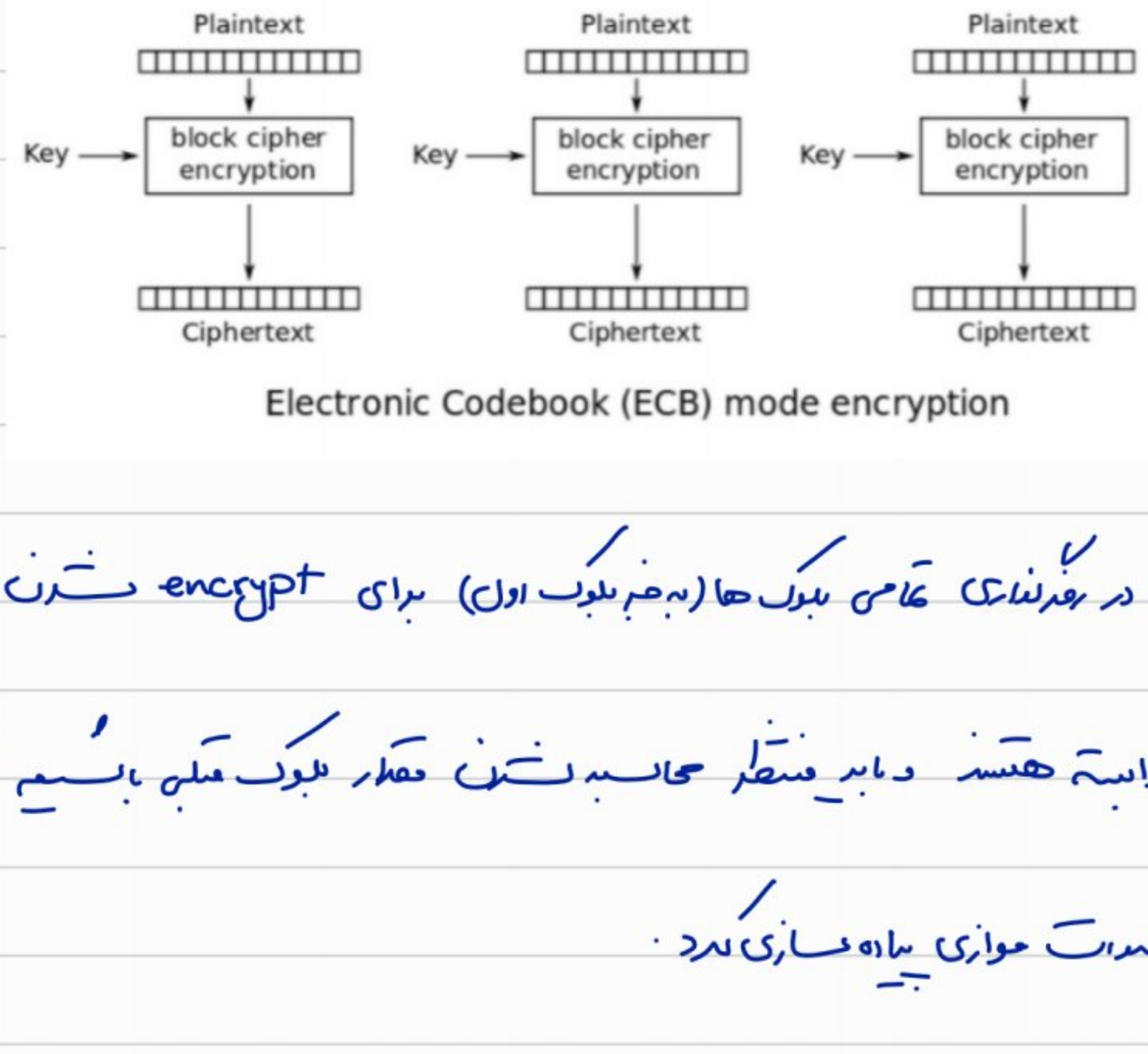
سوبل عالمی هلق « تمرین سری اول امنیت » ۹۸۵۳۶۱۰۴۷

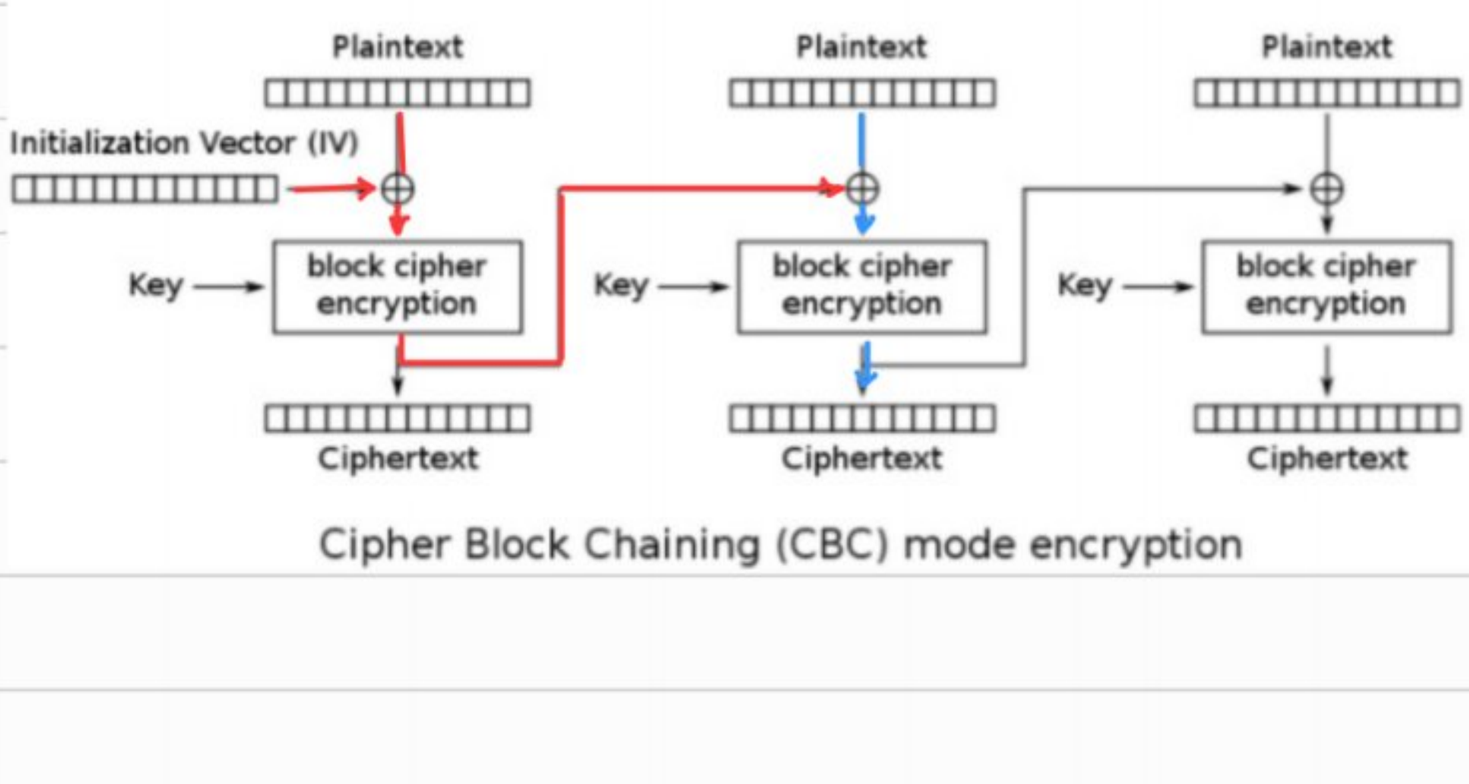
① درمورد مدهای کاری OFB و CBC و ECB ترجیح دهید که آیا می‌توان عملیات رمزنگاری و رمزگشایی را به صورت موازی انجام داد یا نه .

ECB : از آنجایی که در این مدکاری تمامی بلوک ها از همدیگر مستقل می باشند و هیچ فیدبکی از یکدیگر نمی گیرند می توان همه بلوک ها را به صورت مستقل و موازی رمزنگاری و رمزگشایی کرد .



Electronic Codebook (ECB) mode encryption

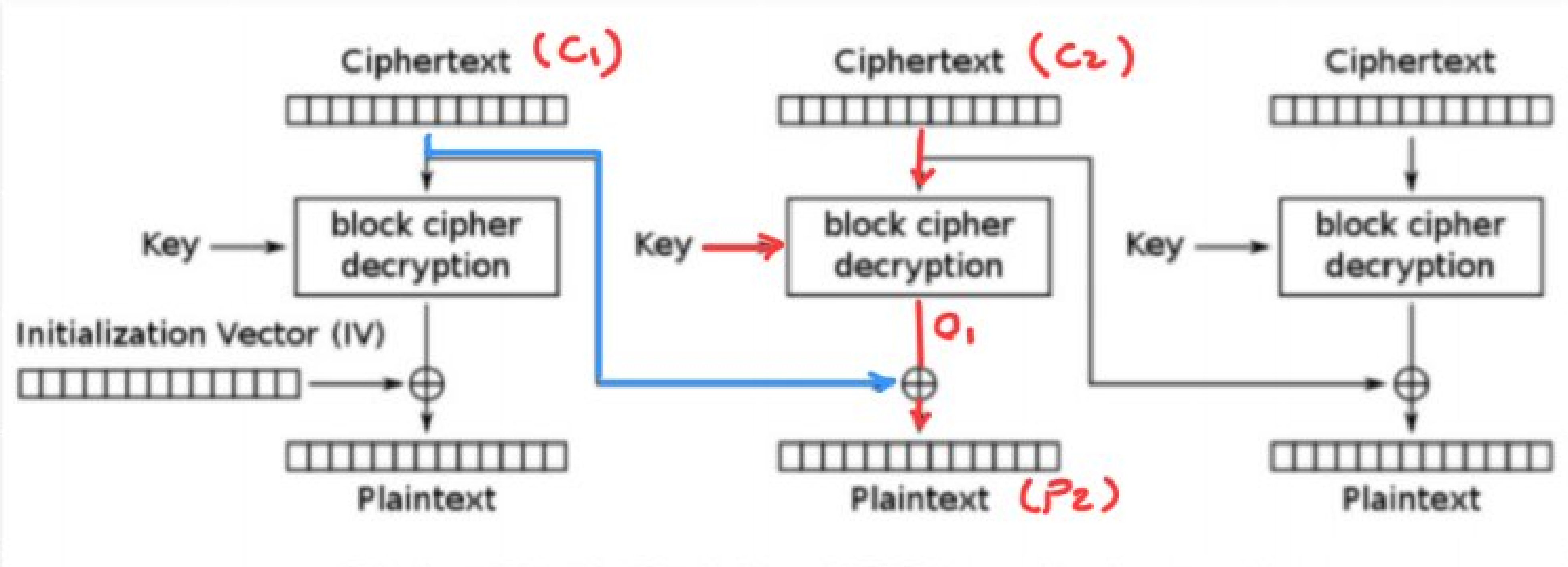
CBC : در این مدکاری در رمزنگاری تمامی بلوک ها (به جز بلوک اول) برای encrypt شدن به نتایج حاصل از بلوک های قبل از خود وابسته هستند و باید منتظر محاسبه شدن مقدار بلوک قبلی باشیم بنابراین نمی توان عملیات رمزنگاری را به صورت موازی پیاده سازی کرد .



Cipher Block Chaining (CBC) mode encryption

اما در حالت رمزگشایی برای گرفتن خروجی قسمت decryption تنها به بلوک ciphertext و کلید نیاز داریم و وابستگی بین بلوک ها دیده نمی شود ، خروجی این بلوک رمزگشایی هم با بلوک رمز شده قبلی XOR می شود ، بنابراین نیازی به صبر کردن برای خروجی بلوک قبل نداریم. برای مثال برای بدست آوردن خروجی شماره O_1 تنها به C_2 و Key نیاز داریم ، در نهایت هم برای به دست آوردن P_2 به خروجی شماره O_1 و C_1 نیاز داریم که C_1 موجود و بدون نیاز به محاسبه و تاخیر است و خروجی O_1 می تواند به صورت مستقل محاسبه شود.

بنابرین تمامی بلوک ها می توانند به صورت موازی رمزگشایی شوند.

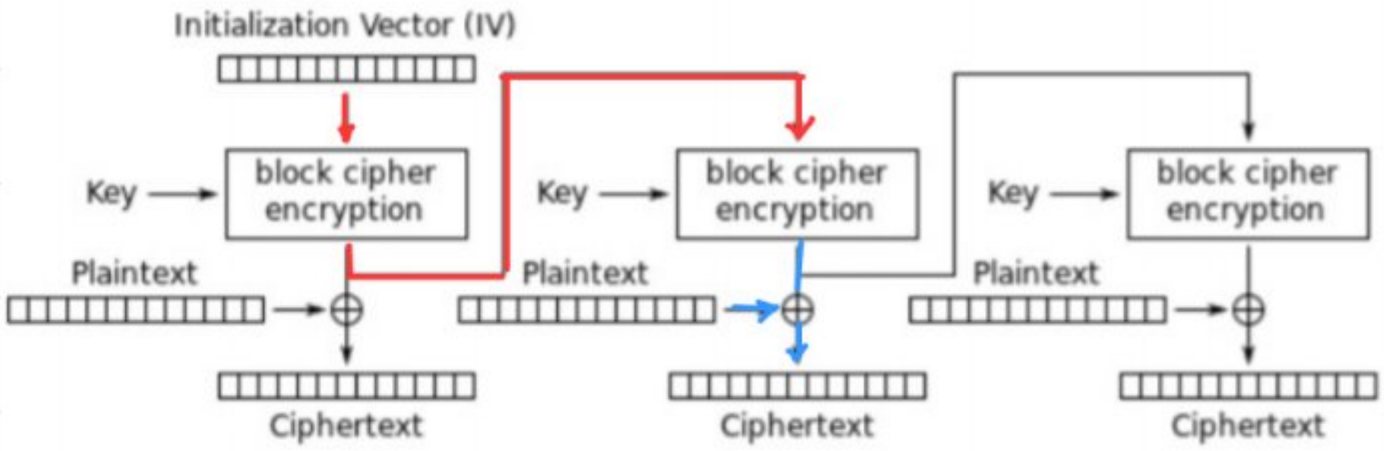


Cipher Block Chaining (CBC) mode decryption

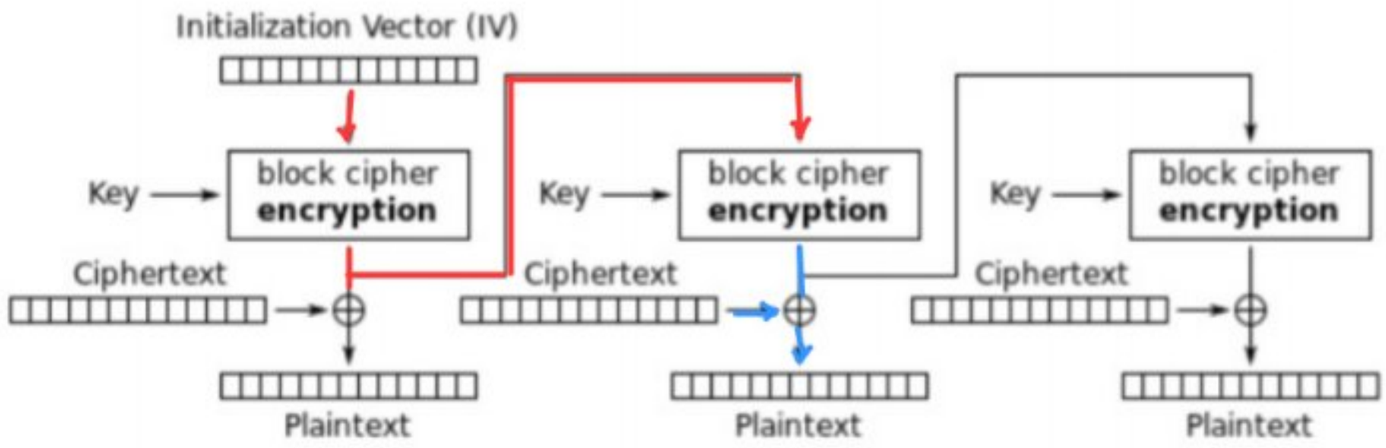
OFB : در این حالت کاری در صورتی که IV از قبل مشخص باشد می توان خروجی های بلوک ها را به صورت آفلاین محاسبه کرد و نگه داری کرد تا در زمان رمزگذاری ، خروجی های آن ها با بلوک های plaintext ، XOR شود و حاصل محاسبه شود ، که در این صورت موازی سازی در رمزگذاری صورت گرفته ، اما اگر IV را از قبل نداشتیم ، هر بلوک برای به دست آوردن ورودی خود ، نیازمند محاسبه خروجی بلوک قبل از خود می ماند ، در نتیجه نمی توان موازی سازی را انجام داد.

در رمزنشانی هم از آنجایی که ساختار بلوک ها شبیه افزونگی است درصورت مشخص بودن IV از قبل می‌توان

موازی سازی کرد و در غیر این صورت موازی سازی نخواهیم داشت.



Output Feedback (OFB) mode encryption



Output Feedback (OFB) mode decryption

② درمورد کلیدهای ضعیف DES تحقیق کنید و آن‌ها را توضیح دهید.

در DES از کلیدی به طول 56 بیت ، 16 زیرکلید به طول 48 بیت تولید می‌کنیم ، کلید ضعیف باعث ایجاد زیرکلیدهای یکسان می‌شوند.

وقتی کلید به حالت‌های زیر باشد می‌گوییم کلید ضعیف است :

- صفرونک های متناوب 0x0101010101010101

- F و E های متناوب 0xFEFEFEFEFEFEFEFE

- 0x E0E0E0E0F1F1F1F1

- 0x1F1F1F1F0E0E0E0E

با استفاده از کلیدهای ضعیف خروجی F1 در تابع تولید زیرکلید تمام صفر ، تمام یک یا صفرونک متناوب می‌شود.

از آنجایی که تمامی زیرکلیدها یکسان هستند و DES ساختار فیستل دارد در رمزنگاری در دور اول داده رمز نمی‌شود ولی در دور بعدی از آنجایی که کلید یکسان است داده رمز شده ، رمزنگاشایی می‌شود ، درواقع مراحل ، دو تا دو تا آنتی کلیدیگر را خنثی می‌کنند و باعث کاهش پیچیدگی رمزنگاری DES می‌شود.

③ حمله‌ای الگوریتم رمزنگاری $DES_{K_1}(DES_{K_2}(m))$ که سریع‌تر از جستجوی جامع باشد را افشا کند.

در Double DES ایده‌ی اصلی برای این بود که به رمزنگاری با کلید به طول 112 برسیم.

که در این صورت به 2^{112} تلاش برای به دست آوردن کلید نیاز خواهیم بود.

فرض کنیم حمله متن آشکار (Known plaintext attack) باشد، یعنی مهاجم به ازای یک P_1، C_1 معادل آن را داند و هدف آن پیدا کردن key استفاده شده برای رمزنگاری می‌باشد.

رابطه‌ی زیر برای متن رمزشده در Double DES برقرار می‌باشد:

$$C_1 = DES_{K_1}(DES_{K_2}(P_1)) \quad (1)$$

در این حمله ابتدا به صورت جستجوی جامع تمامی مقادیر ممکن به ازای تمامی کلیدهای 56 بیتی را برای $DES_{K_2}(P)$ محاسبه کرده و در حافظه ذخیره می‌کنیم. (2^{56} حالت می‌شد)

رابطه‌ی (1) را می‌توان به صورت زیر باز نویسی کرد:

$$DEC_{K_1}(C_1) = DES_{K_2}(P_1)$$

بنابراین C_1 را به ازای تمامی حالت‌های ممکن برای کلید 56 بیتی محاسبه کرده و آن را هم ذخیره می‌کنیم (2^{56} حالت می‌شد)

مقادیر به دست آمده از دو عمل فوق را با هم مقایسه می‌کنیم تا به ازای یک کلید رابطه‌ی $DEC_{K_1}(C) = DES_{K_2}(P_2)$ برقرار باشد. اگر برابر بودند جهت اطمینان بیشتر روی یک P_2 و C_2 دیگر امتحان می‌کنیم.

اگر بازهم برای Pهای دیگر هم با کلید به دست آمده درست بود به کلید درست یافته‌ایم. مراحل فوق به اندازه‌ی 2×2^{56}

یعنی 2^{57} خواهد بود که سریع‌تر از جستجوی جامع (2^{112}) می‌باشد.

④ چرا روند حریان LFSR برای روند حریان مناسب نیست.

در روند نگاری علاقه مند هستیم که ciphertext ها غیرقابل پیشبینی و تصادفی بنوافت (تا حدممکن) باشد.

LFSR با اینکه درصورت انتخاب درست فیدبک ها ازمون های تصادفی بودن را به خوبی پاس می کند، اما تنها

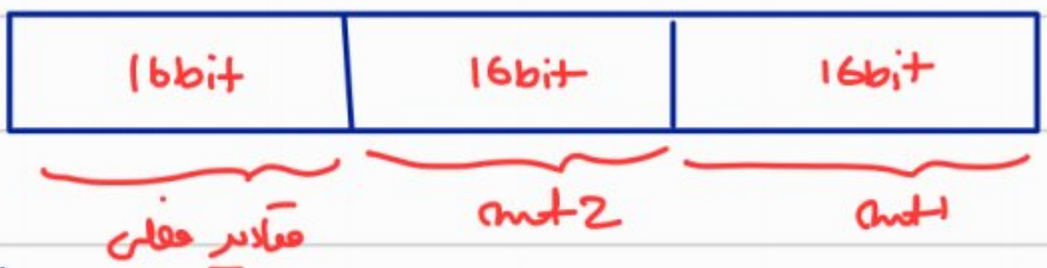
XOR در آن استفاده شده که ایمنی خطرناک است ودرنتیجه حریان چند دستگاه معادله نوشت و seed را

به دست آورد.

درواقع اگر دوبرابر ورودی ها خروجی داشته باشیم حریان seed را محاسبه کرد، از آن جایی که الگوریتم برای همه

آشکارا است، حریان به عنوان دلخواه خروجی را به ازای seed یافته شده محاسبه کرد، درنتیجه این خاصیت

غیرقابل پیشبینی بودن را نداریم.



درواقع از آنجایی که ابتدا مقادیر را از روی seed محاسبه کرده و در انتهای مقادیر ورودی قرار می دهیم و با شیفت یک

عدد را تولید می کنیم در ابتدای کار هربار یک رقم از seed از LFSR خارج می شود، درنتیجه پس از 16

شیفت، کل مقدار seed (out1) را به عنوان خروجی خواهیم داشت با داشتن 16 شیفت بعدی (out2)

و seed (out1) با استفاده از معادلات خطی می توان محل XOR ها را نیز به دست آورد.

درصورتی که seed را داشته باشیم ادامه ی key stream را تولید می کنیم و با شنود کردن C و درنهایت

با محاسبه ی C⊕K، اطلاعات plaintext به دست می آید. همچنین خروجی بودن LFSR درصورتی که

مقادیر ثقلی و محل XOR مشخص باشد، معلوم است که با خاصیت تصادفی بودن تناقض دارد و ضعیف است.

⑤ در مورد پدینگ PKCS#7 تحقیق و توضیح دهید.

منظور این نوع از padding به شکل زیر می باشد :

- بایت های padding قبل از رمزنگاری به متن اصلی افزوده می شوند.

- مقدار هر بایت افزوده شده برابر است با تعداد بایت های لازم برای padding.

- حداقل تعداد بایت های padding ، 1 می باشد.

برای مثال فرض کنیم بلوک های ما 16 بایتی می باشد و متن ما شامل 18 بایت زیر می باشد :

F14ADBDA019D6DB7 EFD91546E3FF8444 9BCB

بنابراین باید 14 بایت (0E) به انتهای متن اضافه کنیم تا مضربی از 16 شود :

F14ADBDA019D6DB7 EFD91546E3FF8444 9BCB0E0E0E0E0E0E

0E0E0E0E0E0E0E0E

درصورتی که متن ما خودش 16 بایت باشد از آنجایی که حتما باید padding انجام شود وحداقل تعداد براساس قاعده ی

سوم برابر 1 بایت است پس باید 16 بایت پد کنیم :

F14ADBDA019D6DB7 EFD91546E3FF8444

10101010 10101010 10101010 10101010

⑥ فرض کنید که کاربری از سروری پیام رمزشده $C = Enc_K(m)$ را دریافت می‌کند. مهاجمی در میان این راه این پیام را شنود می‌کند و می‌خواهد که به محتوای پیام دسترسی داشته باشد. مهاجم می‌داند که الگوریتم رمزنگاری استفاده شده در سرور از الگوریتم رمزنگاری قالبی AES در مد کاری CBC به همراه پدینگ pkcs#7 است.

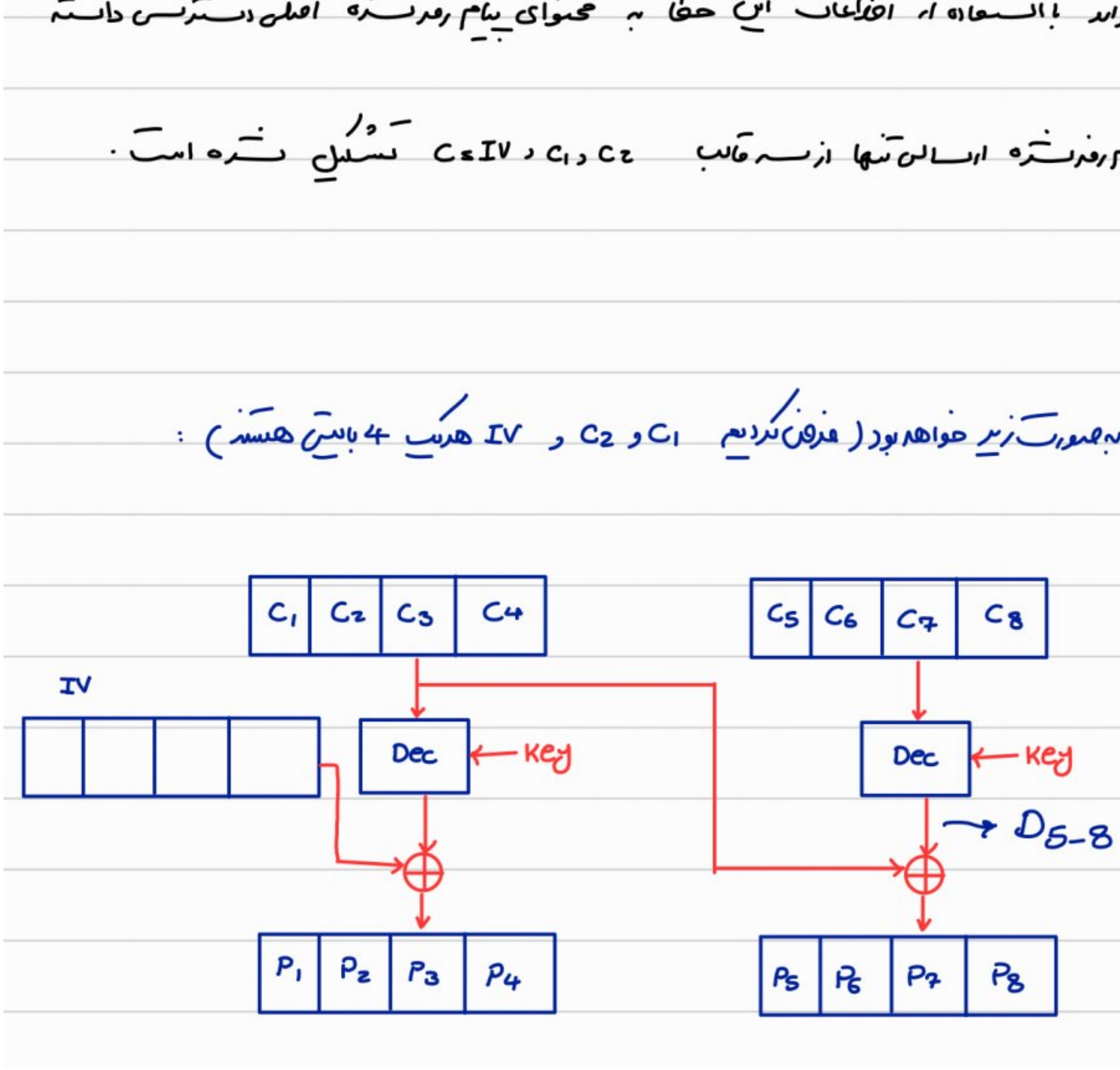
```
cipher = AES.new(KEY, AES.MODE_CBC)
encrypted = cipher.encrypt(pkcs7padding(plaintext))
```

این سرور همچنین پیام‌های رمزشده کاربران را دریافت و رمزگشایی می‌کند و در زمان رمزگشایی سرور ابتدا پیام را رمزگشایی و سپس unpad می‌کند تا به متن اصلی برسد و اگر الگوی پدینگ مناسب نباشد خطای "error in padding" را به کاربر برمی‌گرداند و در غیر این صورت سرور هیچ پیام خطایی را برنمی‌گرداند.

توضیح دهید که مهاجم چگونه می‌تواند با استفاده از اطلاعات این خطا به محتوای پیام رمزشده اصلی دسترسی داشته باشد. برای سادگی فرض کنید که پیام رمزشده از سه قالب c_1 و c_2 و $c=IV$ تشکیل شده است.

(padding_oracle attack).

بنابر فرض مسئله، شکل قالب‌ها به صورت زیر خواهد بود (فرض کردیم c_1 و c_2 و IV هرکدام 4 بایتی هستند):



مهاجم قسمت هایی از ciphertext را تغییر می دهد تا بر اساس اروری که دریافت می کند متن رمزشده را رمزگشایی کند.

اگر مهاجم C_4 را یک واحد افزایش دهد P_4 و P_8 تحت تاثیر این تغییر قرار خواهند گرفت و فرض می کنیم در این حالت P_8 به یک مقدار رندوم 0xAB تبدیل می شود در این حالت سرور انتظار padding را دارد چک می کند که آیا 0xAB با padding ای مطابقت دارد یا خیر و اگر نداشت padding error برمی گرداند.

مهاجم تمام 256 حالت ممکن را برای C_4، بدون تغییر سایر بایت ها ارسال می کند و برای اکثر آن ها padding error دریافت خواهد کرد ولی برای یک مقدار مخصوص error padding دریافت نمی کند.

برای این مقدار سرور P_8 را برابر 0x01 بدست می آورد و فرض می کنند padding درست است ولی MAC ارور خواهیم داشت (چون ciphertext تغییر یافته). فرض کنیم مقداری که مهاجم قرار داده برابر C'_4 باشد داریم:

$$P_8 = D_8 \oplus C_4$$

$$D_8 = P_8 \oplus C_4 \Longrightarrow P'_8 = D_8 \oplus C'_8 \Longrightarrow P'_8 = P_8 \oplus C_4 \oplus C'_4 = 0x01$$

$$P'_8 = 0x01$$

از آنجایی که C_4 و C'_4 را داریم می توان P_8 را به صورت زیر محاسبه کرد:

$$P_8 = 0x01 \oplus C_4 \oplus C'_4$$

بنابراین بایت آخر متن به دست آمده است.

با داشتن مقدار P_8 می توان C'_4 را طوری محاسبه کرد که هر مقدار دلخواه مادر P_8 ظاهر شود. (P''_8)

فرض نیم می‌خواهیم $0x02$ در P_8 ظاهر شود داریم:

$$C_4'' = C_4 \oplus P_8 \oplus 0x02$$

$$P_8'' = C_4'' \oplus D_8$$

$$P_8'' = \underbrace{C_4} \oplus \overbrace{P_8} \oplus 0x02 \oplus \underbrace{C_4} \oplus \overbrace{P_8} = 0x02$$

پس با جایگذاری C_4'' می‌توان $0x02$ را در P_8 ظاهر کرد.

حالا می‌خواهیم بایت یکی مانده به آخر (P_7) را پیدا کنیم.

ابتدا P_8 را به مقدار $0x02$ تغییر می‌دهیم تا سرور فکر کند 2 تا بایت padd شده و P_7 را چک کند تا P_7 هم برابر $0x02$ بود پدینگ را درست در نظر بگیرد.

پس C_4'' را ثابت نگه می‌داریم، C_3 را تا جایی تغییر می‌دهیم که padding error دریافت نکنیم (تا 256 حالت را بررسی می‌کنیم)، زمانی که P_7 برابر $0x02$ شد، MAC اور دریافت می‌کنیم در این حالت داریم:

$$C_3' \longrightarrow \text{MAC error}$$

$$P_7' = 0x02$$

$$P_7' = C_3' \oplus D_7$$

$$P_7 = C_3 \oplus D_7$$

$$D_7 = C_3 \oplus P_7$$

$$P_7' = 0x02 = C_3' \oplus C_3 \oplus P_7 \Longrightarrow P_7 = 0x02 \oplus C_3' \oplus C_3$$

با به‌دست آوردن P_7 می‌توان عشاب کاری که در P_8 کردیم، P_6 و P_5 را هم به‌دست آورد.

از آنجایی که می‌دانیم تعداد بایت‌های پدینگ برابر تعداد بایت‌های یک بلوک است پس می‌توان مقادیر $0x05$ را به‌عنوان

عدد padding در بایت ها قرار داد ، در نتیجه برای به دست آوردن مقادیر بلوک اول فرض می کنیم بلوک

دوم کلا وجود ندارد و با تفسیر بایت های IV مشابه بلوک قبل به محاسبه ی P_4 و P_3 و P_2 و P_1 می پردازیم .

۷) به صورت مختصر حملات شامیر به رمز جریانی RC4 را توضیح دهید.

حملات شامیر یا همان FMS بر پایه‌ی IV های ضعیف می‌باشد که در RC4 استفاده می‌شود.

RC4 از کلید برای ایجاد یک ماشین حالت استفاده می‌کند و سپس از آن با آپدیت کردن حالت یک بایت جدید از keystream حالت جدید ایجاد می‌کند.

با برخی IV های مشخص اگر مهاجم بایت اول از keystream و m بایت اولیه از key را بداند می‌تواند m+1 بایت از کلید را به دست بیاورد.

از آنجایی که اولین بایت از متن رمز نشده از هدر WEP SNAP است، می‌توان فرض کرد که مهاجم می‌تواند بایت اول keystream را از B⊕ 0xAA به دست بیاورد. (SNAP header = 0xAA)

سپس به یک بردار IV به صورت (x و 255 و 3) نیاز داریم. (هر IV در WEP به صورت 24 بیتی می‌باشد)

مهاجم ابتدا از IV به عنوان 3 مولفه‌ی اول K استفاده می‌کند و S-box ها را با مقادیر متوالی از 0 تا n پر می‌کند.

سپس 3 تکرار اول KSA را برای مقداردهی به S-box ها انجام می‌دهد. بعد از این تکرار ها احتمالاً می‌توان بایت چهارم را از O (خروجی keystream) با رابطه‌ی $k[3] = (O - j - S[3]) \bmod n$ به دست بیاورد.

در این حالت مهاجم بایت چهارم از key را ندارد و گاهی تنها مقادیری ممکن برای key را تولید می‌کند.

با جمع‌آوری پیام‌های رد و بدل شده و تکرار مراحل فوق مهاجم مقادیر ممکن را تولید می‌کند که احتمالاً مقداری که صحیح است بیشتر تکرار می‌شود. درصورتی که مهاجم بتواند بایت چهارم را پیدا کند می‌تواند به بایت بعدی حمله کند.