering.com/2012/02/02/multiple-encryption/

215 https://educatedguesswork.org/posts/multiple_encryption/

## • چجور رمزنگار بشیم؟

+ Memo to the Amateur Cipher Designer[216]

+ So, You Want to be a Cryptographer[217] (Old but good)

+ "The code-breakers, The comprehensive History of Secret Communication from Ancient Times to the Internet" by "David Kahn"

+ Book: "Applied Cryptography" by "Bruce Schneier" (Second Edition)

+ "Cryptography I"[218] By "Stanford School of Engineering" (کلاس)

کتاب‌های دیگه‌ای هم هستن که نمی‌دونم دربارشون ولی معروفن:

+ "Serious Cryptography" by "Jean-Philippe Aumasson"

+ "Crypto101"[219] by "Laurens Van Houtven (lvh)" *(Free!)*

+ "The Joy of Cryptography"[220] by "Mike Rosulek" *(Free!)*

# سوالاتم:

۱- چرا md5 فقط ۱۲۸ بیته؟ خب چرا ۲۵۶ طراحیش نکرد ronald rivest؟ خب collistion کمتر و دیرتر رخ میداد! (به خاطر اینکه دیزاین نشده بود برای عدم کالیژن؟ فقط برای کامپرس‌کردن بود؟) یا بـه خاطر اینکه بلاک سایفر ها ۱۲۸ بیتی بودن و طراحیش ساده‌تر بود؟!

- نمی‌دونست ممکنه کالیژن رخ بده. کامپیوترها قوی نبودن که بدونه کالیژن رخ میده.

+ خب همون سالها SHA1 اومد که ۱۶۰ بیت بود!

۲- هش اگر ۲۵۶ بیتی باشه، ۲ به توان ۲۵۶ حالت خروجی داره. اما اگر ۲ به توان ۱۲۸ حـالت ورودی بدیم، کایژن پیدا میشه. این معتبره. همه جا میگن و از کتاب applied cryptography نقل میشه:

"Finding a message that hashes to a given hash value would require hashing 2^M random messages. Finding two messages that hash to the same value would only require hashing 2^(M/2) random messages."

216 https://www.schneier.com/crypto-gram/archives/1998/1015.html#cipherdesign

217 https://www.schneier.com/crypto-gram/archives/1999/1015.html#SoYouWanttobeaCryptographer

218 https://online.stanford.edu/courses/soe-y0001-cryptography-i

219 https://www.crypto101.io/

220 https://joyofcryptography.com/

"For example, if you want to drop the odds of someone Breaking your system to less than 1 in 2^80 use a 160-bit one way hash function."

https://en.wikipedia.org/wiki/Birthday_problem#Approximations
Second one:
https://en.wikipedia.org/wiki/Birthday_problem#Probability_of_a_shared_birthday_(collision)

کد پایتونش هم زدم و نتیجه[۲۲۱]:

*Code:*

```
from decimal import Decimal

def f2(n, d):  # second formula
    n = Decimal(n)
    d = Decimal(d)
    base = (d-1)/d
    exp_ = (n*(n-1))/2
    return (1-base**exp_)

print(f2(2**32, 2**64))
```

*Output:*

```
0.39346934008095526304284536 34
```

خب سوال من اینه چطوری؟ طبق birthday problem، اگر نصف تعداد بیتا امتحان کنیم، به احتمال ۰.۳۹ کالیژن رخ میده. اما ۰.۳۹ که کمه! چرا پس میگن نصف بیتا! کمتر از ۰.۵ که اصلا کافی نیست!
البته اگر ۲ به توان ۳۳ بدیم، مقدار زیر به دست میاد که نشون میده که به احتمال خیلی زیاد کالیژن داریم:

```
0.864664716610671151522204 9937
```

۲۲۱ از «Decimal» استفاده کردم که دقت اعداد اعشاری بره بالا.