Agenzia per la
Cybersicurezza Nazionale

LINEE GUIDA FUNZIONI CRITTOGRAFICHE

Transport Layer Security (TLS)

MAGGIO 2026

Versione	Data di pubblicazione	Note
1.0	**26/02/2025**	**Prima pubblicazione**
2.0	**12/05/2026**	**Aggiunta sezione “Scopo del documento”, introdotte soluzioni post-quantum, ulteriori modifiche minori**

Sommario

Indice delle figure

Indice delle tabelle

Scopo del documento

Questo documento costituisce parte della serie "Linee Guida Funzioni Crittografiche" e fornisce le raccomandazioni di ACN in merito al protocollo **Transport Layer Security** e alle configurazioni considerate sicure. Per ulteriori informazioni sulle Linee Guida e sulle utilità delle funzioni crittografiche in base ai contesti specifici, si faccia riferimento al documento introduttivo della serie [1].

Ogni documento tiene in considerazione le minacce presenti al giorno della sua pubblicazione. Data la diversa natura dei sistemi informativi di destinazione, non è possibile garantire che queste raccomandazioni possano essere utilizzate senza adattamenti specifici. In qualsiasi caso, la pertinenza dell'attuazione delle soluzioni proposte deve essere sottoposta, preventivamente, a valutazione e validazione da parte dei responsabili della sicurezza dei sistemi informativi di destinazione.

I contenuti delle Linee Guida sono indirizzati a sviluppatori, produttori di dispositivi e fornitori di servizi digitali, al fine di promuovere l'utilizzo di primitive crittografiche e relativi parametri di configurazione sicuri fin dalla fase di progettazione di prodotti, reti, applicazioni e servizi. Questi documenti sono altresì rivolti a security manager, referenti e responsabili della cybersicurezza di soggetti pubblici e privati affinché verifichino che i sistemi utilizzati dalla propria organizzazione siano conformi alle raccomandazioni fornite.

La legge 28 giugno 2024, n. 90 relativa a "Disposizioni in materia di rafforzamento della cybersicurezza nazionale e di reati informatici", all'articolo 9, stabilisce a tal fine che *«le strutture di cui all'articolo 8 della presente legge nonché quelle che svolgono analoghe funzioni per i soggetti di cui all'articolo 1, comma 2-bis, del decreto-legge 21 settembre 2019, n. 105, convertito, con modificazioni, dalla legge 18 novembre 2019, n. 133, e al decreto legislativo 18 maggio 2018, n. 65, verificano che i programmi e le applicazioni informatiche e di comunicazione elettronica in uso, che utilizzano soluzioni crittografiche, rispettino le linee guida sulla crittografia nonché quelle sulla conservazione delle password adottate dall'Agenzia per la cybersicurezza nazionale e dal Garante per la protezione dei dati personali e non comportino vulnerabilità note, atte a rendere disponibili e intellegibili a terzi i dati cifrati»*.

Il documento è stato curato dal Centro Nazionale di Crittografia istituito presso ACN.

1 Introduzione

Una delle principali classificazioni degli schemi crittografici si basa sulle caratteristiche delle chiavi utilizzate per proteggere la comunicazione tra due utenti: nella crittografia a chiave pubblica ogni partecipante possiede una coppia di chiavi, una pubblica e una privata, mentre nella crittografia simmetrica i due partecipanti possiedono una stessa chiave segreta, che deve essere scambiata in maniera sicura precedentemente.

La crittografia simmetrica è generalmente migliore della controparte in quanto a prestazioni, ma la necessità di uno scambio sicuro della chiave è una complicazione rilevante. Una soluzione molto utilizzata consiste nello scambiare la chiave simmetrica da utilizzare durante la comunicazione mediante uno schema di cifratura a chiave pubblica. Per questo motivo, in particolare nelle applicazioni pratiche, è comune che i due tipi di crittografia interagiscano fra di loro. Ne sono un esempio importante le comunicazioni a grandi distanze, come quelle online.

Uno dei protocolli più utilizzati nella navigazione web, nei servizi di messaggistica istantanea e nelle comunicazioni nelle reti VoIP è chiamato Transport Layer Security (TLS). Questo protocollo permette di proteggere le comunicazioni tra un server e un client tramite l'utilizzo di diverse primitive crittografiche e ha subito diverse modifiche nel tempo, sia per aggiornare gli schemi sulla base dei risultati della ricerca crittografica, sia per risolvere le vulnerabilità del protocollo stesso che sono state trovate negli anni. Attualmente, la versione più moderna è la 1.3, ma rimane ancora largamente utilizzata la precedente versione 1.2. Questo, normalmente, non inficia la sicurezza del protocollo, ma la versione 1.2 accetta un insieme di suite crittografiche più ampio, alcune delle quali comprendono schemi che sono diventati meno sicuri con i recenti sviluppi tecnologici e della ricerca.

Il documento presenta la seguente struttura: nel capitolo 2 si presenta il protocollo TLS, evidenziando le differenze tra la versione 1.2 e la 1.3; nel capitolo 3 si definiscono ed elencano le suite crittografiche accettate nelle due versioni in esame; nel capitolo 4 si presentano le tipologie di attacco più note, spiegando il motivo delle vulnerabilità e proponendo delle contromisure per mitigarle; infine, nel capitolo 5 si fornisce una raccomandazione dettagliata sulle versioni di TLS da utilizzare e sulle relative suite crittografiche. Le raccomandazioni includono anche un'indicazione sulle estensioni suggerite e quelle sconsigliate.

2 TLS e le sue versioni

Il **Transport Layer Security** (TLS) è un protocollo crittografico ideato per assicurare la sicurezza della comunicazione in rete e sviluppato a partire dalla struttura del precedente **Secure Sockets Layer** (SSL). La versione 1.0 di TLS, pubblicata nel 1999 [2], e la versione 1.1 del 2006 [3] sono considerate obsolete dal 2021 e sono state aggiornate con le versioni 1.2 e 1.3, pubblicate rispettivamente nel 2008 e 2018 [4, 5]. Il protocollo viene modellato tramite il classico sistema **client/server**, secondo il quale un client, cioè un utente o un dispositivo, si connette a un server per fruire di un particolare servizio da esso fornito.

In generale, il protocollo TLS è organizzato in due fasi:

- inizializzazione della comunicazione, detta **handshake**;
- comunicazione sicura tramite cifratura simmetrica e autenticazione dei messaggi.

La prima fase ha lo scopo di definire i dettagli necessari per instaurare un canale sicuro, in particolare: la versione di TLS da utilizzare, gli algoritmi crittografici da utilizzare per la cifratura e l'autenticazione dei messaggi e la chiave di sessione condivisa, utilizzata per generare la chiave segreta per la cifratura autenticata. In alcuni casi, si può optare per utilizzare una chiave simmetrica scambiata precedentemente, cioè una **Pre-Shared Key** (PSK) [6].

Inoltre, durante questa prima fase, il server fornisce al client un **certificato digitale**, il quale contiene la chiave pubblica dalla quale il client deriva il segreto condiviso utilizzato per generare la chiave di sessione.

I certificati digitali vengono rilasciati da una **Certification Authority** (CA). Contengono, oltre a dati accessori, il nome dell'utente, la sua chiave pubblica, il periodo di validità del certificato (crittoperiodo) e la firma della CA. In TLS, il formato più utilizzato è X.509 [7].

La seconda fase è la comunicazione vera e propria tra server e client, che possono scambiarsi messaggi in modo sicuro utilizzando le chiavi e gli algoritmi di cifratura selezionati in fase di negoziazione.

La fase di handshake è la parte fondamentale del protocollo, durante la quale vengono svolti passi diversi in base alla versione impiegata.

2.1 Handshake con TLS 1.2

La fase di handshake di TLS 1.2 prevede i seguenti passaggi:

1. il client invia al server il primo messaggio, detto **ClientHello**, con il quale comunica la versione di TLS che intende utilizzare, le suite crittografiche supportate e un numero casuale necessario per la generazione della chiave segreta condivisa;

2. la risposta del server si struttura in più messaggi:
   - **ServerHello**, che conferma la versione di TLS adottata e la scelta della suite crittografica da utilizzare, scelta tra quelle nella lista fornita dal client, assieme a un valore numerico casuale necessario per la generazione della chiave segreta condivisa;
   - **Certificate**, cioè il proprio certificato che, come già specificato, contiene la propria chiave pubblica e identifica il server come suo legittimo proprietario;
   - se lo schema per lo scambio di chiave scelto lo necessita, il server invia la propria parte di chiave (ServerKeyExchange);
   - nelle situazioni in cui si richiede l'autenticazione del client, il server potrebbe inviare una richiesta del suo certificato (CertificateRequest);
   - **ServerHelloDone**, che specifica la fine delle negoziazioni tra le due parti;
3. il client risponde inviando:
   - il suo certificato, se richiesto dal server, e un eventuale messaggio firmato digitalmente che permette di identificare il client come legittimo proprietario della chiave pubblica contenuta nel certificato (Certificate, CertificateVerify);
   - **ClientKeyExchange**, cioè la sua parte per lo scambio di chiave, che determina la chiave segreta (eventualmente insieme a ServerKeyExchange);
   - l'avviso **ChangeCipherSpec**, che determina l'inizio delle comunicazioni cifrate e autenticate tramite gli algoritmi di cifratura selezionati nella fase di negoziazione;
   - il messaggio **Finished**, contenente la cifratura autenticata di tutti i messaggi precedentemente scambiati;
4. il server deriva la chiave segreta condivisa, che utilizza immediatamente per verificare e autenticare il messaggio Finished. Se la procedura si conclude con successo, allora invia:
   - l'avviso **ChangeCipherSpec**;
   - il messaggio **Finished**, analogo a quello del client.

La fase di handshake si conclude con il client che a sua volta verifica e autentica il messaggio Finished. Se non si riscontrano errori, il canale sicuro è stato instaurato e la comunicazione può continuare utilizzando l'algoritmo di cifratura autenticata concordato e la chiave segreta scambiata. Il protocollo completo è rappresentato a sinistra in Figura 1.

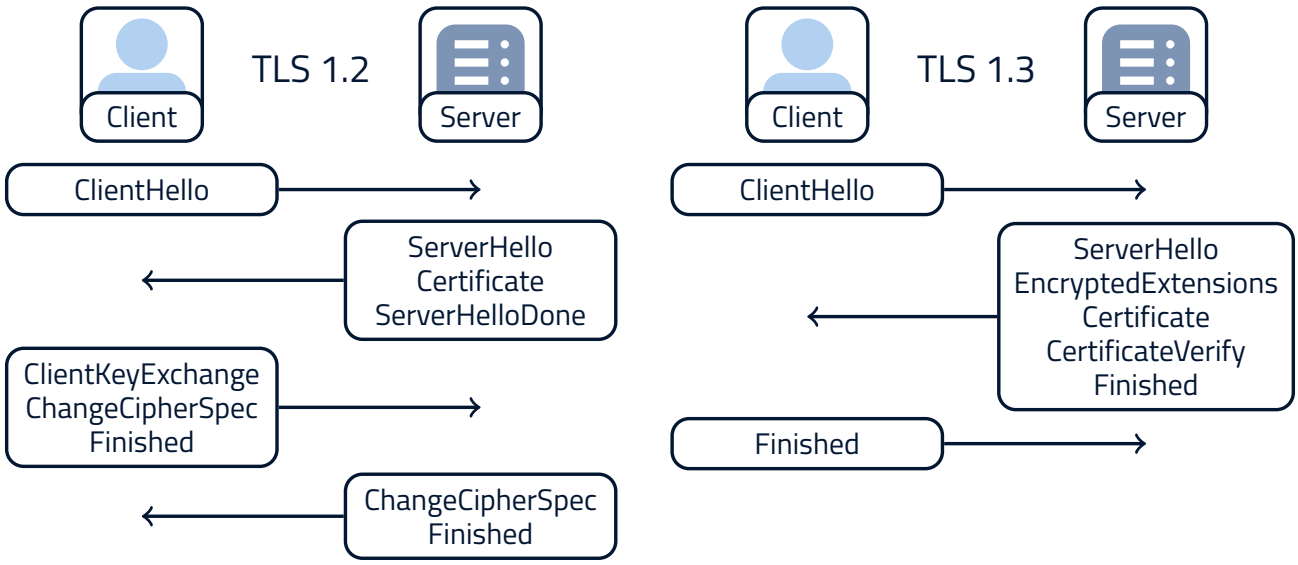


Figura 1 - Handshake con TLS 1.2 a sinistra e TLS 1.3 a destra

2.2 Handshake con TLS 1.3

TLS 1.3 presenta diversi miglioramenti rispetto alla versione precedente. Tra i più importanti vi è la dismissione di schemi vulnerabili o non adatti a garantire la **Perfect Forward Secrecy** (PFS). La PFS assicura che, nel caso in cui una chiave privata a lungo termine venga compromessa, le precedenti comunicazioni cifrate risultino comunque sicure. Senza tale precauzione, la sicurezza del protocollo dipende unicamente da quella delle chiavi private che, una volta ottenute, permetterebbero a chiunque di decifrare anche le comunicazioni passate. In particolare, quando si utilizzano i classici schemi per lo scambio di chiave, per esempio Diffie-Hellman (DH) su campi finiti o curve ellittiche (ECDH), si può ottenere la PFS semplicemente considerando le chiavi scambiate come **effimere**, cioè generate per una singola cifratura o sessione e valide solamente per quel singolo utilizzo.

In aggiunta, è stata velocizzata la fase di handshake riducendo le comunicazioni necessarie. Con TLS 1.3, non è più necessario concordare gli schemi per scambio di chiave e firma, che vengono già proposti dal client durante l'invio del primo messaggio. Inoltre, l'autenticazione è stata unita alla cifratura simmetrica in un unico algoritmo di **Cifratura Autenticata con dati associati** (AEAD, Authenticated Encryption with Associated Data), mentre tutte le funzioni di derivazione di chiave si basano sulla primitiva **HMAC-based Key Derivation Function** (HKDF) [8], che richiede solamente la scelta di una funzione di hash.

La fase di handshake per TLS 1.3 prevede i seguenti passi:

1. il client invia il proprio **ClientHello**, che contiene un valore casuale per generare una chiave simmetrica condivisa, la versione del protocollo e la lista di algoritmi AEAD supportati e delle funzioni di hash da utilizzare nella HKDF. Inoltre, il messaggio contiene una lista di valori, detti **keyshare**, per la generazione della chiave condivisa tramite i possibili schemi di scambio di chiave supportati;
2. il server risponde inviando più messaggi:
   - sceglie i parametri della suite di cifratura e della versione del protocollo appropriati, un valore casuale per generare la chiave simmetrica condivisa e racchiude questi dati nel **ServerHello**. In base alle keyshare fornite dal client, include anche la propria keyshare per uno degli schemi di scambio di chiave supportati. Queste determinano la chiave segreta condivisa che verrà utilizzata per cifrare tutti i messaggi che seguono;
   - un messaggio **EncryptedExtensions**, contenente la lista delle estensioni (si veda il paragrafo successivo) che non sono necessarie per avviare il protocollo crittografico, ma che devono comunque essere protette per la comunicazione;
   - il suo **Certificate**, che consente al client di identificarlo;
   - **CertificateVerify**, cioè la firma dei messaggi scambiati finora tramite la chiave privata relativa alla chiave pubblica contenuta nel certificato appena inviato;
   - se previsto dall'istanza del protocollo, la richiesta del certificato del client (CertificateRequest);
   - il messaggio **Finished**, contenente un MAC per tutti i messaggi scambiati come prova di autenticazione. La chiave per questo MAC viene generata utilizzando l'algoritmo HKDF;
3. infine, il client può generare la chiave segreta condivisa utilizzando i keyshare, controllare la validità dei messaggi ricevuti e inviare a sua volta:
   - il proprio certificato e la firma di verifica, se richiesti dal server;
   - il messaggio **Finished**, per garantire la propria identità.

A questo punto, se nessuna delle verifiche ha prodotto esito negativo, si può avviare la comunicazione cifrata e autenticata. Il protocollo viene confrontato con la versione 1.2 nella Figura 1.

2.3 Estensioni

Uno dei metodi principali per personalizzare il protocollo TLS in base alle varie situazioni e richiedere funzionalità aggiuntive è l'utilizzo delle **estensioni** [9]. In particolare, durante la fase di handshake, il client indica tutte le estensioni che supporta nel ClientHello, così che il server possa poi selezionare quali attivare tra queste e specificarle nel ServerHello. Può accadere che l'assenza di supporto per alcune estensioni possa risultare in un errore nel protocollo.

2.4 Rinegoziazione

La **rinegoziazione** (renegotiation) è una procedura prevista nella versione 1.2 di TLS nel momento in cui una sessione di comunicazione scade ed è necessario ripetere la fase di handshake per reinstaurare un canale sicuro.

Essa può avvenire su richiesta del server o del client.

Per quanto riguarda la rinegoziazione in TLS 1.2, si **raccomanda** di consentire l'utilizzo dell'estensione dedicata [10] solamente se tale richiesta proviene da un **server**.
Al contrario, si **sconsiglia** di eseguire una rinegoziazione richiesta da un **client** per non introdurre vulnerabilità agli attacchi descritti nella sezione 4.4.

Tuttavia esistono attacchi che sfruttano la mancanza di legame tra le credenziali stabilite in diversi handshake per la stessa sessione (descritti nel capitolo 4). Per questo motivo è importante seguire delle operazioni specifiche e attivare l'estensione dedicata alla rinegoziazione [10]. Adottando questa estensione, le nuove credenziali dipendono da quelle vecchie in quanto viene utilizzato il contenuto dei messaggi Finished, che permette di ricavare quando è avvenuta l'ultima negoziazione.
Nella versione 1.3 di TLS la possibilità di una rinegoziazione è stata rimossa. In particolare, quando un server riceve una richiesta di rinegoziazione, la connessione viene automaticamente interrotta.

2.5 Ripresa della sessione

Si parla di **ripresa della sessione** (resumption) quando server e client vogliono riconnettersi dopo una prima comunicazione e, per risparmiare tempo, vorrebbero evitare la fase di handshake.
In TLS 1.2 questo può essere effettuato in vari modi:

- inserendo all'interno del ClientHello un valore che consente di identificare l'utente e che viene memorizzato dal server, in modo che in una seconda riconnessione si possa ignorare la fase di handshake e riutilizzare le credenziali utilizzate durante la prima connessione. Se il server non dovesse riconoscere l'identificativo inviato dal client, si procederebbe a una nuova fase di handshake;
- utilizzando un'estensione dedicata [11] che prevede l'introduzione dei **ticket** di sessione. Questi "biglietti", cifrati e con un meccanismo per verificarne l'integrità, vengono generati dal server e condivisi con il client prima del messaggio ChangeCipherSpec. Una volta ricevuto, il client deve allegare il ticket a ogni riconnessione successiva, così che il server possa verificarne la validità. Il server può sempre richiedere una nuova fase di handshake oppure aggiornare il valore con un nuovo ticket da inviare nuovamente al client.

In TLS 1.3 è possibile riprendere una sessione utilizzando una PSK. In particolare, da questa si deriva una chiave, eventualmente specificandone una validità temporale, da utilizzare per l'identificazione nelle sessioni successive tra client e server. In questo modo, non serve rinegoziare gli algoritmi nella fase di handshake, ma sarà comunque necessario utilizzare uno schema per lo scambio di chiave effimero per proteggere la nuova sessione. Il server può comunque richiedere una nuova fase di handshake se lo ritiene necessario o se la chiave di autenticazione iniziale risulta scaduta.

2.6 0-RTT

Una delle modifiche introdotte in TLS 1.3 è la possibilità di ridurre il numero di round quando si utilizza una PSK proveniente da una precedente fase di handshake o calcolata internamente. In questo caso, il client può aggiungere direttamente dei dati da inviare durante la prima interrogazione al server, cifrandoli con la PSK. Questo procedimento assume il nome di **0-RTT**. Il resto della fase di handshake prosegue come nel caso generale.

La sicurezza dei dati inviati con il 0-RTT è inferiore rispetto a quella del classico TLS, in quanto la cifratura avviene solamente utilizzando la PSK ottenuta in precedenza e quindi i dati non sono protetti dalla PFS. Inoltre, questo scenario risulta più sensibile ai replay attacks (descritti nel capitolo 4), in quanto i dati inviati all'inizio non dipendono dal ServerHello. Per questi motivi, l'utilizzo di 0-RTT è **sconsigliato**.

3 Suite crittografiche

All'interno del protocollo TLS è previsto l'utilizzo di diverse funzioni crittografiche, in particolare per effettuare lo scambio di chiave e garantire la confidenzialità e l'integrità dei dati in transito. Queste funzioni sono organizzate in cosiddette **suite crittografiche**. Come descritto precedentemente, nella fase di handshake di una sessione TLS, il client presenta la lista delle suite supportate e in seguito il server ne seleziona una da utilizzare. Per i dettagli delle singole funzioni crittografiche si faccia riferimento ai rispettivi documenti dedicati [12, 13, 14, 15, 16, 17]. Fino alla versione 1.2 di TLS, la suite crittografica comprende funzioni per: scambio della chiave segreta, metodi di autenticazione (firma digitale), cifrari a blocchi o a flusso, codici di autenticazione dei messaggi (MAC). Invece, come già descritto nel capitolo precedente, TLS 1.3 ha unito le primitive di cifratura e autenticazione, richiedendo solo metodi di AEAD. Inoltre, la parte riguardante scambio di chiave e autenticazione del messaggio è stata rimossa dalla scelta nella suite crittografica, permettendo di ridurre il numero di scambi tra client e server.

3.1 Suite crittografiche per TLS 1.2

In TLS 1.2, le suite crittografiche contengono quattro funzioni che rendono sicuro il canale durante tutte le comunicazioni. Il formato standard viene indicato comunemente con IANA, dalla Internet Assigned Numbers Authority che coordina tutti gli identificativi globali di internet. Il nome di una suite crittografica in questo formato si presenta come:

TLS_scambio_autenticazione_WITH_cifratura_hash,

al cui interno sono presenti i nomi delle funzioni utilizzate. Un esempio può essere:

TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256.

Tuttavia, in alcuni contesti vengono utilizzati formati differenti. La libreria OpenSSL [18] usa un altro formato molto comune, che prevede dei nomi più corti, del tipo:

scambio-autenticazione-cifratura-hash,

con il quale l'esempio precedente diventa:

ECDHE-RSA-AES128-GCM-SHA256,

anche se in alcuni casi parte del nome viene omessa. Al fine di evitare ambiguità tra i diversi formati, IANA ha associato univocamente a ogni suite crittografica un valore di due byte espresso in notazione esadecimale [19]. Nel seguito, tutte le suite crittografiche considerate verranno nominate utilizzando solamente il formato standard e valore IANA. Secondo gli standard IETF [4, 20, 21, 22], le funzioni sono:

- scambio di chiave e autenticazione:
  - RSA, in cui il server fornisce la propria chiave pubblica tramite il certificato e questa viene utilizzata dal client per cifrare il segreto da condividere;

  - DH_RSA, cioè scambio di chiave Diffie-Hellman su campi finiti con firma RSA [23];
  - DH_DSS, Diffie-Hellman su campi finiti con il Digital Signature Standard cioè Digital Signature Algorithm (DSA) sui campi finiti [23];
  - ECDH_RSA, cioè scambio di chiave Diffie-Hellman su curve ellittiche con firma RSA;
  - ECDH_ECDSA, Diffie-Hellman su curve ellittiche con DSA su curve ellittiche [23];
  - DHE_RSA, DHE_DSS, ECDHE_RSA, ECDHE_ECDSA, analoghi ai precedenti ma con chiavi effimere;
  - PSK, che indica l'utilizzo di una chiave scambiata precedentemente [6];
  - RSA_PSK, che indica l'utilizzo di RSA per lo scambio di un nuovo segreto da utilizzare assieme alla PSK per ottenere una nuova chiave segreta condivisa;
  - DHE_PSK, ECDHE_PSK, analoghi al precedente ma con l'utilizzo di DHE o ECDHE;
  - SRP, cioè Secure Remote Password, uno scambio di chiave autenticato tramite password [24];
  - SRP_RSA, SRP_DSS, cioè SRP con firma RSA o DSS;
- cifratura simmetrica:
  - 3DES, standardizzato come TDEA dal NIST, seppure risulti obsoleto dal 2024 [25];
  - AES in modalità CBC, CCM [26, 27] o GCM [28, 29];
  - Camellia [30] (standard utilizzato in Giappone) in modalità CBC o GCM;
  - ARIA [31] (standard utilizzato in Corea del Sud) in modalità CBC o GCM;
  - SEED [32] (standard utilizzato in Corea del Sud) in modalità CBC;
  - ChaCha20-Poly1305 [33];
- hash da usare per HMAC: SHA256, SHA384.

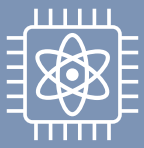
3.2 Suite crittografiche per TLS 1.3

TLS 1.3 considera solo alcune delle funzioni utilizzate nella versione 1.2 [5] e inoltre introduce soluzioni post-quantum:

- scambio di chiave: ML-KEM, DHE, ECDHE, PSK o una combinazione tra questi;
- firma digitale: ECDSA, EdDSA, RSA;
- cifratura autenticata: ChaCha20-Poly1305, AES in modalità CCM o GCM;
- hash da usare per HMAC: SHA256, SHA384.

Considerando anche che gli schemi di scambio di chiave e firma vengono proposti direttamente dal client, le suite crittografiche disponibili per TLS 1.3 sono solo cinque:

- TLS_AES_128_GCM_SHA256 (IANA: 13 01);
- TLS_AES_256_GCM_SHA384 (IANA: 13 02);
- TLS_CHACHA20_POLY1305_SHA256 (IANA: 13 03);
- TLS_AES_128_CCM_SHA256 (IANA: 13 04);
- TLS_AES_128_CCM_8_SHA256 (IANA: 13 05), nella quale si tronca il tag di autenticazione a 8 byte.



Minaccia quantistica

Con gli ultimi aggiornamenti alle suite di TLS 1.3, si introduce la resistenza agli attacchi perpetrati da computer quantistici.
La principale attenzione ricade sullo scambio di chiave, in quanto primitiva soggetta allo scenario "harvest now, decrypt later", secondo cui un attaccante in grado di intercettare le comunicazioni ora sarà in grado di ricavare le chiavi crittografiche trasmesse una volta che computer quantistici sufficiente potenti saranno disponibili. La soluzione è adottare uno **scambio di chiave ibrido**, ottenuto concatenando le chiavi scambiate con uno schema classico e ML-KEM [34, 35].
Invece, per gli schemi di firma digitale, sorgono complicazioni dovute alle possibili strategie di ibridazione [36] e alle dimensioni di chiavi pubbliche e firme generate. Di conseguenza, non esistono ancora standard per l'ibridazione di schemi di firma post-quantum con schemi classici, seppur nell'ambito dei certificati X.509 si è già provveduto all'aggiunta di ML-DSA [37], SLH-DSA [38], LMS e XMSS [39].
Il presente documento verrà **aggiornato** tempestivamente in base agli sviluppi degli standard riguardanti crittografia post-quantum, meccanismi di ibridazione e applicazioni a TLS 1.3.

4 Attacchi a TLS

Questo capitolo descrive le principali tipologie di attacco a TLS, fornendo per ognuna di esse una panoramica sugli attacchi più conosciuti. Per ulteriori approfondimenti si rimanda al documento fornito da IETF [40]. Verranno approfondite solamente le vulnerabilità riguardanti il protocollo, mentre per quelle dei sistemi crittografici sottostanti, qualora insicuri e considerati oramai obsoleti, si rimanda ai documenti dedicati [12, 13, 14, 15, 16, 17].

4.1 Padding oracle attack

Quando per ottenere una cifratura autenticata si affianca un algoritmo di sola cifratura (ad esempio un cifrario a blocchi in modalità CBC) alla generazione di un MAC, è importante evitare il paradigma MAC-then-Encrypt in quanto risulta vulnerabile rispetto al **padding oracle attack**. Questa tipologia di attacco ricava informazioni dai messaggi di errore inviati dal server in caso di padding non validi.
Nella storia di TLS ci sono stati alcuni attacchi specifici che si basano sullo stesso principio. Uno di questi è **ROBOT** (Return Of Bleichenbacher's Oracle Threat), il quale si basa su una vulnerabilità di SSL scoperta da Daniel Bleichenbacher nel 1998 [41]. In particolare, se si utilizza lo schema RSA standardizzato nella versione 1.5 di PKCS#1 (ritenuto ormai obsoleto), il messaggio d'errore fornito in caso di padding non valido dal server SSL può essere sfruttato per un CCA (Chosen-Ciphertext Attack) che rompe completamente la confidenzialità dello schema RSA.
Un famoso attacco, composto in parte da un padding oracle e in parte da un side-channel, è il cosiddetto **Lucky Thirteen**. L'idea alla base consiste nello sfruttare il fatto che, nelle fasi di decifratura di TLS 1.1 e 1.2, la verifica del MAC si esegue anche quando si riceve un messaggio con un formato di padding non valido. Questo serve per evitare vulnerabilità ad attacchi side-channel, ma è importante scegliere bene i dati che vengono controllati. Le vecchie versioni del protocollo prevedevano la verifica di validità di una stringa di soli zeri, la quale comporta una differenza sui tempi di risposta d'errore da parte del server rispetto al caso generale. Questa variazione, seppur minima, può essere sfruttata per ottenere i messaggi in chiaro tramite analisi statistica sui dati intercettati [42].
Un'altra importante minaccia basata sul padding oracle attack è il POODLE, descritto nella sezione seguente in quanto correlato a un'altra tipologia di attacco.
Una soluzione immediata per questo tipo di attacchi è quella di attivare sempre l'estensione che introduce il paradigma **Encrypt-then-MAC** (EtM) per TLS 1.2 [43] oppure un metodo di cifratura autenticata.

4.2 Downgrade attacks

Gli attacchi di tipo **downgrade** sono tutti quelli in cui l'attaccante riesce, durante la fase di handshake, a forzare l'utilizzo di una versione più vecchia e quindi generalmente meno sicura del protocollo. In particolare, per fare ciò,

l'attaccante intercetta un handshake tra un client e un server e si finge quest'ultimo, modificando i messaggi scambiati in questa fase. Questi scenari di attacco, in cui il malintenzionato si intromette tra server e client, vengono chiamati attacchi **man-in-the-middle**. Dopo aver manipolato la comunicazione in tal modo, l'attaccante può sfruttare attacchi esistenti alla versione di TLS che verrà utilizzata e compromettere dunque la sicurezza della conversazione. La contromisura più semplice per questo tipo di attacchi è rimuovere completamente versioni e funzioni crittografiche più deboli, in modo da non poter essere negoziate in nessun modo nella fase di handshake.
Uno degli attacchi downgrade più diffusi è il **POODLE** (Padding Oracle On Downgraded Legacy Encryption) già citato nella sezione precedente. In questo caso, l'attaccante forza l'utilizzo di SSL 3.0, ormai deprecato a causa di varie vulnerabilità. L'attacco POODLE sfrutta quindi un padding oracle attack per rompere facilmente i cifrari a blocchi in modalità CBC utilizzati da SSL 3.0 (RC2, DEA, IDEA, TDEA, FORTEZZA) [44].
Un altro attacco di questo tipo è il **FREAK** (Factoring RSA Export Keys) [45], che sfrutta l'utilizzo nelle versioni commerciali di TLS di chiavi RSA da 512 bit o inferiori, chiamate appunto export keys. Data la dimensione ridotta, queste chiavi pubbliche sono vulnerabili agli algoritmi di fattorizzazione che, ad esempio, possono essere implementati su servizi di cloud computing per ricavare efficientemente la chiave privata. Effettuando quindi un downgrade attack di tipo man-in-the-middle è possibile costringere server e utente a negoziare queste dimensioni di chiavi e ricavare la chiave privata, per poi recuperare il contenuto dei messaggi cifrati.
Un attacco analogo a FREAK è il **Logjam** [46], basato sempre sulla rottura di export keys da 512 bit inserite nelle versioni commerciali di TLS, ma per lo scambio di chiave con lo schema Diffie-Hellman. Queste chiavi pubbliche sono suscettibili ad attacchi che risolvono il problema del logaritmo discreto, ad esempio utilizzando dei servizi computazionali dedicati, e quindi ricavano in tempi rapidi le chiavi segrete scambiate. Inoltre, dato che il gruppo e il generatore utilizzati nello schema Diffie-Hellman sono spesso scelti tra una lista di opzioni standard, è possibile effettuare delle precomputazioni che riducono ulteriormente i tempi di calcolo dell'attacco.

L'ultimo caso di downgrade attack su cui si pone l'attenzione è l'attacco **DROWN** (Decrypting RSA with Obsolete and Weakened eNcryption) [47]. L'attaccante costringe server e utente a rinegoziare la versione 2.0 di SSL e di scegliere RSA come schema per lo scambio di chiave. In questo caso, la versione supportata è la 1.5 di PKCS #1 e quindi si riesce a effettuare un CCA di tipo Bleichenbacher, analogamente a quanto descritto per ROBOT.
La soluzione migliore per mitigare questo tipo di attacchi è **rimuovere le versioni obsolete** come SSL 2.0 da quelle supportate dal protocollo.

4.3 Compression attacks

Gli attacchi di compressione sono una tipologia di attacchi side-channel che sfruttano una funzionalità opzionale di TLS che permette di comprimere i dati trasferiti dal server al browser.
Uno dei primi attacchi di questa tipologia è **CRIME** (Compression Ratio Info-leak Made Easy) [48], proposto nel 2012, con il quale un attaccante prova a decifrare i cookie di autenticazione di un utente durante una connessione con un'applicazione web tramite HTTP. Una volta ottenuti i dati in essi contenuti, l'attaccante può sfruttarli per accedere all'applicazione web al posto dell'utente e rubare dati sensibili. Più in dettaglio, l'attaccante induce il browser del client a inviare all'applicazione una serie di richieste HTTP in cui cerca di indovinare il cookie di autenticazione della vittima. Osservando la lunghezza dei cookie compressi e cifrati, l'attaccante è in grado di derivare uno dopo l'altro tutti i caratteri del cookie di autenticazione. Si tratta quindi di un CPA (Chosen-Plaintext Attack) e inoltre è necessario che l'attaccante sia in grado almeno di misurare la dimensione dei messaggi cifrati, rendendolo un man-in-the-middle. Questo rende CRIME più complicato da perpetrare e in aggiunta, seppur efficace nelle condizioni descritte, si tratta di un attacco facilmente mitigabile: è sufficiente, ad esempio, **disabilitare la compressione** nel protocollo TLS.
Nel 2013, è stato pubblicato un nuovo attacco, denominato **TIME** (Timing Info-leak Made Easy) [49], che migliora CRIME proprio sugli aspetti descritti, che lo rendevano poco applicabile nella pratica. L'attacco TIME cambia il bersaglio dell'attacco dalle richieste HTTP da parte del browser alle risposte HTTP del sito web. In questo modo, la possibile

superficie d'attacco aumenta in modo significativo in quanto, almeno ai tempi dell'attacco, l'utilizzo della compressione da parte dei server era molto più frequente al fine di risparmiare banda e diminuire la latenza. Inoltre, il nuovo attacco non analizza la dimensione delle risposte inviate, bensì misura il tempo impiegato nella trasmissione. In questo modo, seppur l'attaccante debba ancora controllare il testo in chiaro (CPA), l'attacco non deve necessariamente essere un man-in-the-middle in quanto non serve intercettare i messaggi scambiati. Tuttavia, come per CRIME, per mitigare l'effetto di TIME è sufficiente **disabilitare la compressione** in TLS.
Un ulteriore miglioramento degli attacchi descritti è **BREACH** (Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext) [50], il quale procede in modo analogo a CRIME ma sfrutta solamente le compressioni eseguite nelle risposte del protocollo HTTP. A differenza di CRIME e TIME, disabilitare la compressione di TLS non è più sufficiente per mitigare l'attacco. In particolare, tutte le soluzioni conosciute attualmente riguardano precauzioni a livello applicativo e al momento non esistono contromisure a livello del protocollo TLS.

4.4 Renegotiation attack

La **rinegoziazione** è una procedura prevista nel protocollo TLS nel momento in cui una sessione di comunicazione scade ed è necessario ripetere la fase di handshake per reinstaurare un canale sicuro. Tuttavia, senza le necessarie precauzioni, un attaccante potrebbe creare una connessione con il server, fornire i dati malevoli, e infine inserire una nuova connessione da un client ignaro. In questo modo, un server vulnerabile tratta la fase di handshake del nuovo client come una rinegoziazione e quindi ritiene che i dati iniziali trasmessi dall'aggressore provengano dalla stessa entità dei dati client successivi. La soluzione consiste nel collegare indissolubilmente la rinegoziazione con la connessione TLS nella quale avviene [10]. Questo si ottiene introducendo una nuova estensione chiamata **renegotiation_info** da includere nel ClientHello e nel ServerHello. Nel primo handshake, questa assume il valore di una stringa vuota mentre, in eventuali rinegoziazioni future, ad essa vengono associati dei dati di verifica dipendenti da chi sta facendo ripartire la sessione.

4.5 Exhaustion attack

Gli attacchi del tipo **Distributed Denial of Service** (DDoS) mirano a esaurire le risorse del server, rallentando o bloccando il traffico dati e stanno diventando sempre più frequenti negli ultimi anni. Un tipo comune di attacco ottiene questo risultato simulando numerosi client che tentano di instaurare connessioni con il server. Poiché, come spiegato in precedenza, la fase di handshake richiede uno scambio di messaggi tra client e server, il server viene sovraccaricato da un numero elevato di richieste che restano aperte per lunghi periodi, causando un rallentamento o l'interruzione del servizio.
Per contrastare questi attacchi, esistono diverse contromisure: limitare il numero di connessioni simultanee gestibili dal server e impostare una durata massima per le sessioni di handshake. Inoltre, vari provider di sicurezza offrono strumenti avanzati per rilevare e bloccare il traffico anomalo, prevenendo così l'impatto degli attacchi DDoS.

5 Conclusioni

La scelta della versione di TLS e della suite crittografica da utilizzare non si può risolvere semplicemente scegliendo l'opzione più moderna e sicura. Purtroppo, nei contesti odierni, esistono ancora molti utenti che non possono aggiornare i propri sistemi alle ultime versioni. Di conseguenza, è sempre importante effettuare uno studio sul contesto specifico che consenta di raggiungere un equilibrio tra sicurezza e compatibilità.

In ogni caso, per quanto detto nelle sezioni precedenti, qualsiasi versione di SSL e le versioni 1.0 e 1.1 di TLS sono considerate **obsolete** e quindi devono essere evitate.

Le versioni raccomandate sono TLS 1.3 e TLS 1.2, dove la prima è da preferire alla seconda, in quanto molte delle suite di TLS 1.2 sono attualmente considerate insicure.

Di seguito, si elencano le configurazioni raccomandate suddividendole in base al livello di compatibilità desiderato, tenendo anche in considerazione le valutazioni a livello nazionale [51] e internazionale [52, 53, 54, 55, 56, 57].

5.1 Configurazioni raccomandate

A meno di avere vincoli dovuti alla compatibilità, si raccomanda l'utilizzo di TLS 1.3 con le suite crittografiche raccolte nella Tabella 1.

Per quanto riguarda gli schemi non appartenenti alla suite di cifratura e i relativi parametri, la Tabella 2 contiene le descrizioni IANA [58] degli schemi raccomandati e accettati. Nello specifico:

- per lo scambio di chiave, come dettagliato nella sezione 3.2, al fine di resistere ad attacchi della tipologia "harvest now, decrypt later", si raccomanda l'utilizzo di schemi ibridi, cioè **X25519MLKEM768**, **SecP256r1MLKEM768** e **SecP384r1MLKEM1024** dati dall'ibridazione di **ML-KEM** con **ECDHE**, utilizzando rispettivamente i parametri 768 e le curve Curve25519 o secp256r1, oppure 1024 e la curva secp384r1.
  Gli schemi classici ECDHE e DHE [59] sono considerati accettabili con i parametri elencati in Tabella 2;

Versione	Suite crittografica	Valore IANA
TLS 1.3	TLS_AES_128_GCM_SHA256	13 01
	TLS_AES_256_GCM_SHA384	13 02
	TLS_CHACHA20_POLY1305_SHA256	13 03
	TLS_AES_128_CCM_SHA256	13 04

Tabella 1 - Configurazioni di TLS raccomandate

Schema	Raccomandati	Accettati
Scambio di chiave	X25519MLKEM768 SecP256r1MLKEM768 SecP384r1MLKEM1024	secp256r1 secp384r1 secp521r1 brainpoolP256r1tls13 brainpoolP384r1tls13 brainpoolP512r1tls13 x25519 x448 ffdhe3072 ffdhe4096 ffdhe6144 ffdhe8192
Firma digitale	ecdsa_secp256r1_sha256 ecdsa_secp384r1_sha384 ecdsa_secp521r1_sha512 ecdsa_brainpoolP256r1tls13_sha256 ecdsa_brainpoolP384r1tls13_sha384 ecdsa_brainpoolP512r1tls13_sha512 ed25519 ed448	rsa_pss_rsae_sha256 rsa_pss_rsae_sha384 rsa_pss_rsae_sha512 rsa_pss_pss_sha256 rsa_pss_pss_sha384 rsa_pss_pss_sha512 Solo per la firma della CA sui certificati: rsa_pkcs1_sha256 rsa_pkcs1_sha384 rsa_pkcs1_sha512

Tabella 2 - Descrizioni IANA degli schemi crittografici raccomandati e accettati in TLS 1.3

- per la firma digitale, si raccomandano **ECDSA** con curve secp256r1, secp384r1, secp521r1, brainpoolP256r1, brainpoolP384r1 o brainpoolP512r1, oppure **EdDSA** con Curve25519 o Curve448. La firma RSA è accettata nella versione PSS con chiave pubblica di almeno 3072 bit. Lo stesso vale per la versione PKCS1, che tuttavia è considerata accettabile solo come schema di firma utilizzato dalle CA per autenticare i certificati dei server, anche tenendo in considerazione che TLS 1.3 non ne consente l'utilizzo in fase di handshake.

5.2 Configurazioni retrocompatibili

Se le configurazioni raccomandate risultano troppo restrittive, ad esempio se serve un compromesso tra sicurezza e compatibilità per comunicare con un bacino di utenti più ampio, è considerato accettabile aggiungere alle configurazioni descritte nella sezione precedente la versione 1.2 di TLS con le suite crittografiche raccolte in Tabella 3. Quando si utilizza AES, le opzioni raccomandate per la cifratura autenticata sono le modalità **GCM** e **CCM**, si accetta anche **CBC** solo con **Encrypt-then-MAC**.
Si raccomandano inoltre i seguenti parametri:

- scambio di chiave: per ECDHE, le curve secp256r1, secp384r1, secp521r1, brainpoolP256r1, brainpoolP384r1, brainpoolP512r1, Curve25519 o Curve448, mentre per DHE, i campi finiti predeterminati [59] con cardinalità lunga almeno 2048 bit;
- firma digitale: per ECDSA, le curve secp256r1, secp384r1, secp521r1, brainpoolP256r1, brainpoolP384r1 o brainpoolP512r1 mentre per RSA si intendono le firme RSASSA-PKCS1 o RSASSA-PSS con chiave pubblica di almeno 2048 bit.

Versione	Suite crittografica	Valore IANA
TLS 1.2	TLS_DHE_RSA_WITH_AES_128_CBC_SHA256	00 67
	TLS_DHE_RSA_WITH_AES_256_CBC_SHA256	00 6B
	TLS_DHE_RSA_WITH_AES_128_GCM_SHA256	00 9E
	TLS_DHE_RSA_WITH_AES_256_GCM_SHA384	00 9F
	TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256	C0 23
	TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384	C0 24
	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256	C0 27
	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384	C0 28
	TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256	C0 2B
	TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384	C0 2C
	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	C0 2F
	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	C0 30
	TLS_DHE_RSA_WITH_AES_128_CCM	C0 9E
	TLS_DHE_RSA_WITH_AES_256_CCM	C0 9F
	TLS_ECDHE_ECDSA_WITH_AES_128_CCM	C0 AC
	TLS_ECDHE_ECDSA_WITH_AES_256_CCM	C0 AD
	TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256	CC A8
	TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256	CC A9
	TLS_DHE_RSA_WITH_CHACHA20_POLY1305_SHA256	CC AA

Tabella 3 - Configurazioni retrocompatibili accettate

5.3 Estensioni raccomandate

Come ribadito in più occasioni, se si ottiene la cifratura autenticata combinando un algoritmo di cifratura sicuro con un MAC, è fondamentale impiegare il paradigma **Encrypt-then-MAC**.

Questa soluzione può essere integrata in TLS tramite un'estensione [43] di cui è sempre raccomandato l'utilizzo con TLS 1.2. In mancanza di tale estensione, infatti, il protocollo TLS utilizza il paradigma MAC-then-Encrypt, che risulta vulnerabile e quindi non è raccomandato in nessun contesto.

Una seconda estensione raccomandata per TLS 1.2 è la **extended master secret** [60]. Questa estensione descrive un modo più robusto per calcolare la chiave principale (master key) da cui derivare le singole chiavi di sessione, inglobando il digest di tutti i messaggi scambiati durante la fase di handshake.

Come anticipato anche in precedenza, nel caso in cui si intenda consentire la rinegoziazione dei parametri, si raccomanda l'utilizzo dell'estensione **renegotiation_info** [10]. Inoltre, si raccomanda di consentire la rinegoziazione in TLS 1.2 solamente quando la richiesta proviene da un server, mentre in caso di richiesta da parte del client si raccomanda di procedere avviando una nuova connessione.

5.4 Estensioni sconsigliate

L'estensione **Heartbeat** [61] permette di mantenere la connessione attiva tra client e server senza dover nuovamente ricorrere a una fase di handshake. Tuttavia, un bug nell'implementazione di OpenSSL è stato responsabile di una importante vulnerabilità nota con il nome di **Heartbleed**, la quale permette a un attaccante di accedere ad aree della memoria di un server in cui possono essere contenute informazioni riservate, come alcune chiavi private

tra cui quella del server stesso. Per questo motivo l'estensione Heartbeat è sconsigliata.
Una delle prime estensioni introdotte in TLS è chiamata **Truncated HMAC** [9] e consente di utilizzare solamente i primi 80 bit del MAC utilizzato in fase di autenticazione, scartando i bit rimanenti. Lo scopo di questa estensione era risparmiare banda durante la trasmissione dei dati, ma questo introduce una consistente riduzione del livello di sicurezza. Di conseguenza, l'estensione Truncated HMAC è sconsigliata.

Nel caso in cui si utilizzino delle librerie che implementano il protocollo TLS, si raccomanda di verificare che le estensioni riportate risultino abilitate (o disabilitate) oppure di contattare i responsabili del servizio per farle attivare (o disattivare).

Bibliografia

[1] ACN. *Introduzione alla Crittografia e alle Linee Guida*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

[2] T. Dierks e C. Allen. *The TLS Protocol, Version 1.0*. RFC 2246. 1999. DOI: 10.17487/RFC2246.

[3] T. Dierks e E. Rescorla. *The Transport Layer Security (TLS) Protocol, Version 1.1*. RFC 4346. 2006. DOI: 10.17487/RFC4346.

[4] T. Dierks e E. Rescorla. *The Transport Layer Security (TLS) Protocol, Version 1.2*. RFC 5246. 2008. DOI: 10.17487/RFC5246.

[5] E. Rescorla. *The Transport Layer Security (TLS) Protocol, Version 1.3*. RFC 8446. 2018. DOI: 10.17487/RFC8446.

[6] P. Eronen e H. Tschofenig. *Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)*. RFC 4279. 2005. DOI: 10.17487/RFC4279.

[7] ITU-T. *Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks*. X.509. 2019. URL: https://www.itu.int/rec/t-rec-x.509.

[8] H. Krawczyk e P. Eronen. *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. RFC 5869. 2010. DOI: 10.17487/RFC5869.

[9] D. E. Eastlake 3rd. *Transport Layer Security (TLS) Extensions: Extension Definitions*. RFC 6066. 2011. DOI: 10.17487/RFC6066.

[10] E. Rescorla et al. *Transport Layer Security (TLS) Renegotiation Indication Extension*. RFC 5746. 2010. DOI: 10.17487/RFC5746.

[11] J. A. Salowey et al. *Transport Layer Security (TLS) Session Resumption without Server-Side State*. RFC 5077. 2008. DOI: 10.17487/RFC5077.

[12] ACN. *Firme Digitali*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

[13] ACN. *Cifrari a Blocchi e Modalità di Funzionamento*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

[14] ACN. *Cifrari a Flusso*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

[15] ACN. *Cifratura Autenticata*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

[16] ACN. *Funzioni di Hash*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

[17] ACN. *Codici di Autenticazione di Messaggi (MAC)*. Linee Guida Funzioni Crittografiche, 2026. URL: https://www.acn.gov.it/portale/crittografia.

[18] O. Documentation. *openssl-ciphers*. 2024. URL: https://docs.openssl.org/master/man1/openssl-ciphers/.

[19] IANA. *Transport Layer Security (TLS) Parameters*. 2024. URL: https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml.

[20] A. Popov. *Prohibiting RC4 Cipher Suites*. RFC 7465. 2015. DOI: 10.17487/RFC7465.

[21] Y. Nir, S. Josefsson e M. Pégourié-Gonnard. *Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier*. RFC 8422. 2018. DOI: 10.17487/RFC8422.

[22] L. Velvindron, K. Moriarty e A. Ghedini. *Deprecating MD5 and SHA-1 Signature Hashes in TLS 1.2 and DTLS 1.2*. RFC 9155. 2021. DOI: 10.17487/RFC9155.

[23] NIST. *Digital Signature Standard (DSS)*. FIPS 186-5. U.S. Department of Commerce, 2023. DOI: 10.6028/NIST.FIPS.186-5.

[24] D. Taylor et al. *Using the Secure Remote Password (SRP) Protocol for TLS Authentication*. RFC 5054. 2007. DOI: 10.17487/RFC5054.

[25] NIST. *NIST to Withdraw Special Publication 800-67 Revision 2*. 2023. URL: https://csrc.nist.gov/news/2023/nist-to-withdraw-sp-800-67-rev-2.

[26] D. McGrew e D. Bailey. *AES-CCM Cipher Suites for Transport Layer Security (TLS)*. RFC 6655. 2012. DOI: 10.17487/RFC6655.

[27] D. McGrew et al. *AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS*. RFC 7251. 2014. DOI: 10.17487/RFC7251.

[28] J. A. Salowey, D. McGrew e A. Choudhury. *AES Galois Counter Mode (GCM) Cipher Suites for TLS*. RFC 5288. 2008. DOI: 10.17487/RFC5288.

[29] E. Rescorla. *TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)*. RFC 5289. 2008. DOI: 10.17487/RFC5289.

[30] S. Kanno e M. Kanda. *Addition of the Camellia Cipher Suites to Transport Layer Security (TLS)*. RFC 6367. 2011. DOI: 10.17487/RFC6367.

[31] W.-H. Kim et al. *Addition of the ARIA Cipher Suites to Transport Layer Security (TLS)*. RFC 6209. 2011. DOI: 10.17487/RFC6209.

[32] H. Lee, J. Yoon e J. Lee. *Addition of SEED Cipher Suites to Transport Layer Security (TLS)*. RFC 4162. 2005. DOI: 10.17487/RFC4162.

[33] A. Langley et al. *ChaCha20-Poly1305 Cipher Suites for Transport Layer Security (TLS)*. RFC 7905. 2016. DOI: 10.17487/RFC7905.

[34] D. Stebila, S. Fluhrer e S. Gueron. *Hybrid key exchange in TLS 1.3*. RFC draft. 2024. URL: https://datatracker.ietf.org/doc/draft-ietf-tls-hybrid-design.

[35] K. Kwiatkowski et al. *Post-quantum hybrid ECDHE-MLKEM Key Agreement for TLSv1.3*. RFC draft. 2026. URL: https://datatracker.ietf.org/doc/draft-ietf-tls-ecdhe-mlkem/.

[36] N. Bindel et al. *Hybrid signature spectrums*. RFC draft. 2026. URL: https://datatracker.ietf.org/doc/draft-ietf-pquip-hybrid-signature-spectrums/.

[37] J. Massimo et al. *Internet X.509 Public Key Infrastructure – Algorithm Identifiers for the Module-Lattice-Based Digital Signature Algorithm (ML-DSA)*. RFC 9881. 2025. DOI: 10.17487/RFC9881.

[38] K. Bashiri et al. *Internet X.509 Public Key Infrastructure – Algorithm Identifiers for the Stateless Hash-Based Digital Signature Algorithm (SLH-DSA)*. RFC 9909. 2025. DOI: 10.17487/RFC9909.

[39] D. V. Geest et al. *Use of the HSS and XMSS Hash-Based Signature Algorithms in Internet X.509 Public Key Infrastructure*. RFC 9802. 2025. DOI: 10.17487/RFC9802.

[40] Y. Sheffer, R. Holz e P. Saint-Andre. *Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)*. RFC 7457. 2015. DOI: 10.17487/RFC7457.

[41] D. Bleichenbacher. «Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1». In: *Advances in Cryptology - CRYPTO '98*. Lecture Notes in Computer Science. 1998, pp. 1–12. DOI: 10.1007/BFb0055716.

[42] N. J. Al Fardan e K. G. Paterson. «Lucky Thirteen: Breaking the TLS and DTLS Record Protocols». In: *IEEE Symposium on Security and Privacy*. 2013, pp. 526–540. DOI: 10.1109/SP.2013.42.

[43] P. Gutmann. *Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)*. RFC 7366. 2014. DOI: 10.17487/RFC7366.

[44] R. Barnes et al. *Deprecating Secure Sockets Layer Version 3.0*. RFC 7568. 2015. DOI: 10.17487/RFC7568.

[45] B. Beurdouche et al. «A messy state of the union: taming the composite state machines of TLS». In: *Communications of the ACM* 60.2 (2017), pp. 99–107. DOI: 10.1145/3023357.

[46] D. Adrian et al. «Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice». In: *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. CCS '15. 2015, pp. 5–17. DOI: 10.1145/2810103.2813707.

[47] N. Aviram et al. «DROWN: Breaking TLS Using SSLv2». In: *USENIX Security Symposium*. 2016. URL: https://www.usenix.org/system/files/conference/usenixsecurity16/sec16_paper_aviram.pdf.

[48] J. Rizzo e T. Duong. *The CRIME Attack*. ekoparty security conference. 2012. URL: https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU.

[49] A. Shulman. *A Perfect CRIME? Only TIME Will Tell*. Black Hat Europe. 2013. URL: https://media.blackhat.com/eu-13/briefings/Beery/bh-eu-13-a-perfect-crime-beery-wp.pdf.

[50] A. Prado, N. Harris e Y. Gluck. *The BREACH Attack*. 2013. URL: https://www.breachattack.com/.

[51] AgID. *Raccomandazioni AgID in merito allo standard Transport Layer Security (TLS)*. 2020. URL: https://cert-agid.gov.it/wp-content/uploads/2020/11/AgID-RACCSECTLS-01.pdf.

[52] ANSSI. *Recommandations de sécurité relatives à TLS*. 2020. URL: https://cyber.gouv.fr/publications/recommandations-de-securite-relatives-tls.

[53] BSI. *Technical Guideline TR-02102-2 Cryptographic Mechanisms: Recommendations and Key Lengths*. 2024. URL: https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TG02102/BSI-TR-02102-2.

[54] NCSC-NL. *IT Security Guidelines for Transport Layer Security (TLS) v2.1*. 2021. URL: https://english.ncsc.nl/publications/publications/2021/january/19/it-security-guidelines-for-transport-layer-security-2.1.

[55] CCN-PYTEC. *Requisitos de Seguridad para TLS*. 2020. URL: https://www.ccn.cni.es/es/docman/378-pildorapytec-nov2020-seguridad-tls.

[56] SOG-IS. *Crypto Evaluation Scheme Agreed Cryptographic Mechanisms v1.3*. 2023. URL: https://www.sogis.eu/documents/cc/crypto/SOGIS-Agreed-Cryptographic-Mechanisms-1.3.pdf.

[57] K. McKay e D. Cooper. *Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations*. SP 800-52. NIST, 2019. DOI: 10.6028/NIST.SP.800-52r2.

[58] I. A. N. A. (IANA). *Transport Layer Security (TLS) Parameters*. 2025. URL: https://www.iana.org/assignments/tls-parameters/tls-parameters.xml.

[59] D. K. Gillmor. *Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for Transport Layer Security (TLS)*. RFC 7919. 2016. DOI: 10.17487/RFC7919.

[60] R. Bhargavan et al. *Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension*. RFC 7627. 2015. DOI: 10.17487/RFC7627.

[61] R. Seggelmann, M. Tüxen e M. Williams. *Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension*. RFC 6520. 2012. DOI: 10.17487/RFC6520.