Tokens

Nicolò Pellegrinelli

April 15, 2024

1 Introduzione

JSON Web Token (JWT) è uno standard aperto (RFC 7519) che definisce un modo compatto per trasmettere informazioni in modo sicuro tra due parti come oggetti JSON. Queste informazioni possono essere verificate e attendibili perché sono firmate digitalmente.

Le firme digitali possono essere generate utilizzando algoritmi HMAC (chiave segreta), RSA (coppia chiave pubblica/privata) o ECC (curve ellittiche).

2 Usi

- **Autorizzazione**: Una volta che un utente si è autenticato, il server può generare un JWT, che può essere utilizzato per accedere a risorse specifiche senza dover autenticare l'utente ogni volta. Questo è utile per le API RESTful, dove l'autenticazione è necessaria per ogni richiesta. Single Sign On (SSO) è un altro esempio di utilizzo di JWT per l'autenticazione.
- **Scambio di informazioni**: Poiché i JWT possono essere firmati, si può essere sicuri che il mittente sia chi dice di esserlo e che i contenuti del messaggio non siano stati alterati.

3 Struttura di un JWT

Un JWT è composto da tre parti separate da punti: header, payload e signature.

$$header.payload.signature$$

3.1 Header

L'header contiene il tipo di algoritmo di firma utilizzato e il tipo di token. Questo JSON viene poi codificato in Base64URL, che è una codifica Base64 sicura per gli URL.

3.2 Payload

Il payload contiene le informazioni che si desidera trasmettere. Queste informazioni riguardano un'entità (solitamente l'utente) e i metadati. Anche questo JSON viene codificato in Base64URL.

È importante notare che il payload non è crittografato, quindi non dovrebbe contenere informazioni sensibili in quanto possono essere lette da chiunque.

3.3 Signature

La firma viene generata utilizzando l'header codificato a cui viene aggiunto il payload codificato e una chiave segreta. Questo risultato viene firmato utilizzando l'algoritmo specificato nell'header. La firma viene quindi aggiunta al token.

La firma è utilizzata per verificare che il messaggio non sia stato modificato. Inoltre, se il token è stato firmato con una chiave privata, può anche verificare l'autenticità del mittente.

4 Come funziona un JWT

Quando un utente si autentica, un token JWT viene generato e inviato al client. Ogni volta che l'utente fa una richiesta al server, il token viene inviato insieme alla richiesta. Il server può quindi verificare la firma del token per assicurarsi che sia valido e che l'utente abbia i permessi necessari per accedere alla risorsa richiesta.

Generalmente i token JWT hanno una scadenza breve per garantire un livello di sicurezza alto. Questi token sono chiamati Access Tokens e sono quelli utilizzati per l'accesso alle risorse protette. Un altro tipo di token è il Refresh Token, che viene utilizzato per ottenere un nuovo Access Token una volta che il precedente è scaduto. I Refresh Token hanno una vita più lunga degli Access Token e interagiscono con gli authorization server invece di quelli dei resource server.

5 JSON Web Encryption (JWE)

Il JSON Web Encryption standard stabilisce un modo per crittografare, e quindi rendere oscuri, i contenuti di un JWT.

A primo impatto, potrebbe sembrare che la crittografia abbia le stesse garanzie della firma, con l'aggiunta della riservatezza dei dati. Tuttavia la crittografia non garantisce l'autenticità dei dati, ma solo la loro riservatezza.

Come per JWS, JWE supporta principalmente due schemi: uno schema a chiave segreta e uno a chiave pubblica.

Lo schema a chiave segreta funziona in modo simile a quello in JWS, in cui ogni parte che detiene la chiave segreta può crittografare e decrittografare i dati.

Lo schema a chiave pubblica, invece, funziona in maniera differente. In questo caso sono le parti che detengono la chiave pubblica a crittografare i dati, mentre la parte che detiene la chiave privata può decrittografarli. In pratica, in JWE chi detiene la chiave pubblica può creare nuovi messaggi crittografati, mentre in JWS può solo verificarne l'autenticità. Questo vuol dire che JWE non fornisce le stesse garanzie di JWS e, quindi, non può rimpiazzare il ruolo della firma.

Per questi motivi JWE è spesso utilizzato in combinazione con JWS: un token JWT crittografato funziona da container per un token JWT firmato in modo da ottere i benefici di entrambi.

6 JSON Web Signature (JWS)

Il JSON Web Signature standard stabilisce un singolo algoritmo di firma supportato da tutte le implementazioni: HMAC con SHA-256, chiamato HS256. In aggiunta, secondo il JSON Web Algorithms standard (JWA), anche altri algoritmi sono raccomandati per l'uso con JWT. Tra questi RSASSA PKCS1 v1.5 con SHA-256 (RS256) e ECDSA con P-256 a curva ellittica e SHA-256 (ES256).

7 HMAC

Keyed-Hash Message Authentication Code (HMAC) è un algoritmo di firma che combina un certo messaggio con una chiave segreta utilizzando una funzione hash crittografica. Il risultato è un codice di autenticazione che può essere utilizzato per verificare un messaggio solo se le parti generatrice e verificatrice condividono la stessa chiave segreta.

La robustezza della funzione hash garantisce che il messaggio non possa essere modificato senza conoscere la chiave segreta.

In sostanza, HMAC permette di verificare l'integrità e l'autenticità di un messaggio attraverso chiavi segrete condivise.

Sia:

- H la funzione hash crittografica
- B la lunghezza del blocco di H
- K la chiave segreta
- K' la vera chiave utilizzata da H e derivata da K
- *ipad* il byte 0x36 ripetuto B volte (chiamato anche padding interno)
- *opad* il byte 0x5C ripetuto B volte (chiamato anche padding esterno)

- M il messaggio
- || l'operatore di concatenazione

L'algoritmo HMAC è definito come:

```
HMAC(K, M) = H((K' xor opad) || H((K' xor ipad) || M))
```

dove K' è definito come segue:

- Se K è più corto di B, vengono aggiunti zeri a sinistra per raggiungere la lunghezza di B.
- Se K è più lungo di B, viene calcolato $H(K)$.
- Se K è esattamente di B byte, K' è uguale a K.

8 RSASSA

RSASSA è una variazione dell'algoritmo RSA, che è un algoritmo di crittografia a chiave pubblica, adattato alle firme.

Gli algoritmi a chiave pubblica si basano sulla generazione di due chiavi, una privata e una pubblica, per crittografare e decrittografare rispettivamente i dati.

8.1 RSA

L'algoritmo RSA si basa sul fatto che la fattorizzare un numero intero composto è considerato un problema difficile. Questo problema può essere rappresentato come segue:

$$(m^e)^d \equiv m \mod n$$

In cui m è il messaggio originale ed e, d e n sono scelti nel seguente modo:

1. p e q sono due grandi primi generati casualmente.
   - Un Random Number Generator (RNG) crittograficamente sicuro dovrebbe essere utilizzato per generare questi numeri.
   - Non esistendo modo di generare randomicamente numeri primi, è necessario verificare che i numeri generati siano effettivamente primi.
   - I numeri primi devono essere grandi e simili per ordine di grandezza.
2. n è il risultato del prodotto $p \cdot q$. Questo numero è il modulo e il suo numero di bit equivale alla lunghezza della chiave.
3. $\phi(n) = (p-1) \cdot (q-1)$ è il quoziente di Eulero ($phi(n)$).

4. e è un numero intero scelto in modo che $1 < e < \phi(n)$ e e sia coprimo con $\phi(n)$.
5. d deve soddisfare la congruenza $d \equiv e^{-1} \mod \phi(n)$.
    - d è l'inverso moltiplicativo di e modulo $\phi(n)$.
    - L'equazione può essere riscritta come $e \cdot d \equiv 1 \mod \phi(n)$.

La chiave pubblica è composta dai valori di n ed e, mentre la chiave privata è composta dai valori di n ed d. p, q e $\phi(n)$ sono valori che devono essere mantenuti segreti.

8.2 Firma e Verifica con RSA

Per firmare un messaggio RSA funziona come segue:

1. Un digest del messaggio viene calcolato da una funzione hash.
2. Il digest viene poi elevato alla potenza di d modulo n (chiave privata).
3. Il risultato è aggiunto al messaggio come firma.

Per verificare la firma invece:

1. La firma viene elevata alla potenza di e modulo n (chiave pubblica). Questo restituisce il digest originale.
2. Viene calcolato il digest del messaggio ricevuto, utilizzando la stessa funzione hash del passaggio di firma.
3. Se i due digest sono uguali, la firma è valida.

Questo processo è conosciuto come Signature Scheme with Appendix (SSA). La firma è un "appendice" del messaggio originale in quanto è necessario nel processo di verifica della firma.

In una firma con RSA la chiave privata viene utilizzata creare una firma che per verificarne l'autenticità. In contrasto, la chiave pubblica può essere utilizzata solo per verificare l'autenticità di un messaggio. Questo schema permette dunque la possibilità di distribuire in modo sicuro uno-a-molti messaggi firmati: le parti riceventi possono verificare l'autenticità del messaggio mantenendo una copia della chiave pubblica, ma non possono creare nuovi messaggi con essa.

9 ECC

L'esistenza di algoritmi di fattorizzazione efficienti come il General Number Field Sieve (GNFS) o il Quadratic Sieve (QS) ha reso RSA con chiavi brevi vulnerabile. Tuttavia, all'aumentare dei numeri da fattorizzare, non solo gli algoritmi di fattorizzazione diventano più efficienti, ma anche l'algoritmo RSA diventa più lento, richiedendo chiavi che crescono ancora più velocemente. Questo

diventa un problema insostenibile sul lungo periodo, in particolare per dispositivi con risorse limitate. Ciò può potenzialmente rendere RSA insicuro e non più utilizzabile.

Algoritmi a curva ellittica sono un'alternativa più efficiente a RSA. Tra questi, l'Elliptic Curve Diffie-Hellman (ECDH) per lo scambio di chiavi e l'Elliptic Curve Digital Signature Algorithm (ECDSA) per la firma digitale sono i più comuni. Anche questo tipo di algoritmi genera una coppia di chiavi privata/pubblica, ma utilizza, invece della fattorizzazione di numeri semiprimi, il logaritmo discreto su curve ellittiche. Le curve ellittiche per la crittografia sono definite dalla seguente equazione di terzo grado:

$$y^2 = x^3 + ax + b$$

dove a e b sono parametri della curva.

Gli algoritmi a curve ellittiche sono definiti su campi primi finiti, ovvero insiemi di numeri interi su cui sono definite due operazioni binarie: somma e moltiplicazione. Con campo primo finito si intende che il numero di elementi nel campo è un numero primo p, quindi una quantità finita. Tutti gli elementi e le operazioni sono definite modulo p.

Rendendo il campo finito, gli algoritmi usati per le operazioni matematiche cambiano. In particolare, il logaritmo discreto viene usato al posto del logaritmo normale.

Essendo il campo composto da un numero finito di elementi, si potrebbe pensare che il logaritmo (discreto) diventi un problema semplice, tuttavia, non esiste ad oggi un algoritmo efficiente per la risoluzione del logaritmo discreto su curve ellittiche. Questa proprietà rende il logaritmo discreto ideale per la crittografia e la firma digitale.

Per rendere però l'algoritmo sicuro, è necessario che la curva, ovvero i parametri a e b, sia scelta in modo corretto. In passato, infatti, si sono verificati casi in cui certi parametri hanno reso le curve deboli.

L'aspetto interessante degli algoritmi a curva ellittica è che la lunghezza della chiave può essere ridotta di molto rispetto a RSA per ottenere lo stesso livello di sicurezza. Ad esempio, una chiave EC a 256 bit è considerata sicura quanto una chiave RSA a 3072 bit.

Ciò, oltre che a rendere più efficienti gli algoritmi, semplifica la trasmissione e la memorizzazione delle chiavi. Il motivo per cui la chiave EC può essere più corta risiede nell'inesistenza, ad oggi, di algoritmi che risolvono il logaritmo discreto più efficientemente dell'approccio naive o brute-force, diversamente da RSA che è più vulnerabile a fattorizzazione efficiente.

In un suo studio, il matematico Arjen K. Lenstra ha esposto il concetto di "Universal Security" calcolando l'energia necessaria per rompere determinti al-

goritmi crittografici e comparandola con la quantità di acqua che quell'energia sarebbe in grado di portare a ebollizione. Con questo metodo, Lenstra ha dimostrato che per rompere una chiave RSA di 228 bit è richiesta meno energia di quella necessaria per portare a ebollizione un cucchiaino d'acqua. Al contrario, per rompere una chiave EC a 228 bit è necessaria una quantità di energia che porterebbe a ebollizione l'intera acqua presente sulla Terra. Per questo livello di sicurezza con RSA è necessaria una chiave di almeno 2380 bit.

9.1 Usi di ECC

Nonostante la sua sicurezza e la sua efficienza, l'uso di ECC è ancora limitato rispetto a RSA. Tuttavia negli ultimi anni si è assistito ad un aumento dell'interesse per questa tecnologia e ad un incremento del suo utilizzo in varie applicazioni. Il governo degli Stati Uniti, ad esempio, ha adottato ECC per la crittografia delle comunicazioni, il progetto Tor lo utilizza per aiutare a garantire l'anonimato degli utenti e il protocollo TLS lo utilizza per garantire la sicurezza delle comunicazioni su Internet tramite HTTPS. Anche i servizi di messaggistica di iMessage e Whatsapp utilizzano ECC per garantire la sicurezza delle loro comunicazioni. ECC è inoltre utilizzato in molte applicazioni di blockchain, come Bitcoin ed Ethereum, al fine di approvare le transazioni.

9.2 Elliptic Curve Digital Signature Algorithm (ECDSA)

L'Elliptic Curve Digital Signature Algorithm è un algoritmo di firma digitale basato su curve ellittiche. Il funzionamento è simile a quello di RSA, ma utilizza curve ellittiche al posto dei numeri interi. Immaginiamo che User A voglia inviare un messaggio firmato a User B. A sceglie in modo randomico un numero d_A nel range $[1, n-1]$, che rappresenta la chiave private e dove n è l'ordine del punto base G della curva ellittica ed è un numero primo. A questo punto, A calcola il punto $Q_A = d_A \cdot G$ e la coordinata x di Q_A. Il risultato è un punto della curva che rappresenta la chiave pubblica. La curva ellittica e i suoi parametri (inclusi G e n) vengono prestabiliti in base a uno standard crittografico.

Dunque, per firmare un messaggo User A procede come segue:

1. Calcola il digest $H(m)$ del messaggio m con una funzione hash crittografica.
2. $H(m)$ viene poi convertito in un numero intero intero h ridotto modulo n.
3. Un numero k viene scelto randomicamente nel range $[1, n-1]$.
4. Calcola il punto $k \cdot G$ e la sua coordinata x viene ridotta modulo n, ottenendo r. Se $r = 0$, viene scelto un nuovo k.
5. Calcola $s = (h + rd_A)/k \mod n$. Se $s = 0$, viene scelto un nuovo k.
6. La coppia (r, s) è la firma del messaggio.

È importante notare che k deve essere scelto in modo randomico e non deve essere mai riutilizzato con la stessa chiave privata. L'utilizzo di k più volte con la stessa chiave privata può portare alla compromissione della chiave privata.

Per verificare la firma User B procede come segue:

1. Verifica che r e s siano nel range $[1, n-1]$ e che Q_A sia un punto sulla curva il cui prodotto con n è l'elemento neutro.
2. Calcola il digest $H(m)$ del messaggio m con una funzione hash crittografica.
3. $H(m)$ viene poi convertito in un numero intero intero h ridotto modulo n.
4. Calcola $w = s^{-1} \mod n$.
5. Calcola $u_1 = hw \mod n$ e $u_2 = rw \mod n$.
6. Calcola il punto $u_1 \cdot G + u_2 \cdot Q_A$ e la sua coordinata x viene ridotta modulo n, ottenendo v.
7. La firma è valida se $v = r$.

9.3 ECDSA for Bitcoin

L'algortimo ECDSA è estensivamente utilizzato in Bitcoin, come anche in Ethereum e in molte altre criptovalute, per validare le transazioni.

10 Diffie Hellman Key Exchange (D-H)

Il Diffie-Hellman Key Exchange (D-H) è un protocollo crittografico che permette a due parti che non hanno una conoscenza pregressa di stabilire una chiave segreta condivisa su un canale di comunicazione non sicuro. Questo protocollo fornisce le basi per una varietà di diversi protocolli crittografici. In particolare, è alla base della perfetta segretezza di TLS, il protocollo utilizzato da HTTP(S) per garantire la sicurezza delle comunicazioni. Il protocollo originale, sviluppato da Whitfield Diffie e Martin Hellman nel 1976, non prevedeva la firma digitale e quindi non garantiva l'autenticità delle parti coinvolte nella comunicazione, rendendo il protocollo vulnerabile al Man-in-the-middle Attack (MITM).

L'algoritmo si basa sulla difficoltà di calcolare il logaritmo discreto su campi finiti. Inizialmente, le due parti scelgono un numero primo p e un generatore g del campo finito tale che $g^x \mod p$, per $x = 1, 2, ..., p-1$, generi tutti gli elementi del campo in qualche permutazione. Per ogni intero b minore di p e radice primitiva di p, esiste un unico esponente i tale che $b = a^i \mod p$. Questo esponente è chiamato logaritmo discreto di b in base a modulo p.

Supponiamo che utente A e utente B vogliano stabilire una chiave segreta condivisa. Il protocollo funziona come segue: un numero primo p e un intero α, che è una radice primitiva di p, vengono scelti e resi pubblici. Gli utenti A e B

scelgono in modo casuale rispettivamente due numeri segreti X_a e X_b minori di p e calcolano $Y_a = \alpha^{X_a} \mod p$ e $Y_b = \alpha^{X_b} \mod p$. Dopodiché, A e B scambiano i loro valori pubblici Y_a e Y_b mantenendo segreti i loro valori privati X_a e X_b. Infine, A calcola $K = Y_b^{X_a} \mod p$ e B calcola $K = Y_a^{X_b} \mod p$. Entrambi otterranno lo stesso valore K che sarà la chiave segreta condivisa. Infatti:

$$\begin{aligned} K &= Y_b^{X_a} \mod p \\ &= (\alpha^{X_b} \mod p)^{X_a} \mod p \\ &= \alpha^{X_a \cdot X_b} \mod p \\ &= \alpha^{X_b \cdot X_a} \mod p \\ &= (\alpha^{X_a} \mod p)^{X_b} \mod p \\ &= Y_a^{X_b} \mod p = K \end{aligned} \tag{1}$$

In questo modo i due utenti si sono scambiati una chiave segreta senza che nessuno dei due abbia mai inviato la propria chiave privata. Inoltre, un attaccante che intercetta i valori pubblici Y_a e Y_b non può risalire ai valori privati X_a e X_b senza conoscere il logaritmo discreto su campi finiti. La sicurezza del protocollo si basa sulla relativa facilità di calcolare esponenziali modulari e la difficoltà di calcolare logaritmi discreti. In particolare, per primi grandi, quest'ultima operazione è considerata infattibile.

10.1 Autenticazione in Diffie Hellman

Il protocollo D-H originale non prevedeva l'autenticazione delle parti coinvolte nella comunicazione, rendendolo vulnerabile a MITM e Impersonation Attack.

Nell'Impersonation Attack, l'attaccante intercetta un messaggio di A indirizzato a B, contenente la chiave pubblica Y_a, e invia ad A la propria chiave pubblica Y_{att} con l'User ID di B. In questo modo A genera la chiave segreta condivisa credendo di star comunicando con B.

Nel Man-in-the-Middle Attack, un attaccante si interpone tra le due parti e stabilisce due connessioni separate, una con A e una con B, impersonando entrambe le parti.

Si può notare come il protocollo è soggetto a questo tipo di attacchi in quanto non è presente nessun tipo di autenticazione tra le parti. Esistono generalmente tre metodi differenti per garantire l'autenticità delle parti in D-H:

- **Firma Digitale**: Lo scambio di chiavi è autenticato da una firma, ovvero un hash ottenibile da entrambi gli utenti, ciascuno crittografato con la propria chiave privata. Questo hash viene generato da parametri importanti come l'User ID.
- **Cifratura a chiave pubblica**: Alcuni parametri importanti vengono crittografati con la propria chiave privata.

- **Chiave simmetrica**: Una chiave simmetrica condivisa attraverso altri mezzi viene utilizzata per autenticare la comunicazione.

10.2 Diffie Hellman Key Exchange con Curve Ellittiche (ECDH)

Il Diffie-Hellman Key Exchange può essere implementato anche con curve ellittiche. User A and B scelgono una curva ellittica $E(a, b)$ e un punto G su di essa, che funge da generatore. Questi parametri vengono resi pubblici. A e B scelgono, a questo punto, due numeri casuali e segreti X_a e X_b e calcolano rispettivamente $Y_a = X_a \cdot G$ e $Y_b = X_b \cdot G$, dove $\cdot$ indica il prodotto scalare. A e B si scambiano i valori pubblici Y_a e Y_b e calcolano grazie ad essi la chiave segreta condivisa K.

$$\begin{aligned} K &= X_a \cdot Y_b \\ &= X_a \cdot (X_b \cdot G) \\ &= X_b \cdot (X_a \cdot G) \\ &= X_b \cdot Y_a = K \end{aligned} \tag{2}$$

Anche in questo caso, un utente malevolo che intercetta i valori pubblici Y_a e Y_b non può risalire ai valori privati X_a e X_b senza conoscere il logaritmo discreto su curve ellittiche e quindi non può ottenere la chiave segreta condivisa.

Come per il D-H con numeri interi, anche ECDH non prevede l'autenticazione delle parti coinvolte nella comunicazione e quindi è vulnerabile a MITM Attack.

11 Considerazioni sulla Sicurezza di JWT

Essendo il token firmato, non è possibile modificarlo senza invalidare la firma. Tuttavia, bisogna considerare alcuni aspetti per garantirne la sicurezza, come la protezione delle chiavi, quella privata nel caso asimmetrico e quella segreta nel caso simmetrico. Ciò è ugualmente importante nel caso di token crittografati.

Un altro aspetto da considerare è l'invio di token contenenti informazioni sensibili. In questo caso è obbligatorio utilizzare misure che evitano la divulgazione di queste informazioni. Un modo per farlo è utilizzare token crittografati. Un altro modo è quello di utilizzare protocolli sicuri per la trasmissione di dati come HTTPS, che si basa su TLS (Transport Layer Security) Omettere informazioni sensibili dai token è sicuramente la soluzione più semplice e sicura.

Nel caso in cui firma e crittografia siano necessarie contemporaneamente, è solito firmare il token per crittografarlo successivamente. Questo previene attacchi in cui la firma viene rimossa come il Signature Stripping Attack.

Esistono, inoltre, anche altre vulnerabilità che possono essere sfruttate per compromettere la sicurezza di un'applicazione che utilizza JWT.

11.1 Signature Stripping Attack

Un metodo comune per aggirare la verifica della firma è semplicemente quello di rimuovere la firma dal token. Essendo le tre parti del token codificate separatamente, l'attacco può essere effettuato facilmente rimuovendo la firma e cambiando l'header.

L'attacco è facilmente prevenibile assicurandosi che il server accetti solo token firmati.

11.2 Cross-Site Request Forgery Attack (CSRF)

Attacchi CSRF effettuano richieste non autorizzate verso siti in cui l'utente è autenticato, ingannando il browser a inviare richieste da un sito diverso.

Un elemento nel sito malevolo che contiene l'URL del sito target invia una richiesta ogni volta che viene visitato. Se il sito target utilizza i cookie per la sessione, questa richiesta conterrà le credenziali dell'utente.

JWT token con breve validità di tempo possono mitigare questo attacco. Un'altra soluzione è l'utilizzo di speciali campi che vengono aggiunti alle richieste solo quando provengono da un sito autorizzato. Inoltre, essendo questo un'attacco basato sui cookie, se i token non vengono memorizzati nei cookie, l'attacco non può essere effettuato.

11.3 Cross-Site Scripting Attack (XSS)

Gli attacchi XSS consentono a un attaccante di eseguire script nel browser di un utente. Questo script può essere utilizzato per rubare i token accedendo ai cookie e al localStorage.

Molti attacchi XSS sono causati da una mancata validazione dei dati passati al server. Infatti, se il backend non sanifica i dati, un attaccante può inserire dati in modo che vengano interpretati come script.

Per risolvere questo problema è sempre necessario sanificare i dati in ingresso e, se vengono usati i cookie per salvare i token, è possibile utilizzare la flag HttpOnly per impedirne l'accesso da parte di script.

Fonti

Risorse per questo articolo:

- Documentazione JWT: jwt.io/introduction
- The JWT Handbook: auth0.com/resources/ebooks/jwt-handbook
- Access Tokens vs Refresh Tokens: ietf.org/doc/html/rfc6749
- RFC 7519: ietf.org/doc/html/rfc7519

- ECC Wikipedia: wikipedia.org/wiki/Elliptic-curve-cryptography
- ECC Cloudflare Article: cloudflare.com/elliptic-curve-cryptography
- Universal Security: eprint.iacr.org/2013/635.pdf
- Diffie Hellman Key Exchange: ieeexplore.ieee.org/document/5485276
- Elliptic Curve Diffie Hellman: ieeexplore.ieee.org/document/8192159
- ECDSA: ietf.org/doc/html/rfc6979
- ECDSA Wikipedia: wikipedia.org/wiki/EC-Digital-Signature-Algorithm
- EC Point Multiplication: wikipedia.org/wiki/EC-point-multiplication