Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica Per Il Management

Autenticazione nelle applicazioni Web, analisi di JWT, PASETO e Macaroons nel framework OAuth2.0

Relatore:
Chiar.mo Prof.
Di Iorio Angelo

Presentata da:
Rocca Claudio

Sessione II
Anno Accademico 2023-2024

Introduzione

Negli ultimi anni, l'autenticazione basata su token ha guadagnato crescente popolarità grazie alla sua efficienza, scalabilità e flessibilità, specialmente in contesti di applicazioni distribuite e sistemi moderni. Tra le tecnologie più diffuse per la gestione dell'autenticazione basata su token si trovano JWT[1] (JSON Web Token), PASETO[2] (Platform-Agnostic Security Tokens) e Macaroons[3].
L'uso di token porta vantaggi come la scalabilità in ambienti cloud, dove le applicazioni possono essere distribuite su più server senza la necessità di mantenere una sessione centralizzata. Diversamente dai sistemi session-based, dove le sessioni devono essere memorizzate e gestite a livello di server, i token vengono generati e conservati dal client, o da altri componenti del sistema, consentendo una maggiore libertà e una riduzione del carico di gestione sul backend, in termini di tempo di risposta e di memoria utilizzata.
Ancora, l'utilizzo di *cookie* per l'autenticazione è tipicamente stateful, in quanto richiede che le informazioni di sessione vengano mantenute lato server. Ciò comporta infatti un aumento del carico di gestione e di memoria per il sistema.
È da notare però come non tutti gli utilizzi delle sessioni possono essere sostituiti da token, un esempio può essere un ambiente in cui un utente deve effettuare il logout, operazione che diventa difficile eseguire attraverso l'utilizzo di token, come verrà approfondito successivamente.
Questo rende l'autenticazione basata su token meno adatta a contesti in cui la revoca immediata dell'accesso è un requisito critico.

Al contrario di sessioni e cookie, i token che verranno presentati nel corso dell'elaborato sono stateless, non è dunque richiesto che il server tenga traccia di ogni sessione attiva, favorendo una migliore implementazione del paradigma REST. Il paradigma REST (REpresentational State Transfer) identifica le risorse di una applicazione tramite URI, e descrive come le richieste del client deveono contenere tutte le informazioni necessarie per essere processate, senza che il server debba memorizzare dati relativi alla sessione in corso.
L'utilizzo di sistemi di autenticazione token based favorisce inoltre l'utilizzo di pattern architetturali volti ad introdurre componenti intermedi nella comunicazione tra gli utenti e il server, come il pattern *proxy*. Questo si può tradurre per esempio nell'utilizzo di un componente incaricato di verificare la validità dei token inviati dagli utenti, riducendo il carico di lavoro del server, e aumentando la sicurezza e le performance del sistema in cui si opera.
L'elaborato descrive l'utilizzo di token per l'autenticazione, illustrando come questi siano utilizzati per garantire la sicurezza del sistema. Vengono forniti esempi pratici di costruzione, validazione e gestione di questi token, insieme a confronti delle loro caratteristiche in termini di sicurezza, semplicità di implementazione, performance e diffusione.
Viene approfondito l'utilizzo del framework OAuth2.0, largamente utilizzato per gestire la condivisione di informazioni in modo sicuro tra servizi, e di come questo possa essere implementato attraverso l'utilizzo dei vari token presentati nel corso dell'elaborato.
Un'analisi specifica viene dedicata alle principali vulnerabilità identificate nella OWASP Top 10, lista contenente le più comuni falle nei sistemi di sicurezza delle Web application. L'elaborato esplora come i token possono contribuire a mitigare queste minacce, rendendo l'autenticazione e l'autorizzazione più sicure in ambienti distribuiti. Sono presi in analisi altri aspetti fondamentali nell'utilizzo di queste tecnologie, come la facilità d'uso e la diffusione di librerie per i principali linguaggi di programmazione.
È stato inoltre sviluppato un client per ottenere dei dati relativi al tempo di generazione e di validazione di ogni token. Questo client è in grado di emettere token contenenti i principali claim utilizzati, e di controllare se un token è valido, con approssimazioni dovute al contesto astratto in cui si opera.
Viene infine approfondito l'utilizzo di sistemi di autenticazione token based attraverso i protocolli HTTP (HyperText Transfer Protocol) e HTTPS (HyperText Transfer Protocol Secure), e di quali misure siano da prendere in entrambi gli ambienti per garantire la sicurezza del sistema.

In questo elaborato non verrà analizzato l'uso dei token SAML (Security Assertion Markup Language). Questi token sono largamente utilizzati in ambienti enterprise, specialmente per il SSO (single sign-on) tra vari servizi, ma la loro struttura verbosa data dall'utilizzo di XML non li rende adatti all'utilizzo in

applicazioni che vogliono garantire buone prestazioni in termini di velocità e sicurezza.
Il focus di questo elaborato è pertanto rivolto su soluzioni moderne come JWT, PASETO e Macaroons. Tali tecnologie offrono vantaggi significativi in termini di velocità, leggerezza e flessibilità, caratteristiche che le rendono adatte all'utilizzo in applicazioni mobile, API RESTful e architetture cloud.
Le soluzioni basate su token come quelle trattate nell'elaborato consentono una gestione più efficiente dell'autenticazione in contesti distribuiti e senza stato, riducendo il carico sui server e migliorando le performance in ambienti scalabili.
Pertanto, l'obiettivo è fornire una panoramica dettagliata di tecnologie che si adattano meglio alle esigenze delle applicazioni moderne, dove la compattezza dei token favorisce migliori performance rispetto a SAML.

In sintesi, questo lavoro offre una panoramica dettagliata sull'uso dei token per l'autenticazione, proponendo le tecnologie JWT, PASETO e Macaroons come token da utilizzare all'interno del framework di autenticazione OAuth2.0, ed evidenziando i loro vantaggi rispetto ai sistemi session-based, approfondendo come il loro utilizzo possa prevenire alcune delle più comuni vulnerabilità nel panorama della sicurezza informatica.

Indice

Elenco delle figure

Elenco delle tabelle

1 Strumenti di autorizzazione Token-based

1.1 JWT

JWT(Json Web Token)[1] è uno dei principali metodi utilizzati per autorizzare gli utenti all'interno di una applicazione Web, con lo scopo di verificare l'identità e i permessi che un utente possiede. JWT implementa lo standard JOSE (JavaScript Object Signing and Encryption). Questo tipo di token contiene informazioni codificate in Base64url e firmate per garantire l'identità dell'utente. La struttura compatta lo rende un ottimo metodo per scambiare informazioni in ambiti dove la memoria a disposizione è limitata, come le chiamate HTTP, dove può essere utilizzato come header per l'autorizzazione, o salvato nei cookie del browser per consentire un facile accesso all'utente.
I JWT utilizzano dei dati in formato chiave/valore definiti *claim*, per comunicare diversi dati riguardanti il token, i principali claim utilizzati saranno approfonditi in seguito.
I JWT offrono diversi vantaggi rispetto ai sistemi di autorizzazione basati su sessioni con session ID. La differenza più importante rispetto alle sessioni è che questo tipo di token è stateless, il che significa che non è necessario memorizzare sessioni lato server, riducendo il carico sulla memoria del server e il numero di query effettuate sul database. Questo ne facilita l'utilizzo su applicazioni cloud distribuite, eliminando la necessità di contenere le informazioni sulle sessioni in ogni nodo della rete del sistema, e consentendo l'autenticazione ai vari componenti della rete senza la necessità di sincronizzare i dati della sessione. Tale caratteristica, unita alla ridotta dimensione di ogni token, rende JWT un ottimo candidato per l'utilizzo in applicazioni con architettura a microservizi, dove più componenti necessitano di comunicare in maniera rapida e sicura.
La caratteristica che viene più approfondita nell'analisi di JWT è l'utilizzo che questo tipo di token fa degli algoritmi di firma: lo standard JOSE permette di scegliere l'algorirtmo da utilizzare per la firma dei token, per consentire agli sviluppatori flessibilità in fase di implementazione. Delegare all'utilizzatore la scelta degli algoritmi di firma può però portare con sè problemi derivati da errate configurazioni del sistema, come la scelta di algoritmi poco performanti, o una cattiva configurazione dei componenti incaricati di verificare la validità di un token.
Rispetto all'utilizzo di sistemi basati su sessioni, JWT presenta anche altri svantaggi, che includono l'irrevocabilità e la difficile modifica di un token rilasciato, caratteristiche che verranno approfondite successivamente.
Un token JWT può essere di due tipi differenti: JWS e JWE[4], di seguito vengono rappresentate le principali caratteristiche di ognuno.

Token	**Descrizione**
JWS (JSON Web Signature)	Token firmato digitalmente che garantisce l'integrità dei dati. Non è criptato, quindi i dati sono visibili ma la firma assicura che non siano stati alterati.
JWE (JSON Web Encryption)	Token criptato che protegge la riservatezza dei dati. Oltre a essere criptato, può anche essere firmato per garantire sia la riservatezza che l'integrità.

Tabella 1: Descrizione dei tipi di JWT

L'utilizzo di token firmati (JWS) è raccomandato per ambienti in cui le informazioni contenute nel token non sono di particolare importanza, e possono dunque circolare senza il rischio di essere intercettate. Un JWS garantisce la sua integrità attraverso l'utilizzo di una firma, che certifica l'identità del chiamante e l'integrità del token.
Un token criptato è un JWE, questo tipo di token garantisce una maggiore sicurezza, in quanto le informazioni in esso contenute sono leggibili solamente da chi possiede la chiave necessaria per decifrarle. Questo tipo di token è consigliato per l'utilizzo in casi in cui il token debba necessariamente contenere informazioni sensibili, o più in generale per l'utilizzo in ambienti ritenuti non sicuri.

È inoltre da riportare l'esistenza di un terzo tipo di JWT, i cosiddetti *Unsecured tokens*, cioè quelli che non contengono firma, ma riportano solamente informazioni codificate in Base64url. L'utilizzo di questi token è da evitare, poichè facilmente manipolabili da chiunque. In fase di implementazione di un sistema di difesa, è necessario non autorizzare chiunque effettui una richiesta di autenticazione utilizzando questo tipo di token.
In questo capitolo verranno analizzate la struttura e l'utilizzo dei JWS, successivamente quando sarà discusso l'utilizzo di JWT in ambienti non sicuri sarà approfondita la struttura di un JWE.

Struttura
Il token è composto da una sequenza di parametri codificati in Base64url separati da un punto, il numero di parametri può variare in base alla struttura e all'utilizzo del token.
Nei casi più comuni la struttura di un token è la seguente:

```
<header>.<payload>.<firma>
```

In seguito ogni componente del token sarà approfondita per mostrarne il contenuto e la costruzione.

Un token viene tipicamente rilasciato in seguito alla autenticazione di un utente con credenziali come username e password, il server crea un JWS contenente le informazioni necessarie e lo invia nella risposta all'utente. Per essere autorizzato l'utente dovrà aggiungere il token come header per ogni successiva richiesta effettuata.
Il processo appena discusso è illustrato nella seguente figura:

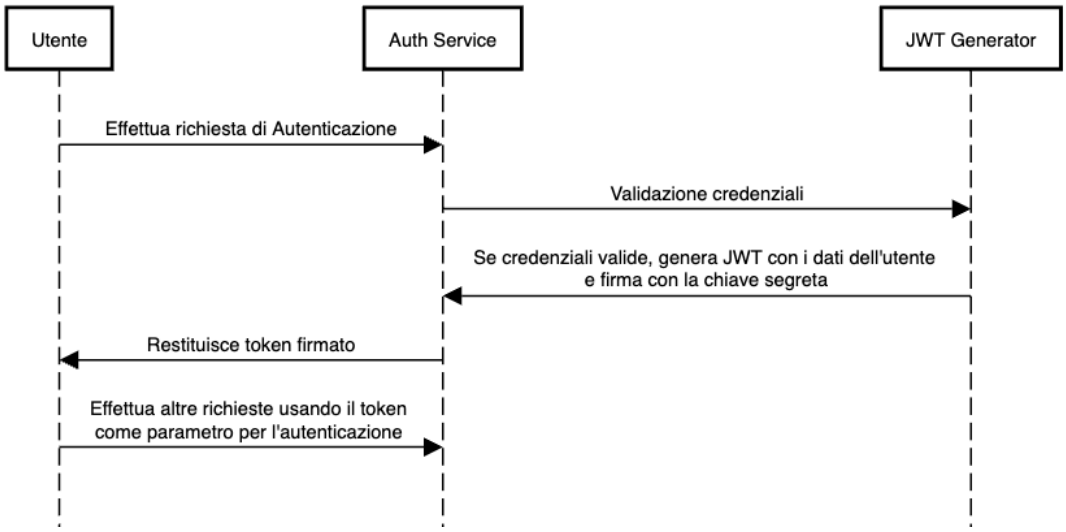

Figura 1: Diagramma di sequenza della creazione di un JWT

Il diagramma riportato illustra il processo di creazione di un JWS firmato con un algoritmo a chiave asimmetrica, vengono dunque utilizzate una coppia di chiavi, una pubblica e una privata, per firmare e verificare la validità del token.
Questo sistema implementa il pattern architetturale *proxy*, che aggiunge un servizio intermedio per gestire le richieste dell'utente prima che queste arrivino al server. L'utilizzo di un componente intermedio permette di suddividere meglio il carico di lavoro, per esempio attraverso l'utilizzo di un load balancer che inoltra le richieste ai diversi server presenti nel sistema.
Nell'esempio riportato, è presente un servizio incaricato di gestire l'autenticazione degli utenti, esso inoltra al server la richiesta di creazione di un nuovo token, e, nel caso in cui nella richiesta ricevuta sia già presente un token, ne verifica la validità e dialoga con il server per restituire all'utente quanto richiesto.
Quando viene effettuata una richiesta di autenticazione, questa viene reindirizzata verso un sevizio incaricato di gestire le richieste dell'utente, chiamato Auth Service. Questo servizio inoltra le credenziali

dell'utente al server per verificarne la validità. In caso di esito positivo, il server restituisce il token contenente le informazioni dell'utente firmato con la chiave privata, altrimenti restituisce un errore e l'utente non viene autorizzato. Il token generato viene dunque inoltrato all'utente che potrà utilizzarlo come header di autorizzazione per le successive richieste.
Quando l'auth service riceve una richiesta già contenente un JWS, per cui non è dunque necessario generarne uno nuovo, sarà direttamente questo componente a verificare la validità del token senza dialogare con il server. Questo sistema favorisce il disaccoppiamento delle componenti, delegando i compiti a più parti invece di gestire tutto tramite un server che si troverebbe a dover svolgere vari compiti, compromettendo le presazioni del sistema.
Di seguito viene riportato approfondimento sulle componenti di un token, con un esempio di costruzione di un JWS con l'utilizzo di chiave segreta condivisa.

Header
L'header contiene informazioni relative al tipo di token e all'algoritmo di firma utilizzato per generare il token. Il tipo di token viene dichiarato con l'utilizzo del claim "typ", mentre l'algoritmo utilizzato viene specificato nel claim "alg", che deve contenere come valore la sigla corrispondente ad uno degli algoritmi descritti nello standard JWT. La scelta dell'algoritmo utilizzato per la firma firma viene delegata al responsabile del contesto in cui si opera, in modo che sia possibile scegliere quello più adatto alle caratteristiche del sistema cercando il giusto compromesso tra sicurezza e facilità d'uso. Questo aspetto è la causa di numerosi problemi imputati a JWT, e sarà oggetto di successive analisi volte a elencare i possibili danni che la scelta di un algoritmo non adatto, o una cattiva configurazione del sistema di decodifica dei token possono portare.

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

I token senza firma, i già citati Unsecured Token, riportano come come claim "alg" la seguente coppia chiave/valore:

```
"alg": none
```

Questi token sono accettati dallo standard JOSE, ma in qualunque sistema con dei requisiti di sicurezza da rispettare devono essere ignorati, a causa della loro scarsa protezione e della facilità di replica da parte di attaccanti.
Per formare il primo parametro del token finale, l'header viene codificato in Base64url, la codifica dell'esempio riportato restituisce il seguente valore:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
```

A questa stringa viene concatenato un punto (".") per indicare la fine dell'header e l'inizio del payload. L'header codificato sarà utilizzato per calcolare l'HMAC(Hash-based Message Authentication Code) del token, in modo da verificarne l'integrità in fase di decodifica.

Payload

Il payload di un token contiene le informazioni che l'utilizzatore vuole comunicare per certificare la propria identità. Contiene in genere alcuni dei claim riservati definiti nello standard JWT[1, par. 4.1]. Un claim riservato non può essere utilizzato per altri scopi rispetto a quelli definiti nello standard, per evitare incomprensioni in fase di decodifica dei token. Di seguito viene riportato un elenco contenente alcuni dei claim riservati e il loro utilizzo all'interno di un JWT

Claim	Descrizione
`iss`	Indica chi ha rilasciato il token
`sub`	Utilizzato per dichiarare a che utente si riferisce il token in oggetto
`aud`	Indica chi deve essere il recipiente dell'oggetto
`iat`	Contiene lo Unix Epoch Time del momento in cui è stato rilasciato il token
`nbf`	Indica lo Unix Epoch Time di inizio di validità del token
`exp`	Contiene lo Unix Epoch Time della scadenza di validità del token
`jti`	Identificatore univoco del token

Tabella 2: Claim riservati nello standard JWT

Il payload può inoltre contenere altre coppie di claim non descritte nello standard JWT, i cosiddetti *custom claims*, ovvero coppie chiave/valore specifiche del sistema in cui si opera.
Questi claim possono essere utilizzati per aggiungere informazioni al token che non sono esprimibili attraverso l'utilizzo di claim riservati.
Proseguendo con la costruzione di un JWS, il payload che sarà utilizzato per formare il token di esempio è il seguente:

```
{
  "exp": 1730246399,
  "sub": "User123",
  "iat": 1728594632
}
```

Il claim "sub" indica dunque che questo token fa riferimento ad un utente il cui username è "User123", il claim "iat" indica che il token diventa valido alla mezzanotte del 29 ottobre 2024, mentre il claim "exp" indica che il token perderà la validità alle 23:59 del 29 ottobre 2024. L'utilizzo di claim come "iat", "exp" o "nbf" è fondamentale per garantire la sicurezza del sistema: i token JWT sono stateless, mantengono infatti la loro validità anche dopo il loro utilizzo se non altrimenti specificato. Limitarne la validità temporale garantisce che un token sia valido strettamente solo per l'arco di tempo necessario, ed è uno dei metodi di invalidazione di un token che verranno approfonditi in seguito.
Il contenuto del payload viene codificato in Base64url e concatenato all'encoding dell'header, in questo modo:

```
base64UrlEncode(header) + "." + base64UrlEncode(payload)
```

Dalla concatenazione di header e payload codificati si ottiene la seguente stringa, da precisare come i riporti a capo contenuti nella stringa siano stati inseriti solamente per visualizzare correttamente i dati, e non facciano parte del token.

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJleHAiOjE3MzAyND
YzOTksInN1YiI6IlVzZXIxMjMiLCJpYXQiOjE3Mjg1OTQ2MzJ9
```

È bene notare, come citato in precedenza, che il payload di un JWS è codificato in Base64url ed è dunque leggibile da chiunque ottiene il token, è pertanto di fondamentale importanza non includere informazioni sensibili nel payload.
Nel caso questo non fosse possibile per requisiti del sistema in cui si opera, è consigliato dallo standard utilizzare protocolli che fanno uso di comunicazioni crittografate, o ricorrere ad un JWE.

Firma
La firma garantisce che il mittente sia effettivamente chi viene dichiarato nel payload, e garantisce che il token non sia stato modificato durante il trasferimento.
L'utilizzo di una chiave simmetrica segreta presuppone che questa sia di una complessità sufficiente, e che sia stata precedentemente condivisa tra le parti in modo sicuro. Nel caso di una API invocata da un utente, la chiave utilizzata per la firma viene mantenuta solamente all'interno del server e mai inviata direttamente.
La firma di un token viene generata a partire dalla stringa ricavata dall'encoding dell'header e del payload, e da una chiave segreta di una lunghezza prefissata, questo metodo permette al sistema di controllare se il contenuto del token è stato modificato durante la trasmissione, proteggendosi dagli attacchi chiamati *man in the middle*. Questo attacco viene effettuato da una entità che si intromette nella comunicazione tra due parti, per intercettare e/o modificare le informazioni scambiate.
È da notare come l'utilizzo di una singola firma per la creazione di token per più utenti possa esporre a pericoli come attacchi di replica. Per garantire maggiore sicurezza, è dunque necessario implementare sistemi di rotazione delle chiavi che vengono usate per la firma dei token.
La chiave utilizzata per la firma del token può essere indicata tramite l'utilizzo del claim "kid". Questo claim deve contenere un identificativo che il server associa alla chiave da utilizzare per decodificare il token.
Un altro potenziale problema è che, a partire da dati identici, il token generato risulta sempre lo stesso. In altre parole, se due coppie di header e payload sono identiche e vengono firmate con la stessa chiave, i token risultanti saranno identici. Questa situazione espone il sistema a potenziali attacchi di replica: un utente che riesce a ottenere un token già utilizzato potrebbe riutilizzarlo senza alcuna modifica. Se non si gestisce correttamente questa eventualità, un utente non autorizzato potrebbe ottenere accesso al sistema. Per proteggersi da questi pericoli, è possibile inserire all'interno del token un identificativo univoco, anche attraverso l'utilizzo del claim riservato "jti", facendo ciò ci si assicura che anche ricevendo due richieste uguali, i token generati saranno differenti.
Continuando nella costruzione del token di esempio, viene di seguito riportata la chiave utilizzata per la firma del JWS, composta da 32 caratteri generati casualmente:

```
yJojIqZismADFUmhEjgB9NJxh20JpP4d
```

Alla stringa codificata in Base64url ricavata dall'encoding di header e payload viene applicato l'algoritmo di firma dichiarato nell'header del token, in questo caso HS256, un algoritmo a chiave simmetrica che usa come chiave condivisa una stringa di almeno 256 bit.
L'algoritmo appena descritto si può rappresentare in questo formato:

```
HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload),
chiave)
```

Il token finale generato utilizzando i dati riportati in precedenza è il seguente:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJle
HAiOjE3MzAyNDYzOTksInN1YiI6IlVzZXIxMjMiLCJpYXQi
OjE3Mjg1OTQ2MzJ9.4hV3kzCOgntWiCoJndb48m_ZStF46c9dc-6UoDxZfq0
```

La firma può infine, anche essere a sua volta codificata in Base64url prima di essere aggiunta al token.

È importante sottolineare come lo standard dichiari che la firma è opzionale in un token, possono essere infatti creati dei cosiddetti *'Unsecured JWT'*, ovvero token che riportano come claim "alg" la seguente coppia chiave/valore:

```
"alg": "none"
```

Un token di questo tipo non possiede la firma e termina con il punto (".") concatenato al termine del payload codificato in Base64url.
Per garantire un buon livello di sicurezza, un client incaricato di decodifcare un token deve poter individuare questa casistica e scartare qualsiasi token non firmato.[1][par. 8]

Valdazione di un token
Il controllo della validità del token parte dalla decodifica da Base64url a testo in chiaro dell'header, per identificare con che algoritmo è stato firmato il token. L'algoritmo dichiarato nel token viene dunque confrontato con quelli accettati dal sistema, se esso è presente si può continuare con la decodifica, altrimenti la richiesta viene scartata.
Il sistema utilizza la chiave necessaria per decifrare il token, a seconda dell'algoritmo utilizzato e della chiave utilizzata, il cui riferimento può essere contenuto nel claim "kid".
La firma del token viene dunque decodificata da Base64url a testo cifrato, e successivamente viene ricavato il testo in chiaro applicando l'algoritmo di hashing.
Se il testo ottenuto corrisponde alla stringa ricavata dalla concatenazione di header e payload codificati in Base64, il token viene considerato valido e si procede con la analisi del contenuto del payload.
Questo processo fornisce i seguenti vantaggi:

- Garantisce l'identità del chiamante, poichè si presuppone che solo lui sia in grado di firmare il token con la chiave segreta condivisa
- Garantisce che le informazioni contenute nel token non siano state modificate durante il transito: se il payload è stato modificato, l'encoding header e payload non corrisponde con il testo ottenuto dalla decodifica della firma del token.

In sintesi, l'utilizzo di token JWT per gestire l'autenticazione consente di ridurre il carico con il server limitando il numero di operazioni da compiere. L'utilizzo di questi token permette inoltre di disaccoppiare le componenti incaricate di gestire l'autenticazione e processare le richieste in arrivo.
I token JWT, nella loro variante JWS, comunicano i dati del payload in chiaro, garantendo l'integrità del payload attraverso algoritmi di firma. JWT permette inoltre di gestire in modo flessibile l'utilizzo degli algoritmi di crittografia, lasciando agli sviluppatori la scelta in base alle loro necessità e alle caratteristiche del sistema. Questa caratteristica è uno dei punti di forza di JWT, ma può portare ad errori di configurazione del sistema, ed è dunque necessario definire regole chiare da adottare in fase di emissione e di lettura dei token.

1.2 PASETO

PASETO (Platform-Agnostic Security Tokens)[2] è un meccanismo di autenticazione token-based alternativo ai JWT, che si pone l'obiettivo di migliorare la sicurezza nello scambio di dati tra due parti. A differenza dei JWT, i PASETO garantiscono una sicurezza predefinita, non lasciando all'utente la scelta dell'algoritmo di firma. Questi token utilizzano infatti uno standard condiviso per minimizzare i rischi legati a scelte di algoritmi non sicuri o cattive configurazioni del client incaricato di decodificare il token. Similmente ai JWT, i PASETO contengono informazioni strutturate in formato JSON e codificate in Base64url, i componenti presenti in un token comprendono obbligatoriamente un header, un payload, e un footer opzionale, la struttura di un PASETO è dunque la seguente:

```
<v[versione]>.<tipo_token>.<payload>.<footer_opzionale>
```

Il contenuto dell'header di un PASETO è ciò che lo differenzia maggiormemte da un JWT: l'header non include riferimenti espliciti all'algoritmo di firma utilizzato, ma solo due elementi, la versione del token e la tipologia dello stesso. Il parametro "version" fa riferimento alla versione del token in uso, le specifiche tecniche di questo token vengono infatti aggiornate, a differenza dello standard JOSE dei JWT. Attualmente, l'ultima versione disponibile è la quarta, i valori che questo campo può assumere sono dunque "v1", "v2", "v3" o "v4", la versione di un token indica quali algoritmi di firma vengono utilizzati in fase di creazione e di lettura del token.
Il tipo indica che algoritmo viene utilizzato per la firma del token, e di conseguenza come questo sarà utilizzato.
Il payload del token contiene le informazioni che l'utente deve comunicare al servizio per essere autorizzato, ed è strutturato in maniera simile a quello dei JWT, ovvero con dati in formato Json che vengono firmati oppure cifrati, a seconda del tipo di token che si sta utilizzando. Le specifiche di PASETO definiscono gli stessi claim riservati dello standard JWT discussi precedentemente, possono essere dunque utilizzati gli stessi claim per comunicare informazioni riguardanti l'utente che si vuole autenticare, oppure claim che limitano la validità temporale del token in oggetto.
L'unica differenza tra i claim riservati di JWT e quelli presenti nelle specifiche di PASETO è che i claim che limitano la validità temporale del token (ovvero "nbf", "iat" ed "exp") riportano la data in formato ISO 8601, e non utilizzano dunque Unix Epoch Time.
È da notare come questi claim diventino pressochè superflui in sistemi in cui i token sono monouso. Limitare la validità temporale del token può essere utile in sistemi che utilizzano sessioni server-side per garantire l'identità dell'utente, oppure nei casi in cui il token non viene utilizzato subito dopo la creazione, ma più avanti nel tempo.

Ogni tipo di token può contenere, in aggiunta al payload, un footer che contiene informazioni non firmate, ma semplicemente codificate in Base64url. Il footer non influisce sulla validità del token, non viene infatti utilizzato per determinare se la richiesta deve essere elaborata o meno.
In un sistema che ruota le chiavi di firma dei token, questo footer può essere utilizzato per comunicare quale chiave è stata utilizzata per la firma. Per questo scopo si rende utile il claim riservato "kid", a cui si può associare l'identificativo univoco della chiave mantenuta in memoria dal server.
Nel footer può essere inoltre contenuto un PASERK, tipo di dato utilizzato per la comunicaziioni di dati relativi alla chiave utilizzata per la firma, questo meccanismo sarà approfondito in seguito.
Siccome le informazioni contenute nel footer non sono firmate ma semplicemente codificate in Base64url, è fondamentale che esso non contenga informazioni sensibili, la cui fuoriuscita può danneggiare l'utente o la sicurezza del sistema.

Lo standard PASETO prevede due tipologie di token, *local* e *public*. La scelta del tipo di token dipende dal contesto in cui si opera e determina l'algoritmo di firma che verrà utilizzato per la costruzione e la lettura dei token. La tabella sottostante sintetizza le caratteristiche di entrambi i tipi di token.

Tipo di Token	**Algoritmi di Firma**	**Caratteristiche**	**Casi d'Uso**
Local	Algoritmi a chiave simmetrica, chiave privata condivisa	Token cifrati con chiavi simmetriche. Ideale per sistemi che possono mantenere chiavi segrete sicure. Il payload è cifrato.	Comunicazione tra server interni, token monouso, sessioni lato server.
Public	Algoritmi a chiave asimmetrica, coppia di chiave pubblica/chiave privata	Token firmati con chiavi asimmetriche. Il payload è visibile, quindi non adatto per informazioni sensibili. Solo la firma garantisce l'integrità.	Autenticazione su sistemi esterni, token firmati verificabili pubblicamente senza condividere chiavi private.

Tabella 3: Caratteristiche token PASETO di tipo Local e Public

Nei paragrafi seguenti viene approfondita la struttura di ogni token, riportando per entrambi i tipi un esempio di costruzione di un token. I token riportati negli esempi sono stati costruiti utilizzando la popolare libreria *pyseto*[5], che permette di costruire, firmare e decodificare dei PASETO in python.

Local

I token "local" vengono firmati tramite algoritmi a chiave simmetrica. Sono infatti utilizzati in sistemi in cui è possibile mantenere una chiave segreta in modo sicuro. Un caso d'uso tipico è la comunicazione tra due server, dove la chiave utilizzata per la firma non viene mai condivisa con l'utente, ma mantenuta al sicuro all'interno del sistema.
Un sistema che usa questo metodo per l'autorizzazione degli utenti può essere implementato in modo da generare token che sono validi solamente per un utilizzo. Un altro scenario possibile nell'implementazione di questi token è tramite la creazione di sessioni server-side che mantengono l'utente loggato per un certo periodo di tempo. Questo meccanismo permette di non dover generare un token a ogni richiesta ricevuta, ma introduce i costi di overhead tipici dei sistemi che utilizzano sessioni, ovvero il dover mantenere in memoria i dati delle sessioni attive, e il dover cercare l'ID della sessione a cui l'utente vuole collegarsi per ogni richiesta ricevuta.

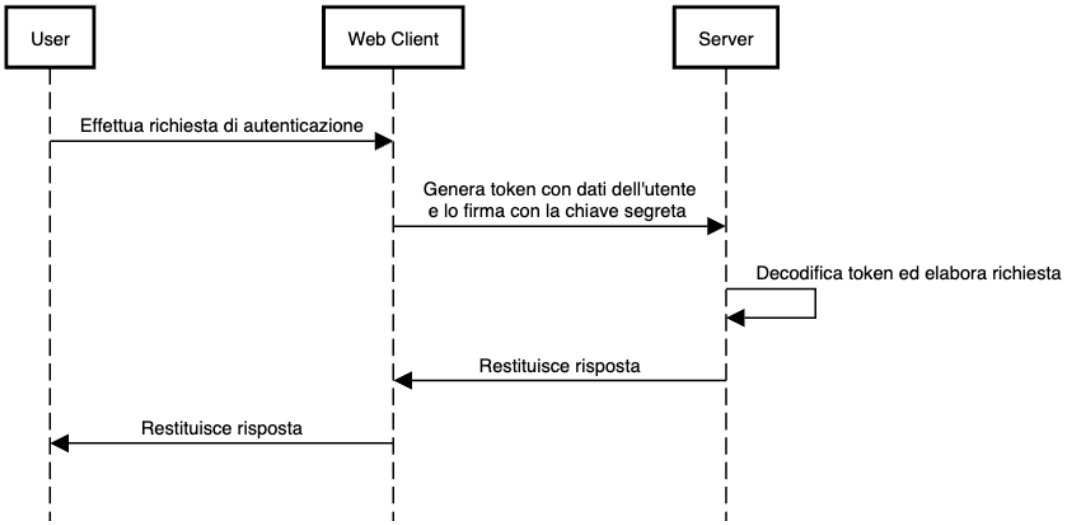

Figura 2: Diagramma di sequenza della creazione di un PASETO local

Il diagramma di sequenza riportato illustra il processo di creazione di un PASETO in un sistema che utilizza token di tipo local per la comunicazione tra due componenti del sistema, in questo caso un client Web e il server. Si assume che questi due componenti possano condividere in modo sicuro una chiave segreta, presupposto necessario per la salvaguardia del sistema.
Il sistema preso in analisi implementa la gestione delle autorizzazioni tramite dei token monouso, questo sistema prevede che ogni token possieda un identificativo univoco. Il server mantiene al suo interno una lista contenente gli identificativi dei token già utilizzati, se l'ID del token ricevuto risulta già utilizzato, la chiamata verrà dunque scartata.
Non ci si aspetta che la generazione di un nuovo token per ogni richiesta influisca eccessivamente sui tempi di risposta del servizio, gli algoritmi utilizzati da PASETO, sono infatti estremamente rapidi e vengono aggiornati nel tempo. Il costo di overhead in termini di tempo impiegato per generare il token viene ampiamente ricompensato dalla sicurezza che questo sistema garantisce.
La creazione di un token parte quando viene ricevuta una nuova chiamata, i dati necessari per la generazione del token vengono raccolti dal client, e codificati in Base64url.
Il processo di creazione di un token prevede inoltre la generazione di un *nonce*, un valore univoco che garantisce che ogni token generato sia unico, anche in presenza di dati identici. Questa stringa, come dichiarato nello standard PASETO, deve essere composta da 32 caratteri esadecimali e creata utilizzando un generatore di caratteri casuali sufficientemente sicuro, per proteggersi da attacchi di replica del token. L'utilizzo del nonce porta due vantaggi molto importanti:

- garantisce che, anche a partire da dati identici, il payload di ogni token sia differente, in modo da poter identificare univocamente ogni token

- permette di verificare se il payload del token è stato modificato durante la trasmissione. Questa stringa viene infatti aggiunta al token prima di inviarlo, e utilizzata per la creazione del payload: se il nonce che si ottiene una volta decodificato il token non è uguale con quello dichiarato, la richiesta viene scartata poichè significa che il payload è stato modificato in seguito alla generazione del token.

Il payload codificato in Base64 viene dunque cifrato utilizzando un algoritmo a chiave simmetrica. Ogni versione di PASETO utilizza algoritmi diversi, questi vengono infatti aggiornati nel tempo per garantire un elevato livello di sicurezza. L'algoritmo utilizzato dalla versione 3, ovvero quella che verrà utilizzata per un esempio di generazione di un token, utilizza l'*AES-256-GCM*[6]

La struttura finale di un PASETO della terza versione di tipo local è dunque la seguente:

```
v3.local.<nonce + payload_cifrato>
```

Il token firmato viene inoltrato al server assieme alla richiesta ricevuta, questo utilizza la chiave condivisa per decifrare il token e determinare se l'utente può essere autorizzato o meno. In questo caso, a differenza di quanto accade con JWT, l'utente non entra mai in possesso del token, esso rimane all'interno del sistema per tutto il suo ciclo di vita, che può essere differente a seconda dell'implementazione. In fase di decodifica, il server salva in memoria l'identificativo del token, per evitare che questo possa essere ri-utilizzato in futuro per altre richieste.
L'identificativo del token può coincidere con il nonce, o può essere utilizzato un altro metodo di identificazione, per esempio l'utilizzo del claim "jti".
Di seguito viene riportato un esempio di costruzione di un token di tipo local, utile per delinearne le caratteristiche, verrà utilizzata la versione 3 del token, i dati di partenza utilizzati saranno gli stessi dell'esempio riportato per JWT, ma con i timestamp riportati in formato ISO 8601, ovvero:

```
{
    "sub": "User123",
    "iat": "2024-10-10T00:00:00",
    "exp": "2024-10-29T23:59:59"
}
```

La chiave utilizzata per la firma è ancora una volta la stringa

`yJojIqZismADFUmhEjgB9NJxh20JpP4d`

Un esempio di token generato a partire dai dati sopra riportati è il seguente:


```
v3.local.hGqEZai7l8Bvjc2hZXKF4PT3aM1DGUu9mMZmBRi-gMhVDz5uoV6T8h1GuUa1IFOOlfvLa5xgZyAxDMfZS
taDqYpfwNhJSuvvS51PlJOcQFMV6yGEvDrH-IwckNdHgTrA_79coKJzOe1T5TkmCkbUUL3C4WrYnugbgzXxkv
x3lXeFygVEPvpyWmIhMQCb2a_Yt8DpY1bkUSSOpxg328HJQ67ZEWk2KE9UhbtiDA
```


In fase di decodifica e verifica di un token local, il server estrae dal payload in nonce e il contenuto cifrato. Il payload viene decodificato da Base64url a testo normale, e successivamente decifrato utilizzando la chiave simmetrica, utilizzata anche in fase di firma del token.
Una volta decifrato, il server verifica che il nonce, anch'esso cifrato e incluso nel token, corrisponda a quello generato in precedenza. Un nonce non corrispondente a quello originale indica che il token è stato manomesso, e che la richiesta è dunque da scartare.
Se il nonce è valido, il server procede con la lettura del payload, analizzando i claim contenuti e verificando la validità del token, per esempio controllando l'ID del token, per vedere se questo è già stato utilizzato.

In sintesi, i token PASETO di tipo local utilizzano algoritmi a chiave simmetrica per cifrare il payload e garantire l'integrità dei token. Per evitare che due token siano identici, viene utilizzata una componente generata in maniera casuale chiamata nonce. Possono essere creati token monouso, oppure token validi per un certo periodo di temppo, in modo da creare una sessione sicura tra l'utente e il server.

Questi token si differenziano da JWT impostando un metodo di crittografia preciso, non lasciando la scelta all'utilizzatore, con lo scopo di facilitare l'implementazione di un sistema di autenticazione sicuro.

Public
I token "public" vengono usati quando non è possibile condividere una chiave segreta in modo sicuro, sono infatti firmati attraverso l'utilizzo di un algoritmo a chiave asimmetrica. Questi algoritmi sfruttano una coppia di chiavi, una pubblicamente disponibile e una privata da mantenere al sicuro.
La differenza che distingue i token "public" dai token "local" è che le informazioni contenute nel payload sono solamente codificate in Base64url e non cifrate. È dunque fondamentale non includere informazioni sensibili all'interno del payload, poichè questo può essere letto liberamente da chiunque riesce ad intercettare il token.
Questi token contengono una firma che attesta l'integrità del token, tramite questo processo si può controllare, come nei casi di JWT e dei PASETO local, se il token ha subito modifiche durante il trasporto.
Per firmare un token "public" in un sistema a chiave asimmetrica, viene utilizzata una chiave privata per generare la firma e una chiave pubblica per verificarla. Il token viene inviato al destinatario che è in possesso della chiave pubblica. Quando viene ricevuto, il destinatario utilizza la chiave pubblica per verificarne l'integrità.
La chiave pubblica permette di decifrare la firma e confrontare l'hash risultante con quello del token ricevuto. Se i due hash corrispondono, il token è integro e autentico.

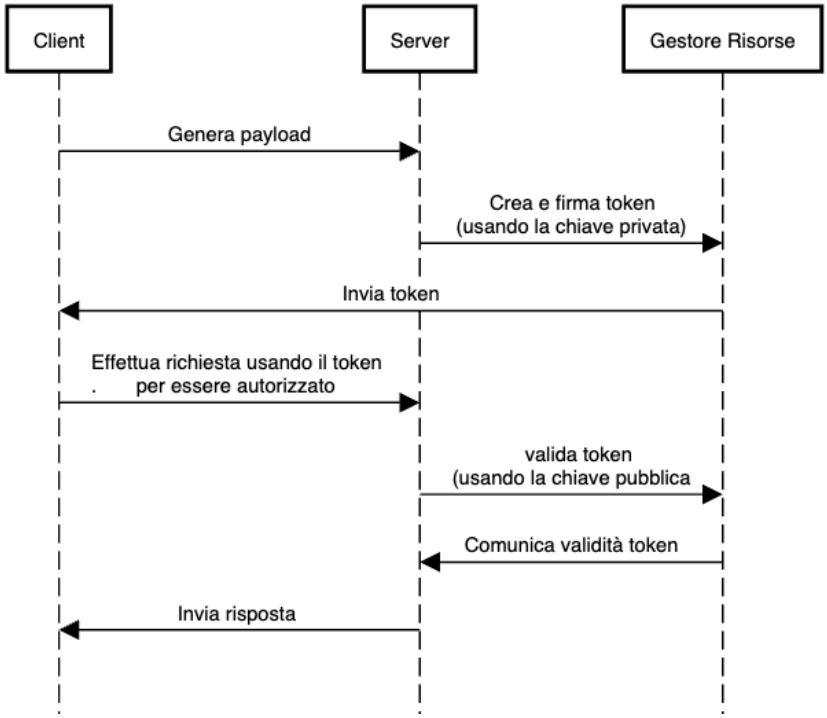

Figura 3: Diagramma di sequenza della creazione di un PASETO public

Il diagramma riportato illustra il processo di creazione di un token in un sistema che utilizza delle sessioni per mantenere l'utente registrato.
Un token viene emesso quando un utente che non ha ancora una sessione attiva richiede al sistema l'autorizzazione. Il payload del token viene generato con i dati necessari e firmato utilizzando la chiave privata, in possesso del server. Viene dunque creata una sessione associata al token e all'utente, che dovrà utilizzare il PASETO appena generato per tutte le successive richieste in modo da riferirsi alla sessione già creata in precedenza.
È possibile, e consigliato, gestire la durata della sessione mettendo delle limitazioni temporali alla vali-

dità del token. Questo meccanismo obbliga l'utente ad effettuare una nuova richiesta di autorizzazione dopo un intervallo di tempo arbitrariamente lungo, in modo da evitare che il token rimanga valido per sempre e possa essere utilizzato da utenti malevoli per scopi impropri.
Viene riportato un esempio di costruzione di un token "public", i dati utilizzati per generare il payload, ovvero Json contenente i claim e il nonce, saranno gli stessi dell'esempio riportato per i PASETO "local". La differenza sarà ovviamente nel sistema di firma, in quanto i token "public" utilizzano algoritmi a firma asimmetrica, che prevedono dunque la creazione di una coppia di chiavi.
I token "public" della terza versione usano l'algoritmo *ECDSA*, un algoritmo di firma che utilizza curve ellittiche per garantire l'integrità dei messaggi. Gli algoritmi basati su curve ellittiche, come ECDSA, offrono lo stesso livello di sicurezza di altri algoritmi asimmetrici (come RSA), ma con chiavi più piccole e operazioni più veloci. Le curve ellittiche utilizzano calcoli matematici di difficile risoluzione, che diventano banali se si è in possesso della chiave privata che è stata usata per firmare il messaggio.
Per non deviare dall'argomento dell'elaborato non si approfondisce il funzionamento degli algoritmi a curve ellittiche, è solo da sottolineare la sicurezza di questi algoritmi e la velocità di firma e di decodifica di token che li utilizzano.
Le chiavi utilizzate per l'esempio sono state generate utilizzando le librerie openssl, direttamente disponibili nel terminale di dispositivi Windows e Linux:

- Comando per generazione chiave privata:

  ```
  openssl ecparam -genkey -name secp384r1 -noout -out private_key.pem
  ```

  Questo comando utilizza le librerie di openssl per generare una chiave privata utilizzata per l'algoritmo *secp384r1*, questa chiave viene poi salvata in un file .pem(Privacy Enhanced Mail), formato utilizzato per il salvataggio e la condivisione di chiavi e certificati.
  Il comando genera una chiave privata con il seguente formato:

  ```
  -----BEGIN EC PRIVATE KEY-----
  MIGkAgEBBDDqH2h9IMc8sWm6CW2dlfoZvmUeiA0jHK+lbJCOKJSZZ8OnlmMWu2Kq
  YlfldBXrWFugBwYFK4EEACKhZANiAAQSitZz+HloNBL8OEYG4RX2+U4/0ES2YJae
  wwcOkyH5nQTavWBBLsOt/+OHn5qknBcuycQYm9IiOxu5LmQqDZKIgZR7n7aCheDh
  yDBCzE6PMrnGooCDttbnhMOn9Frz3iA=
  -----END EC PRIVATE KEY-----
  ```

  Le stringhe che delimitano l'inizio e la fine della chiave non vengono utilizzate per firmare il contenuto del messaggio, servono solamente ad indicare il tipo di contenuto del file.

- Comando per generazione chiave pubblica:

  ```
  openssl ec -in private_key.pem -pubout -out public_key.pem
  ```

  Questo comando utilizza la chiave privata generata in precedenza per generare la chiave pubblica corrispondente, viene anch'essa salvata in un file .pem.
  L'output del comando riportato è il seguente:

  ```
  -----BEGIN PUBLIC KEY-----
  MHYwEAYHKoZIzj0CAQYFK4EEACIDYgAEEorWc/h5aDQS/NBGBuEV9vlOP9BEtmCW
  nsMHNJMh+Z0E2r1gQS7Drf/jh5+apJwXLsnEGJvSIjsbuS5kKg2SiIGUe5+2goXg
  4cgwQsxOjzK5xqKAg7bW54TDp/Ra894g
  -----END PUBLIC KEY-----
  ```

Al token creato nell'esempio viene infine aggiunto un footer che contiene la stringa: "Footer opzionale", in modo da illustrare anche l'utilizzo del footer nel processo di creazione.
Il token risultante è il seguente:

```
v3.public.eyJzdWIiOiAiVXNlcjEyMyIsICJpYXQiOiAxNzMwMTYwMDAwLCAiZXhwIjog
MTczMDI0NjM5OX3b80Er1w5RtrY4fLoKwl1c8Lg4fHhfttE8a5nw-_Ldsur-h7MCCVQy2aOmiZSp_8BFyKtRB2b9
CuJ_rdOA6pyXqqj3C-L0KRJc_P4vLM3BQmuqB
dXAc44bADZ_52RVETI.Rm9vdGVyIG9wemlvbmFsZQ
```

Per verificare la validità di questo token, il server ricalcola la firma a partire dalla chiave privata, dai dati dell'header, contenente il tipo e la versione del token, e dall payload, contenente il nonce e i dati codificati.
Se la firma ricalcolata corrisponde a quella riportata nel token, allora l'integrità è verificata, e si può procedere con l'analisi dei claim contenuti nel payload.

In sintesi, PASETO rappresenta una soluzione alternativa rispetto ai JWT che mira ad aumentare la sicurezza del sistema mantenendo la facilità di implementazione. La sua struttura protegge dai rischi che JWT può comportare, come la scelta errata degli algoritmi di firma, e integra la crittografia come componente essenziale. Sebbene questa tecnologia sia meno diffusa rispetto a JWT, il crescente interesse la rende una valida scelta, specialmente in contesti che trattano informazioni sensibili, per migliorare la sicurezza delle applicazioni moderne.
La tecnologia PASETO è stata presentata all'IETF[1] nel 2022 per essere riconosciuta come standard condiviso, è tutt'ora in attesa di approvazione.
L'approvazione da parte dell'IETF, e la conseguente pubblicazione di uno standard condiviso, consentirebbe agli sviluppatori di avere un documento su cui basarsi per implementare PASETO, permettendone l'utilizzo in vari ambiti in cui non è ancora stato impiegato.

[1]Internet Engineering Task Force, ente incaricato della pubblicazione degli standard condivisi

1.3 Macaroons

Macaroons[3] è un tipo di token utilizzato per autorizzare e controllare l'accesso alle risorse in un sistema distribuito. A differenza di JWT o PASETO, i macaroons introducono una flessibilità maggiore grazie alla possibilità di essere attenuati, cioè di poter restringere i diritti associati a un token senza che esso debba essere riemesso. Un macaroon è composto da un insieme di dati firmati e codificati in Base64url, ciò che lo contraddistingue è la capacità di aggiungere *caveat*, ovvero delle clausole che limitano l'uso del token in determinati contesti, come specifici orari, IP, o risorse.
I caveat per macaroons svolgono lo stesso ruolo dei claim per JWT e PASETO, ma possono essere aggiunti ad un token in seguito alla sua creazione, modificando i permessi ad esso associati.
Per essere considerato valido, un macaroons deve essere utilizzato in una richiesta che rispetta tutti i caveat al suo interno.

Questo tipo di token aiuta a rispettare il *principio di privilegio minimo*, che indica come ogni attore all'interno del sistema debba avere accesso solo alle risorse strettamente necessarie, per evitare di incorrere in fuoriuscite di dati. La struttura modulare dei macaroons li rende adatti ad applicazioni dove l'autorizzazione necessita di controlli granulari e modificabili nel tempo.
La loro firma garantisce che il token non possa essere alterato senza invalidarlo, ma allo stesso tempo permette di aggiungere o ridurre permessi man mano che il token viene propagato tra i vari componenti di un sistema.
Rispetto a JWT e PASETO, i macaroons offrono un vantaggio significativo in termini di flessibilità e sicurezza. Ad esempio, in JWT una volta generato il token non può essere modificato senza invalidare la firma, il che limita le possibilità di adattarlo a diverse situazioni dinamiche.
I macaroons, al contrario, possono essere iterativamente "attenuati" aggiungendo caveat che non richiedono la rigenerazione del token, consentendo un controllo d'accesso più preciso e dinamico. Questa capacità di attenuazione rende i macaroons particolarmente utili in sistemi complessi come quelli basati su microservizi o in applicazioni multi-tenant, dove la gestione differenziata dei permessi diventa essenziale per la sicurezza e l'efficienza.
I permessi associati ad un macaroons possono solamente essere attenuati attraverso l'utilizzo di caveat, e mai ampliati. Sapendo che un token per essere considerato valido deve rispettare tutte le restrizioni poste al suo interno, è pertanto impossibile aggiungere permessi ad un macaroons senza invalidarlo.
I macaroons permettono di modificare o aggiungere caveat senza invalidare il token originale. Ogni macaroon viene infatti firmato utilizzando una chiave segreta, quando vengono aggiunti nuovi caveat non è necessario rigenerare il token completo. Questo significa che il token può essere trasmetto con i nuovi caveat aggiunti, e il sistema verificherà semplicemente che tutti i caveat (sia quelli originali che quelli attenuati) siano rispettati. Poiché il macaroon non cambia nella sua parte firmata, il token rimane valido, ma con accessi più limitati.

L'utilizzo di chiavi crittografiche consente inoltre di verificare che il token è stato emesso da un utente autorizzato, e non creato da utenti esterni con scopi malevoli. Ogni macaroon contiene una firma HMAC(Hash-based Message Authentication Code) generata con la chiave segreta del sistema, e solo chi possiede questa chiave può emettere macaroons validi. Anche se un utente malintenzionato dovesse ottenere una copia del macaroon, non potrebbe modificarlo o rimuovere caveat senza invalidare la firma.
Macaroons introduce anche un ulteriore vantaggio rispetto a JWT e PASETO, è possibile infatti creare un token contenente dei cosiddetti *third-party caveats*[3][par2.A], ovvero caveat che per essere soddisfatti necessitano l'approvazione di un servizio terzo. Questi caveat sono necessari per verificare la validità di un token quando sono richieste informazioni non all'interno del server.
Un esempio di utilizzo di third-party caveats può essere un sistema che richiede l'autenticazione a due fattori per autorizzare l'utente. Il server a cui viene richiesto l'accesso riceve un token con un caveat dove viene specificato che la richiesta è valida solamente se l'utente ha effettuato l'autenticazione a due fattori.Tuttavia, il server non possiede queste informazioni, è dunque costretto ad affidarsi a servizi terzi per verificare se l'utente ha effettuato l'autenticazione a due fattori.
Assieme alla richiesta con il token macaroon, l'utente dovrà presentare un token aggiuntivo, detto *discharge macaroon*, emesso da un servizio terzo la cui affidabilità è verificata, per certificare che uno dei

caveat all'interno del token originale è stato soddisfatto.
Questo processo introduce il sostanziale vantaggio che il server è ora in grado di verificare molte più informazioni senza dover comunicare con altri servizi.
In fase di validazione del token, saranno verificati tutti i caveat e l'integrità di eventuali discharge macaroons richiesti per l'autenticazione, questo processo verrà approfondito in seguito.

Per spiegare meglio il processo di costruzione e di lettura di un macaroons, viene riportato un esempio di utilizzo all'interno di un ambiente cloud, dove un sistema contiene risorse che possono essere richieste da più utenti a cui sono associati permessi differenti.

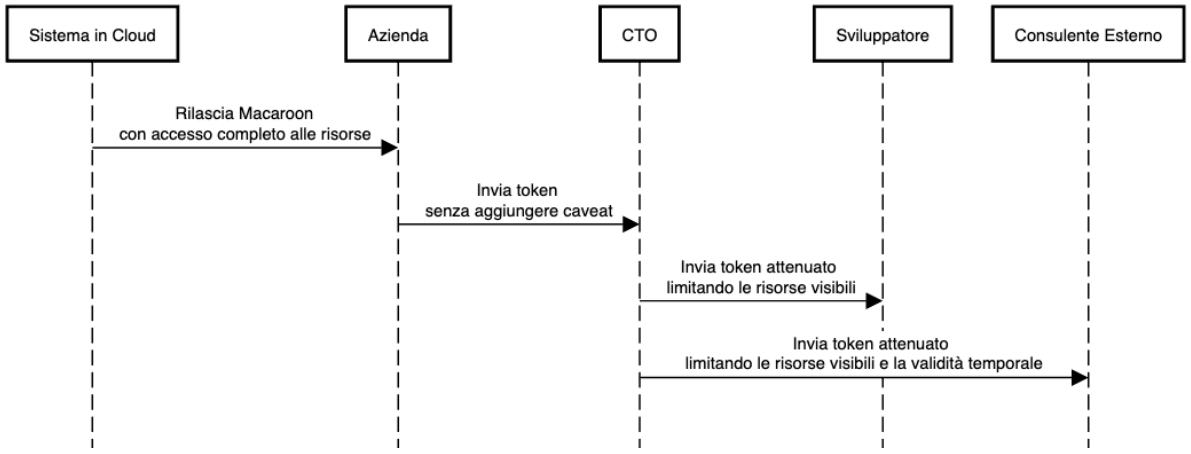

Figura 4: Diagramma di sequenza della creazione e attenuazione di un token macaroons

Il diagramma riportato descrive uno scenario in cui una azienda utilizza un servizio di storage in cloud per conservare i propri dati, questo servizio può essere acceduto da vari dipendenti dell'azienda, che possono visualizzare informazioni differenti a seconda del ruolo che ricoprono.
Il CTO dell'azienda deve poter visualizzare anche informazioni confidenziali, per esempio i costi che l'azienda sostiene per lo storage dei dati in cloud, mentre gli sviluppatori devono visualizzare solamente i dati a loro necessari.
Il servizio di storage cloud rilascia all'azienda un macaroons, con i permessi necessari per visualizzare tutte le informazioni a essa relative, ovvero senza nessun caveat che limita la validità del token. L'azienda invia lo stesso token al CTO, che sarà successivamente in grado di attenuare il token e inviarlo agli altri utenti del sistema.
Agli sviluppatori può essere inviato un macaroons con dei caveat che limitano l'accesso soltanto a determinati tipi di dati, per esempio quelli del servizio a cui stanno lavorando. Ci si immagini ora che l'azienda si serva di un consulente esterno per lo sviluppo di un prodotto.
A questo nuovo utente può essere rilasciato un token che contiene gli stessi caveat del token rilasciato agli sviluppatori, ma con un caveat aggiuntivo che ne limita la validità temporale, in modo che una volta scaduto il contratto, esso non sia più in grado di accedere ai dati aziendali.

L'azienda può dunque gestire i permessi associati ad ogni utente in maniera più flessibile rispetto all'utilizzo di JWT e PASETO grazie alla capacità dei macaroons di essere attenuati progressivamente, aggiungendo nuovi caveat senza dover rigenerare il token o invalidare la firma.
Per verificare la validità di un token vengono utilizzati algoritmi a chiave simmetrica, come avviene in alcuni casi per JWT e PASETO, ma con la necessità aggiuntiva di verificare la validità dei caveat aggiunti in seguito all'emissione del token.
Un macaroons viene creato a partire da una chiave segreta, che viene conservata all'interno del server, e da un *identifier*, un campo che identifica univocamente il token. La firma del token viene calcolata a partire da questi due dati utilizzando un algoritmo HMAC, nel seguente modo:

```
firma=HMAC(chiave_segreta, identifier)
```

In seguito a ciò, per ogni caveat contenuto nel token viene calcolata incrementalmente la nuova firma del token, effettuando l'hash della firma calcolata precedentemente e dei dati del caveat, come descritto di seguito:

```
nuova_firma=HMAC(firma_attuale, caveat)
```

La firma risultante verrà dunque aggiunta al token, questa verrà utilizzata in fase di validazione per verificare l'integrità del token.
Quando un macaroon viene verificato, il server (che è in possesso della chiave segreta con cui è stato firmato il token) ricalcola la firma originale a partire dall'identificatore del token. Viene poi ripetuto il processo di creazione di una nuova firma per ogni caveat, se la stringa risultante coincide con la firma riportata nel token, allora questo è integro, e si può procedere con la verifica dei caveat.
Questo processo permette di aggiungere vincoli incrementalmente senza compromettere la validità del token, operazione non possibile con altri token come JWT e PASETO, dove all'aggiunta di un nuovo claim, è necessario riemettere un nuovo token con la firma aggiornata.

In sintesi, l'utilizzo dei macaroons offre una serie di vantaggi rispetto ad alternative come JWT e PASETO. Grazie alla loro capacità di essere attenuati tramite l'aggiunta di caveat senza invalidare la firma originale, i macaroons garantiscono una flessibilità superiore nella gestione dei permessi e delle restrizioni dinamiche. Questo si traduce in una maggiore adattabilità per le applicazioni distribuite e basate su microservizi, dove è fondamentale poter modulare i permessi in base a condizioni che cambiano nel tempo. Inoltre, la possibilità di delegare la verifica dei caveat a terze parti, tramite l'uso di third-party caveat, rende i macaroons particolarmente adatti per scenari in cui sono richieste informazioni non in possesso del server per validare i token.

La sicurezza offerta dall'algoritmo HMAC per la firma garantisce che i token non possano essere alterati senza che il sistema ne rilevi l'invalidità, fornendo un'ulteriore protezione contro tentativi di modifica o compromissione dei token. Tutte queste caratteristiche rendono i macaroons una soluzione ottimale per ambienti cloud distribuiti, dove è richiesta una gestione flessibile e sicura dei permessi e delle risorse.
È da citare come questa tecnologia si presti ad applicazioni differenti rispetto a JWT e PASETO, pertanto è da preferire solo in determinate circostanze. Per casi d'uso più tradizionali, è ad oggi più utilizzato JWT, data la maggiore familiarità della community e la facilità in fase di implementazione.
Macaroons non è stata ancora codificata in uno standard tramite un documento RFC, ma si basa su una pubblicazione, citata in precedenza, di alcuni ricercatori di Google Research. La pubblicazione di uno standard di questa tecnologia ne aumenterebbe sicuramente la diffusione, permettendo agli sviluppatori di avere un punto di partenza nell'adottare macaroons all'interno dei loro sistemi.

1.4 OAuth 2.0

OAuth 2(Open Authorization 2.0)[7] è uno standard aperto utilizzato dai più popolari servizi Web per delegare l'accesso tra applicazioni. Consente a un'applicazione, chiamata client, di accedere a risorse protette da un altro sistema, come una API, per conto di un utente, senza condividere informazioni confidenziali come la password.
OAuth viene principalmente utilizzato per gestire lo scambio di informazioni tra servizi in modo sicuro, il caso più comune di utilizzo è quando si vuole permettere a un'applicazione di accedere ai propri dati su un'altra piattaforma (ad esempio, permettere ad un'app di terze parti di leggere le proprie videochiamate fissate su Google Calendar).
Nei tradizionali sistemi client-server, nel caso in cui si verifichi il bisogno di effettuare l'accesso al server tramite un servizio di terze parti, l'utente si troverebbe costretto a dover comunicare le proprie credenziali ad un servizio esterno, con i rischi annessi di fuoriuscita di informazioni. OAuth 2 risolve questo problema definendo un framework tramite il quale un server che contiene informazioni riservate può condividerle con servizi terzi, senza mai rivelare le credenziali dell'utente per cui viene effettuato l'accesso.
Questo sistema migliora notevolmente la sicurezza del sistema e facilita l'integrazione tra servizi, definendo una interfaccia tramite la quale è possibile interagire per richiedere accesso in maniera uniforme.
In questo capitolo verrà approfondita la struttura del framework OAuth2.0, e come i token precedentemente illustrati possano essere utilizzati per gestire le richieste di accesso in maniera flessibile.

I principali componenti definiti nel framework OAuth2.0 sono quattro:

- Resource owner: colui che è in grado di permettere l'accesso ad una risorsa protetta. Tipicamente coincide con l'utente che cerca di accedere ad una risorsa di sua proprietà tramite un servizio di terze parti
- Client: il componente che richiede l'accesso alle risorse protette. Il client si interfaccia, per conto del resource owner, con i servizi incaricati di gestire l'accesso alle risorse richieste.
- Authorization server: server che assegna al client un token di accesso dopo che questo si è autenticato correttamente. Il client effettua all'authorization server una richiesta di autenticazione per conto di un utente, se la richiesta è valida riceve in cambio un *access token*, che verrà utilizzato per richiedere l'accesso alle risorse vere e proprie.
- Resource: serverserver che contiene le risorse protette ed è in grado di ricevere e processare richieste di accesso. Questo server riceve delle richieste contenenti un access token, ne analizza la validità, e in caso positivo restituisce al client la risorsa richiesta.

All'interno di un sistema, authorization server e resource server possano coincidere, in questo caso un unico server si trova a dover gestire le richieste di autenticazione, analizzare la validità degli access token ricevuti e restituire i dati richiesti. Negli esempi seguenti verranno però analizzati sistemi in cui l'authorization server e il resource server saranno separati.
Nei prossimi paragrafi viene descritta l'interazione tra i vari componenti di un sistema che utilizza OAuth2.0 per l'interazione con servizi terzi. Viene inoltre analizzato l'utilizzo di JWT, PASETO e Macaroons per scambiare informazioni in maniera compatta, evidenziando i vantaggi e gli svantaggi delle varie soluzioni.

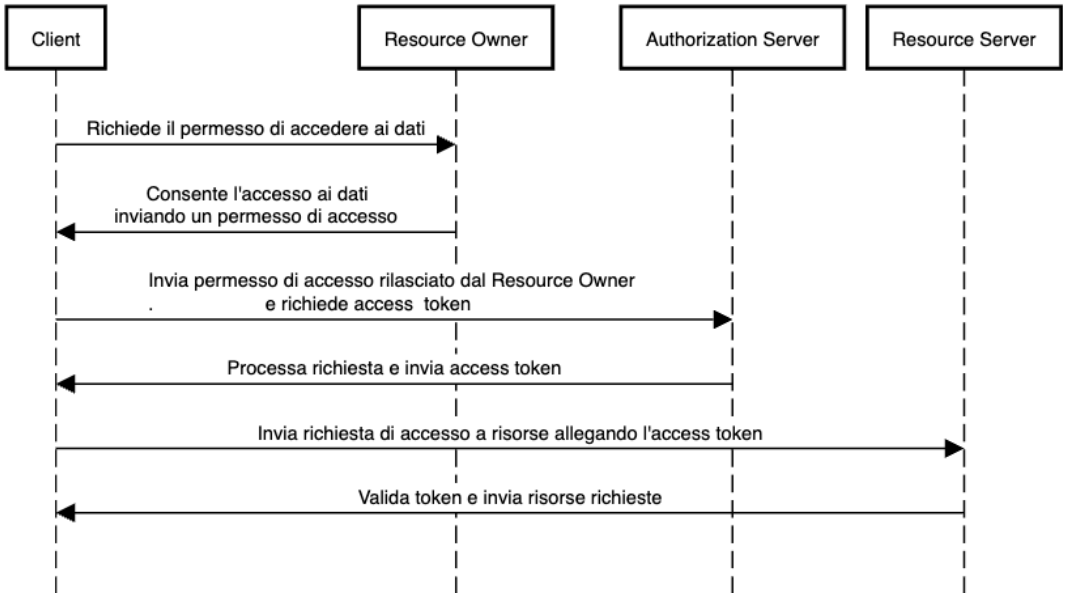

Figura 5: Diagramma di sequenza dello scambio di informazioni in un sistema che utilizza framework OAuth 2

Nel diagramma di sequenza riportato viene descritto il processo che il client segue per ottenere il permesso di accedere a delle risorse per conto di un utente.
Precedentemente a qualsiasi scambio di informazioni, è necessario che il client, che nel sistema che si sta analizzando coincide con un servizio di terze parti, si autentichi presso l'authorization server. Questo step permette di monitorare tutte le richieste che il client effettua, per gestire, e nel caso in cui si rendesse necessario per motivi di sicurezza, disabilitare tutte le chiamate effettuate da un client.
Una volta autenticato, il client può richiedere ai propri utenti il permesso di accedere ai dati contenuti nel resource server. L'utente rilascia dunque un permesso di accesso al client, lo standard IETF definisce quattro forme che questo permesso può assumere, di seguito viene riportata una tabella che le riassume, riportando i casi d'uso consigliati per ciascuna forma.

Tipo di Access Grant	Descrizione	Caso d'uso
Codice di Autorizzazione	Il client indirizza il resource owner al server di autorizzazione per ottenere un codice di autorizzazione. Il proprietario della risorsa si autentica con il server di autorizzazione, che poi emette il codice di autorizzazione. Il client utilizza questo codice per ottenere un token di accesso. Le credenziali del proprietario della risorsa non vengono mai condivise con il client.	Adatto per applicazioni Web dove il client non può essere considerato affidabile per gestire le credenziali del proprietario della risorsa.
Implicito	Un flusso semplificato per i client implementati in un browser. Il server di autorizzazione emette direttamente il token di accesso al client senza credenziali intermedie come un codice di autorizzazione. Questo tipo di grant non richiede l'autenticazione del client.	Tipicamente utilizzato per applicazioni eseguite in un browser, dove ridurre la latenza è importante.
Credenziali del resource Owner	Il client utilizza il nome utente e la password del proprietario della risorsa per richiedere un token di accesso. Questo tipo di grant dovrebbe essere utilizzato solo quando il proprietario della risorsa ha un alto grado di fiducia nel client e nessun altro tipo di grant è disponibile. Le credenziali vengono scambiate con un token di accesso, eliminando la necessità di memorizzarle per usi futuri.	Adatto per client altamente fidati, come app native o dispositivi con accesso diretto alle credenziali del proprietario della risorsa.
Credenziali del Client	Il client utilizza le proprie credenziali per richiedere l'accesso alle proprie risorse contenute nel resource server. Questo è utilizzato quando il client agisce per proprio conto o per risorse predefinite.	Ideale per la comunicazione tra servizi dello stesso ecosistema (es. diversi servizi Google che comunicano), o quando il client stesso è il proprietario della risorsa.

Tabella 4: Tipi di access grant presenti in OAuth2.0 e casi d'uso

Ottenuto il permesso di accesso del resource owner, il client lo può presentare all'authorization server per scambiarlo con un access token.
Utilizzare un access token con una struttura predefinita introduce un livello di astrazione che consente all'authorization server di poter gestire numerosi tipi di richieste in un solo modo, diminuendo lo sviluppo necessario per implementare un sistema sicuro.
È da sottolineare il fatto che OAuth non descrive la struttura che un access token deve avere. All'interno dello standard IETF precedentemente citato vengono soltanto riportate alcune informazioni che un access token deve avere, come il tipo di token, e alcune caratteristiche che sono consigliate per migliorare la sicurezza del sistema, come la data di scadenza di un token[7][par 4.2.2].
L'access token ottenuto, sarà infine presentato al resource server, che ne verifica la validità, e restituisce al client le risorse richieste.
L'IETF ha emesso un documento dove viene descritto l'utilizzo raccomandato dei token per l'autenticazione all'interno di OAuth2.0, chiamati *Bearer Token*[8], dall'inglese *'to bear'*, ovvero farsi carico.
In questo documento non viene descritta la struttura che un token deve avere, ma esempi di richieste di autenticazione, assieme ad una lista di minacce da cui i token utilizzati devono essere in grado di pro-

teggere[8][par.5]. È possibile notare come tutti tipi di token discussi fin'ora implementino perfettamente le caratteristiche sopra elencate, e come siano dei buoni candidati per essere utilizzati come access token.

Refresher Token
Un componente del framework OAuth2.0 non ancora analizzato è il *refresher token*, token aggiuntivo che può essere utilizzato per richiedere un nuovo access token nel caso quello in possesso del client sia scaduto.
Un sistema può scegliere di non utilizzare refresher token, forzando così il client a ripetere tutto il processo di rilascio di un access token.
Un refresher token contiene generalmente informazioni relative all'utente per cui si effettua la richiesta e all'access token di cui si sta chiedendo una nuova emissione, in modo tale che l'authorization server possa verificarne la validità.
Di seguito viene riportato un diagramma di sequenza che illustra il processo di rilascio e l'utilizzo di un refresher token per ottenere un nuovo access token, nel momento in cui questo viene rifiutato perchè scaduto.

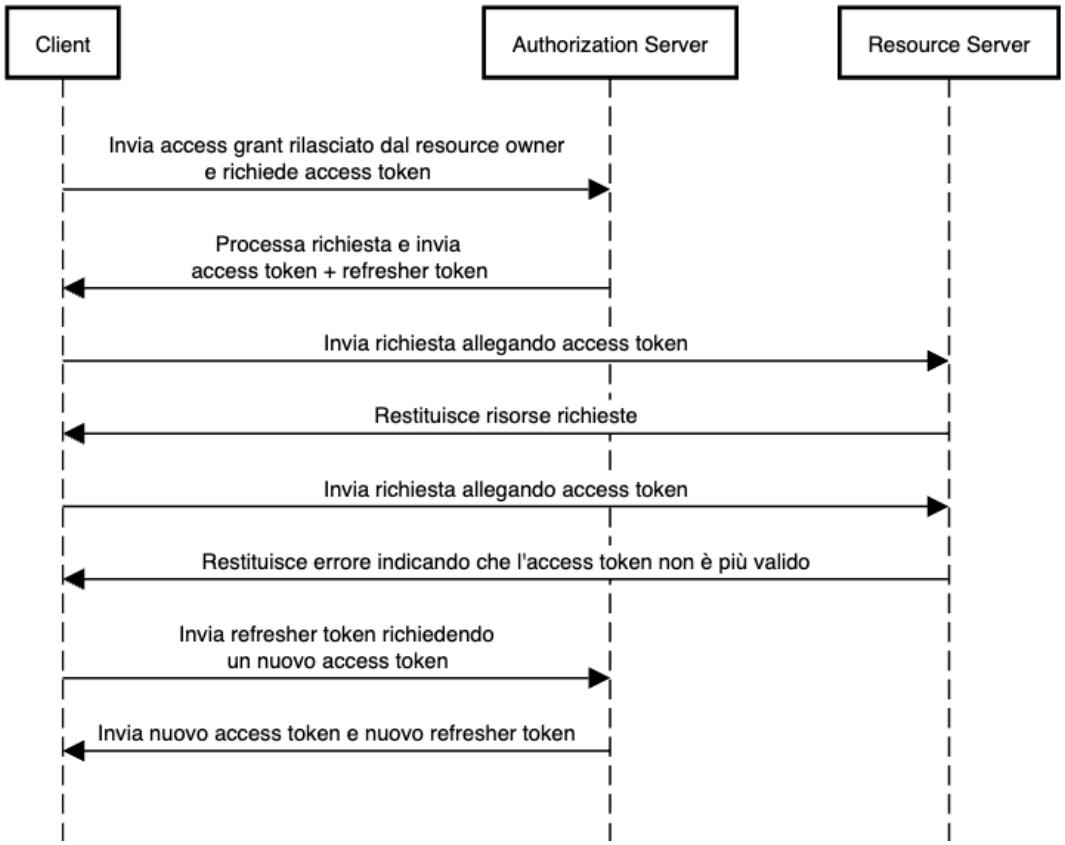

Figura 6: Diagramma di sequenza dell'utilizzo di un refresher token nel framework OAuth2.0

Contestualmente al rilascio del nuovo access token, l'authorization server può rilasciare un nuovo refresher token, per consentire al client di ripetere il processo ogni volta che ne avrà bisogno. Il refresher token deve contenere informazioni opache al client, per garantire che qualsiasi client non sia in grado di utilizzarlo per scopi malevoli.

1.4.1 JWT in OAuth2.0

I token JWT possono essere utilizzati nel framework OAuth2.0 come Bearer token, il loro utilizzo è da preferire rispetto all'utilizzo di parametri in query, poichè anche comunicando attraverso il protocollo HTTPS che protegge le informazioni scambiate, numerosi cambi di connessione possono portare alla fuoriuscita di informazioni sensibili.
L'RFC 9101[9], documento della IETF pubblicato definitivamente nel 2021, sancisce regole e best practices per l'utilizzo di JWT come Bearer token all'interno di OAuth2.0.

Questo documento descrive l'utilizzo di JWS e JWE come token di autenticazione in OAuth2.0, delineando la struttura che le richieste e le risposte devono avere per massimizzare la sicurezza e la scalabilità dei sistemi. La struttura di un JWT è già stata ampiamente discussa, una componente fondamentale viene introdotta nella scelta dell'algoritmo di firma per i token.
L'RFC 9191, descrivendo l'utilizzo di JWT all'interno di OAuth2.0, introduce elementi che fin'ora non sono stati citati, di seguito vengono riportati i più significativi:

Algoritmi di firma comuni tra client e auth server
Il documento descrive come gli algoritmi utilizzati per la firma debbano essere comuni tra il client e l'authorization server[9][par.4], limitando così la scelta e garantendo una maggiore sicurezza. Attraverso questo sistema possono dunque essere implementati solo gli algoritmi più sicuri, decidendo di non utilizzarne alcuni descritti nello standard JOSE.

Utilizzo di claim con riferimento al client
Come già citato precedentemente, un sistema che utilizza OAuth2.0 può richiedere l'autenticazione del client presso il resource server, per verificare l'identità del client.
I token utilizzati in sistemi di questo tipo devono dunque contenere un claim con un riferimento al client che effettua la chiamata, come un identificativo stabilito precedentemente tra client e Authentication server.
L'utilizzo di questo claim permette al server di verificare la fonte del token, scartando tutte le richieste provenienti da client non autorizzati. Questo sistema aumenta inoltre la agilità del sistema nel rispondere ad attacchi effettuati attraverso un client.
Nel caso in cui un client venga corrotto, e utilizzato da utenti malevoli per scopi impropri, sarà dunque sufficiente invalidare tutte le richieste contenenti l'identificativo del client corrotto, semplificando la difesa da questi attacchi.

Richieste che utilizzano riferimenti e non valori
Per garantire la compattezza di un token, e di conseguenza la velocità nello scambio di informazioni, l'RFC9101 dichiara che una richiesta effettuata da un client può utilizzare un URI che fa riferimento ad un JWT, invece che un token vero e proprio. L'URI utilizzato nella richiesta dovrà essere accessibile da entrambe le parti, e fare riferimento ad un token JWT valido.
Questo sistema consente di inviare richieste compatte, riducendo il numero di informazioni passate. Per garantire la massima sicurezza del sistema, l'URI dovrebbe essere accessibile soltanto dall'authorization server, e il contenuto dovrebbe essere lasciato disponibile solo per il tempo strettamente necessario.

1.4.2 PASETO in OAuth2.0

PASETO può essere utilizzato come access token e refresher token, sia i token local che i token public possono essere utilizzati.
I token public sono da preferire nel caso non ci siano le condizioni adatte per condividere le chiavi segrete, negli altri casi possono essere utilizzati anche i token local.
L'utilizzo di PASETO in OAuth2.0 è tuttavia limitato dalla mancanza di uno standard PASETO condiviso, e da un documento che delinei l'utilizzo di questo token all'interno del framework OAuth2.0.
Mentre JWT è stato soggetto di varie pubblicazioni da parte della IETF, alcune delle quali sono state citate nei precedentementi approfondimenti, PASETO non ha ancora ricevuto la approvazione della IETF per la pubblicazione di uno standard condiviso, documento necessario su cui si basano tutte le

implementazioni.
Sul Web sono presenti numerose richieste di implementazioni di librerie per l'utilizzo di PASETO in OAuth2.0. Di seguito cito la risposta che la Paragonie-Security, società attivamente coinvolta nello sviluppo di PASETO, ha fornito a chi chiedeva spiegazioni sulla mancata implementazione di PASETO in OAuth2.0[2]:

Prima di tutto, è necessaria la pubblicazione di un RFC per PASETO. Questo dipende dalla pubblicazione di un RFC per XChaCha20-Poly1305. Una volta superati entrambi gli ostacoli, sarà necessario redarre delle specifiche d'uso per PASETO in OAuth2.0.

PASETO è tutt'ora in attesa della pubblicazione di un RFC per XChaCha20-Poly1305, algoritmo di firma a chiave simmetrica utilizzato per la costruzione di token PASETO local nelle versioni 2 e 4.
Solo in seguito alla pubblicazione di tale documento sarà possibile vedere un RFC su PASETO, e la conseguente adozione in OAuth2.0, questo step è dunque fondamentale per la diffusione di PASETO all'interno di un framework open-source come OAuth2.0.

1.4.3 Macaroons in OAuth2.0

I token macaroons possono essere utilizzati come access token e refresher token in ambienti cloud distribuiti che utilizzano il framework OAuth2.0 per gestire le autorizzazioni degli utenti e dei servizi all'interno del sistema.
In questo specifico scenario, contestualmente all'autenticazione del client presso il resource server, questo rilascia al client un macaroon con accesso a tutte le risorse del sistema. Sarà dunque il client ad emettere token attenuati in base alle necessità del sistema, per garantire una gestione flessibile degli accessi.
Questo processo consente di delegare la gestione dei permessi al client, evitando che sia il resource server ad assegnare un token con i giusti permessi per ogni utente.
Un esempio di utilizzo di macaroons come access token e refresher token può essere un sistema documentale in cloud per visualizzare e modificare documenti condivisi.
In questo scenario, quando l'utente esegue il login tramite OAuth 2.0, l'applicazione riceve un macaroon come access token. Questo macaroon consente l'accesso temporaneo a determinati documenti e include un caveat che limita la sua validità temporale a 30 minuti dall'emissione.
Quando il tempo di validità dell'access token scade, l'utente utilizza un refresh token (può essere anch'esso un macaroon) per ottenere un nuovo access token senza dover effettuare nuovamente l'autenticazione.
Il refresher token può essere strutturato per avere una durata maggiore, evitando che l'utente debba effettuare numerose richieste di emissione di un nuovo token.
Durante l'accesso, se l'utente cerca di condividere il documento con un collega, può aggiungere caveat al proprio access token per limitare ulteriormente i permessi, ad esempio concedendo solo l'accesso in lettura ad uno specifico documento. Il nuovo token attenuato è poi inviato al collega, mantenendo intatta la firma crittografica necessaria per la validazione del token.
In sintesi, l'utilizzo dei macaroons in ambienti cloud che implementano OAuth 2.0 offre diversi vantaggi significativi. Grazie alla loro natura attenuabile, i macaroons permettono una gestione flessibile e granulare dei permessi, consentendo di aggiungere caveat senza invalidare il token originale. Questo processo facilita la delega della gestione delle autorizzazioni dal resource server al client, riducendo il carico sui server e migliorando la scalabilità del sistema in ambienti distribuiti.
La possibilità di generare facilmente access token e refresh token con durata e permessi differenti rende i macaroons una soluzione ideale per sistemi basati su OAuth 2.0, sfruttando la flessibilità e la sicurezza che questo metodo di autenticazione offre.

[2]Disponibile al seguente link: https://github.com/paseto-standard/paseto-spec/issues/15

2 Analisi dei parametri di confronto

2.1 Sicurezza

In questa sezione verrà analizzata la sicurezza delle soluzioni analizzate, approfondendo le potenziali vulnerabilità agli attacchi elencati nella *OWASP top 10*[10], studio che analizza le vulnerabilità più diffuse nei sistemi di sicurezza delle applicazioni Web.
Verranno inoltre approfonditi i processi di revoca e di modifica di un token già emesso, aspetti da considerare per un sistema flessibile in grado di adattarsi alle circostanze in cui si opera.

2.1.1 Analisi delle vulnerabilità OWASP top 10

All'interno della OWASP top 10 vengono elencate le falle nella sicurezza che le applicazioni Web presentano più spesso, dimostrate analizzando centinaia di migliaia di progetti.
È possibile costruire un sistema in grado di proteggersi dai pericoli causati da queste falle, direttamente tramite il corretto utilizzo dei token di autenticazione approfonditi in precedenza.
Di seguito verranno analizzati gli attacchi da cui token come JWT e PASETO proteggono, quali sono le best practices per difendersi, e i potenziali pericoli in cui si incorre in caso di una cattiva configurazione del sistema di difesa.

Broken Access Control
Meccanismi di autenticazione mal configurati permettono agli attaccanti di violare identità o rubare sessioni ad altri utenti correttamete autenticati.
Limitare la validità temporale dei token contribuisce a proteggere il sistema da un uso malevono di un token, mentre un corretto utilizzo degli algoritmi di firma garantisce che un token in transito non venga alterato senza essere invalidato.
L'uso di token di autenticazione al cui interno possono essere inserite informazioni riguardo ai permessi dell'utent, consentono un buon controllo degli accessi.
È tuttavia fondamentale implementare meccanisimi che scartano richieste contenenti token rubati, o utilizzati in contesti diversi da quelli previsti.

Cryptographic failure
Errori nell'implementazione della crittografia, come l'uso di algoritmi deboli o chiavi insicure, possono compromettere la sicurezza dei dati trasportati nei token.
Nell'utilizzo di token JWT è da priorizzare l'utilizzo di algoritmi sicuri, come HS256 se si vuole utilizzare un algoritmo a chiave simmetrica, o RS256 come algoritmo a chiave asimmetrica. È inoltre fondamentale proteggersi da attacchi che modificano il contenuto del claim "alg", utilizzato per indicare con che algoritmo è stato firmato il token. Questi attacchi possono sfruttare falle dello standard JOSE per effettuare richieste malevoli verso applicazioni configurate in maniera errata. Tra questi sono presenti l'attacco che utilizza come valore del claim "alg" la stringa *"none"*. L'utilizzo di questo claim è consentito dallo standard JOSE, per permettere di utilizzare token non firmati, questa caratteristica porta però alcuni client malconfigurati ad ignorare la fase di verifica della firma del token.
Utilizzare token come PASETO, che forniscono regole più stringenti nella firma dei token, può portare ad una maggiore sicurezza, se comparati con JWT.
È inoltre fondamentale gestire nella maniera più sicura possibile le chiavi utilizzate per le firme, proteggendole da attacchi esterni, condividendole solo con entità fidate e utilizzando un sistema di rotazione frequente, per minimizzare i rischi causati da una fuoriuscita di informazioni.

Insecure Design
La mancanza di protezioni strutturali a livello di progettazione espone il sistema a vulnerabilità prevedibili. Un design sicuro include una corretta gestione dei token, controllandone l'origine e assicurandosi che il payload contenga solo informazioni strettamente necessarie. I token devono essere conservati in modo sicuro e non devono mai essere esposti nel client in modo tale da evitare che possano essere utilizzati intercettati.

Ogni applicazione Web che vuole rispettare elevati standard di sicurezza, deve gestire comunque l'eventualità che un token cada in mani di utenti malevoli, ed evitare che questi possano essere considerati validi. Per questo scopo si rendono utili metodi di difesa già approfonditi in precedenza, come l'utilizzo di token monouso, o l'emissione di token validi solo per il periodo di tempo necessario per l'utilizzo.

Security Misconfiguration
Configurazioni di sicurezza errate o non aggiornate, come librerie o server che contengono vulnerabilità, espongono il sistema a pericolosi attacchi. I token di autenticazione devono essere configurati correttamente per evitare che vengano usati in contesti insicuri. Ad esempio, è consigliato utilizzare HTTPS rispetto ad HTTP per trasmettere token, ed emettere token validi solo per un breve periodo di tempo.

Vulnerable And Outdated Components
L'uso di componenti vulnerabili o non aggiornati espone il sistema a potenziali pericoli. Anche se l'utilizzo di token offre protezione da questi attacchi, l'utilizzo di librerie obsolete o insicure per la loro gestione può aprire nuove falle. È essenziale mantenere aggiornate le librerie di autenticazione e seguire le pratiche di sicurezza più recenti per prevenire attacchi noti.
Nel caso di PASETO, in cui vengono rilasciate nuove versioni di token, è necessario prevedere questa eventualità e costruire un sistema in grado di effettuare l'aggiornamento dei token emessi, con modalità che verranno approfondite successivamente.

Identification And Authentication Failures
Problemi nei meccanismi di identificazione e autenticazione, come sessioni mal protette o man-in-the-middle attacks, possono esporre i token a intercettazioni. L'uso di tecniche come la validazione di token basati su firma, o tecniche come il token binding che saranno approfondite in segutio, riducono il rischio che un token possa essere utilizzato per scopi impropri.

Server-Side Request Forgery (SSRF)
Attacchi in cui un server vulnerabile viene indotto ad effettuare richieste HTTP verso destinazioni non previste, spesso interne alla rete.
Un utilizzo errato del framework OAuth2.0 può esporre ad attacchi di SSRF, per esempio nel caso si consenta ad un client non sicuro di effettuare richieste al server che contiene le risorse protette. Per proteggersi da questi attacchi, è possibile implementare un sistema di autenticazione del client, con lo scopo certificarne la sicurezza e riconoscere tutte le chiamate da esso provenienti, in modo da poterle bloccare tempestivamente una volta identificata una minaccia.

Di seguito vengono inoltre citate le vulnerabilità presenti nella OWASP top 10 non direttamente collegate con l'utilizzo di token per l'autenticazione, ma comunque da tenere in considerazione in fase di implementazione.

- Injection: Attacchi che sfruttano la possibilità di inserire istruzioni di codice maligne all'interno di input non validati (es. query SQL/NoSQL), permettendo l'esecuzione di codice arbitrario o l'accesso a dati sensibili.
  I dati contenuti in un token devono essere correttamente utilizzati per proteggersi da attacchi di injection. Un attaccante può teoricamente inserire codice malevolo all'interno di un token, una corretta lettura dei dati impedisce che questo codice venga eseguito. È da notare come l'utilizzo delle più diffuse librerie contribuisce a proteggersi da questo tipo di attacchi.

- Software and Data Integrity Failures: Mancata verifica dell'integrità di codice e dati, espone al rischio di eseguire software non autorizzato o compromesso, oppure di trattare dati alterati.

- Security Logging and Monitoring Failures: La mancanza di registrazione e monitoraggio degli eventi di sicurezza. Impedisce di rilevare in maniera tempestiva attacchi e anomalia, limitando la agilità del sistema nel rispondere alle minacce.
  Un sistema di logging deve inoltre tenere conto delle informazioni che vengono stampate, tra cui è presumibile si trovi il contenuto dei token. Un sistema di logging dovrebbe essere considerato

sensibile tanto quanto i token che vengono scambiati, in quanto esso può contenere informazioni confidenziali come il contenuto dei token.

2.1.2 Revoca di un token

Un sistema che utilizza un'autenticazione token-based deve essere progettato per rispondere ai cambiamenti delle esigenze di sicurezza e delle autorizzazioni degli utenti. La revoca dei token esistenti e l'emissione di nuovi token con modifiche al payload consentono di gestire situazioni come la modifica dei privilegi di accesso o il cambiamento di credenziali. Questo approccio garantisce che gli utenti abbiano sempre accesso solo alle risorse autorizzate, migliorando la sicurezza complessiva del sistema. Consente di mantenere la flessibilità necessaria per adattarsi a requisiti normativi in continua evoluzione, assicurando una protezione adeguata dei dati sensibili. La revoca di un token può essere necessaria in seguito a vari eventi, per esempio può essere necessario revocare un token assegnato ad un utente non più autorizzato nel sistema. In questo paragrafo verranno approfondite le modalità di revoca delle diverse soluzioni proposte, evidenziando la facilità di questa operazione per ogni tecnologia, ed eventuali costi, in termini di sviluppo e/o di tempo, che essa comporta.

Limitazione della validità temporale dei token
Uno dei metodi più comuni per invalidare un token è limitarne la validità temporale, garantendo che possa essere utilizzato solo per un periodo di tempo prestabilito. In un contesto JWT, i claim come "nbf" (not before), "iat" (issued at), ed "exp" (expiration) sono utilizzati per definire con precisione l'intervallo di tempo in cui il token è valido. Questo approccio è utile in sistemi dove la scadenza temporale dei token è cruciale per gestire la sicurezza dinamica, assicurando che, anche in caso di compromissione, un token sia utile solo per un breve periodo. Per i PASETO di tipo 'local', la limitazione temporale può essere applicata nei casi in cui il token non è monouso, come un sistema che utilizza PASETO per la creazione di una sessione, o in sistemi dove il token potrebbe non essere consumato immediatamente. Anche in questo caso, l'utilizzo dei claim volti a limitare la validità temporale del token consente di rafforzare la sicurezza, costringendo l'utente a richiedere un nuovo token dopo un determinato intervallo di tempo.

Blacklisting
Il blacklisting è un metodo efficace per invalidare i token memorizzando gli identificativi di quelli non più validi. Nel contesto JWT, questo approccio può essere implementato utilizzando il claim "kid" per identificare univocamente ogni token. Il sistema conserva un elenco di identificativi dei token invalidati e, durante il processo di validazione, verifica se il token ricevuto è presente in questa lista. Se l'ID del token è incluso nell'elenco, il token viene considerato non valido e la richiesta viene rifiutata. Tuttavia, questo metodo introduce un costo aggiuntivo, in quanto è necessario memorizzare e gestire l'elenco di token invalidati, reintroducendo la necessità di eseguire query per ogni richiesta. Anche per i PASETO di tipo 'public', si può applicare lo stesso principio, dato che questi token non memorizzano alcuna informazione sullo stato lato server.
Per sistemi che utilizzano PASETO di tipo 'local' monouso, la blacklist viene utilizzata per salvare tutti gli ID dei token già utilizzati, per garantire che ogni token non possa essere utilizzato più di una volta. Utilizzando invece OAuth2.0, il blacklisting di un token obbliga il client a richiedere nuovamente l'access grant al resource owner per ottenere un nuovo access token, o a presentare il refresher token. L'utilizzo di un refresher token consente di gestire l'emissione di nuovi token in maniera flessibile, minimizzando il numero di volte che il resource owner deve gestire.

2.1.3 Modifica di un token

Un sistema di autenticazione basato su token deve prevedere la possibilità di modificare rapidamente i token già emessi qualora emergano nuove esigenze di sicurezza o cambiamenti nei privilegi dell'utente. Sebbene alcuni tipi di token siano progettati per essere immutabili una volta creati, esistono meccanismi che consentono di sostituire un token con uno aggiornato, garantendo che le modifiche necessarie siano

riflesse immediatamente. Un approccio comune consiste nell'invalidare il token originale e generarne uno nuovo, incorporando nel payload le informazioni aggiornate.
In questo paragrafo vengono analizzati i metodi disponibili per modificare un token già emesso, esaminando per ogni tecnologia quando questo si può rendere neessario, e quanto sia facile o complesso operare tali modifiche.

Modifica di token JWT
Nel caso dei JWT, una volta emesso un token, esso non può essere modificato direttamente a causa della sua natura stateless e immutabile. Tuttavia, una tecnica comune per "modificare" un token consiste nel generare un nuovo token con le informazioni aggiornate. Il processo prevede l'invalidazione del vecchio token tramite blacklisting o riduzione della sua validità temporale e l'emissione di un nuovo token che riflette i cambiamenti. Ad esempio, se i privilegi di un utente sono stati aggiornati, il nuovo token includerà queste modifiche nel payload, mentre il vecchio token, anche se ancora in possesso dell'utente, non sarà più valido. Questo approccio richiede un'infrastruttura adeguata per gestire i token invalidati e garantire che le richieste siano effettuate solo con i token più recenti.

Modifica di token PASETO
Similmente ai JWT, anche i token PASETO sono progettati per essere immutabili dopo la loro emissione. Tuttavia, è possibile emettere un nuovo token per riflettere modifiche alle informazioni contenute nel payload. Nel caso dei PASETO 'local', questo processo è particolarmente semplice poiché i token sono monouso e stateless, permettendo una rapida sostituzione senza la necessità di memorizzare informazioni di stato lato server. Per i PASETO 'public', la tecnica di modifica comporta l'emissione di un nuovo token con i cambiamenti necessari, mentre il vecchio token può essere invalidato tramite blacklisting.

Aggiornamento di versione PASETO
Nell'utilizzo di PASETO si rende necessario considerare una casistica che con JWT non si verifica, ovvero l'aggiornamento della versione dei token che si utilizza.
PASETO, a differenza di JWT, viene infatti continuamente aggiornato per migliorarne l'efficienza e la sicurezza, è pertanto necessario configurare un sistema che possa rimanere al passo con le nuove versioni che vengono pubblicate.
È possibile costruire sistemi di autenticazione che accettano token di più versioni, che vanno aggiornati all'uscita di ogni nuova versione, ma questo creerebbe complessità in fase di implementazione e di gestione delle chiavi.
Ci si aspetta dunque che la procedura più efficiente rimanga l'emissione di nuovi token della versione aggiornata, con la conseguente cancellazione dei token già emessi. La migrazione tra versioni richiede una completa rigenerazione del token, in quanto la struttura crittografica e i metodi di firma o cifratura possono variare significativamente tra le versioni.
Convertire un token di versione 3 a versione 4 non porta vantaggi sostanziali rispetto all'emissione di uno nuovo, poiché richiederebbe comunque un passaggio di validazione e riemissione, mantenendo invariati i costi operativi e infrastrutturali legati a questo processo.
Di conseguenza, è più conveniente emettere un nuovo token anziché tentare di aggiornare uno esistente.

Modifica di un macaroons
Macaroons, come citato in precedenza, offre il vantaggio di poter modificare facilmente i permessi associati ad un token senza che questo debba essere revocato e riemesso.
È pertanto possibile modificare un token aggiungendo caveat che ne restringono la validità.
L'unico scenario in cui si rende necessaria la ri-emissione di un token è il caso in cui ad un utente debbano essere aggiunti dei permessi che prima non aveva. Un token macaroon, infatti, può essere solamente attenuato aggiungendo caveat. Nel caso in cui si rende necessario aumentare i permessi associati ad un utente, l'aggiunta di un caveat non risolve il problema, poichè le restrizioni imposte dai caveat precedenti rimarrebbero valide.
La soluzione a questo problema è l'emissione di un nuovo token con i caveat modificati, in modo da permettere all'utente di raggiungere tutte le risorse necessarie.

Aggiornamento di un Access Token in OAuth2.0
Il framework OAuth2.0 facilita le operazioni di modifica, come avviene per la revoca di un token, grazie all'utilizzo del refresh token rilasciato al client.
Un sistema può infatti utilizzare una blacklist per salvare tutti i token che è necessario modificare, per evitare che possano essere ancora utilizzati. Se il resource server riceve uno dei token contenuti nella blacklist, forza il client a ripetere il processo di emissione di un nuovo token, o richiedendo un nuovo access grant al resource owner, o attraverso l'utilizzo di un refresher token. Il nuovo token rilasciato conterrà le modifiche necessarie per gestire l'accesso dell'utente.

2.2 Diffusione tecnologie e librerie disponibili

Tra le tecnologie discusse fin'ora, la più utilizzata per la gestione delle autenticazioni nelle applicazioni Web, sono i JWT. Questi token sono molto diffusi a causa della loro longevità, lo standard di JWT è stato infatti pubblicato nel 2015. La comunità di sviluppatori ha avuto tempo per fare propria questa tecnologia e di individuare i differenti casi d'uso che possono sfruttare la flessibilità di JWT.
Sono disponibili numerose librerie per ognuno dei maggiori linguaggi di programmazione, di seguito verranno elencate alcune delle librerie disponibili per i linguaggi più utilizzati nello sviluppo delle applicazioni Web.
I dati riportati in seguito fanno riferimento agli utilizzi delle librerie citate in progetti di grandi dimensioni liberamente consultabili. Non sono pertanto conteggiati progetti di piccole dimensioni o progetti in ambito enterprise, che sicuramente utilizzano ampiamente JWT, PASETO e Macaroons, ma di cui non si hanno dati a disposizione.
Nel caso di Java, la libreria più utilizzata è Nimbus, liberamente disponibile sulla repository Maven Central[11]. Questo progetto implementa gli standard JOSE e JWT per permettere la costruzione e la decodifica di token in maniera facile e intuitiva. Nimbus viene importata come dipendenza in più di 1000 artefatti disponibili sulla repository, ed è una delle più utilizzate nell'ambito della sicurezza, mentre la libreria 'paseto4j', che implementa la gestione dei token PASETO in Java non viene utilizzata da nessun'altro progetto di grandi dimensioni presente su Maven Central. L'interesse della community per questo tipo di token sta comunque aumentando, la repository del progetto (disponibile su github al seguente link: https://github.com/nbaars/paseto4j) viene aggiornata continuamente e registra un grande contributo da molti utenti. La libreria più utilizzata per Macaroons sviluppata in Java è jmacaroons, questa contiene classi per costruire macaroons, aggiungere caveat e verificare la validità di un token
Per i linguaggi JavaScript e TypeScript, e per i framework basati su di essi, le librerie più utilizzate sono 'jsonWebtoken' e 'jwt-decode'[12], entrambe contano migliaia di utilizzi all'interno del package manager più utilizzato, *npm*. Da notare invece come le librerie di PASETO e Macaroons più utilizzate in JavaScript contino solamente qualche utilizzo.
Il package manager 'Packagist', utilizzato per la gestione delle dipendenze di progetti in PHP e nei relativi framework, riporta che la libreria che implementa gli standard JWT e JOSE più utilizzata è 'php-jwt', è sviluppata da firebase e conta migliaia di utilizzi in altri progetti. Per quanto riguarda PASETO, la libreria più utilizzata in PHP viene sviluppata dalla 'Paragon Initiative Enterprise', società di consulenza nell'ambito della cybersecurity molto impegnata nello sviluppo di librerie a supporto di questa tecnologia. Come avviene per gli altri linguaggi citati, PASETO e Macaroons registrano un utilizzo molto minore rispetto a JWT.

Nella tabella sottostante vengono riportati le principali librerie disponibili per i linguaggi più utilizzati nello sviluppo Web:

Token	**Linguaggio**	**Libreria**	**Numero di utilizzi in grandi progetti open-source**
JWT	Java	Nimbus	1083
PASETO	Java	paseto4j	0
Macaroons	Java	jmacaroons	4
JWT	JavaScript	jsonWebtoken	31.389
PASETO	JavaScript	paseto	19
Macaroon	Javascript	macaroons.js	3
JWT	Python	PyJWT	n.d.
PASETO	Python	pyseto	n.d.
Macaroons	Python	pymacaroons	n.d.
JWT	PHP	php-jwt	1.896
PASETO	PHP	paseto	14
Macaroons	PHP	php-macaroons	1

Tabella 5: Librerie disponibili per l'utilizzo di JWT, PASETO e Macaroons nei principali linguaggi di programmazione, dati aggiornati al 23/09/2024

La tabella riporta il numero di progetti presenti sui principali package manager di ogni linguaggio che utilizza la libreria in questione come dipendenza.
I dati relativi all'utilizzo in python non sono disponibili, poichè il package manager PyPi, quello più utilizzato per Python, non riporta l'utilizzo delle librerie al suo interno.
I numeri riportati mostrano come JWT rimanga la soluzione dominante per l'autenticazione nello sviluppo di applicazioni Web, grazie alla flessibilità che offre in fase di implementazione e alla maggiore familiarità della community.
Al contrario, nonostante i vantaggi che PASETO vuole portare, come la sicurezza e la semplicità d'uso, non ha ancora raggiunto un'adozione significativa in progetti di grosse dimensioni. Questo dato però non significa che PASETO non sia utilizzato all'interno di vari progetti, la crescente attenzione della comunità verso PASETO porta a pensare che questa tecnologia potrebbe vedere un maggiore utilizzo in futuro.
La pubblicazione di uno standard RFC, e la adozione di PASETO da parte di qualche popolare servizio online contribuirebbero dare slancio a questa tecnologia, aumentandone la diffusione e delineando le basi per l'implementazione di questa tecnologia.
Gli sviluppi futuri aiuteranno a capire se il minore utilizzo rispetto a JWT è dato dalla relativa 'giovane età' del progetto, o se la community non vede PASETO come una reale alternativa.
Anche Macaroon riporta un minore utilizzo, ma le sue caratteristiche lo rendono maggiormente utile in alcuni ambiti, come le applicazioni cloud distribuite, dove anche JWT e PASETO possono essere utilizzati, ma non risultano flessibili come Macaroon.
Data l'estensione che gli ambienti cloud raggiungono, basti pensare a provider come AWS, Microsoft Azure o Google Cloud Computing, è dunque necessario esplorare anche l'utilizzo di macaroons all'interno di progetti enterprise, che non sono riportati sui siti da cui sono stati ricavati i dati. Sono stati infatti ricercatori di Google a pubblicare nel 2014 il primo paper sui Macaroons, proponendoli come alternativa ai token classici per l'utilizzo in ambienti cloud. Tra i progetti di larga scala che utilizzano Macaroon è inoltre da citare *Lightning Network*[3], protocollo di pagamento utilizzato per gestire transazioni nella blockchain di bitcoin.

[3]https://docs.lightning.engineering/the-lightning-network/l402/macaroons

2.3 Velocità di generazione e di verifica

Per calcolare questi dati è stato utilizzato il progetto disponibile al seguente link:

`https://github.com/ClaudioRocca/AmazingTokenGenerator`

Questo client è stato scritto in Java, precisamente attraverso il framework Spring Boot, in modo da poter creare e validare token contenenti i claim riservati dai vari standard.
Per l'implementazione sono state utilizzate le librerie sopra citate per Java, ovvero Nimbus, paseto4j e jmacaroons.
Le classi contenute al suo interno sono inoltre state utilizzate per la generazione e la validazione di tutti i token creati nel corso dell'elaborato.
Di seguito vengono riportati i tempi di creazione e verifica, misurati su i 3 tipi di token variando il numero di claim inseriti nel payload. Per interpretare i dati è dunque necessario tenere in conto il tempo di overhead aggiunto da un linguaggio di alto livello come Java.
All'aumentare del numero di claim ci si aspetta un lieve aumento nei tempi di generazione e di verifica dovuto alla maggiore quantità di testo da cifrare, tuttavia questo non dovrebbe risultare significativo.
Per la generazione di token JWT verrà utilizzato l'algoritmo a chiave simmetrica HS256, mentre i token PASETO generati saranno di tipo local e della terza versione.

Token	**Numero Claim**	**Tempo di Generazione**	**Tempo di verifica**
JWT	1	21 ± 0.465 ms	11 ± 2.326 ms
JWT	3	21.4 ± 0.589 ms	9.8 ± 3.368 ms
JWT	5	21.4 ± 0.589 ms	9.6 ±2.815 ms
PASETO Local	1	160 ± 5.94 ms	21.4 ± 6.04 ms
PASETO Local	3	165 ± 3.92 ms	32.2 ± 2.55 ms
PASETO Local	5	165 ± 2.76 ms	32.4 ± 3.19 ms
Macaroons	3	6.4 ± 0.465 ms	n.d.
Macaroons	5	6.9 ± 0.465 ms	n.d.

Tabella 6: Tempi di generazione e verifica token

I tempi riportati sono stati calcolati su un campione di 10 generazioni, calcolando la deviazione standard con un intervallo di confidenza del 90%.
È possibile notare come i tempi di generazione e verifica di un JWT siano nettamente minori di quelli rilevati per PASETO, questa differenza di performance può essere imputata alla lunghezza dei token generati, in quanto un token PASETO è mediamente più lungo di un JWT. Tuttavia, la notevole differenza di tempi non può essere solamente spiegata dalla maggiore lunghezza del token, ma è da imputare anche al diverso algoritmo di firma utilizzato, evidentemente più lento di HS256, utilizzato per JWT.
I token Macaroons sono quelli che riportano i tempi di generazione più bassi, questo dato può essere attribuito alla flessibilità dell'algoritmo di creazione di un Macaroon. Per questo token non è possibile effettuare tentativi di creazione con un solo claim (o caveat, per essere più precisi), in quanto ogni token richiede almeno due caveat chiamati location e identifier.
Ancora, i tempi di verifica di questo token non sono disponibili poichè non esiste uno standard condiviso che definisca quale struttura essi devono avere, ed è dunque impossibile verificarne la struttura, cosa che non avviene per JWT e PASETO, dove sono presenti claim riservati.

3 Utilizzo di token in HTTP e HTTPS

L'utilizzo di servizi Web basati sull'architettura client-server, come le applicazioni Web analizzate nel corso dell'elaborato, basano il loro funzionamento sull'utilizzo del protocollo HTTP(HyperText Transfer Protocol).
Il protocollo HTTP opera a livello applicativo del modello TCP/IP(Transmission Control Protocol/Internet Protocol), permettendo la comunicazione tra client e server attraverso la rete. HTTP risolve il problema della trasmissione di dati strutturati, come pagine Web e risorse multimediali, garantendo un formato standard per la richiesta e la risposta tra un client (ad esempio, un browser) e un server.
Questo protocollo è stato progettato per essere stateless, il che significa che ogni richiesta è indipendente per non costringere il server a conservare informazioni sulle richieste precedenti, facilitando la scalabilità e la gestione delle risorse.
Ogni sistema espone delle risorse, che vengono mappate con degli URI(Uniform Resource Identifier), che sono raggiungibili dall'esterno attraverso chiamate HTTP. Ogni risorsa è raggiungibile tramite uno o più metodi descritti dal protocollo HTTP (get, post, put, patch, delete, options...), per facilitare le interazioni con il server.
HTTP presenta tuttavia sfide significative in termini di sicurezza. Il protocollo, nella sua versione base, non prevede la crittografia delle informazioni trasmesse, rendendo vulnerabili i dati sensibili come credenziali o informazioni personali, a potenziali attacchi come l'intercettazione (man-in-the-middle) o il furto di sessione. Per mitigare questi rischi, è necessario utilizzare il protocollo HTTPS (HTTP Secure), che integra la crittografia SSL/TLS per garantire che le informazioni siano protette durante il trasferimento. Inoltre, l'assenza di uno stato implicito in HTTP richiede soluzioni aggiuntive, come i token di sessione o l'autenticazione tramite OAuth, per assicurare che l'utente rimanga autenticato in modo sicuro durante più interazioni con il server.
Nel corso di questo capitolo verrà analizzato l'utilizzo dei token per l'autenticazione nei protocolli HTTP e HTTPS.
Saranno analizzate delle best practices per l'utilizzo di token nel protocollo HTTP, e saranno approdondite tecnologie che garantiscono una sicurezza aggiuntiva in ambienti dove la comunicazione non è cifrata.

3.1 Scambio di token in HTTP

Nell'ambito dell'autenticazione, l'uso di token nei protocolli HTTP non cifrati può comportare significativi rischi di sicurezza.
È infatti raccomandato utilizzare il protocollo HTTPS, che utilizza sistemi di crittografia SSL/TLS per certificare l'identità degli utenti, in modo da proteggere da numerosi attacchi.
Alcuni di questi attacchi possono essere utilizzati per mettere le mani su token di autenticazione validi, è pertanto necessario tenere in conto queste problematiche e agire di conseguenza.
Nei sistemi in cui l'utilizzo di HTTP non è evitabile, sono da prendere misure di protezione aggiuntive per difendersi da attacchi mirati per intercettare informazioni sensibili e/o token da ri-utilizzare per scopi malevoli.
Una delle priorità di un sistema di questio tipo deve essere l'utilizzo di token che non portano al loro interno informazioni sensibili riguardo gli utenti e/o il sistema, poichè, come già accennato, tutti i token (tranne i PASETO di tipo public), contengono informazioni codificate in Base64url. Il contenuto dei token è infatti leggibile da chiunque riesca ad intercettarne uno, ed è pertanto necessario che un sistema non scambi informazioni confidenziali in ambienti non sicuri, come il protocollo HTTP.
Nei casi dove ciò è assolutamente necessario per il sistema, è consigliato l'utilizzo di metodi di autenticazione che utilizzano sistemi di crittografia per cifrare il contenuto dei token.
Tra questi, troviamo JWE(Json Web Encryption)[4], variante di JWT che utilizza chiavi per cifrare il payload, e PASERK (Platform Agnostic SERialized Key), formato utilizzato per scambiare le chiavi utilizzate per le firme in maniera sicura. L'uso di JWE e di PASERK permette di crittografare il payload dei token, e comunicare informazioni riguardo alle chiavi utilizzate, rendendo più difficile per un attaccante l'accesso ai dati anche in caso di intercettazione.

Nel caso in cui un sistema scelga invece di utilizzare token non criptati, è fondamentale adottare meccanismi di difesa che limitano la validità del token, come il blacklisting o l'utilizzo di claim riservati come "exp" e "iat".
Di seguito viene approfondito l'utilizzo dei token con contenuto criptato, analizzandone i casi d'uso e i vantaggi che essi introducono rispetto alle loro varianti, solamente firmate e non criptate.

3.1.1 JWE

JWE è un formato di token che viene utilizzato per scambiare informazioni cifrate, in modo che il contenuto dei token sia protetto durante la trasmissione. Rispetto a JWS, che si focalizza sulla verifica dell'integrità dei dati attraverso una firma, JWE assicura la riservatezza delle informazioni attraverso l'utilizzo di algoritmi crittografici. Un JWE può anche includere una firma per garantire sia la riservatezza che l'integrità del messaggio, aumentando la sicurezza per adattarsi a scenari in cui la protezione delle informazioni è fondamentale.
I JWE offrono un significativo vantaggio in termini di riservatezza se comparati a JWS, poichè il loro contenuto non è leggibile in chiaro se non si è in possesso delle chiavi utilizzate per firmare il token.
Inoltre, come accade per JWS, questi token supportano diversi algoritmi crittografici, delegando agli implementatori la scelta degli algoritmi di firma più adatti al sistema in cui si opera.
JWE risultano quindi adatto per applicazioni che utilizzano token al cui interno sono contenute informazioni riservate, e devono dunque essere protette per garantire la sicurezza del sistema. Questi token possono dunque essere utilizzati per evitare che una trasmissione nel protocollo HTTP venga intercettata con il fine di ottenere un token contenente informazioni confidendiali.
JWE è inoltre supportato dalle principali librerie che implementano JWT, come Nimbus per Java. Questa caratteristica ne favorisce l'utilizzo, alleggerendo il carico di lavoro in fase implementativa.
La struttura di un JWE è diversa da quella di un JWS, questo tipo di token è infatti più complesso, e necessita di più di una chiave per la sua creazione. Un JWE è diviso in sezioni di dati codificati in Base64url, come accade per JWS.

Di seguito viene riportata la struttura di un JWE e approfondito il contenuto di ogni sezione:

```
<Header>.<Chiave_Crittografata>.<Initialization_vector>.<Testo_Cifrato>.<Authentication_Tag>
```

Header
Come per JWS, l'header di un JWE contiene informazioni riguardanti il tipo di token e gli algoritmi utilizzati per la sua costruzione. È costituito da un oggetto JSON che definisce l'algoritmo utilizzato per la crittografia della chiave simmetrica e l'algoritmo per la cifratura del contenuto del token. Il claim "alg" indica l'algoritmo asimmetrico impiegato per cifrare la chiave simmetrica, mentre il claim "enc" rappresenta l'algoritmo di crittografia simmetrica utilizzato per cifrare il payload. L'header può inoltre contenere altri claim riservati dallo standard RFC7516[4][par 4.1], per esempio per far riferimento alle chiavi utilizzate nella costruzione del token.

Chiave Cifrata
Questa sezione contiene la chiave di cifratura, che viene utilizzata per decifrare il payload del token. Questa chiave è generata in modo casuale e viene cifrata utilizzando l'algoritmo asimmetrico specificato nel campo "alg" dell'header. Il risultato viene poi codificato in Base64Url, generando un valore che permette al destinatario del token di recuperare la chiave simmetrica necessaria per decifrare il contenuto cifrato.

Initialization Vector (IV)
Il vettore è una sequenza di 96 bit generati randomicamente, che vengono codificati in Base64url.
L'IV viene utilizzato in combinazione con un algoritmo a chiave simmetrica per garantire che, anche a partire da dati uguali, il token generato sia unico. Questo vettore spesso generato casualmente, viene crittografato con lo stesso algoritmo indicato nel claim "enc" e successivamente codificato in Base64url. Questa sezione è dunque fondamentale per evitare che due token possano essere identici, al fine di garantire che il contenuto del payload rimanga sicuro anche se il token viene riutilizzato.

Testo Cifrato
Il testo cifrato è l'equivalente del payload per JWS, ed è il contenuto del token che si vuole tenere protetto. Il payload viene cifrato utilizzando l'algoritmo specificato nel campo "enc" dell'header con la chiave simmetrica ottenuta dalla sezione "Chiave Cifrata". L'output di questa cifratura è codificato in Base64Url, generando una rappresentazione del testo cifrato del token. In generale, il contenuto del payload può essere costituito da qualsiasi dato che si desidera proteggere, incluso testo, JSON, o altri formati.
È da notare come, data la natura dell'algoritmo di firma utilizzato per codificare il payload, è necessario che la chiave segreta venga condivisa e conservata in maniera sicura. Lo scambio di questa chiave deve avvenire in maniera sicura, è pertanto fortemente sconsigliato l'utilizzo del protocollo HTTP.

Authentication Tag
L'ultima sezione del token è il tag di autenticazione, che viene generato con l'utilizzo di un algoritmo a chiave simmetrica, per garantire l'integrità del messaggio cifrato. Questo tag consente di rilevare eventuali modifiche non autorizzate al payload o alle altre parti del token. Anche il tag di autenticazione viene codificato in Base64Url e fornisce una protezione aggiuntiva contro possibili manomissioni.

JWE è dunque una valida soluzione per l'utilizzo in ambienti dove la comunicazione non è sicura. Grazie alle sue caratteristiche consente di evitare token duplicati, e di cifrare il contenuto dei token, garantendo che le informazioni siano visualizzabili solo da chi è in possesso delle chiavi per decifrare il token.

3.1.2 PASERK

PASERK (Platform-Agnostic Serialized Key) è un formato complementare a PASETO, progettato per la serializzazione sicura delle chiavi, rendendolo ideale per ambienti in cui il trasferimento sicuro delle chiavi è necessario.
Citando il commento riportato nella presentazione di PASERK su github[13]

"Tra i casi d'uso che PASETO non copre si trovano il wrapping delle chiavi, la crittografia a chiave asimmetrica e la crittografia basata su password".

Rispetto a PASETO, che si concentra sulla generazione e sulla verifica di token per garantire l'integrità e la riservatezza dei dati, PASERK si focalizza dunque sulla gestione delle chiavi, assicurando che siano scambiate in maniera sicure e in un formato standardizzato.
PASERK offre un significativo vantaggio in termini di sicurezza rispetto ai formati di chiave non strutturati. Ogni chiave è identificata da un tipo, e da una versione di PASETO, questo sistema permette di ridurre la possibilità di errori nell'utilizzo delle chiavi.

La struttura di un PASERK è definita da diverse sezioni, che comprendono un identificatore della versione, il tipo di chiave, e i dati della chiave in forma codificata. Ogni sezione viene rappresentata in Base64url, separata da punti. Di seguito viene riportata la struttura di un PASERK e descritto il contenuto di ogni sezione:

`k[versione].[tipo_chiave].[dati_chiave]`

Versione
L'identificatore di versione di PASERK inizia con la lettera "k" seguita dal numero di versione di PASETO a cui la chiave si riferisce. Un token PASERK deve riportare lo stesso indicatore di versione del PASETO a cui è associato. Ad esempio, la versione `k1` indica compatibilità con PASETO v1, mentre `k2` è utilizzato per la versione 2. Questo campo permette di definire facilmente il tipo di token associato alla chiave. **Tipo di Chiave**
Il tipo di chiave identifica il ruolo della chiave e l'algoritmo utilizzato per la crittografia o la firma. Tra i formati definiti da PASERK si trovano i tipi public e local, già approfonditi in precedenza, più altri tipi di chiavi:

- **local**: una chiave simmetrica utilizzata per la cifratura di token local.
- secret e public: chiavi private e pubbliche per la firma e la verifica di token public tramite crittografia asimmetrica.
- seal: una chiave sigillata, crittografata per essere trasferita in modo sicuro e decrittabile solo dal destinatario.
- wrap: una chiave avvolta, che può essere utilizzata per trasferire altre chiavi crittografate.

Dati della Chiave
I dati della chiave sono rappresentati come una stringa codificata in Base64url e contengono la chiave effettiva, pronta per l'uso con PASETO. La chiave può essere simmetrica o asimmetrica, e il formato di questi dati può variare in base al tipo di chiave e alla versione specifica. In alcuni casi, i dati includono un hash della chiave, come accade per gli identificatori `pid` (per chiavi pubbliche) e `sid` (per chiavi segrete), che fungono da riferimento a una chiave senza esporre la chiave stessa.

PASERK rappresenta dunque una soluzione sicura e standardizzata per la gestione delle chiavi PASETO, riducendo i rischi di configurazioni errate e garantendo un'alta interoperabilità tra piattaforme. Inoltre, grazie alla sua struttura modulare, PASERK è in grado di supportare diversi algoritmi e applicazioni, offrendo flessibilità nella scelta dell'algoritmo da utilizzare per la firma dei token.
L'utilizzo di PASERK nei contesti di comunicazione insicura rappresenta dunque un notevole vantaggio in termini di sicurezza, poiché consente la condivisione sicura delle chiavi.

In conclusione, l'utilizzo di HTTP per lo scambio di informazioni sensibili è sconsigliato, poichè espone a numerosi pericoli.
Nel caso si scegliesse di designare HTTP come protocollo utilizzato per lo scambio di informazioni, è necessario adottare misure di sicurezza, come ridurre la validità dei token al minimo intervallo di tempo necessario. È inoltre possibile utilizzare varianti di token che utilizzano algoritmi di crittografia, come JWE, o sistemi di comunicazione delle chiavi in maniera sicura e standardizzata, come PASERK.
L'applicazione di queste best practices aiuta a mitigare i rischi associati all'utilizzo di token su HTTP, rendendo la trasmissione dei dati più sicura anche in assenza di protezione delle comunicazioni.

3.2 Scambio di token in HTTPS

L'utilizzo del protocollo HTTPS è fortemente consigliato rispetto ad HTTP.
Questo protocollo fornisce un canale di comunicazione sicuro, grazie all'utilizzo di crittografia per impedire che le informazioni scambiate vengano intercettate. HTTPS utilizza inoltre dei certificati, per identificare univocamente le entità coinvolte nella conversazione, eliminando il rischio di attacchi come il man in the middle, già citato in precedenza.

In una conversazione tramite HTTPS, si è sicuri che il contenuto dei messaggi non può essere letto in chiaro da entità terze, ed è dunque possibile utilizzare token non cifrati.
L'utilizzo di sistemi per certificare l'identità dei vari attori fornisce inoltre diversi vantaggi, che portano maggiore sicurezza al sistema.
Tra questi è presente il token binding, sistema che introduce sostanziali vantaggi, impedendo che un token possa essere ri-utilizzato in contesti diversi da quello in cui è stato rilasciato.

Token Binding
Il token binding[14],[15] permette al server di associare un token alla sessione in cui questo è stato rilasciato, in modo da poter controllare che esso venga utilizzato solamente dall'utente a cui è stato effettivamente rilasciato. Questo viene reso possibile tramite l'utilizzo di un header aggiuntivo per ogni richiesta, chiamato *Sec-Token-Binding*. Questo header contiene valori cifrati con una chiave segreta in possesso del client, e successivamente codificati in Base64url, che associano il token alla connessione TLS tramite la quale client e server comunicano. L'header Sec-Token-Binding viene creato dal client una volta stabilita una connessione HTTPS con il server. Esso contiene un messaggio di binding del

token (*TokenBindingMessage*), che include un identificatore di binding (*Token Binding ID*) generato dal client durante la connessione TLS utilizzata per comunicare con il server.
Il client invia questo header in ogni successiva richiesta HTTP verso il server, permettendo a quest'ultimo di verificare l'integrità della connessione e la sicurezza del token. Questo meccanismo offre il vantaggio di proteggere i token di autenticazione da attacchi di replica, poiché il token è legato in modo crittografico alla connessione TLS specifica tra client e server, rendendo impossibile il riutilizzo su altre connessioni.

Questo header può inoltre essere utilizzato all'interno di sistemi che coinvolgono servizi esterni, come provider di identità nel framework OAuth2.0, per comunicare al server dei dati relativi ad una connessione TLS che il client ha stabilito con server di terze parti.
In questo scenario, il client richiede ad un servizio, definito *Provider* l'emissione di un token, dimostrando di essere in possesso di una chiave privata specifica, in modo da certificare al server originale, definito *Consumer*, la propria identità.
Il diagramma di seguito riportato illustra il processo di rilascio di un token in un sistema che si interfaccia con servizi di autenticazione esterni e utilizza token binding per garantire maggiore sicurezza.

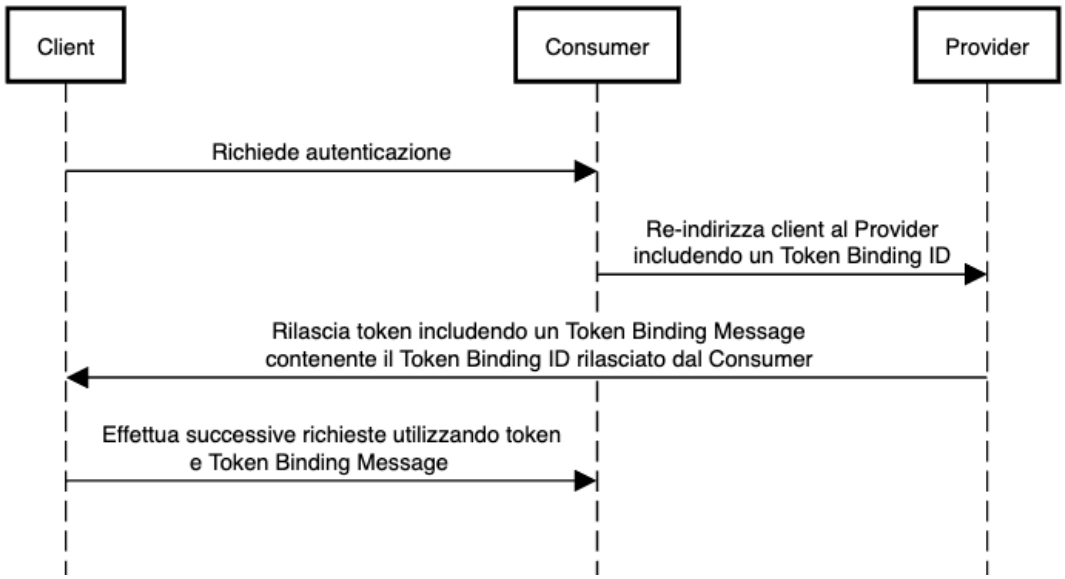

Figura 7: Diagramma di sequenza del rilascio di token con l'utilizzo di binding

In questo processo, il client certifica la propria identità attraverso l'utilizzo di una coppia di chiavi asimmetriche: utilizza la chiave privata per firmare le richieste al Provider durante il rilascio del token, mentre la chiave pubblica associata può essere usata dal Consumer per verificare la validità della firma. Un attaccante deve dunque essere in possesso della chiave privata del client per poter replicare l'header, aggiungendo un ulteriore livello di sicurezza al sistema.
Il token binding risulta dunque utile in tutti i casi in cui la connessione tra client e consumer non è diretta, dove esiste dunque un ulteriore livello intermedio, come un token provider o un load balancer. In tutti questi casi il token viene associato ad una connessione HTTPS differente rispetto a quella che il client ha stabilito con il consumer, permettendo di comunicare informazioni tra servizi di terze parti in maniera sicura.
Il token binding è dunque una pratica molto utile per aggiungere un livello di sicurezza ai meccanismi di autenticazione basati su token, poichè permette di associare un token alla sessione in cui esso viene rilasciato, permettendo inoltre di comunicare con servizi terzi in grado di rilasciare token di accesso.

Questo protocollo è la scelta migliore per lo scambio di informazioni in maniera sicura, di gran lunga preferibile rispetto ad HTTP. L'utilizzo di HTTPS rappresenta un'importante misura di sicurezza, grazie ai meccanisimi di protezione delle comunicazioni, e all'utilizzo di certificati per identificare le

entità coinvolte nella conversazione.
È tuttavia essenziale considerare l'intero ciclo di vita di un token per prevenire attacchi e garantire la protezione delle risorse. Mentre HTTPS protegge principalmente la comunicazione, il sistema dovrebbe implementare sistemi di conservazione e di logging sicuri in modo da minimizzare il rischio di intercettazione dei dati.

4 Conclusioni

Nel corso dell'elaborato sono state presentate tre tecnologie token based: JWT, PASETO e Macaroons. L'utilizzo di queste tecnologie permette di gestire in maniera sicura le richieste nelle moderne applicazioni Web distribuite. Rispetto all'utilizzo di sessioni o cookie, l'utilizzo di queste tecnologie riduce il carico sul server, favorendo la scalabilità dell'applicazione e permettendo di utilizzare risorse distribuite sul cloud senza bisogno di una centralizzazione delle informazioni.

JWT permette di creare token firmati in maniera flessibile, scegliendo tra i numerosi algoritmi di firma descritti dallo standard JOSE. Questa caratteristica può anche essere vista come una falla nella sicurezza, poichè richiede una implementazione molto precisa per evitare attacchi che coinvolgono l'utilizzo di algoritmi non sicuri, o addirittura inesistenti.
PASETO si pone dunque l'obiettivo di semplificare la gestione dei token, utilizzando solamente due algoritmi: uno a chiave simmetrica e uno a chiave asimmetrica, utilizzati rispettivamente dai PASETO local e dai PASETO public.
L'utilizzo di PASETO è però tutt'ora limitato della mancanza di uno standard RFC redatto dalla IETF. La pubblicazione di tale documento spingerebbe PASETO per essere considerato una reale alternativa a JWT, che è attualmente l'opzione più popolare.
Macaroons viene presentata come tecnologia pensata per la gestione delle risorse nel cloud, dove è richiesto un granulare controllo dei permessi di ogni utente. Questa tecnologia permette di gestire in maniera flessibile i token, aggiungendo caveat che modificano la validità del token, senza che questo debba essere riemesso. I macaroons sono dunque consigliati per l'utilizzo in applicazioni dove i permessi da concedere agli utenti possono variare in base a vari parametri, come risorsa richiesta
Viene infine presentato il framework OAuth2.0, utilizzato per la comunicazione con servizi terzi senza condividere le credenziali. Questo framework introduce sostanziali vantaggi nella gestione dei permessi degli utenti, permettendo di comunicare con app di terze parti, e di conseguenza aumentare l'interconnessione dei servizi. OAuth2.0 consente inoltre di gestire l'emissione di nuovi token tramite l'utilizzo di un refresher token.

Tutti e tre i tipi di token presentati risultano utilizzabili come access token e refresher token all'interno del framework OAuth2.0, JWT e PASETO sono utilizzabili in tutte le architetture moderne, mentre l'utilizzo di Macaroons è specialmente adatto ad applicazioni con risorse accessibili in cloud.
Le tecnologie presentate vengono analizzate approfondendone i vantaggi in termini di sicurezza, analizzando come possono proteggere dagli attacchi più comunemente effettuati verso le applicazioni Web.
Viene inoltre analizzato come i vari token siano soggetti a meccanismi di revoca e modifica, caratteristiche essenziali per la flessibilità di un sistema e per aumentare l'agilità nella risposta ad attacchi informatici.
È stato sviluppato un client in Java per la creazione e la verifica dei vari token, in modo da verificare i tempi richiesti per la generazione e la lettura dei vari token, da cui emerge come JWT sia più veloce rispetto a PASETO, e come Macaroon offra ottime performance a causa della sua struttura flessibile.
Viene inoltre analizzata la diffusione di JWT, PASETO e Macaroons, riportando le librerie più popolari che consentono di integrare queste tecnologie in tutti i progetti. Sotto questo aspetto JWT rimane di gran lunga il token più utilizzato dalla community, a causa della longevità del progetto e della grande quantità di risorse disponibili online.
Un approfondimento finale è dedicato allo scambio di token nei protocolli HTTP e HTTPS, descrivendo le misure di sicurezza da prendere in entrambi casi.
In una comunicazione tramite HTTP possono essere utilizzati token con payload cifrato, in modo da proteggerne il contenuto da intercettazioni di utenti terzi.
La scelta consigliata è dunque HTTPS, che tramite l'utilizzo di certificati garantisce che la comuicazione non venga intercettata, e utilizza crittografia per proteggere i dati in transito.

Possibili sviluppi futuri di questo elaborato comprendeono un approfondimento su altri framework per l'autenticazione, come OpenID connect, protocollo che mira a condividere l'identità degli utenti per centralizzare l'autenticazione su più piattaforme. Altri approfondimenti interessanti sarebbero senza

dubbio sull'utilizzo dell'autenticazione a due fattori (MFA) tramite dispositivi fisici o meccanismi digitali, come i codici temporanei o le notifiche push, per osservare se i token citati nel corso dell'elaborato possono essere utilizzati in maniera sicura anche in questi contesti.

5 Bibliografia

Riferimenti bibliografici

[1] M. Jones, J. Bradley e N. Sakimura. *RFC 7519: JSON Web Token (JWT)*. USA, 2015.

[2] Paseto Standard. *PASETO - Platform-Agnostic Security Tokens*. Consultato in data 2024-09-18. 2024. URL: https://github.com/paseto-standard/paseto-spec.

[3] Arnar Birgisson et al. "Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud". In: *Network and Distributed System Security Symposium*. 2014.

[4] Michael B. Jones e Joe Hildebrand. *JSON Web Encryption (JWE)*. RFC 7516. Mag. 2015. DOI: 10.17487/RFC7516. URL: https://www.rfc-editor.org/info/rfc7516.

[5] Daisuke Ajitomi. *Pyseto: PASETO for Python*. https://pypi.org/project/pyseto/. Consultato in data 2024-09-27. 2024.

[6] Kiteworks. *What is AES 256 Encryption?* Consultato in data 2024-09-24. 2024. URL: https://www.kiteworks.com/risk-compliance-glossary/aes-256-encryption/.

[7] Dick Hardt. *The OAuth 2.0 Authorization Framework*. RFC 6749. Ott. 2012. DOI: 10.17487/RFC6749. URL: https://www.rfc-editor.org/info/rfc6749.

[8] Michael B. Jones e Dick Hardt. *The OAuth 2.0 Authorization Framework: Bearer Token Usage*. RFC 6750. Ott. 2012. DOI: 10.17487/RFC6750. URL: https://www.rfc-editor.org/info/rfc6750.

[9] N. Sakimura, J. Bradley e M. Jones. *RFC 9101: The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR)*. USA, 2021.

[10] OWASP Foundation. *OWASP API Security Top 10 - 2021*. https://owasp.org/www-project-top-ten/. Consultato in data 2024-09-20. 2021.

[11] Nimbus Project. *Nimbus JOSE+JWT - Maven Repository*. Consultato in data 2024-09-21. 2024. URL: https://mvnrepository.com/artifact/com.nimbusds/nimbus-jose-jwt.

[12] OAuth. *jwt-decode - NPM Package*. Consultato in data 2024-09-22. 2024. URL: https://www.npmjs.com/package/jwt-decode.

[13] Paseto Standard. *PASERK: Platform Agnostic Serialized Keys*. Consultato in data 2024-10-07. 2022. URL: https://github.com/paseto-standard/paserk.

[14] Andrei Popov et al. *The Token Binding Protocol Version 1.0*. RFC 8471. Ott. 2018. DOI: 10.17487/RFC8471. URL: https://www.rfc-editor.org/info/rfc8471.

[15] Andrei Popov et al. *Token Binding over HTTP*. RFC 8473. Ott. 2018. DOI: 10.17487/RFC8473. URL: https://www.rfc-editor.org/info/rfc8473.

Ringraziamenti

Vorrei infine scrivere un sentito ringraziamento per tutte le persone che mi hanno accompagnato nel mio percorso di vita e di studi. Voglio ringraziare innanzitutto il professor Angelo Di Iorio, relatore di questa tesi che mi ha aiutato nella stesura dell'elaborato, fornendo consigli e spunti sempre puntuali e stimolanti. Lo ringrazio per il tempo speso e per l'aiuto fornito.

Ringrazio tutti i miei familiari per avermi sempre fatto sentire il loro affetto, e per avermi supportato con gioia ed entusiasmo durante tutta la mia vita, i miei genitori Cristiana e Domenico, mio fratello Francesco, i nonni Anna, Carlo, Lia e Martino, gli zii Jonathan, Micaela e Ivano.

Voglio ringraziare la mia fantastica fidanzata Giorgia, per amarmi e per farmi crescere come persona tutti i giorni.
Ringrazio tutti i compagni di università con cui ho condiviso questo cammino, per le avventure passate insieme e per i legami costruiti in questi anni, oltre che per l'aiuto indispensabile durante lo studio per gli esami.

Ringrazio infine tutti i colleghi che ho incontrato nella mia, seppur breve, esperienza lavorativa, in particolare Fabrizio , tutor durante il mio tirocinio curriculare, per l'aiuto fornito sia dal punto di vista tecnico che umano.