Eindopdracht Applied Cryptography

Frans Schippers f.h.schippers@hva.nl
Versie 2223b4 1.2 2023-06-12
Eindopdracht blok 4 Voltijd
Herkansing Eindopdracht Blok 2 Deeltijd

Versie 1.1

Een aantal spelfouten is verbeterd.

Versie 1.2

Naam geveing aangepast en skelewton-file aangepast

Versie 1.3

Het commando voor de in te leveren files is aangepast.

Versie 1.4

Op een aantal plaatsen in `AC2223b4EO.pdf` en `cryptomail_skel.py` stond `cbf` ipv `cfb`. Het versie nummer van het `.xmsg` is ook aangepast. Dus zorg dat je deze wijziging meeneemt in je code en in `cm.mode`.

Versie 1.5

Inleveren in Codegrade via DLO.

Versie 1.6

De inlever commando's waren niet geheel zichtbaar.

Introductie

De eindopdracht bestaat uit een aantal delen waarmee je aangeeft cryptografie te kunnen toepassen met behulp van Python.

In het document wordt gesproken van ontvangers/verzenders (receivers/senders). Afhankelijk van het doel wordt hier of ontvangers of verzenders bedoeld. Het is aan jou om de juiste te kiezen.

De onderdelen zijn:

- Aanmaken van een private key, publieke key en jouw certificaat met jouw publieke key [crt 1-2].
- Het kunnen gebruiken van de standaard cryptografische functies, zoals hash, symmetrische encryptie/decryptie, asymmetrische encryptie/decryptie en signing/verificatie [crt 3-7].
- Het toepassen van deze technieken in een cryptografische probleem: CryptoMail [crt 7-16]

- Overige eisen [crt 17-20]

We gebruiken de python3 `cryptography` library, bekijk de documentatie op: https://cryptography.io

CryptoMail

Deze opdracht simuleert S/MIME en PGP. Dit zijn twee standaarden om versleutelde en ondertekende mail uit te wisselen. De meeste email-programma hebben support voor deze technieken. De echte implementatie van S/MIME en/of PGP is redelijk complex, Deze simpele versie heeft wel alle functionaliteit om berichten uit te wisselen. Met de `encode` optie creëer je een `.xmsg` -file met het beschermde bericht. Je kunt de `.xmsg` file via mail versturen en de andere partij kan deze dan lezen met de `decode` optie van het programma. Het is echter noodzakelijk om de private keys of certificaat op een veilige manier eenmalig te transporteren. Hierdoor kan de geldigheid van de publiek-key/certificaat worden vastgesteld.

Met een CryptoMail wordt een bericht versleuteld en ondertekend opgeslagen. Hierdoor is het bericht niet te lezen zijn door anderen (net bevoegden), maar door slechts een bepaald aantal beoogde ontvangers/verzenders. Ook biedt het de mogelijkheid om het bericht te ondertekenen door een aantal ontvangers/verzenders. De beoogde ontvangende/verzendende partij kan van het gelezen bericht van de ontvanger/verzenders verifiëren.

Let op
De verzenders en/of ontvangers kunnen ontbreken, dan is het bericht niet getekend en/of versleuteld. De hash wordt ook in dit geval berekend.

Een deel van de opdracht bestaat uit het creëren van de benodigde keys en het afmaken/aanvullen van een skeleton programma. Op een aantal plaatsen moet de code worden aangevuld in de skeleton file. Dit is aangegeven door de volgende regels in de code van de skeleton file:

```
# Student Work {{
  # Vervang dit door jouw code
# Student Work }}
```

Het programma behoeft op andere plaatsen geen wijzigingen, denk je dat dit wel nodig is, kijk nogmaals naar de code, want het wijzigingen van bestaande code is **niet** nodig.

Voor de eindopdracht heb je de tijd tot 18 juni 2023 23:55. Wacht niet tot het laatste moment. Begin vroeg zodat je er steeds een beetje aan kunt werken.

De opdracht moet ingeleverd worden met behulp van CodeGrade (zie DLO). Na het inleveren worden een aantal tests uitgevoerd en krijg je een indruk van de kwaliteit van de code. Er wordt hier gelet of het resultaat het juiste resultaat is. Tijdens het inleveren worden niet alle aspecten getest. Overige functionele testen worden uitgevoerd na de deadline. Deze beide functionele tests bepalen 90 punten van het cijfer, de laatste 10 punten is ter beoordeling van de docent. Het eindcijfer is het aantal punten gedeeld door 10. Zie de rubriek. CodeGrade is beschikbaar vanaf 14 juni. Het kunt programma maken zonder de hulp van CodeGrade, het helpt je wel door vanaf 14 juni te testen met CodeGrade. Inleveren moet altijd via CodeGrade, we beoordelen de laatste versie die ingeleverd is voor de deadline.

Samenwerken en bronnen

Je mag gebruik maken van de sheets en informatie op het internet. Echter mag je het werk niet door een andere laten maken of op aanwijzing van anderen het maken. Het dient je eigen werk te zijn. Ook generative AI is niet toegestaan om je programma te maken.

Bij twijfel wordt dit gecontroleerd door een aanvullend assessment waarvoor je zal worden uitgenodigd. Als wij de indruk hebben dat niet niet je eigen werk is zullen wij dit bij de examencommissie aanhangig maken.

Python Type Hints

De methodes hebben een `signatuur` (vorm van de methode): We gebruiken hiervoor **Python Type Hints**. Hierbij geven we achter de naam van het argument na de `:` het type van de variabel.

```
def meth(arg1: int, arg2: str) -> bool:
```

Hiermee wordt aangegeven dat het eerste argument een integer moet zijn, het tweede een string en dat het resultaat van de methode een boolean is. https://docs.python.org/3/library/typing.html

Beschrijving

Voor de opdracht is het maken van een programma `cryptomail.py` waarmee je een tekst-bericht (`mesg`) beschermd kunt overdragen met mail. De verzender maakt met dit programma een file waarin het tekst-bericht beschermd is opgeslagen. Dit bericht kan aan de ontvanger worden overgedragen, Deze ontvanger kan met dit programma kan het tekst-bericht lezen en een aantal eigenschappen van het tekst-bericht vaststellen.

De functionaliteit is:

- Het bericht kan versleuteld worden voor een aantal beoogde ontvangers/verzenders, alleen deze kunnen het tekst-bericht lezen.
- Het bericht kan ondertekend worden door een aantal ontvangers/verzenders, en gecontroleerd worden of de handtekeningen juist zijn..

Hiervoor moeten twee methoden geschreven worden `encode` en `decode` .

encode

De methode `encode` encrypt het tekst-bericht zodat een aantal ontvangers/verzenders het bericht kunnen lezen, de hash van het bericht wordt bepaald en het bericht word ondertekend door de ontvangers/verzenders. De benodigde informatie kan de CryptoMail-instance (`cm`) gehaald worden, de verkregen resultaten moeten ook hierin weer worden opgeslagen. Bijvoorbeeld het encrypted bericht in `cm.code` .

De signatuur van deze belangrijke methoden is:

```
def encode(cmFname: str, mesg: str, senders: list=None, receivers: list=None) -> tuple:
    """ Encode (encrypt and/or sign) the message (`mesg`) for the `receivers` and `senders`.
        The cryptomesg-structure (HvaCryptoMesg) is created and filled with encode-information
        and written to the file `cmFname`.
        Returns a tuple (sendersState, receiversState).
    """
```

decode

De methode `decode` decrypt het bericht voor de ontvangers/verzenders. Tevens geeft het aan welke ontvanger/verzender het bericht heeft getekend en of de ondertekening juist is.
De methode heeft de volgende `signatuur` (vorm van de methode):

```
def decode(cmFname: str, receivers: list=None, senders: list=None) -> tuple:
    """ Decode (decrypt and/or verify) the coded message `cmFname` for the `receivers` and `send
        The cryptomeg-structur (HvaCryptoMesg) is read from the file `cmFname` and
        used to decode de containing message.
        Returns a tuple (mesg, sendersState, receiversState, secretState).
    """
```

Rubric / Beoordeling

De rubric (beoordeling) is als volgt:

Voor de verschillende onderdelen kun je maximaal het opgegeven aantal punten krijgen. Het totaal aantal punten is 100. Het eindcijfer is aantal punten gedeeld door 10, afgerond op 1 cijfer achter de komma.

1. **10 pnt** Creëren van het eigen certificaat
2. **3 pnt** Alle benodigde informatie meesturen
3. **2 pnt** Implementatie genKey / genIv
4. **4 pnt** Implementatie encryptkey / decryptKey
5. **5 pnt** Implementatie encryptMesg / decryptMesg / padding
6. **4 pnt** Implementatie signMesg / verifyMesg
7. **3 pnt** Implementatie calcHash / chckHash
8. **3 pnt** Niet signen en niet encrypten (alleen hash)**(test01)**
9. **3 pnt** Alleen signen/verifieren van de message (1 signer)**(test02)**
10. **3 pnt** Alleen encrypten/decrypten van de message (1 encryptor)**(test03)**
11. **6 pnt** Encrypt/decrypt sign/verify van de message (1 signer en 1 encryptor)**(test04)**
12. **6 pnt** Signed en encrypted (namens jezelf) van een bericht voor user1**S.xmsg**
13. **3 pnt** Alleen sign/verify van de message (2 signers)**(test05)**
14. **3 pnt** Alleen encrypt/decrypt van de message (2 encryptors)**(test06)**
15. **6 pnt** Encrypt/decrypt sign/verify van de message (2 signers en 2 encryptors)**(test07)**
16. **6 pnt** Signed en encrypted (namens jezelf en user3) van een bericht voor jezelf en user1**M.xmsg**
17. **5 pnt** Aangeven of er geheimen in een bericht worden geopenbaard.
18. **10 pnt** Geen geheimen prijs geven in encoded bericht en uploaded bestanden.
19. **5 pnt** Fout afhandeling van cryptografisch foutieve berichten
20. **10 pnt** De wijze waarom het programma is gecodeerd en becommentarieerd (teachers points).

De opdracht

Aanmaken asymmetrische keys

Je moet voor deze opdracht een eigen private key, public key en certificaat genereren. Deze hebben de uitgang `.prv`, `.pub` en `.crt`. Hierbij moet de asymmetrische keys **1024** bits rsa-keys zijn. Je kunt de keys genereren met openssl.

Tevens moet op basis van de key een **self-signed certificaat** maken.
Het subject-veld moet zijn `"/CN=${email}@hva.nl/C=NL/O=hva/OU=hbo-ict/ST=NH/L=Amsterdam"`. Hierbij is `${email}@hva.nl` je email adres (bv f.h.schippers@hva.nl). De drie files beginnen met je studentnummer, dus de files heten:

- `${studNr}.prv`, bijvoorbeeld `123456789.prv`
- `${studNr}.pub`, bijvoorbeeld `123456789.pub`
- `${studNr}.crt`, bijvoorbeeld `123456789.crt`

HvaCryptoMesg

Deze `class` vormt de belangrijkste datastructuur van het programma. De verschillende velden worden gebruikt in de verschillende hulp-methoden en deze maken ook nieuwe waarden aan in deze structuur. Na `encode` bevat de structuur (class-instance) alle benodigde informatie om met `decode` het bericht weer te krijgen als mede informatie over de ondertekening.

Er is een serialisatie van deze structuur (`class-instance`) als een json-structuur. De routines `HvaCryptoMesg.load` en `HvaCryptoMesg.dump` worden gebruikt deze json-structuur in te lezen en weg te schrijven.

De class `HvaCryptoMesg` heeft de volgende velden: Zie ook het commentaar in het skeleton programma. Sommige velden zijn alleen nodig tijdens het cryptografische proces. Deze waarden moeten op `None` gezet worden om te voorkomen dat informatie wordt gelekt tijden het bewaren van deze structuur.

`def __init__(self) -> None:`

Deze methode initialiseert alle (class) variabelen. Zie de uitleg van de instance variabelen hieronder.

```
    def __init__(self) -> None:
        """ Initialise the used variables """
        self.version = '1.0'    # Version number
        self.modes   = []       # Specifies the used algorithms
        self.snds    = {}       # keys: names of senders, values: relevant data
        self.rcvs    = {}       # keys: names of receivers, values: relevant data
        self.sesIv   = None     # (optional) session Iv     (bytes)
        self.sesKey  = None     # (optional) session key    (bytes)
        self.prvs    = {}       # keys: names of user, values: prvKey-object
        self.pubs    = {}       # keys: names of user, values: pubKey-object
        self.code    = None     # (optional) the encrypted message  (bytes)
        self.mesg    = None     # (optional) the message            (bytes)
        self.dgst    = None     # (optional) the hash the message   (bytes)
```

`HvaCryptoMesg.version`

Het is handig om een versie nummer in de structuur op te nemen. Deze is momenteel `1.0` .

`CryptpMesg.modes`

Het bericht kan encrypted en/of signed zijn.

- Met de mode wordt aangegeven of het bericht encrypted is en welke cryptografische methoden zijn gebruikt.
- Met de mode wordt ook aangegeven of het bericht is getekend en welke cryptografische methoden zijn gebruikt
- Met de mode wordt ook aangegeven of er en hash is berekend en welke cryptografische methoden zijn gebruikt.

Voor deze opgave worden de volgende modes gebruikt, hierbij worden de verschillende parameters voor de cryptografische operaties bepaald

```
    'hashed:sha256'
    'signed:rsa-pss-mgf1-sha384',
    'crypted:aes256-cfb:pkcs7:rsa-oaep-mgf1-sha256',
```

`HvaCryptoMesg.snds`

Dit is een dict (dictionary) met als key de verzender als waarde relevante informatie van/voor de verzender. Het is een dict omdat een bericht in theorie namens meerdere partijen verzonden kan worden.

`HvaCryptoMesg.rcvs`

Dit is een dict met als key een ontvanger en als waarde relevante informatie van/voor de ontvanger. Het is een dict omdat een bericht in theorie door meerdere partijen te lezen kan zijn.

`HvaCryptoMesg.sesIv`

Deze moet op de juiste waarde worden geïnitialiseerd (`HvaCryptoMesg.genIv`). Deze wordt gebruikt in encryptMesg en decryptMesg. Het is een bytes-string, geen hex-string.

`HvaCryptoMesg.sesKey`

Deze moet op de juiste waarde worden geïnitialiseerd (`HvaCryptoMesg.genSesKey`). Deze wordt gebruikt in `encryptMesg` en `decryptMesg` . Het is een bytes-string, geen hex-string.

`HvaCryptoMesg.prvs`

Deze dict bevat voor een verstuurder/ontvanger de privateKey (type `rsa._RSAPrivateKey`).

`HvaCryptoMesg.pubs`

Deze dict bevat voor een verstuurder/ontvanger de publicKey (type `rsa._RSAPublicKey`).

`HvaCryptoMesg.mesg`

Dit is het onversleutelde bericht (type `bytes`)

`HvaCryptoMesg.code`

Dit is het versleutelde bericht (type `bytes`)

`HvaCryptoMesg.dgst`

Dit is de hash van het onversleutelde bericht (type `bytes`)

HvaCryptoMesg (buildin) methodes

De class heeft ook nog de volgende reeds gemaakt methodes.

`def load(self, cmFname: str) -> None:`

De methode laadt de file aangeduid met `cmFname` en vult de verschillende class-variabelen. De file moet een representatie zijn van de HvaCryptoMesg class.

`def dump(self, cmFname: str) -> None:`

De methode dumpt de verschillende class-variabelen in de file aangeduid met `cmFname` . De file is een representatie van de HvaCryptoMesg class.

`def addMode(self, mode: str):`

Hiermee wordt aangegeven of de inhoud van HvaCryptoMesg encrypted en/of signed is. Tevens worden de gebruikte cryptografische methoden aangeduid. Deze methoden zijn in de sheets beschreven.

`def hasMode(self, mode: str):`

Hierbij kan gekeken worden of de inhoud van HvaCryptoMesg encrypted, hashed en/of signed is. Bijvoorbeeld om te kijken of het bericht getekend is:

```
if cm.hasMode('signed'): ...
```

HvaCryptoMesg (te maken) methodes

Al deze routines hebben een `# Student Work {{ ... # Student Work }}` sectie die moet worden aangevuld. Daar buiten is geen aanpassing nodig.

`def loadPrvKey(self, name: str) -> None:`

Deze methode laad een Private-key in pem-formaat van file. De filenaam is de naam gevolgd door `.prv` . De privateKey (type `rsa._RSAPrivateKey`) wordt opgeslagen in `self.prvs[name]` .

`def loadPubKey(self, name: str) -> None:`

De code om vanuit een certificaat de publieke key te halen is al gegeven. Je hoeft alleen de code voor een `.pub` file te schrijven. Deze methode laad een public-key in pem-formaat vanuit een file. De filenaam is de naam gevolgd door `.pub` . De publicKey (type `rsa._RSAPublicKey`) wordt opgeslagen in `self.pubs[name]` .

`def genSesKey(self, n: int) -> None:`

Genereer een goede sessieKey (type bytes) en assigneer deze aan `self.sesKey` De parameter `n` geeft de key-lengte in bytes aan.

`def genSesIv(self, n: int) -> None:`

Genereer een goede sessieIntialVector (type bytes) en assigneer deze aan `self.sesIv` . De parameter `n` geeft de iv-lengte in bytes aan.

`def encryptSesKey(self, user: str) -> bool:`

Encrypt de sessieKey (`self.sesKey`) met de juiste public of private key voor persoon `user` . Deze is eerder opgeslagen in `self.prvs` of `self.pubs` door de methoden `loadPrvKey` of `loadPubKey` . De beschermde key (`encKey`) wordt bewaard in `self.rcvs[user]` of `self.snds[user]` .
Let op
De sessieKey (`sesKey`) en de beschermde sessieKey (`encKey`) zijn van type bytes. **Let op**
De class-variable `self.mode` geeft aan hoe er encrypt/decrypt moet worden.

def `decryptSesKey(self, user: str) -> bool:`

Deze methode decrypt de beschermde sessieKey (`encKey`) met de juiste key voor persoon `user` . Deze is eerder opgeslagen in `self.prvs` of `self.pubs` door de methoden `loadPrvKey` of `loadPubKey` . De beschermde sessieKey (`encKey`) is opgeslagen in `self.rcvs[user]` of `self.snds[user]` .

`def encryptMesg(self) -> bool:`

Deze methode doet het de symmetrische encryptie op basis van de cryptografische methoden aangegeven in `self.modes` . De plainBytes staat in `self.mesg` en de encrypted data moet worden opgeslagen in `self.code` , beide zijn van type `bytes` .
Let op
De `cm.mode` geeft aan hoe er encrypt/decrypt moet worden.

`def decryptMesg(self) -> bool:`

Deze methode doet het de symmetrische decryptie op basis van de cryptografische methoden aangegeven in `self.modes`. De encrypted data staat in `self.code` en de decrypted data moet worden opgeslagen in `self.mesg`, beide zijn van type `bytes`.

Let op

De `cm.mode` geeft aan hoe er encrypt/decrypt moet worden.

`def signMesg(self, user: str) -> bool`:

Deze methode ondertekent het bericht `self.mesg` voor `user`. De key staat in `self.prvs` of `self.pubs` Deze is eerder opgeslagen in `self.prvs` of `self.pubs` door de methoden `loadPrvKey` of `loadPubKey`. De te ondertekenen data zit in `self.mesg` en de signature (type `bytes`) wordt opgeslagen in `self.rcvs[user]` of `self.snds[user]`

Letop

De `cm.mode` geeft aan hoe er signed/verified moet worden.

`def verifyMesg(self, user: str) -> bool`:

Deze methode verifieert de signature (`self.rcvs[user]` of `self.snds[user]`) voor `user` met het bericht `self.mesg`. Afhankelijk van correctheid van de signature wordt `True` of `False` teruggeven. En in andere gevallen `None`.

`def calcHash(self, : str) -> None`:

Deze functie berekent de hash van het bericht (`self.mesg`) en bewaard deze in de HvaCryptoMesg structuur (`self.dgst`)

Let op

De `cm.mode` geeft aan hoe er gehashed moet worden.

`def chckHash(self, : str) -> bool`:

Deze functie controleert of de hash (`self.dsgt`) klopt met het bericht (`self.mesg`). Als de hash correct is, is de return waarde `True` anders `False`

Let op

De `cm.mode` geeft aan hoe er gehashed moet worden

Als alle boven genoemde methoden zijn geïmplementeerd heb je de basis functionaliteit geïmplementeerd. de juist werking van deze methoden wordt getest in CodeGrade als je opdracht inlevert

Applicatie Functionaliteit

De applicatie functionaliteit wordt gerealiseerd in twee methoden. `encode` en `decode`. De verschillende tests testen de werking van de aspecten van deze methode.

- encode
  De methode `encode` zorgt er voor dat het bericht `mesg` door de ontvangende/versturende partijen wordt ondertekend en versleuteld zodat alleen de beoogde partijen het kunnen lezen.
- decode
  De methode `decode` controleert op de opgeven ontvangers/verzenders het bericht kunnen decrypten en controleert of het bericht is ondertekend door de opgegeven partijen (users). Ook wordt aangegeven of in het bericht ten onrechte geheimen worden prijsgegeven,

`def encode(cmFname: str, mesg: str, senders: list, receivers: list) -> tuple:`

De methode `encode` zal een HvaCryptoMesg bericht bewaren in de file `cmFname` met `cm.dump` .

Er moeten vier stukken code worden ingevuld/geprogrammeerd. Gebruik hiervoor de routines die je eerder hebt geïmplementeerd.

0) Conversie
De te coderen string is een unicode-string, deze moet worden opgezet naar `bytes` .

1) Encrypt
Als er ontvangers/verzenders zijn (bedenk welke van de twee) moet het bericht worden versleuteld. Dus onder andere het creëren van sessie key en sessie iv, het encrypten van de message (`cm.mesg`) en het beschermen van de sessie key. Het gecodeerde bericht komt in `cm.code` De beschermde keys sla je op in de dict (`cm.rcvs` of `cm.snds`) met als key de `user` en als waarde de beschermde key (`bytes`).

Gebruik de hulproutines: `genSesKey` , `genSesIv` , `encryptSesKey` , `encryptMesg` en mogelijk andere.

2) Hash
Bereken de hash met `calcHash` Deze wordt door de methode opgeslagen in `cm.dgst`

3) Sign
Als er ontvangers/verzenders zijn (bedenk welke van de twee) moet het bericht (`cm.mesg`) ondertekend worden. De signatures sla je op in de dict (`cm.rcvs` of `cm.snds`) met als key de `user` en als waarde de signature (`bytes`).

Gebruik de hulproutines: `signMesg` en mogelijk andere.

4) Strip
Er staat te veel (vertrouwelijke) informatie in de HvaCryptoMesg class. Deze class-variabelen moet op `None` worden gezet om te voorkomen dat deze in het bestand worden weggeschreven.
Let Op
De files `test1.xmsg` , `test2.xmsg` , `test3.xmsg` bevatten nog dergelijke geheimen.

A) Return
Vul de dict `receiverState` en `senderState` voor elke opgegeven ontvangers (`receivers`) en verzenders (`senders`) met een van de volgende waarden:

- None: geen informatie beschikbaar om te encrypten/signeren
- True: operatie geslaagd.
- False: operatie gefaald.

Deze twee dicts vormen het return-tuple

```
    return (senderState, receiverState)
```

`def decode(cmFname: str, rcvs: list, snds: list) -> tuple:`

De methode `decode` zal een HvaCryptoMail bericht inlezen vanuit de file `cmFname` met `cm.load` .

Er moeten vier stukken code worden ingevuld. Gebruik hiervoor de routines die je eerder hebt geïmplementeerd.

1) Secrets Het kan zijn dat er in het ingelezen bestand informatie bevat die geheim had moet blijven (dus niet mee verzonden had moeten worden). De variable `secretState` heeft dat er geen geheimen gelekt zijn.

2) Decrypt
Voor de opgegeven ontvanger/verzender controleer of het bericht decrypted kan worden. Zorg er voor dat het ontsleutelde bericht in `cm.mesg` komt te staan. Daarnaast moet aangegeven worden of voor elke ontvanger/verzender het bericht

leesbaar is.

Gebruik de hulproutines: `decryptSesKey` , `decryptMesg` en mogelijk andere.

3) CheckHash Controleer of the hash van `cm.mesg` juist is. Gebruik de hulproutines `checkHash` en mogelijk andere. De variable `hashState` geeft aan of de hash al dan niet correct is. Deze waarde is `None` als de hash niet controleerbaar is.

2) Verify Voor de opgegeven verzenders/ontvangers controleer of het bericht juist ondertekend is. Gebruik `cm.mesg` als het te controleren bericht, bedenk waar de signature is opgeslagen. Daarnaast moet aangegeven worden of het bericht voor elke ontvanger/verzender leesbaar is.

Gebruik de hulproutines: `verifyMesg` en mogelijk andere.

A) Return
Vul de dict `secretState` , `receiverState` , `hashState` en `senderState` voor elke opgegeven ontvangers (`receivers`) en verzenders (`senders`) met een van de volgende waarden:

- None: geen informatie beschikbaar om te decrypten/verifiëren
- True: operatie geslaagd.
- False: operatie gefaald.

De conversie van `bytes` naar `str` is voor je gedaan. Het bericht (`mesg`) en de vier states vormen het return-tuple

```
    return (mesg, receiversState, sendersState, hashState, secretState)
```

Inhoud AC2223b4Eo.zip

De opgave zip bevat de volgende files:

```
AC2223b4Eo.pdf
cryptomail.py
user1.prv user1.pub
user2.prv user2.pub user2.crt
user3.prv user3.pub
user4.prv user4.pub user4.crt
user5.prv user5.pub
user6.prv user6.pub
test0a.xmsg
test1a.xmsg
test2a.xmsg
test3a.xmsg
```

Starten

Volg de volgende richtlijn om de starten met de opgave.

Probeer eerst op papier te bedenken wat je allemaal moet doen. Maak een takenlijstje en teken het proces. Wat is de rol van de verzenders en ontvangers, Wat moet getekend worden en door wie, wat moet encrypted worden en door wie.

1. Creeer de `${studnr}.prv` , `${studnr}.pub` en `${studnr}.crt` files met openssl (of python)
2. Copier `cryptomail_skel.py` naar `cryptomail.py` , pas je naam en studnr aan.
3. Run `python3 cryptomail.py -f test0.xmsg decode` , dit zou moeten werken.
4. Vul de standaard methoden aan, na 13 juni kun je deze testen via CodeGrade.
5. Als deze standaard methoden goed werken, implementeer `decode` en dan `encode`

Testen

Het is natuurlijk van de belang dat je programma hebt getest voor je het inlevert. In de voorbeeld output staat `user?` voor een van de users.

Dit zijn de test-cases waaraan je moet voldoen.

Test 0

Bij deze test is er geen encryptie of signing of hashing. Deze test werkt zonder enige verandering aan `cryptomail_skel.py`.

In **vet font** het commando, in normal font de verwachte output.

```
$ python3 cryptomail.py -f test0a.xmsg decode
Decoded:file:      test0a.xmsg
Decoded:receivers:
Decoded:senders:
Decoded:hash:      no-info
Decoded:secrets:   no-info
Decoded:mesg:      Dit is de message.\n
```

De inhoud van de xmsg-file kun je zien door:

```
$ python3 cryptomail.py -f test0.xmsg info
{
    "mesg": "Dit is de message.\\n",
    "vers": "1.0"
}
```

Test 1

Door geen verzenders of ontvangers op te geven wordt alleen de hashing methodes getest. Kun dit op de volgende manier testen

```
$ python3 cryptomail.py -f test1.xmsg encode
Encoded:file:      test1.xmsg
Encoded:receivers:
Encoded:senders:
Encoded:mesg:      Dit is de message.\n
```

```
$ python3 cryptomail.py -f test1.xmsg decode
Decoded:file:      test1.xmsg
Decoded:receivers:
Decoded:senders:
Decoded:hash:      success
Decoded:secrets:   success
Decoded:mesg:      Dit is de message.\n
```

Test 2

Hier wordt 1 ontvanger (`user1`) getest.

```
$ python3 cryptomail.py -f test2.xmsg -r user1 -m test.txt encode
Encoded:file:      test2.xmsg
Encoded:receivers: user1=success
Encoded:senders:
Encoded:mesg:      Dit is de message.\n

$ python3 cryptomail.py -f test2.xmsg -r user1 decode
Decoded:file:      test2.xmsg
Decoded:receivers: user1=success
Decoded:senders:
Decoded:hash:      success
Decoded:secrets:   success
Decoded:mesg:      Dit is de message.\n
```

De file `test2a.xmsg` is een test-file om `decode` te testen, de secrets zijn niet verwijderd en er is geen hash-info.

Test 3

Hier wordt 1 verzender (`user3`) getest.

```
$ python3 cryptomail.py -f test3.xmsg -s user3 -m test.txt encode
Encoded:file:      test3.xmsg
Encoded:receivers:
Encoded:senders:   user3=success
Encoded:mesg:      Dit is de message.\n

$ python3 cryptomail.py -f test3.xmsg -s user3 decode
Decoded:file:      test3.xmsg
Decoded:receivers:
Decoded:senders:   user3=success
Decoded:hash:      success
Decoded:secrets:   success
Decoded:mesg:      Dit is de message.\n
```

De file `test3a.xmsg` is een test-file om `decode` te testen, de secrets zijn niet verwijderd en er is geen hash-info.

Test 4

Hiermee wordt 1 ontvanger en 1 verzender getest.

```
$ python3 cryptomail.py -f test4.xmsg -m test.txt  -r user1 -s user3 encode
Encoded:file:      test4.xmsg
Encoded:receivers: user1=success
Encoded:senders:   user3=success
Encoded:mesg:      Dit is de message.\n

$ python3 cryptomail.py -f test4.xmsg decode
Decoded:file:      test4.xmsg
Decoded:receivers: user1=success
Decoded:senders:   user3=success
Decoded:hash:      success
Decoded:secrets:   success
Decoded:mesg:      Dit is de message.\n
```

Test 5

Hier worden 2 ontvangers (user1,user2) getest.

```
$ python3 cryptomail.py -f test5.xmsg -m test.txt -r user1,user2 encode
Encoded:file:      test5.xmsg
Encoded:receivers: user1=success,user2=success
Encoded:senders:
Encoded:mesg:      Dit is de message.\n

$ python3 cryptomail.py -f test5.xmsg decode
Decoded:file:      test5.xmsg
Decoded:receivers: user1=success,user2=success
Decoded:senders:
Decoded:hash:      success
Decoded:secrets:   success
Decoded:mesg:      Dit is de message.\n
```

Test 6

Hiermee wordt 2 verzenders (`user3` en `user4`) getest.

```
$ python3 cryptomail.py -f test6.xmsg -m test6.txt -s user3,user4 encode
Encoded:file:      test6.xmsg
Encoded:receivers:
Encoded:senders:   user3=success,user4=success
Encoded:mesg:      Dit is de message.\n

$ python3 cryptomail.py -f test6.xmsg decode
Decoded:file:      test6.xmsg
Decoded:receivers:
Decoded:senders:   user3=success,user4=success
Decoded:hash:      success
Decoded:secrets:   success
Decoded:mesg:      Dit is de message.\n
```

Test 7

Hiermee wordt 2 ontvangers (`user1` en `user2`) en 2 verzenders (`user3` en `user4`) getest.

```
$ python3 cryptomail.py -f test7.xmsg -m test.txt -r user1,user2 -s user3,user4 encode
Encoded:file:      test7.xmsg
Encoded:receivers: user1=success,user2=success
Encoded:senders:   user3=success,user4=success
Encoded:mesg:      Dit is de message.\n

$ python3 cryptomail.py -f test7.xmsg decode
Decoded:file:      test7.xmsg
Decoded:receivers: user1=success,user2=success
Decoded:senders:   user3=success,user4=success
Decoded:hash:      success
Decoded:secrets:   success
Decoded:mesg:      Dit is de message.\n
```

Controle

Controleer of je overall tussen `# Student work }}` en `# Student work {{` de juiste code geschreven hebt.
Deze functies moet worden afgemaakt:

- loadPrvKey

- loadPubKey
- genSesKey
- genSesIv
- encryptSesKey
- decryptSesKey
- encryptMesg
- decryptMesg
- signMesg
- verifyMesg
- calcHash
- chckHash
- encode
- decode

Inleveren

In je programma heb je bij `__author__` je naam en `${studnr}` (studentnummer) ingevuld. Inlever deadline is **18 jun 2023 23:59**

Voor de volgende unix commando's uit om de files `${studNr}S.xmsg` en `${studNr}M.xmsg` te maken:

```
# jouw studnr gebruiken
$ studnr="123456789"
$ echo "Dit is de geheime tekst van ${studnr}." > ${studnr}.txt
$ python3 cryptomail.py -f ${studnr}S.xmsg -r user1 -s ${studnr} \
    -m ${studnr}.txt encode
$ python3 cryptomail.py -f ${studnr}M.xmsg -r user1,${studnr} -s user3,${studnr} \
    -m ${studnr}.txt encode
```

Het is wel handig als je een en ander controleert met:

```
$ python3 cryptomail.py -f ${studnr}S.xmsg decode
$ python3 cryptomail.py -f ${studnr}M.xmsg decode
```

Je levert in op Codegrade:

1. Certificaat (`${studnr}.crt)
2. HvaCryptoMesg-files (`${studnr}S.xmsg`) en (`${studnr}M.xmsg`)
3. Je programma (`cryptomail.py`)

Dus vier files inleveren !!

Succes