ΚΡΥΠΤΟΓΡΑΦΙΑ | FERNET ΚΑΙ X.509

ΣΥΜΜΕΤΡΙΚΗ ΚΡΥΠΤΟΓΡΑΦΗΣΗ

Πρακτικά Ζητήματα

Εισαγωγή: Η Fernet[1] είναι μια python recipe (συνταγή) με σκοπό να διευκολύνει τους προγραμματιστές σε Python να εφαρμόσουν κρυπτογράφηση και έλεγχο ταυτότητας στις εφαρμογές τους.

Τι είναι τα recipes στην Python; Οι συνταγές είναι μικρά σενάρια python που χρησιμοποιούνται για την επίλυση κοινών προβλημάτων.

Τι παρέχει η Fernet; Παρέχει συμμετρική κρυπτογράφηση και έλεγχο ταυτότητας στα δεδομένα. Αποτελεί μέρος της βιβλιοθήκης κρυπτογραφίας για την Python (cryptography module/package), η οποία έχει αναπτυχθεί από την Python Cryptographic Authority (PYCA). Η Fernet εγγυάται ότι ένα μήνυμα που έχει κρυπτογραφηθεί δεν μπορεί να χειριστεί ή να διαβαστεί χωρίς το κλειδί. Η Fernet είναι μια εφαρμογή συμμετρικής (επίσης γνωστής ως «μυστικό κλειδί») επαληθευμένης κρυπτογραφίας.

Βασική χρήση της Fernet (key generation and token creation):

```
# Filename: fernet.py
from cryptography.fernet import Fernet

key = Fernet.generate_key()
f = Fernet(key)
token = f.encrypt(b"my deep dark secret")

print(token)
print(f.decrypt(token))
```

key (bytes or str): Ένα κλειδί 32 bytes με κωδικοποίηση βάσης 64 (base64). Αυτό το κλειδί είναι το μυστικό κλειδί (private key). Το κλειδί αυτό αξιοποιείται για την κρυπτογράφηση και αποκρυπτογράφηση των μνημάτων.

generate_key(): Δημιουργεί ένα νέο κλειδί fernet. Εάν χαθεί τότε το μήνυμα δεν θα μπορεί να αποκρυπτογραφηθεί. Εάν διαρρεύσει τότε τα αντίστοιχα μηνύματα μπορούν να αποκρυπτογραφηθούν και θα μπορούν επίσης να πλαστογραφηθούν (με την ένταξη ψηφιακής υπογραφής).

Encrypt(): Κρυπτογραφεί τα δεδομένα που τοποθετούνται σαν όρισμα. Το αποτέλεσμα αυτής της κρυπτογράφησης είναι γνωστό ως "Fernet token (κουπόνι)" και έχει ισχυρές εγγυήσεις απορρήτου και γνησιότητας. Το κρυπτογραφημένο μήνυμα περιέχει την τρέχουσα ώρα που δημιουργήθηκε σε απλό κείμενο, επομένως η ώρα που δημιουργήθηκε ένα μήνυμα θα είναι ορατή σε έναν πιθανό εισβολέα.

Decrypt(): Αποκρυπτογραφεί ένα διακριτικό Fernet. Εάν αποκρυπτογραφηθεί επιτυχώς, θα λάβετε ως αποτέλεσμα το αρχικό απλό κείμενο, διαφορετικά θα δημιουργηθεί εξαίρεση. Είναι ασφαλές να χρησιμοποιήσετε αυτά τα δεδομένα καθώς η Fernet επαληθεύει ότι τα δεδομένα δεν έχουν παραποιηθεί πριν τα επιστρέψει.

MultiFernet(): Η MultiFernet αντίστοιχα εκτελεί όλες τις επιλογές κρυπτογράφησης χρησιμοποιώντας το πρώτο κλειδί στη λίστα που παρέχεται. Η MultiFernet προσπαθεί να αποκρυπτογραφήσει διακριτικά με κάθε κλειδί με τη σειρά του.

```
# Filename: multifernet.py
from cryptography.fernet import Fernet,
MultiFernet

key1 = Fernet(Fernet.generate_key())
key2 = Fernet(Fernet.generate_key())
f = MultiFernet([key1, key2])
token = f.encrypt(b"Secret message!"

print(token)
print(f.decrypt(token))
```

HMAC: Είναι δυνατή η χρήση κωδικών πρόσβασης με την αξιοποίηση Fernet. Για να το κάνετε αυτό, πρέπει να εκτελέσετε τον κωδικό πρόσβασης μέσω μιας συνάρτησης εξαγωγής κλειδιών όπως PBKDF2HMAC, bcrypt ή Scrypt.

```
# Filename: hmac.py
import base64
import os
from cryptography.fernet import Fernet
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.kdf.pbkdf2
import PBKDF2HMAC

password = b"password"
```

[1] github.com/pyca/cryptography/blob/main/src/cryptography/fernet.py

```
salt = os.urandom(16)
kdf = PBKDF2HMAC(algorithm=hashes.SHA256(),
length=32, salt=salt, iterations=390000)
key =
base64.urlsafe_b64encode(kdf.derive(password))
f = Fernet(key)
token = f.encrypt(b"Secret message!")
print (token)
print (f.decrypt(token))
```

Επεξήγηση: Σε αυτή την υλοποίηση το σχήμα και το salt πρέπει να αποθηκευτεί σε μια θέση με δυνατότητα ανάκτησης προκειμένου να εξαχθεί το ίδιο κλειδί από τον κωδικό πρόσβασης στο μέλλον. Ο αριθμός επαναλήψεων που χρησιμοποιείται θα πρέπει να ρυθμιστεί ώστε να είναι όσο υψηλότερος μπορεί να ανεχτεί ο διακομιστής σας. Μια καλή προεπιλογή είναι τουλάχιστον 480.000 επαναλήψεις (Django , July 2022).

Βασικές αρχές της Fernet: Η Fernet είναι βασισμένη πάνω σε μια σειρά από βασικές αρχές της κρυπτογραφίας. Συγκεκριμένα χρησιμοποιεί τα εξής:

- AES σε λειτουργία CBC με κλειδί 128-bit για κρυπτογράφηση. χρησιμοποιώντας padding PKCS7.
- HMAC χρησιμοποιώντας SHA256 για έλεγχο ταυτότητας.

Τα διανύσματα αρχικοποίησης δημιουργούνται χρησιμοποιώντας os.urandom(). Η Fernet είναι ιδανική για κρυπτογράφηση δεδομένων όταν υπάρχει περιορισμός στη μνήμη. Ως χαρακτηριστικό σχεδιασμού, δεν εκθέτει bytes χωρίς έλεγχο ταυτότητας. Αυτό σημαίνει ότι το πλήρες περιεχόμενο του μηνύματος πρέπει να είναι διαθέσιμο στη μνήμη, καθιστώντας το Fernet γενικά ακατάλληλο για πολύ μεγάλα αρχεία.

Πιστοποιητικά X.509

Τα πιστοποιητικά X.509 χρησιμοποιούνται για τον έλεγχο ταυτότητας πελατών και διακομιστών. Η πιο συνηθισμένη περίπτωση χρήσης είναι για διακομιστές ιστού που χρησιμοποιούν HTTPS. Τα πιστοποιητικά X.509 είναι μια γενική, εξαιρετικά ευέλικτη μορφή ως έννοια. Άλλες προσεγγίσεις είναι το SSL (τώρα γνωστό ως "TLS") που στην ουσία αξιοποιεί πιστοποιητικά X.509.

Certificate Signing Request (CSR): Κατά τη λήψη πιστοποιητικού από μια αρχή έκδοσης πιστοποιητικών (CA) εκτελούνται τα ακόλουθα βήματα:

1. Δημιουργείται ένα ζεύγος ιδιωτικού/δημόσιου κλειδιού.
2. Δημιουργείται ένα αίτημα για πιστοποιητικό, το οποίο υπογράφεται από το κλειδί σας (για να αποδείξετε ότι σας ανήκει αυτό το κλειδί).
3. Δίνετε την CSR σας σε μια Αρχή Πιστοποίησης – CA (αλλά όχι το ιδιωτικό κλειδί).
4. Η αρχή έκδοσης πιστοποιητικών (CA) επικυρώνει ότι σας ανήκει ο πόρος (π.χ. τομέας) για τον οποίο θέλετε ένα πιστοποιητικό.
5. Η αρχή αρχής σάς δίνει ένα πιστοποιητικό, υπογεγραμμένο από αυτούς, το οποίο προσδιορίζει το δημόσιο κλειδί σας και τον πόρο για τον οποίο έχετε πιστοποιηθεί.
6. Ρυθμίζετε τις παραμέτρους του διακομιστή σας ώστε να χρησιμοποιεί αυτό το πιστοποιητικό, σε συνδυασμό με το ιδιωτικό σας κλειδί, για την κυκλοφορία διακομιστή.

Εάν θέλετε να αποκτήσετε ένα πιστοποιητικό από μια τυπική CA, αρχικά, θα χρειαστεί να δημιουργήσετε ένα ιδιωτικό κλειδί (ακολουθεί παράδειγμα).

```
# Filename: private-key.py
from cryptography.hazmat.primitives import
serialization
from cryptography.hazmat.primitives.asymmetric
import rsa
# Generate our key
key = rsa.generate_private_key(
    public_exponent=65537,
    key_size=2048,
)
# Write our key to disk for safe keeping
with open("key.pem", "wb") as f:
    f.write(key.private_bytes(
        encoding=serialization.Encoding.PEM,

format=serialization.PrivateFormat.TraditionalO
penSSL,

encryption_algorithm=serialization.BestAvailabl
eEncryption(b"passphrase"),
    ))
```

Εάν έχετε ήδη δημιουργήσει ένα κλειδί, μπορείτε να το φορτώσετε με το load_pem_private_key(). Στη συνέχεια πρέπει να δημιουργήσουμε ένα αίτημα υπογραφής πιστοποιητικού (CSR). Μια τυπική αίτηση περιέχει τα εξής:

- Πληροφορίες σχετικά με το δημόσιο κλειδί μας (συμπεριλαμβανομένης της υπογραφής ολόκληρου του σώματος).
- Πληροφορίες για την ταυτότητα.
- Πληροφορίες σχετικά με τους τομείς για τους οποίους προορίζεται αυτό το πιστοποιητικό.

```
# Filename: csr-generation.py
from cryptography import x509
from cryptography.x509.oid import NameOID
from cryptography.hazmat.primitives import hashes

# Generate a CSR
csr = x509.CertificateSigningRequestBuilder().subject_name(x509.Name([
    # Provide various details about who we are.
    x509.NameAttribute(NameOID.COUNTRY_NAME, u"US"),
    x509.NameAttribute(NameOID.STATE_OR_PROVINCE_NAME, u"California"),
    x509.NameAttribute(NameOID.LOCALITY_NAME, u"San Francisco"),
    x509.NameAttribute(NameOID.ORGANIZATION_NAME, u"My Company"),
    x509.NameAttribute(NameOID.COMMON_NAME, u"mysite.com"),
])).add_extension(
    x509.SubjectAlternativeName([
        # Describe what sites we want this certificate for.
        x509.DNSName(u"mysite.com"),
        x509.DNSName(u"www.mysite.com"),
        x509.DNSName(u"subdomain.mysite.com"),
    ]),
    critical=False,

# Sign the CSR with our private key.
).sign(key, hashes.SHA256())

# Write our CSR out to disk.
with open("path/to/csr.pem", "wb") as f:
    f.write(csr.public_bytes(serialization.Encoding.PEM))
```

Ενώ τις περισσότερες φορές θέλετε ένα πιστοποιητικό που έχει υπογραφεί από κάποιον άλλο (δηλαδή μια αρχή έκδοσης πιστοποιητικών), έτσι ώστε να εδραιωθεί η εμπιστοσύνη, επιτρέπεται να δημιουργήσετε ένα πιστοποιητικό που υπογράφεται από τον εαυτό σας. Τα αυτο-υπογεγραμμένα πιστοποιητικά δεν εκδίδονται από αρχή έκδοσης πιστοποιητικών, αλλά υπογράφονται από το ιδιωτικό κλειδί (self-signed) που αντιστοιχεί στο δημόσιο κλειδί που ενσωματώνουν.

```
# Filename: certification-generate.py
# Various details about who we are. For a self-signed certificate the
# subject and issuer are always the same.
subject = issuer = x509.Name([
    x509.NameAttribute(NameOID.COUNTRY_NAME, u"US"),
    x509.NameAttribute(NameOID.STATE_OR_PROVINCE_NAME, u"California"),
    x509.NameAttribute(NameOID.LOCALITY_NAME, u"San Francisco"),
    x509.NameAttribute(NameOID.ORGANIZATION_NAME, u"My Company"),
    x509.NameAttribute(NameOID.COMMON_NAME, u"mysite.com"),
])
cert = x509.CertificateBuilder().subject_name(
    subject
).issuer_name(
    issuer
).public_key(
    key.public_key()
).serial_number(
    x509.random_serial_number()
).not_valid_before(
    datetime.datetime.utcnow()
).not_valid_after(
    # Our certificate will be valid for 10 days
    datetime.datetime.utcnow() + datetime.timedelta(days=10)
).add_extension(

x509.SubjectAlternativeName([x509.DNSName(u"localhost")]),
    critical=False,
# Sign our certificate with our private key
).sign(key, hashes.SHA256())
# Write our certificate out to disk.
with open("path/to/certificate.pem", "wb") as f:
    f.write(cert.public_bytes(serialization.Encoding.PEM))
```

ΠΑΡΑΔΟΤΕΟ 06 (fernet, multifernet, private-key, csr-feneration, certification-generate)

Εκτελέστε τις εντολές δημιουργώντας ένα αρχείο .py και **σχολιάστε σε κάθε γραμμή τι συντελείται.**

Υποχρεωτικά! Προσθέστε **δικές σας τιμές** σε κάθε περίπτωση. Ανεβάστε το αρχείο py.