باسمه تعالی

پروژه‌ی برنامه‌نویسی رمزنگاری کاربردی

درس: **رمزنگاری کاربردی** تاریخ تحویل: **۱۷ بهمن ماه ۱۳۹۹، ساعت ۲۳:۵۹**

دانشجویان عزیز، لطفاً موارد و نکات زیر را در انجام پروژه در نظر داشته باشید:

- در ادامه چهار بخش پروژه برای شما مطرح شده است که برای حل آن‌ها مجاز به استفاده از هر نوع منبع، کتاب و ابزاری هستید. توصیه می‌شود از ابزار Sage استفاده کنید. برای آشنایی بیش‌تر با این ابزار، به این مستند مراجعه نمایید.
- توجه کنید که شما مجاز به کپی کردن هیچ کدی و به اشتراک‌گذاری روش حل خود با دیگران نیستید.
- به ازای هر سوال لطفاً یک پوشه به نام لاتین سوال ایجاد نمایید، یک پرونده‌ی متنی در قالب Markdown یا PDF ایجاد کنید و سپس توضیحاتی در مورد روش حل سوال و هم‌چنین **روش اجرای کد** خودتان را در این مستند بنویسید.
- در نهایت این چهار پوشه را به صورت فشرده، ترجیحاً در قالب ZIP ارسال نمایید.
- زمانی جهت تحویل حضوری پروژه در نظر گرفته شده است که متعاقباً اعلام می‌شود.
- لطفا سوالات خود را به آدرس s.salimi@sharif.edu ارسال کنید.

سوال ۱. Reminders: در این سوال می‌خواهیم گریزی به یک مقاله‌ی بسیار معروف در حوزه‌ی امنیت کلیدهای عمومی داشته باشیم. توصیه می‌شود این مقاله را با جزییات بیش‌تری مطالعه کنید:

Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Device

به صورت خلاصه، این مقاله با جمع آوری گستره‌ی کلیدهای عمومی از SSH و TLS گزارش کرد که در حدود ۱ درصد از کلیدهای موجود مشترک است. اما نکته‌ی جالب تر این است که ۰٫۵ درصد از کلیدهای عمومی TLS فاکتورهای مشترکی در هنگام تولید عدد n مولفه‌ی کلید دارند که باعث می‌شود کلید خصوصی RSA نیز به دست آید.

در این سوال، شما دو پوشه در اختیار دارید که یکی از آن‌ها به نام pkeys شامل ۱۱۸ کلید عمومی RSA است و پوشه‌ی دیگر به نام encs شامل ۱۱۸ پرونده است که **یک پیام** یکسان با هر یک از ۱۱۸ کلید رمز شده است. دقت کنید که شماره‌ی هر پرونده‌ی رمز شده با شماره‌ی کلید عمومی که آن را رمز کرده است، یکسان است.

شما باید با بررسی کلیدهای عمومی، تلاش کنید یکی از کلیدهای خصوصی را با توجه به مولفه‌های مشترک ساخت عدد بازیابی کنید و سپس پیام رمزشده‌ی مرتبط با آن کلید را رمزگشایی کنید.

خروجی کد شما باید چاپ پیام متنی ساده باشد.

سوال ۲. EThree: در این سوال شما با حمله‌ی Low public exponent attack روی کلید RSA آشنا می‌شوید.

در این سوال، به شما یک اسکریپت پایتون داده شده است که پیام $secret$ را از پرونده‌ی top_secret.py می‌خواند. دقت کنید که شما این پرونده را در اختیار ندارید و می‌توانید یک پرونده با این نام ایجاد کنید که متغیر $secret$ را در آن تعریف می‌کنید، به طور مثال یک پیام به شکل زیر در این پرونده ایجاد می‌کنید:

```
secret = "This is a simple secret!!"
```

سپس می‌توانید کد ethree.py را اجرا کنید تا خروجی سه عبارت رمزشده ببینید که با سه کلید متقاوت تصادفی رمز شده است. شما از این کلیدها مولفه‌ی $e = ۳$ را در اختیار دارید اما مولفه‌ی n در اختیار شما نیست و دقت کنید که این مولفه (با توجه به تعداد بیت‌های عدد) امن است.

در واقع کد ethree.py یک اوراکل است که شما به کد آن دسترسی دارید، این اوراکل به ازای هر پیامی که در پرونده‌ی top_secret.py تعریف می‌کنید، سه عبارت رمزشده که با سه کلید متفاوت ایجاد شده‌اند را، چاپ می‌کند.

```
#!/usr/bin/env sage
# # Problem Set, Applied Cryptography I
# run with this command sage -python ethree.py

from sage.all import *
from top_secret import secret

def keygen(nbit):
PRIMES = [random_prime(2**(nbit - 1), 2**nbit) for _ in range(6)]
n_1, n_2, n_3 = [PRIMES[2*i]*PRIMES[2*i + 1] for i in range(3)]
return (n_1, 3), (n_2, 3), (n_3, 3)

def bytes_to_long(msg):
mhex = msg.encode('utf8-').hex()
mint = int(mhex, 16)
return mint

def encrypt(msg, pubkey):
n, e = pubkey
m = bytes_to_long(msg)
return pow(m, e, n)

nbit = 1024
pubkey_1, pubkey_2, pubkey_3 = keygen(nbit)
```

```
enc_1 =  encrypt(secret, pubkey_1)
enc_2 = encrypt(secret, pubkey_2)
enc_3 = encrypt(secret, pubkey_3)

print('enc_1 =', enc_1)
print('enc_1 =', enc_2)
print('enc_1 =', enc_3)
```

وظیفه‌ی شما این است که با توجه به کوچک بودن مولفه‌ی e و در دست داشتن سه پیام رمزشده‌ی موجود در پرونده‌ی output.txt به پیام اصلی که رمزشده است، دسترسی پیدا کنید. کد حل شما باید این پیام را چاپ کند.

سوال ۳. Sides: هدف از این بخش، درک کارکرد یک سیستم رمزنگاری متقارن است. کد زیر که از کتابخانه‌ی SimplifiedDES مربوط به ابزار Sage استفاده می‌کند، به شما داده شده است:

```
#!/usr/bin/env sage
# Problem Set, Applied Cryptography I

from sage.crypto.block_cipher.sdes import SimplifiedDES
from top_secret import secret

sides = SimplifiedDES()
tobin = BinaryStrings()
ptext = tobin.encoding(secret)
assert len(ptext) % 8 == 0

key = sides.list_to_string(sides.random_key())
ciphertext = sides(ptext, key, algorithm = "encrypt")
enc = int(str(ciphertext), 2)

print('enc =', enc)
print('key =', '?')
```

این کد با صدا زدن کتابخانه‌ی مورد نظر، یک متن با نام متغیر *secret* ذخیره شده در پرونده‌ی top_secret.py را می‌خواند و سپس با استفاده از یک کلید تصادفی آن را رمزگذاری می‌کند و خروجی آن را به صورت زیر چاپ می‌کند:

```
enc = 1102307742996258269973515185089554645271210662538939464770320571930\
4048434293684897555835843297884953478832758403661278136344191451856523567\
6662798389007958502585192941395939477252766831102421911700079643695675515 1
key = ?
```

وظیفه‌ی شما این است که با مطالعه‌ی سیستم رمزنگاری Simplified DES، به نقاط ضعف آن پی ببرید و با استفاده از این نقاط ضعف، عبارت رمزشده‌ی خروجی را بشکنید. خروجی کد شما باید متن ساده و هم‌چنین کلید رمزگذاری باشد. برای مطالعه‌ی بیش‌تر این سیستم رمزنگاری به اینجا مراجعه کنید.

سوال ۴. Last: در این سوال به سراغ الگوریتم رمزنگاری AES در حالت CBC-MAC و روش پدینگ PKCS#7 می‌رویم. کد زیر را در نظر بگیرید:

```
#!/usr/bin/env python3
import os, binascii, struct
from Crypto.Cipher import AES

op = lambda msg: msg + bytes([16 - len(msg) % 16] * (16 - len(msg) % 16))
def cryptocrypto(m):
    o0ps = AES.new(bytes(0x10), AES.MODE_CBC, bytes(0x10))
    return o0ps.encrypt(len(msg).to_bytes(0x10, 'big') + op(msg))[[:0x10-

pre = os.urandom(0x10)
print(binascii.hexlify(pre).decode())

post = binascii.unhexlify(input())

if post.startswith(b'I think cryptography is really cool, you?\0') \
        and cryptocrypto(post) == pre:
    print("Well done!!!")
```

فرض کنید خروجی این کد عبارت a42665fd6b9416f0a67b8b64b358cff4 شده است.

وظیفه‌ی شما این است که کدی توسعه دهید که با دریافت هر عبارتی که کد چاپ می کند، یک پیام تولید کند که شرط آخر این کد را صحیح کند و عبارت Well Done!!! چاپ شود. به طور مثال عبارت زیر یک خروجی معتبر برای مثال است:

```
49207468696e6b2063727970746f677261706879206973207265616c6c7920636f6f6c2c20796f\
753f002e2e2e2e2e9e88385905e6b6c5bddb0ff9ecd06adb
```